Differential gene expression (DGE) with salmon data and DESeq2

Transcipt quantification

transcriptome quantification attempts to estimate expression levels of indvidual transcripts. This is performed by assigning RNAseq reads to transcripts, counting, and normalization.

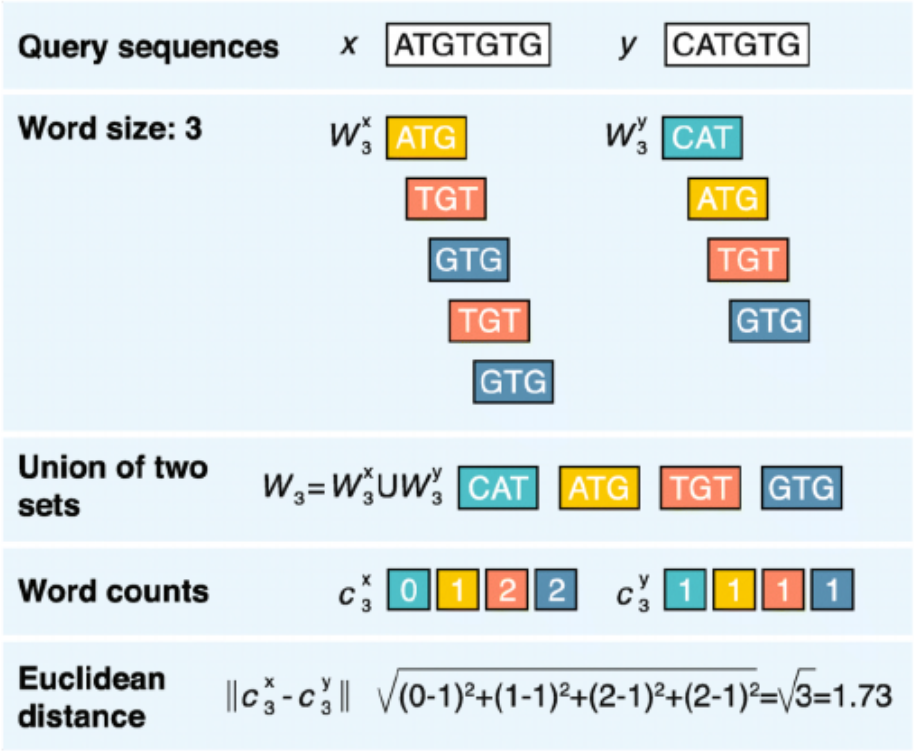
Assigning reads to transcripts

To associate reads with transcripts the reads need to be aligned to the transcriptome. Tools like Cufflinks and StringTie reconstruct transcripts from spliced read alignments generated by other programs (TopHat, HISAT, STAR), so they already have the information about which reads belong to each reconstructed transcript. Other tools such as Sailfish (http://www.cs.cmu.edu/~ckingsf/software/sailfish/), Kalisto (http://pachterlab.github.io/kallisto/) and Salmon (http://combine-lab.github.io/salmon/) perform *lightweight* alignment of RNAseq reads against existing transcriptome sequences. The goal of lightweight alignment is to quickly distribute the reads across transcripts the likely originate from without worrying to much about producing high quality alignments. The upside of this is that the entire procedure can be performed very quickly. The downside is that these tools require high quality transcriptome as inpunt, which is not a problem if you work with humans or mice.

These **alignment-free** methods follow the rationale that similar sequences share similar subsequences (k-mers or words). Counting the shared k-mer occurrences should therefore give a a good relative measure of sequences similarity, irrespective of the precise genome location. This also means that the quantitative information about expression levels of individual genes will be the immediate result of the read mapping step without the need for additional tools.

To assess the seqeunce similarity, the following steps are typically taken:

1. The sequences for comparison (reads, reference) are sliced up into collections of *unique* k-mers of a given length k.
2. For each pairwise comparison, we count the number of times a specific k-mer appears in both sequence strings that are being compared.
3. To assess the similarity between the two strings, some sort of distance function is employed, for example, Euclidean distance; two identical sequences should have a distance of zero.



In practice Salmon will first generate an index of k-mers from all known transcript sequences. These transcript k-mers will then be compared with the k-mers of the sequenced reads, yielding *a pseudoalignment* that describes how many k-mers a read shares with a set of compatible transcripts (based on the distance scores). By grouping all pseudoalignments that belong to the same set of transcripts, they can then estimate the expression level of each transcript model.

While the tools such as Kallisto and Salmon have been shown to perform almost as good as classic alignment tools on *simulatated* data, it is clear that these programs are still under active developtment Srivastava et al. 2019 (https://europepmc.org/abstract/ppr/ppr81065) and are prone to spurious alignments, particularly for lowly expressed genes.

Salmon

Salmon is a tool for quantifying the expression of transcripts using RNASeq data. Salmon uses new algorithms (specifically, coupling the concept of *quasi-mapping* with a two-phase inference procdure) to provide accurate expression estimates very quickly and while using little memory. Salmon performs its inference using an expressive and realistic model of RNAseq data that takes into account experimental attributes and biases commonly observed in *real* RNAseq data.

Salmon requires a set of target transcripts (either from a reference or *de-novo* assembly) to quantify. All you need to run Salmon is a FASTA file containing your reference transcripts and a set of FASTA/FASTQ files containing your reads.

The **mapping-based** mode of Salmon runs in two phases; indexing and quantification. The indexing step is independent of the reads, and only need to be run once for a particular set of reference transcripts.

```
# create index from HG38 FASTA cdna file
salmon index -t Homo_sapiens.GRCh38.cdna.all.fa.gz -i HG38_index
```

This will build the mapping-based index, using an auxiliary k-mer hash over k-mers of lenth 31. While the mapping algorithms will make use of arbitrarily long matches between the query and reference, the k size selected here will act as the *minimum* acceptable length for a valid match. Thus, a smaller value of k may slightly improve sensitivity.

The quantification step, obviously, is specific to the set of RNAseq reads and is thus run more frequently.

```
#!/bin/bash

# run salmon on all the Colo samples
for sample in Colo/*_2.txt.gz
do
samp=`basename ${sample}`
echo "Processing sample ${samp}"
/home/sebastian/biotools/salmon-latest_linux_x86_64/bin/salmon quant -
i HG38_index \
        --libType SF -r Colo/${samp} \
    -p 8 --validateMappings --output quants/${samp}_quant
done
```

You can, of course, pass a number of options to control things such as the number of threads (-p 8) or different cutoffs used for counting reads. If you are using singe-end reads, then you pass them to Salmon with the -r flag.

Salmon, has the user provide a description of the type of sequencing library from which the reads come, and this contains information about e.g. the relative orientation of paired end reads. From the alignment QC I concluded that we have single stranded reads from forward strand (--libType SF)?

Selective alignment, enabled by the *--validateMappins* flag, is a major feature enhancement introduced in recent versions of salmon. When salmon is run with selective alignment, it adopts a cosinderably more sensitive scheme that was developed for finding the potential mapping loci of a read, and score potential mapping loci using the chaining algorithm introduced in minimap2.

After Salmon has finished running, there will be a directory called quants, that contains folders named for every sample and inside besides other files a file called *quant.sf* containing the quantification results.

In [44]:
```
# Load an example quant file to show its content
quant_file <- read.delim("quant.sf")
head(quant_file)
```

A data.frame: 6 × 5

Name	Length	EffectiveLength	TPM	NumReads
<fct>	<int>	<dbl>	<dbl>	<dbl>
ENST00000631435.1	12	3.002	0	0
ENST00000415118.1	8	2.722	0	0
ENST00000448914.1	13	3.040	0	0
ENST00000434970.2	9	2.818	0	0
ENST00000439842.1	11	2.954	0	0
ENST00000390567.1	20	3.185	0	0

Loading libraries

For this analysis several R packages are needed, some of which have been installed from CRAN and others from Bioconductor. To use these packages (and the functions within them), we need to **load the libraries**.

In [3]:
```
# Setup
### Bioconductor and CRAN libraries used
library(DESeq2)
library(tidyverse)
library(RColorBrewer)
library(pheatmap)
library(DEGreport)
library(tximport)
library(ggplot2)
library(ggrepel)
```

Loading data

The main output of Salmon is a *quant.sf* file, and we have one of these for each individual sample in our dataset.

Name	Length	EffectiveLength	TPM	NumReads
ENST00000456328	1657	1785.304	0.054490	3.722479
ENST00000450305	632	250.000	0.000000	0.000000
ENST00000488147	1351	1530.937	3.793490	222.229533
ENST00000619216	68	3.000	34.844416	4.000000
ENST00000473358	712	519.262	0.000000	0.000000
ENST00000469289	535	250.000	0.000000	0.000000
ENST00000607096	138	5.000	0.000000	0.000000
ENST00000417324	1187	250.000	0.000000	0.000000
ENST00000461467	590	250.000	0.000000	0.000000
ENST00000606857	840	250.000	0.000000	0.000000
ENST00000642116	1414	250.000	0.000000	0.000000
ENST00000492842	939	250.000	0.000000	0.000000

The effective gene lenght in a sample is then the average of the transcript lengths after weighting for their relative expression. You may see effective lengths that are larger than the physical length. The interpretation would be that in this case, given the sequence composition of these transcripts (including both the sequence-specific and fragment GC biases), you'd expect a priori tp sample more reads from them, thus they have longer estimated effective length.

The pseudocounts generated by Salmon are represented as normalized TPM (transcripts per million) counts and map to transcripts. These **need to be converted into non-normalized count estimates for performing DESeq2 analysis.** To use DESeq2 we also need to **collapse our abundance estimates from the transcript level to the gene-level.** We will be using the R Bioconductor package *tximport* to do all of the above and get set up for DESeq2.

The first thing we need to do is create a variable that contains the paths to each of our *quant.sf* files. Then we will **add names to our quant files which allow us to easily discriminate between samples in the final output matrix.**

In [ ]:
```
# if quant.sf has ENST with Versionnumbers ENTS000060232.1 use following c

sed -i 's/\(ENST[0-9]*\)\.[0-9]*/\1/g' quant.sf
```

In [5]:
```
## List all directories containing data
samples <- list.files(path = "./data", full.names = T, pattern="\\.salmon$
samples
```

'./data/ColoP14_5.salmon' './data/ColoP14_6.salmon' './data/ColoP28_2.salmon'
'./data/ColoP28_3.salmon' './data/ColoR34_3.salmon' './data/ColoR34_5.salmon'
'./data/ColoR37_1.salmon' './data/ColoR37_5.salmon'

In [6]:
```
## Obtain a vector of all filenames including the path
files <- file.path(samples, "quant.sf")
files
```

'./data/ColoP14_5.salmon/quant.sf' './data/ColoP14_6.salmon/quant.sf'
'./data/ColoP28_2.salmon/quant.sf' './data/ColoP28_3.salmon/quant.sf'
'./data/ColoR34_3.salmon/quant.sf' './data/ColoR34_5.salmon/quant.sf'
'./data/ColoR37_1.salmon/quant.sf' './data/ColoR37_5.salmon/quant.sf'

In [7]:
```
## Since all quant files have the same name it is useful to have names for
names(files) <- str_replace(samples, "./data/", "") %>%
                str_replace(".salmon", "")
names(files)
```

Our Salmon index was generated with transcript sequences listed by Ensembl IDs, but *tximport* needs to know **which genes these transcripts came from**. We will use the annotation table to extract transcript to gene information.

In [9]:
```
# Load the annotation table for GrCh38
tx2gene <- read.delim("tx2gene_grch38_ens94.txt")
head(tx2gene)
```

A data.frame: 6 × 3

tx_id	ensgene	symbol
<fct>	<fct>	<fct>
ENST00000387314	ENSG00000210049	MT-TF
ENST00000389680	ENSG00000211459	MT-RNR1
ENST00000387342	ENSG00000210077	MT-TV
ENST00000387347	ENSG00000210082	MT-RNR2
ENST00000612848	ENSG00000276345	AC004556.1
ENST00000386347	ENSG00000209082	MT-TL1

tx2gene **is a three-column dataframe** linking transcript ID (column 1) to gene ID (column 2) to gene Symbol (column 3). We will take the first two columns as input to *tximport*. The column names are not relevant, but the column order is.

Now we are ready to **run** *tximport*. Note that although there is a column in our quant.sf files that corresponds to the estimated count value for each transcript, those values are correlated by effective length. What we want to do is use the *countsFromAbundance="lengthScaledTPM"* argument. This will use the TPM column, and compute quantities that are on the same sqale as original counts, except no longer correlated with transcript length across samples.

In [10]:
```
# take a look at the arguments for the tximport function
?tximport
```

In [12]:
```
# Run tximport
txi <- tximport(files, type="salmon", tx2gene=tx2gene[,c("tx_id", "ensgene
```

```
reading in files with read_tsv
1 2 3 4 5 6 7 8
transcripts missing from tx2gene: 2041
summarizing abundance
summarizing counts
summarizing length
```

Viewing data

The *txi* object is a simple list containing matrices of the abundance, counts, length. Another list element *countsFromAbundance* carries through the character argument used in the tximport call. The length matrix contains the average transcript length for each gene which can be used as an offset for gene-level analysis.

In [13]:
```
attributes(txi)
```

$names =

'abundance' 'counts' 'length' 'countsFromAbundance'

In [14]:
```
head(txi)
```

$abundance

A matrix: 37896 × 8 of type dbl

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1
ENSG00000000003	2.297887	3.026222	1.075306	3.211711	4.059826	1.683818	5.422947
ENSG00000000005	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
ENSG00000000419	54.030883	43.301661	54.228247	58.495351	75.843579	82.514027	86.011391
ENSG00000000457	0.420783	0.238132	0.573562	0.561369	0.467493	0.173318	0.707600
ENSG00000000460	0.350344	1.591664	1.618641	0.881295	1.139336	1.999766	0.189245
ENSG00000000938	0.000000	0.000000	0.000000	0.000000	0.080315	0.000000	0.000000
ENSG00000000971	0.000000	0.175684	0.878898	0.089471	0.159680	0.483564	1.704614
ENSG00000001036	21.523845	19.123604	27.712694	22.823836	36.519436	38.540865	42.100132
ENSG00000001084	44.139466	17.169458	38.407377	18.162816	18.342237	19.746256	18.923743

In [15]:
```
# Write the counts to file
data <- txi$counts %>%
  round() %>%
  data.frame()
```

We will be using the *txi* object as is, for input into DESeq2.

Creating metadata

Of great importance is keeping track of the information about our data. At minimum, we need to alleast **have a file which maps our samples to the corresponding sample groups that we are investingating**. We will use the columns headers from the counts matrix as the row names of our metadata file and have single column to identify each sample as parental (P) or resistent (R).

In [22]:
```
## Create a sampletable/metadata
sampletype <- factor(c(rep("P",4), rep("R",4)))
meta <- data.frame(sampletype, row.names = colnames(txi$counts))
meta
```

A data.frame: 8 × 1

	sampletype
	<fct>
ColoP14_5	P
ColoP14_6	P
ColoP28_2	P
ColoP28_3	P
ColoR34_3	R
ColoR34_5	R
ColoR37_1	R
ColoR37_5	R

Count normalization of Salmon dataset using DESeq2

1. Ensure the row names of the metadata dataframe are present and in the same order as the column names of the counts dataframe.
2. Create a *DESeqDataSet* object
3. Generate the normalized counts

Match the metadata and counts data

We should make sure that we have sample names that match between the two files, and that the samples are in the right order. DESeq2 will output an error if this is not the case.

In [23]:
```
### Check that sample names match in both files
all(colnames(txi$counts) %in% rownames(meta))
all(colnames(txi$counts) == rownames(meta))
```

TRUE

TRUE

Create DESeq2 object

Bioconductor software packages often define and use a custom class within R for storing (input data, intermediate data and also results). These custom data structures are similar to *lists* in that they can contain multiple different data types/structures within them. But, unlike lists they have pre-specified *data slots*, which hold specific types/classes of data. The data stored in these pre-specified slots can be accessed by using specific package-defined functions.

Let's start by creating the *DESeqDataSet* object and then we can talk a bit more about what is stored inside it. To create the object we will need the **count matrix** and the **metadata** table as input. We will also need to specify a **design formula**. The design formula specifies the column(s) in the metadata table and how they should be used in the analysis.For our dataset we only have one column we are interested in, that is *~sampletype*. This column hast two factor levels, which tells DESEq2 that for each gene we want to evalutate gene expression change with respect to these different levels.

Our count matrix input is stored inside the *txi* list object, and so we pass that in using *DESeqDataSetFromTximport()* function which will extract the counts component and round the values to the nearest whole number.

In [24]:
```
## Create DESeq2Dataset object
dds <- DESeqDataSetFromTximport(txi, colData = meta, design = ~ sampletype
```

```
using just counts from tximport
```

You can use DESeq-specific functions to access the different slots and retrieve information, if you wish. For example, suppose we wanted the original count matrix we would use *counts()*.

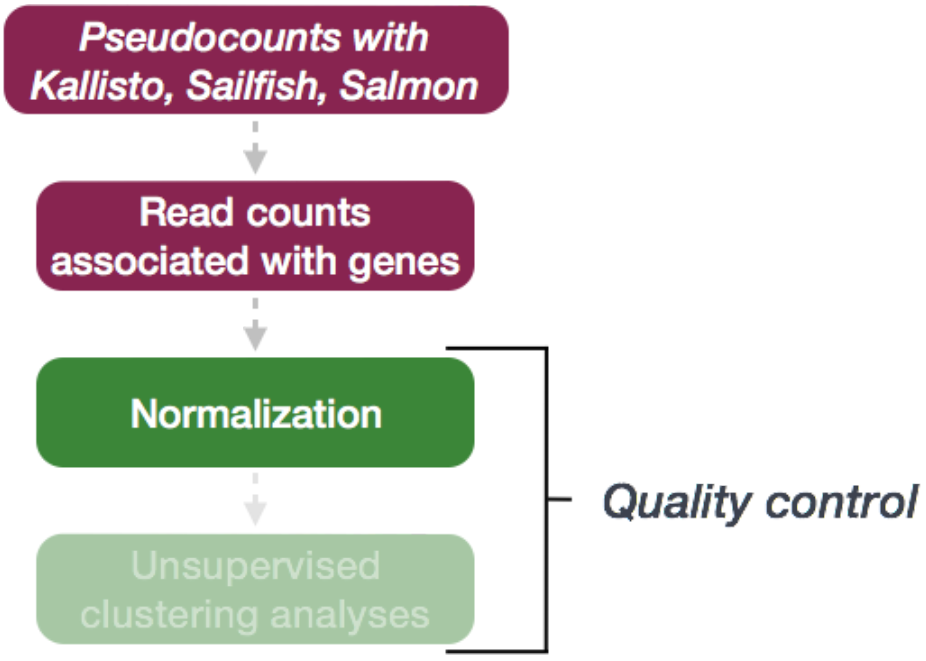
In [25]:
```
head(counts(dds))
```

A matrix: 6 × 8 of type int

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1	Co
ENSG00000000003	57	81	32	121	135	50	134	
ENSG00000000005	0	0	0	0	0	0	0	
ENSG00000000419	412	354	496	677	778	754	655	
ENSG00000000457	14	9	24	29	21	7	24	
ENSG00000000460	11	51	58	40	46	72	6	
ENSG00000000938	0	0	0	0	3	0	0	

Generate the Salmon normalized counts

The next step is to normalize the count data in order to be able to make fair gene comparisons between samples.



To perform the **median of rations method** of normalization, DESeq2 has single *estimateSizeFactors()* function that will generate size factors for us. We will use the function in the example below, but **in a typical RNAseq analysis this step is automatically performed by the DESeq() function**, which we will see later.

In [26]:
```
dds <- estimateSizeFactors(dds)
```

By assigning the results back to the *dds* object we are filling in the slots of the *DESeqDataSet* object with the appropriate information. We can take a look at the normalization factor applied to each sample using:

In [27]:
```
1 sizeFactors(dds)
```

ColoP14_5	0.905695935716028
ColoP14_6	0.909497081989753
ColoP28_2	1.02591818290851
ColoP28_3	1.26223133529604
ColoR34_3	1.26790133175554
ColoR34_5	0.987525571113325
ColoR37_1	1.00720459589524
ColoR37_5	0.90291864588562

Now, to retrieve the normalized counts matrix from *dds*, we use the *counts()* function and add the argument *normalized=TRUE*.

In [28]:
```
1 normalized_counts  <- counts(dds, normalized=TRUE)
```

We can save this normalized data matrix to file for later use:

In [29]:
```
1 write.table(normalized_counts, file="data/normalized_counts.txt", sep="\t"
```

!Note DESeq2 doesn't actually use normalized counts, rather it uses the raw counts and models the normalization inside the Generalized Linear Model (GLM). These normalized counts will be useful for downstream visualization of results, but cannot be used as input to DESeq2 or any other tools that perform differential expression analysis which use negative binomial model.

Quality Control

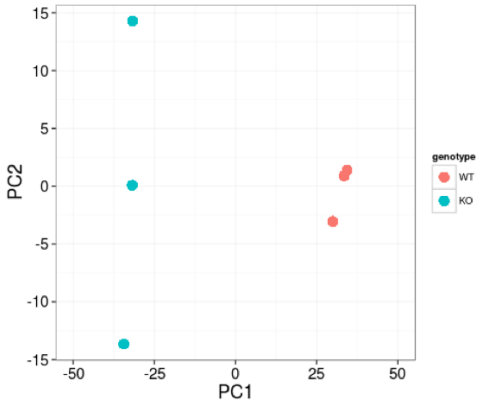
The next step in the DESeq2 workflow is QC, which includes sample-level and gene-level steps to perform QC checks on the count data to help us ensure that the sample/replicates look good.

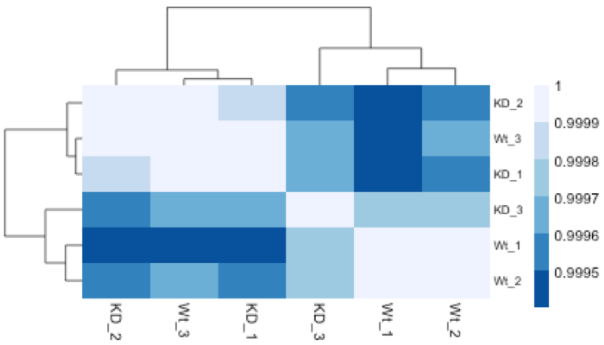
Sample-level QC

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 to the expectation from the experiment's design?
- What are the major sources of variation in the dataset?

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

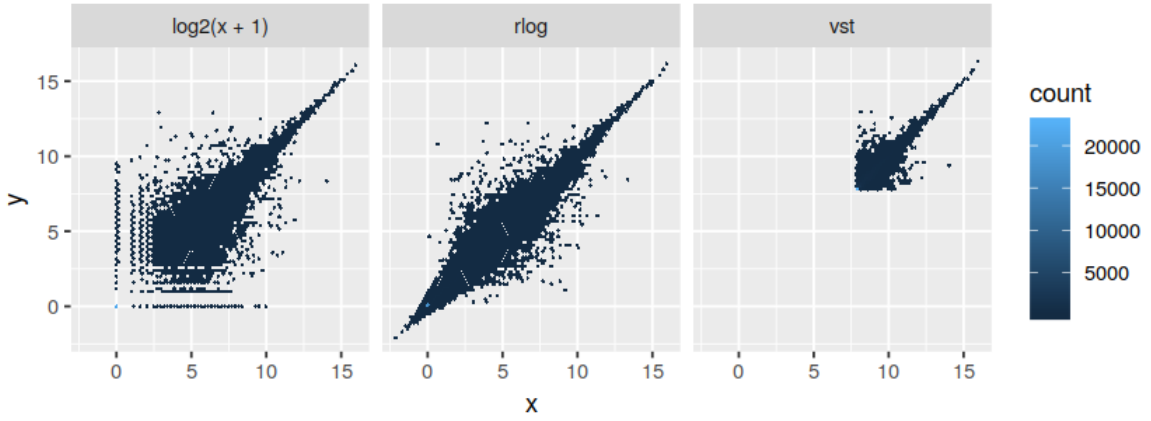




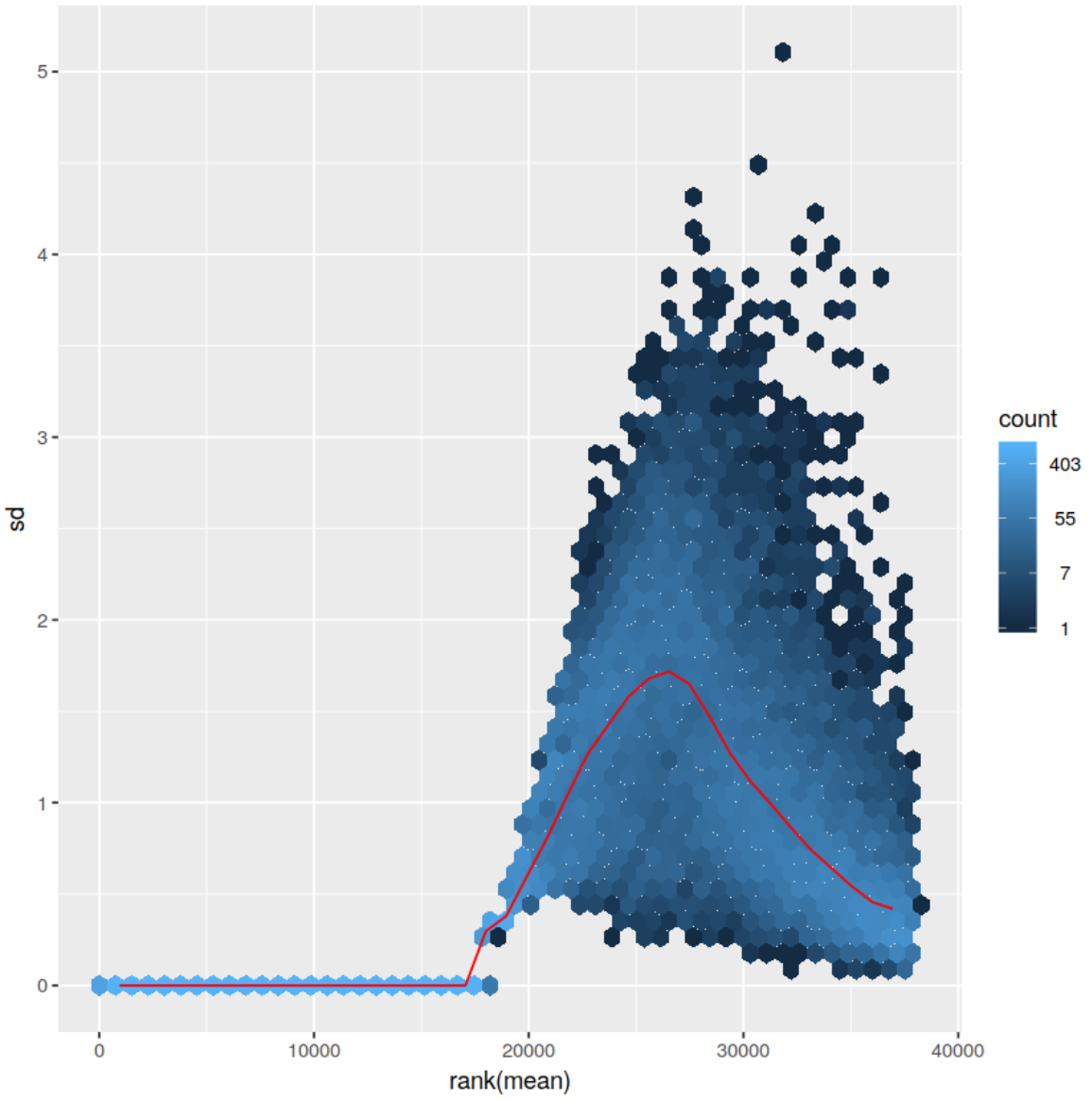
When using these unsupervised clustering methods, log2-transformation of the normalized counts improves the distances/clustering for visualization. DESeq2 uses a **regularized log transform** (rlog) of the normalized counts for sample-level QC as it moderates the variance across the mean, improving the clustering.

In [55]:

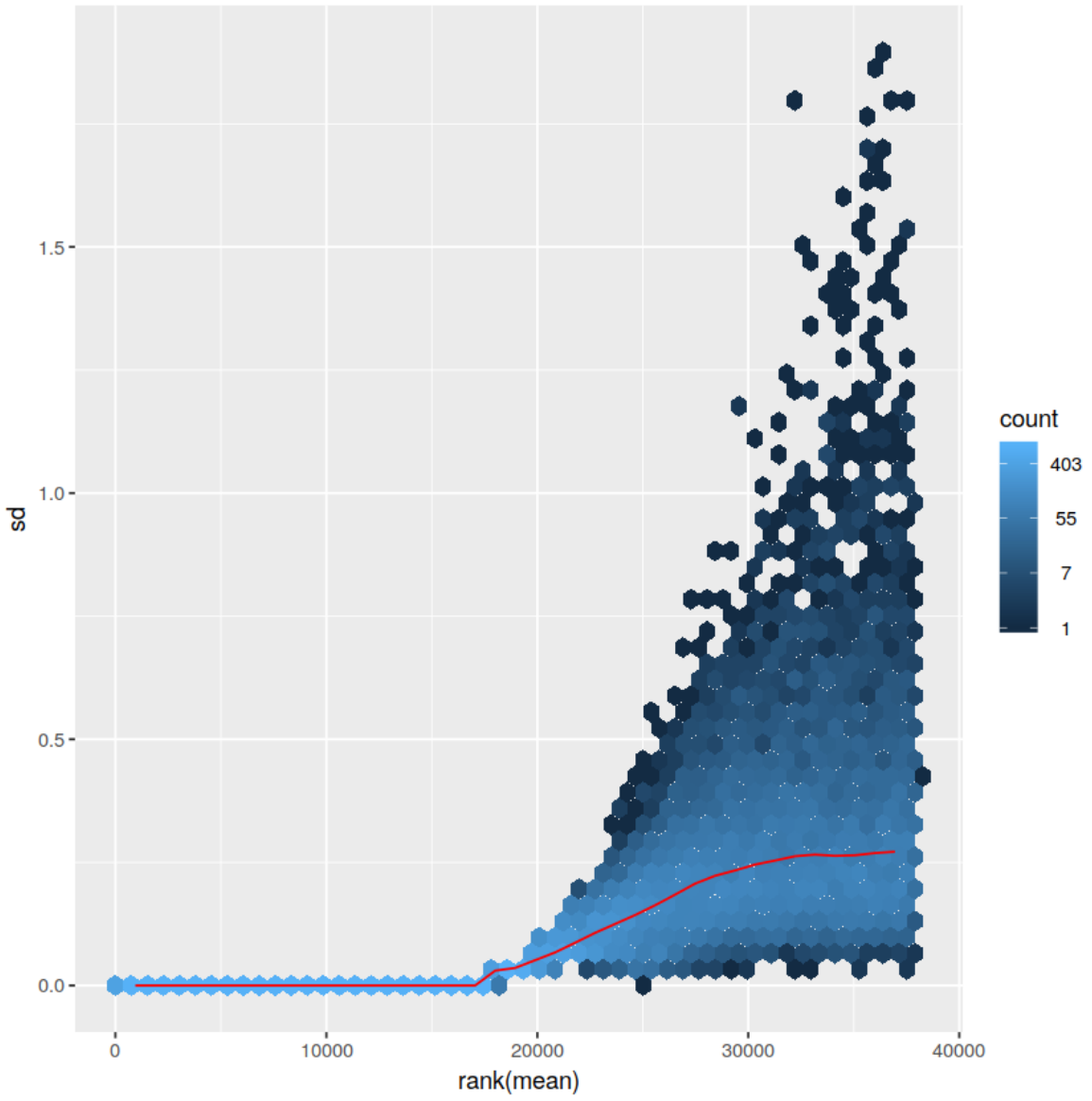
```r
### Transform counts for data visualization
library("vsn")
rld <- rlog(dds, blind = FALSE)
vsd <- vst(dds, blind = FALSE)
ntd <- normTransform(dds)

library("dplyr")
library("ggplot2")

df <- bind_rows(
  as_data_frame(log2(counts(dds, 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)
```



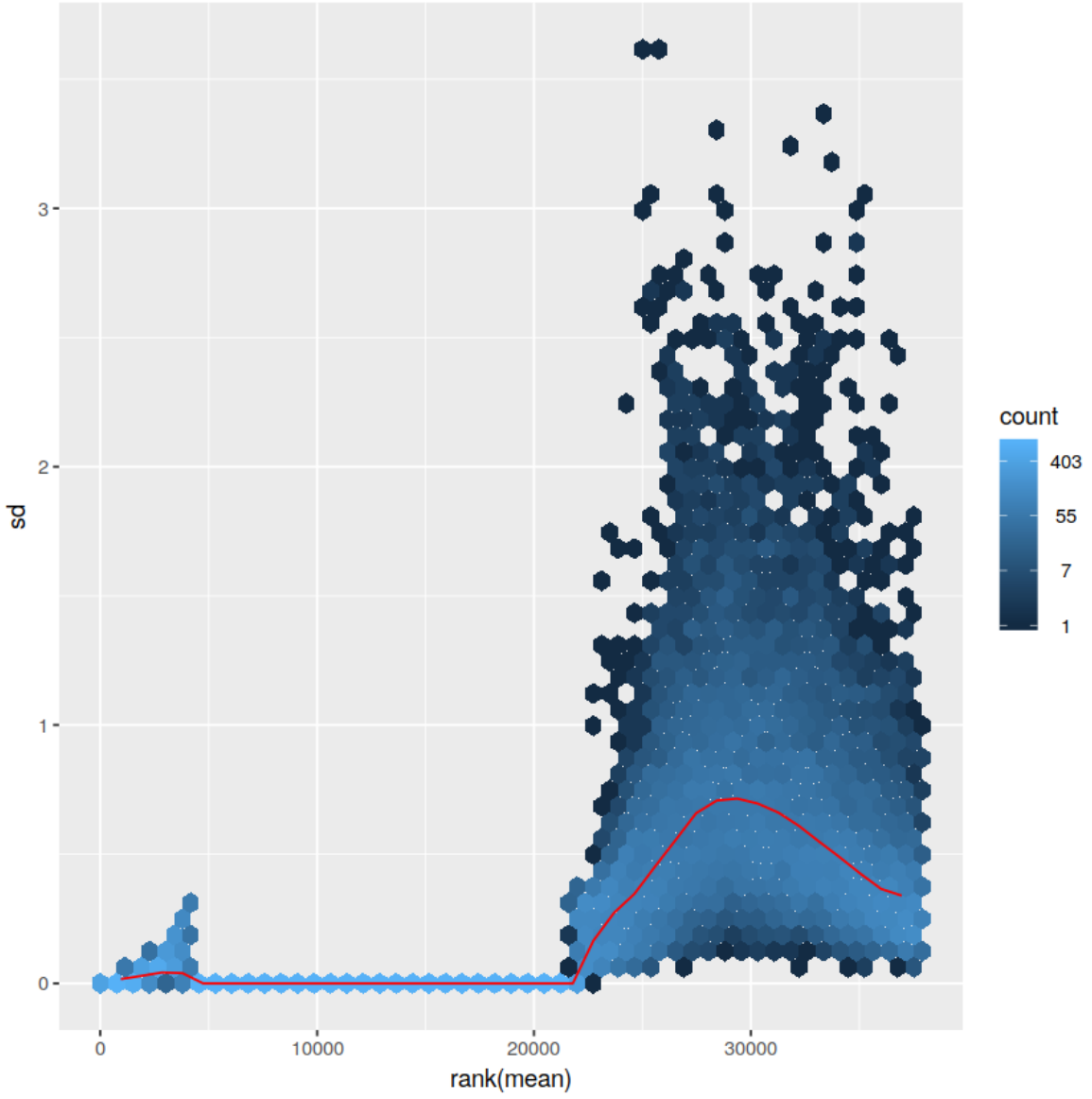
In [56]:
```
1 meanSdPlot(assay(ntd))
```



In [57]:
```
1 meanSdPlot(assay(vsd))
```



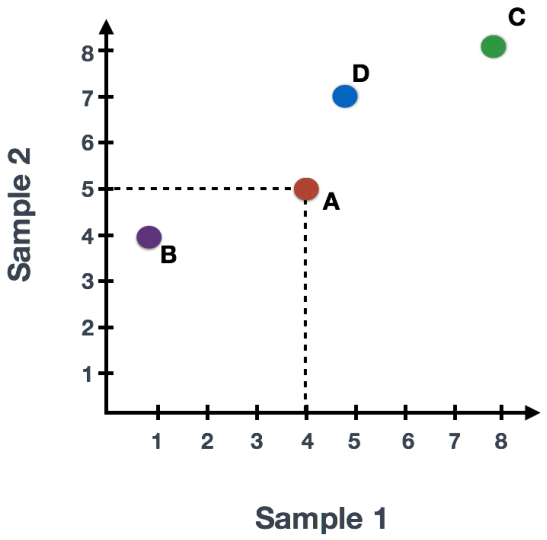
In [58]:
```
1 meanSdPlot(assay(rld))
```



Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a technique used to emphasize variation and bring out strong patterns in a dataset (dimensionality reduction).

Suppose we had a dataset with two samples and four genes. Based on this expression data we want to evalutate the relationship between these samples. We could plot the counts of one sample versus another, with Sample 1 on the x-axis and Sample 2 on the y-axis as shown below:

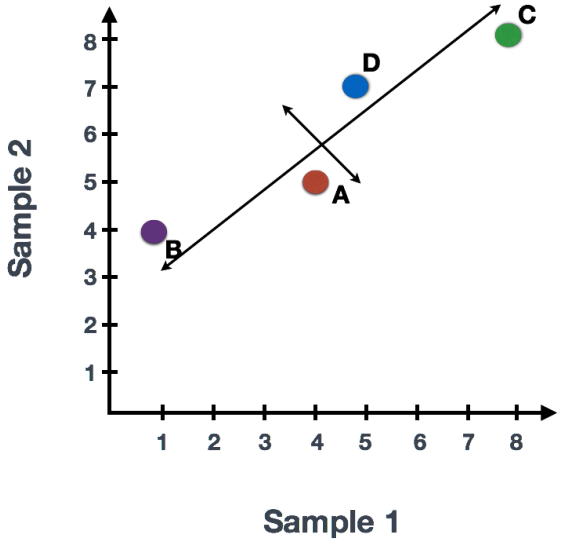


	Sample 1	Sample 2
Gene A	4	5
Gene B	1	4
Gene C	8	8
Gene D	5	7

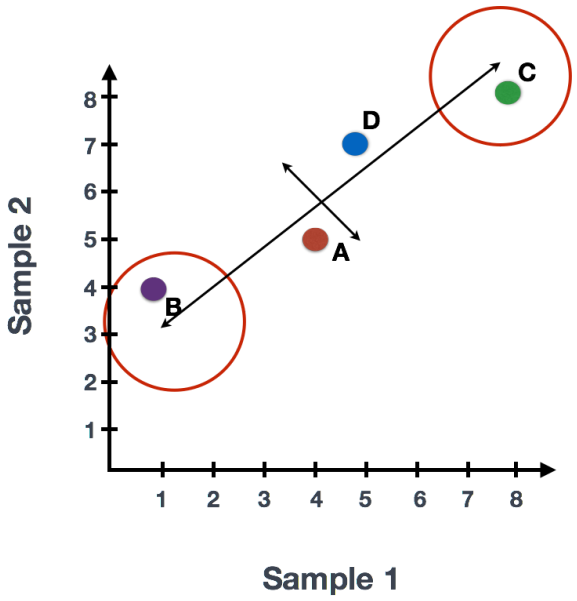
For PCA analysis, the first step is taking this plot and drawing a line through the data in the direction representing the most variation. In this example, the most variation is along the diagonal. That is, the **largest spread in the data** is between the two endpoints of this line. **This is called the first principal component, or PC1**. The genes at the endpoints of this line (Gene B and Gene C) have the **greatest influence** on the direction of this line.

![](PCA_2sample_variation1 (1).png)

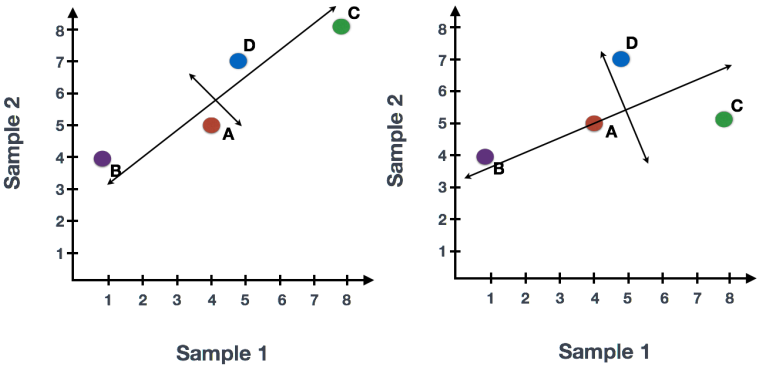
We also see the genes vary somewhat above and below the line. We could draw another line through the data representing the second most amount of variation in the data.



The genes near the ends of the line, which would include those genes with the highest variation between samples (high expression in one sample and low expression in the other), have the greatest influence on the direction of the line.



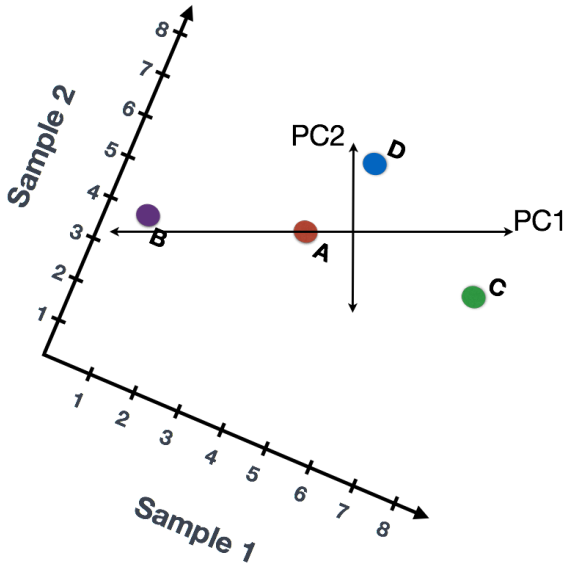
For example, a small change in the value of *GENE C* would greatly change the direction of the line, whereas a small change in *GENE A* or *GENE D* would have little affect.



We could just rotate the entire plot and view the lines representing the variation as left-to-right and up-and-down. We see most of the variation in the data is left-to-right; this is and the second most variation in the data is up-and-down. These axes that represent the variation are "Principal Components", with PC1 representing the most variation in the data and PC2 representing the second most variation in the data.

If we had three sampels, then we would have and extra direction in which we could have variation. Therefore, if we have *N* samples we would have *N* -directions of variation or principal components.

We could give quantitative scores to genes based on how they influence PC1 and PC2. Genes with little influence would get scores near zero, while genes with more influence would receive larger scores. Genes on opposite ends of the lines have a large influence, so they would receive large scores, but with opposite signs.



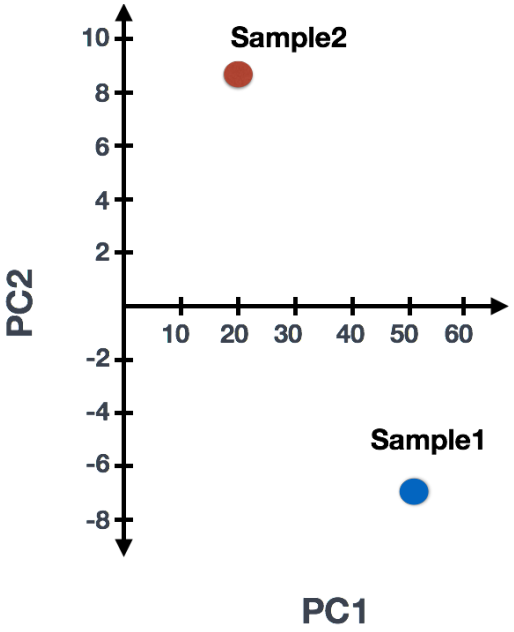
	Sample 1	Sample 2	Influence on PC1	Influence on PC2
Gene A	4	5	-2	0.5
Gene B	1	4	-10	1
Gene C	8	8	8	-5
Gene D	5	7	1	6

To generate a score per sample, we combine the read counts for all genes Using the counts in the table for each gene (assuming we had only 4 genes total) we could calculate PC1 and PC2 values for each sample as follows:

Sample1 PC1 score = (4 * -2) + (1 * -10) + (8 * 8) + (5 * 1) = 51 Sample1 PC2 score = (4 * 0.5) + (1 * 1) + (8 * -5) + (5 * 6) = -7

Sample2 PC1 score = (5 * -2) + (4 * -10) + (8 * 8) + (7 * 1) = 21 Sample2 PC2 score = (5 * 0.5) + (4 * 1) + (8 * -5) + (7 * 6) = 8.5

The scores would then be plotted to examine whether the samples exhibit similar variation across all genes:



	PC1	PC2
Sample1	51	-7
Sample2	21	8.5

Since genes with the greatest variation between samples will have the greatest influence on the principal components, we hope our condition of interest explains this variation (e.g. high counts in one condition and low counts in the other). With PC1 representing the most variation in the data and PC2 representing the second most variation in the data, we can visualize how similar the variation of genes is between samples.

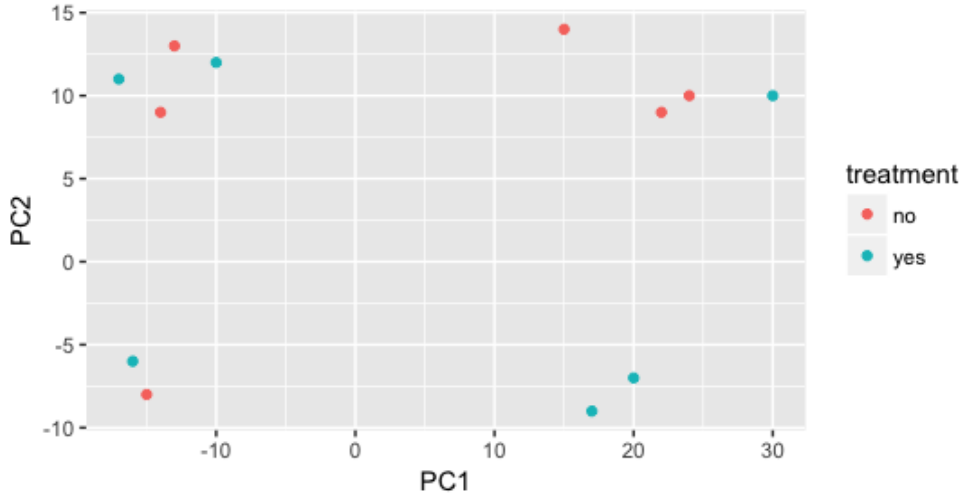
If two samples have similar levels of expression for the genes that contribute significantly to the variation represented by PC1, the will be plotted close together on the PC1 axis. Therefore, we would expect that biological replicates to have similar scores (since the same genes are changing) and cluster together on PC1 and/or PC2, and the samples from different treatment groups to have different score. This is easiest to understand by visualizing example PCA plots.

Interpreting PCA plots

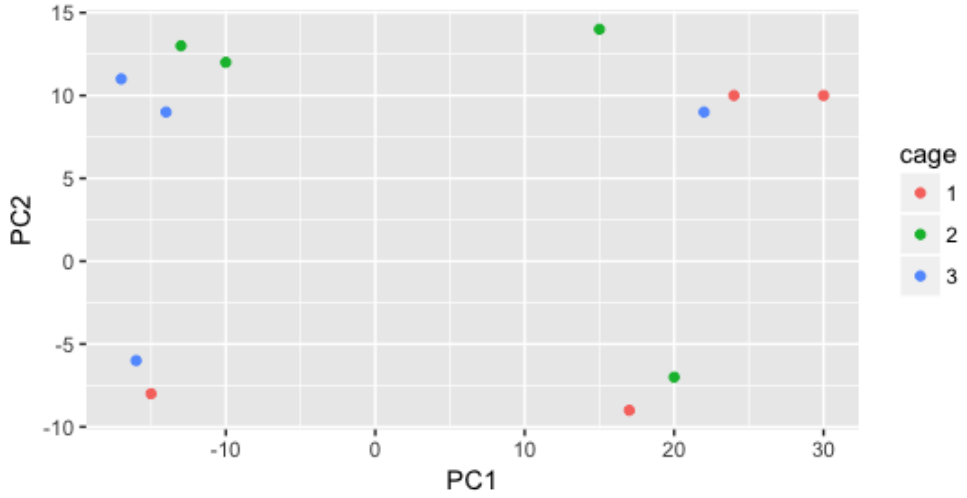
Below is an example dataset and a few PCA plots to get a feel for how to interpret them. The metadata for the experiment is displayed below. The main condition of interest is *treatment*.

sample	strain	date	cage	treatment	replicate	sex
B1	BALB/cJ	20180515	1	yes	1	M
B2	C57BL/6J	20180515	2	yes	1	M
B3	BALB/cJ	20180515	3	no	1	M
B4	C57BL/6J	20180515	1	no	1	F
B5	BALB/cJ	20180515	2	yes	2	F
B6	C57BL/6J	20180515	3	yes	2	M
B7	BALB/cJ	20180515	1	no	2	M
B8	C57BL/6J	20180515	2	no	2	M
B9	BALB/cJ	20180515	3	yes	3	F
B10	C57BL/6J	20180307	1	yes	3	F
B11	BALB/cJ	20180307	2	no	3	M
B12	C57BL/6J	20180307	3	no	3	M

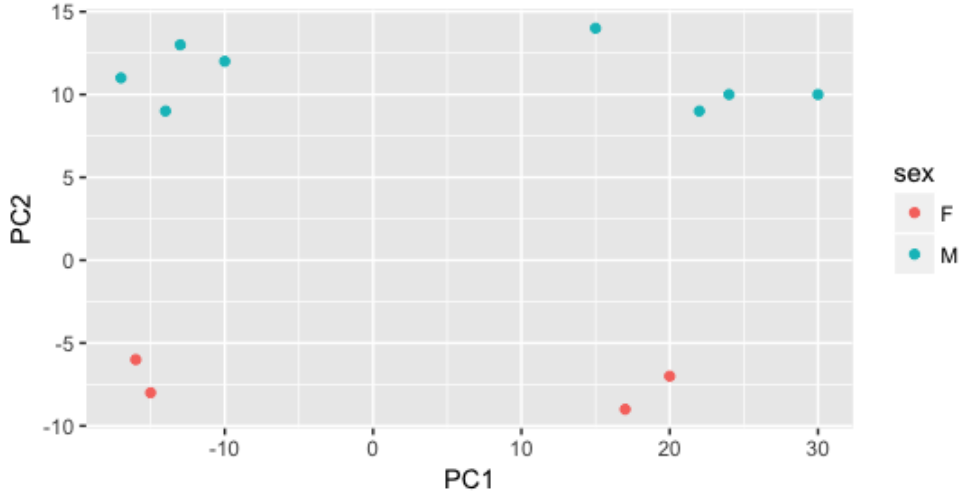
When visualizing on PC1 and PC2, we don't see the samples separate by treatment, so we decide to explore other sources of variation present in the data. We hope that we have included all possible known sources of variation in our metadata table, and we can use these factors to color the PCA plot.



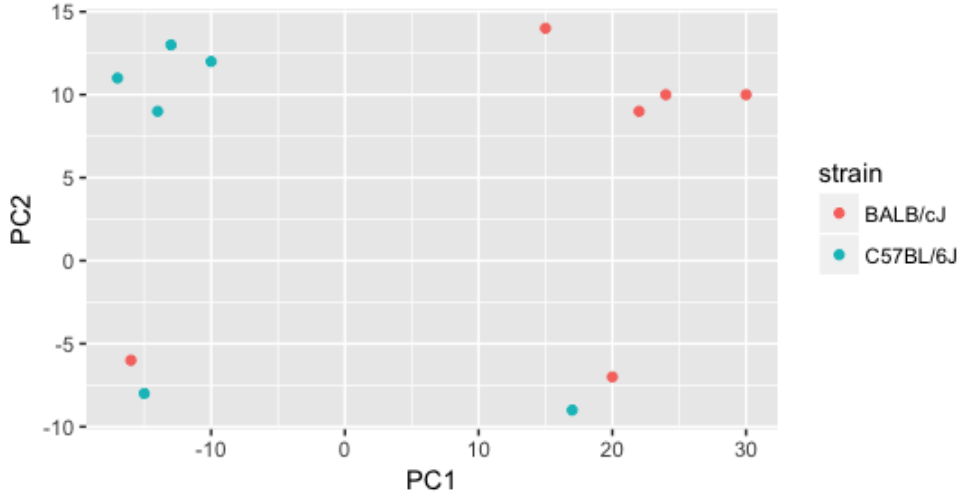
We start with the factor *cage*, but the *cage* factor does not seem to explain the variation on PC1 or PC2.



Then, we color by the *sex* factor, which appears to separate samples on PC2. This is good information to take note of, as we can use it downstream to account for the variation due to sex in the model and regress it out.



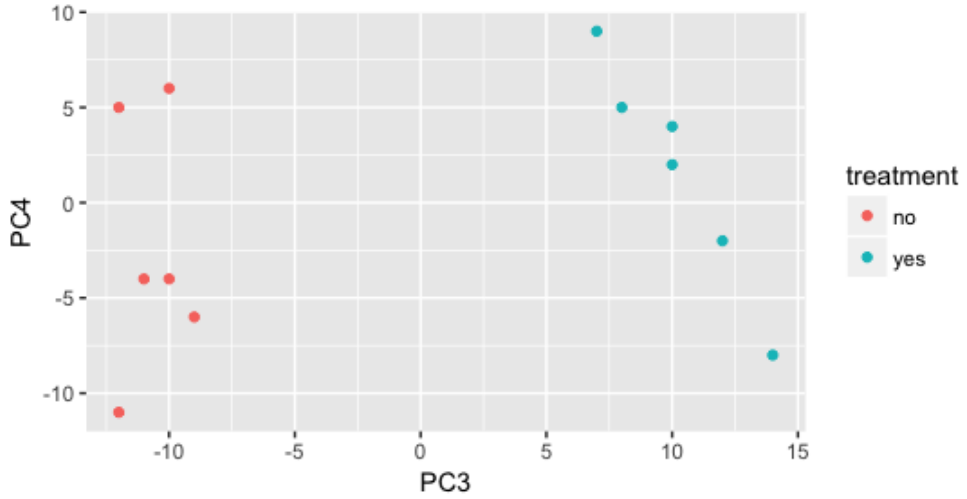
Next we explore the *strain* factor and find that it explains the variation on PC1.



It's great that we have been able to identify the sources of variation for both PC1 and PC2. By accounting for it in our model, we should be able to detect more genes differentially expressed due to *treatment*.

Worrisome about this plot is that we see two samples that do not cluster with correct strain. This would indicate a likely **sample swap** and should be investigated to determine whether these samples are indeed the labeled strains. If we found there was a switch, we could swap the samples in the metadata. However, if we think they are labeled correctly or are unsure, we could just remove the sample from the dataset.

Still we haven't found if *treatment* is a major source of variation after *strain* and *sex*. So, we explore PC3 and PC4 to if *treatment* is driving the variation represented by either of these PCs.



We find that the samples separate by *treatment* on PC3, and are optimistic about our DE analysis since our condition of interest, *treatment*, is separation on PC3 and we can regress out the variation driving PC1 and PC2.

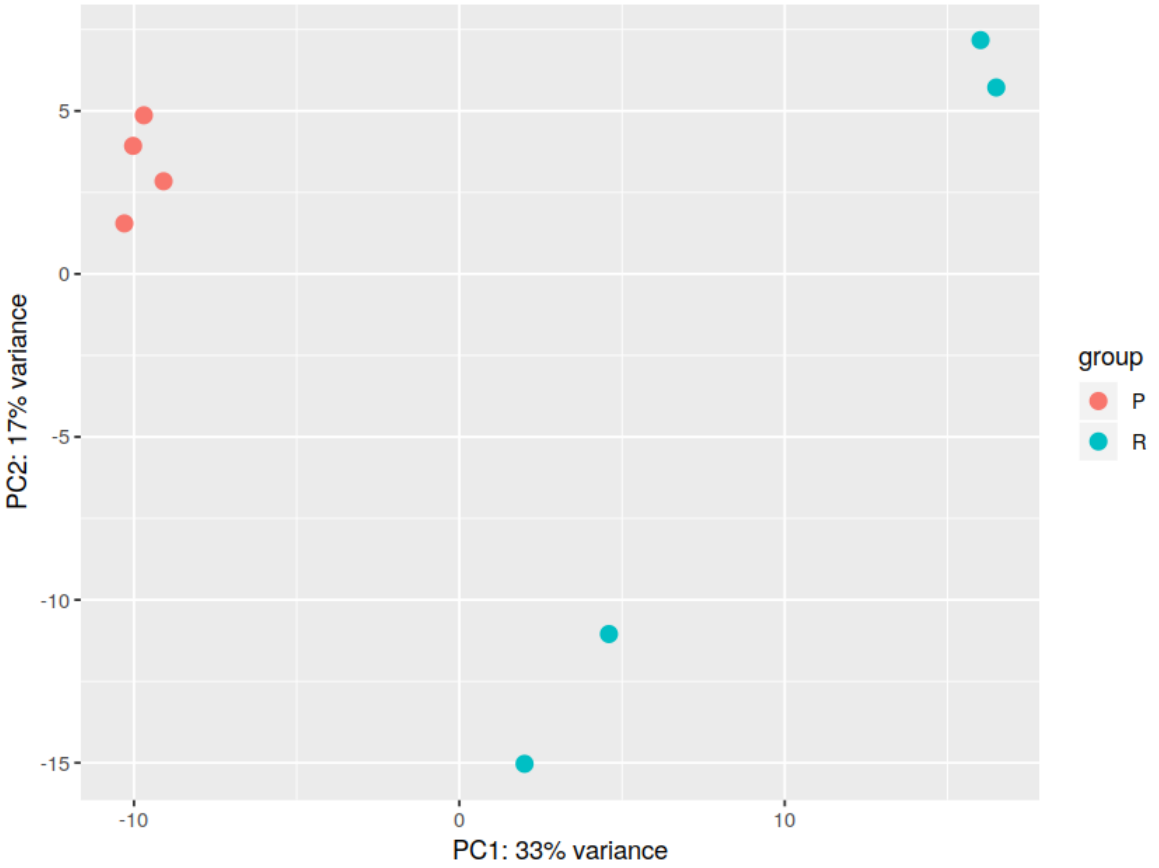
Depending on how much variation is explained by the first few principal components, you **may want to explore more (i.e. consider more components and plot pairwise combinations)**. Even if your samples do not separate clearly by the experimental varible, you may still get biologically relevent results 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 important to account for these in your model**, as it provides more power to the tool for detecting DE genes.

Principal components analysis with DESeq2

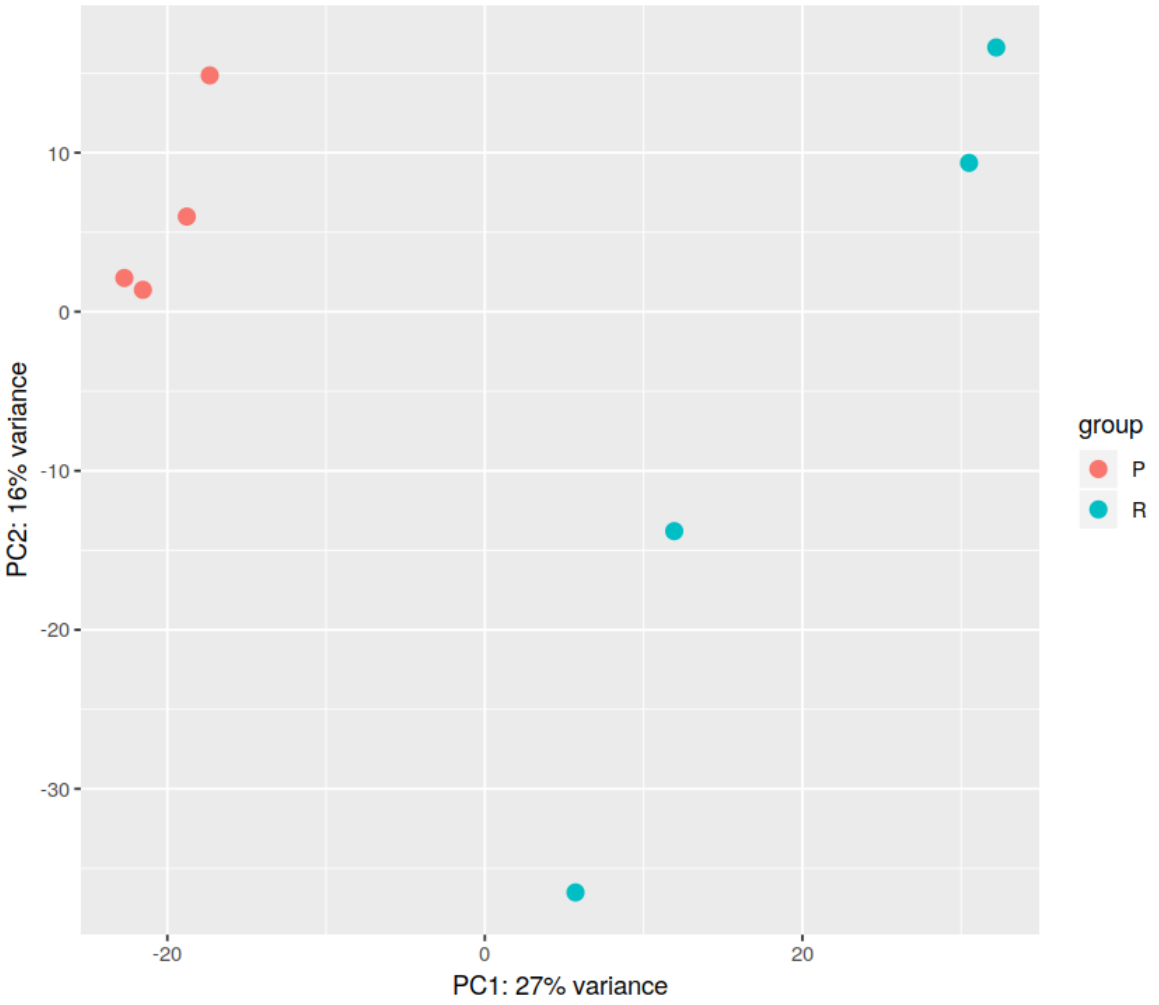
DESeq2 has a built-in function for plotting PCA plots, that uses *ggplot* under the hood. The function *plotPCA()* requires two arguments as input: an *rlog* object and the *intgroup*, the column in the metadata that we are intersted in.

In [51]:

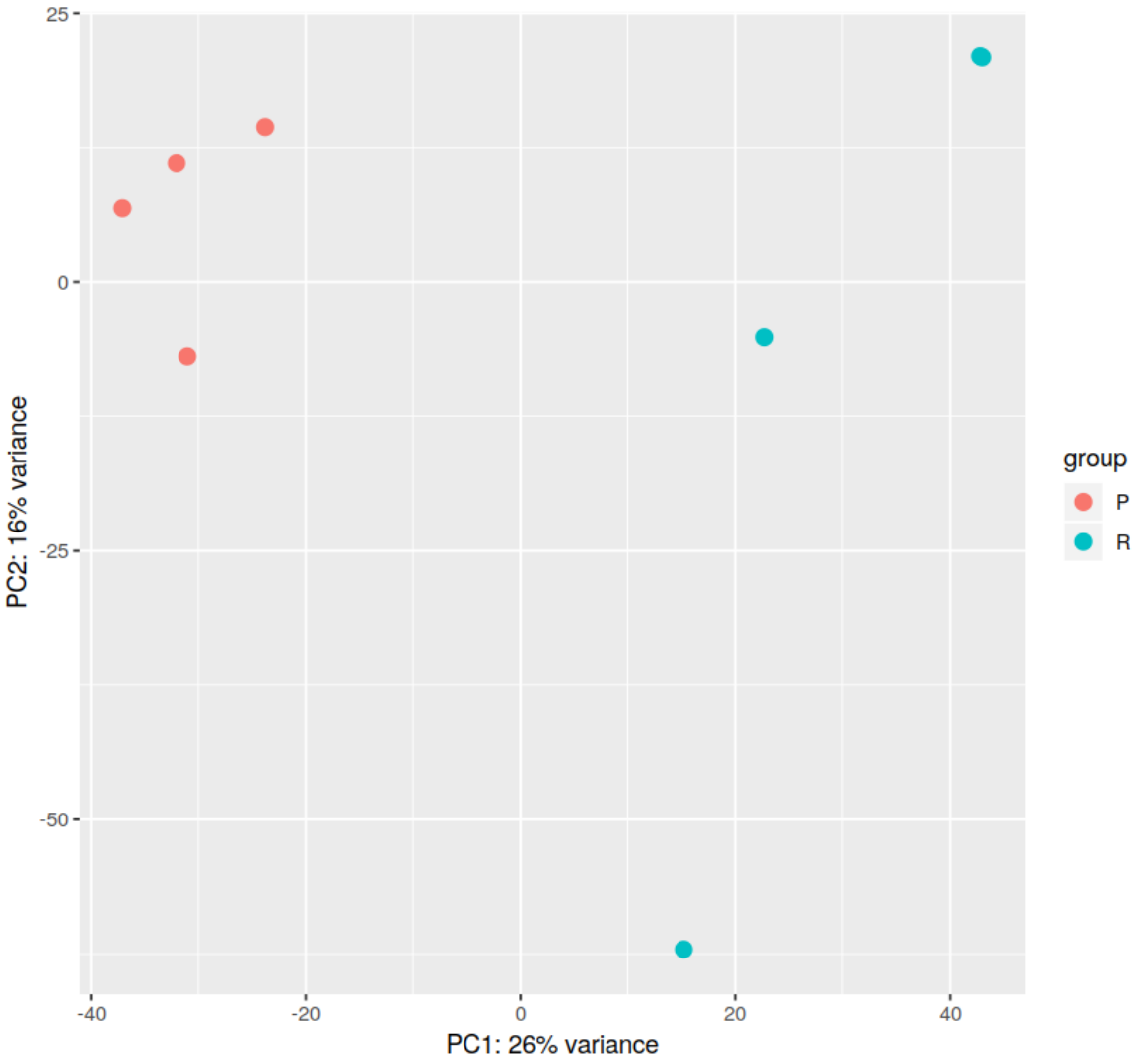
```
### Plot PCA for vsd transformation which looks best see plot above
plotPCA(vsd, intgroup="sampletype")
```



In [52]:
```
1 plotPCA(rld, intgroup="sampletype")
```

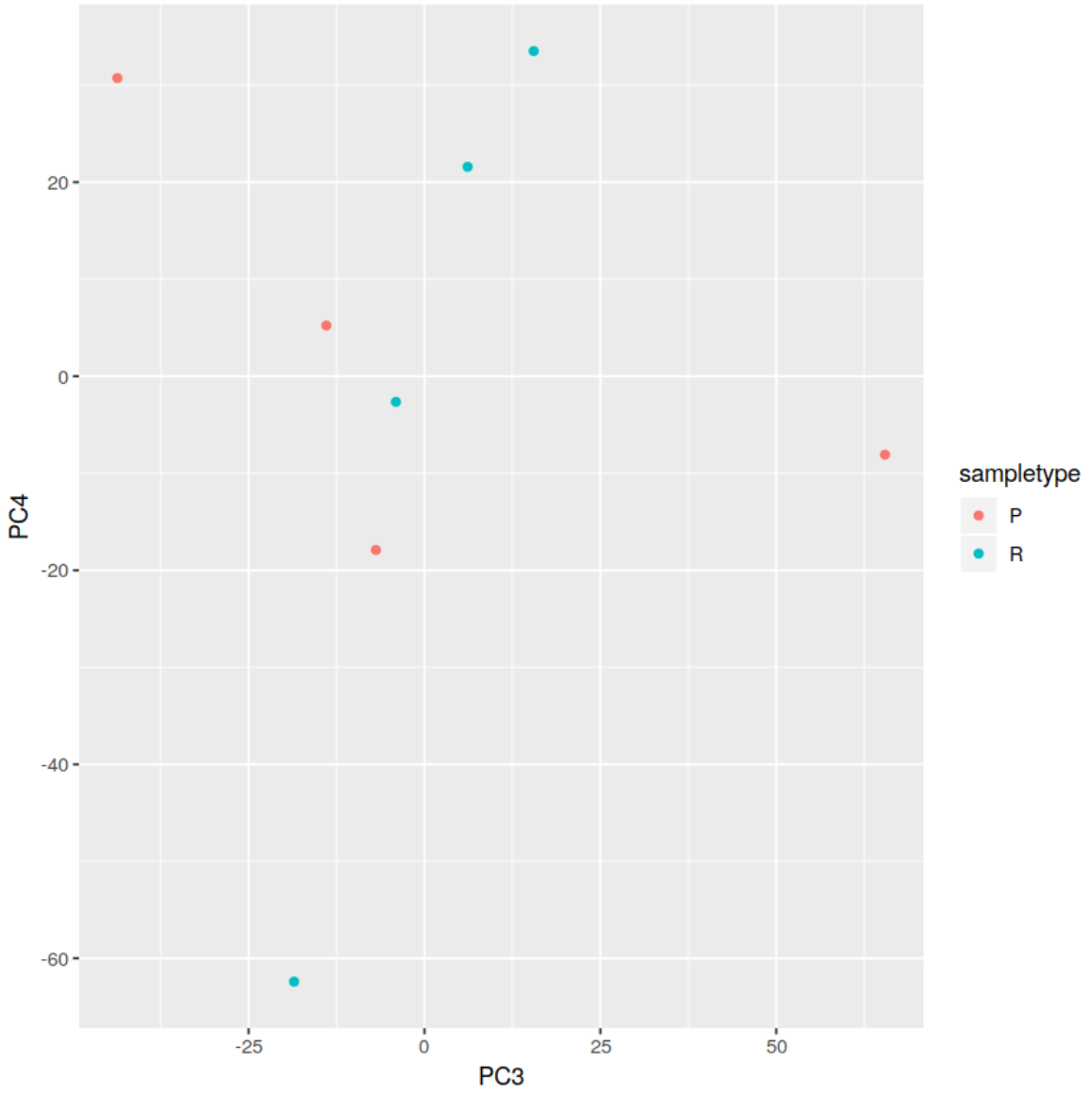


In [53]:
```
1 plotPCA(ntd, intgroup="sampletype")
```



In [40]:

```r
# Input is a matrix of log transformed values
rld <- rlog(dds, blind=T)
rld_mat <- assay(rld)
pca <- prcomp(t(rld_mat))

# Create data frame with metadata and PC3 and PC4 values for input to ggp
df <- cbind(meta, pca$x)
ggplot(df) + geom_point(aes(x=PC3, y=PC4, color = sampletype))
```



Hierarchical Clustering Heatmap

Similar to PCA, hierarchical clustering is another, complementary method for identifying strong patterns in a dataset and potential outliers. The heatmap displays **the correlation of gene expression for all pairwise combinations of samples** in the dataset. Since the majority of genes are not differentially expressed, samples generally have high correlations with each other (values higher than 0.80). Samples below 0.80 may indicate an outlier in your data and/or sample contamination.

The hierarchical tree can indicate which samples are more similar to each other based on the normalized gene expression values. The color blocks indicate substructure in the data, and you would expect to see your replicates cluster together as a block fokr each sample group. Additionally, we expect to see samples clustered similar to the groupings observed in a PCA plot.

Since there is no built-in function for heatmaps in DESeq2 we will be using the *pheatmap()* function from the *pheatmap* package. This function requires a matrix/dataframe of numeric values as input, and so the first thing we need to is retrieve that information from the rld object.

Then we need to compute the pairwise correlation values for samples. We can do this using the *cor()* function.

In [34]:

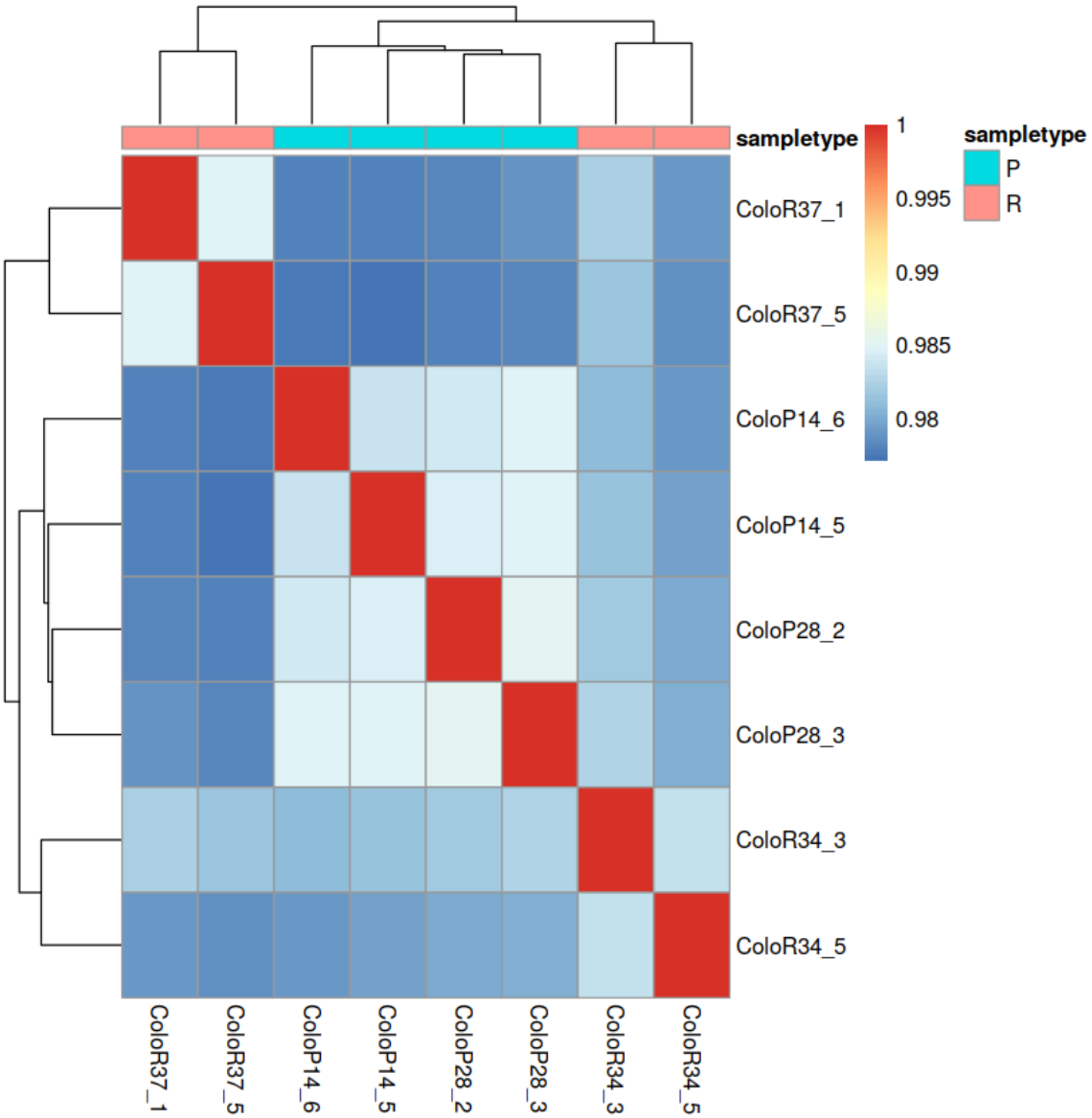
```
### Extract the rlog matrix from the object
rld_mat <- assay(rld)    ## assay() is function from the "SummarizedExperi
### Compute pairwise correlation values
rld_cor <- cor(rld_mat)    ## cor() is a base R function

head(rld_cor)   ## check the output of cor(), make note of the rownames an
```

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	0.9836086	0.9844993	0.9848781	0.9812979	0.9795294	0.9778821	0.977164
ColoP14_6	0.9836086	1.0000000	0.9840829	0.9847247	0.9809314	0.9791539	0.9779389	0.977455
ColoP28_2	0.9844993	0.9840829	1.0000000	0.9853536	0.9818489	0.9800187	0.9781867	0.978001
ColoP28_3	0.9848781	0.9847247	0.9853536	1.0000000	0.9825880	0.9802105	0.9788029	0.978288
ColoR34_3	0.9812979	0.9809314	0.9818489	0.9825880	1.0000000	0.9835117	0.9822156	0.981689
ColoR34_5	0.9795294	0.9791539	0.9800187	0.9802105	0.9835117	1.0000000	0.9790905	0.978745

In [35]:

```
### Load pheatmap package
library(pheatmap)

### Plot heatmap
pheatmap(rld_cor, annotation = meta)
```

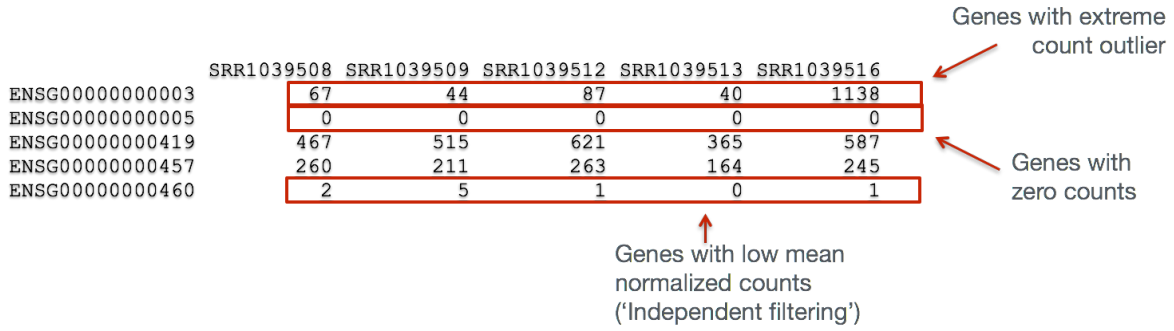


Overall, we observe pretty high correlations across the board suggesting no outlying sample(s). Also, similar to the PCA plot you see the parental samples clustering together. The resistent are a problem but for demonstration purpose we will continue to proceed to differential expression analysis.

Gene-level QC

In addition to examining how well the samples/replicates cluster together, there are a few more QC steps. Prior to differential expression analysis it is beneficial to omit genes that have little or no chance of being detected as differentially expressed. This will increase the power to detect differentially expressed genes. The genes omitted fall into three categories:

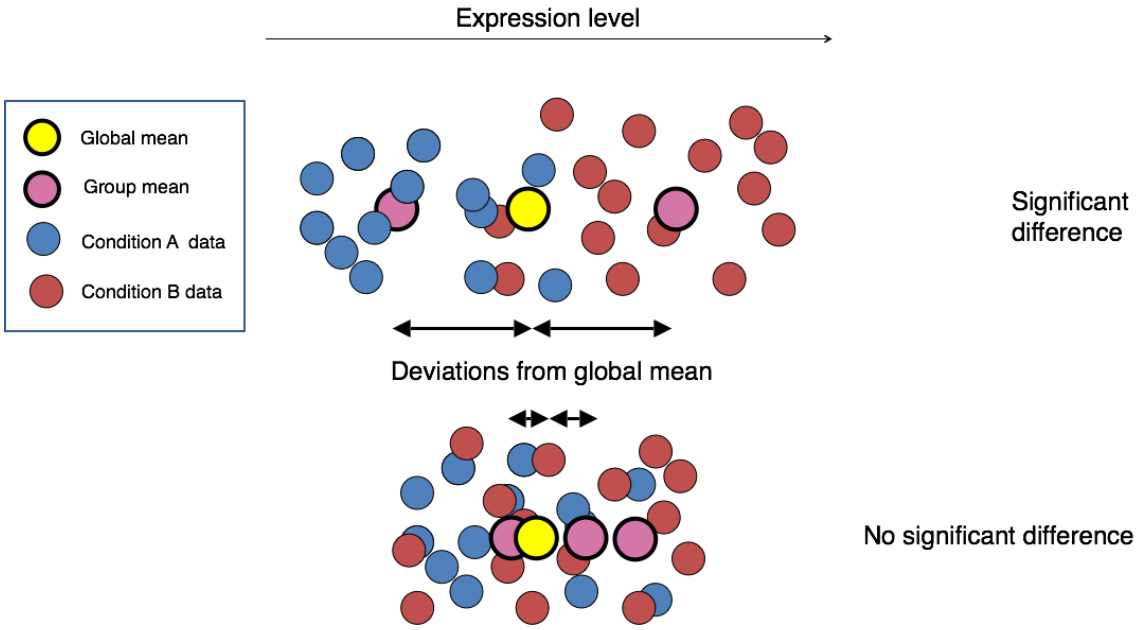
- Genes with zero counts in all samples
- Genes with an extreme count outlier
- Genes with a low mean normalized counts



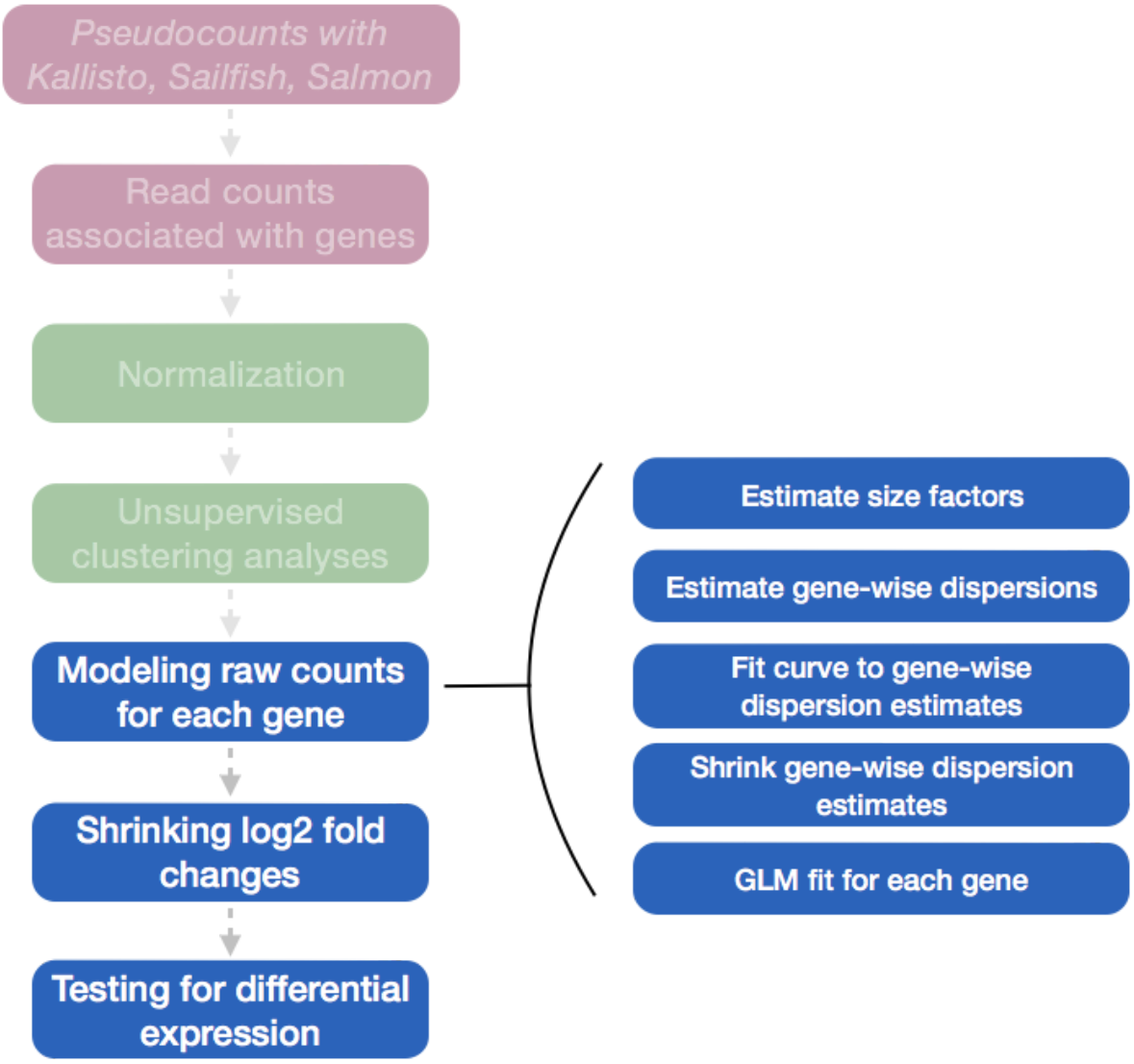
DESeq2 will perform this filtering by default; however other DE tools, such as EdgeR will not. Filtering is a necessary step, even if you are using limma-voom and/or edgeR's quasi-likelihood methods. Be sure to follow pre-filtering steps when using other tools.

Differential expression analysis with DESeq2

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.



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 dispersions 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.



Running DESeq2

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

Complex designs

DESeq2 also allows for the analysis okf complex designs. You can explore interactions or the difference of differences by specifiying 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.

Salmon Colo DE analysis

Now that we know how to specify the model to DESeq2, we can run the differential expression pipeline on the **raw counts**.

First we create a DESeqDataSet as we did in the *Count normalization* and specify the *txi* object which contains our raw counts, the metadata variable, and provide our design formula:

In [59]:
```
# Create DESeq2Dataset object
dds  <- DESeqDataSetFromTximport(txi, colData = meta, design = ~ sampletyp
```

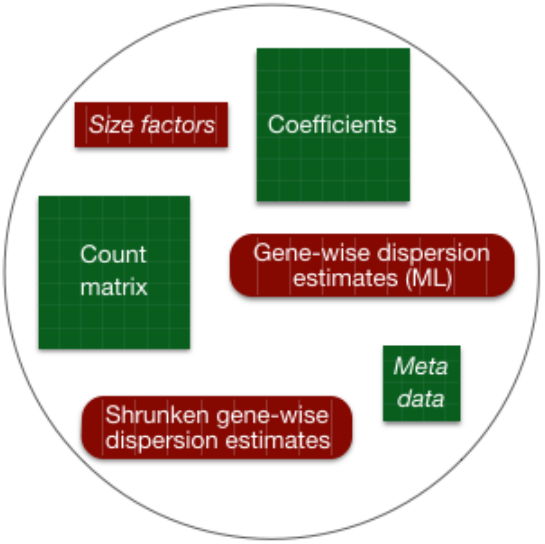
```
using just counts from tximport
```

Then, to run the actual differential expression analysis, we use a single call to the function *DESeq()*

In [60]:
```
# Run analysis
dds  <- DESeq(dds)
```

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

By re-assigning the results of the function back to the same variable name (dds), we can fill in the *slots* of our *DESeqDataSet* object.



Everything from normalization to linear modeling was carried out by the use of a single function

In [61]:
```
## Check the size factors
sizeFactors(dds)
```

ColoP14_5	0.905695935716028
ColoP14_6	0.909497081989753
ColoP28_2	1.02591818290851
ColoP28_3	1.26223133529604
ColoR34_3	1.26790133175554
ColoR34_5	0.987525571113325
ColoR37_1	1.00720459589524
ColoR37_5	0.90291864588562

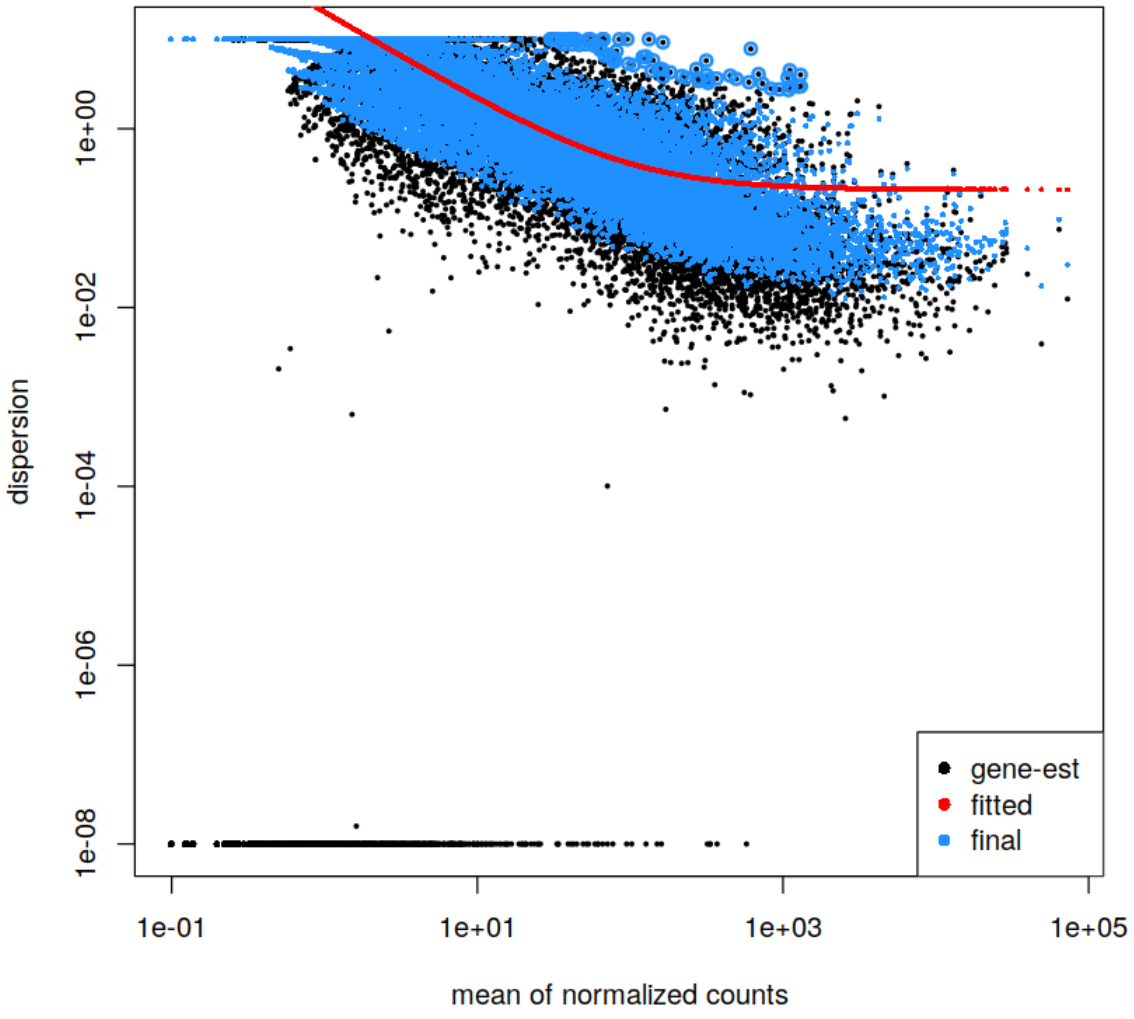
In [62]:
```
## Total number of raw counts per sample
colSums(counts(dds))
```

ColoP14_5	4710892
ColoP14_6	4888656
ColoP28_2	5448695
ColoP28_3	6752146
ColoR34_3	6463883
ColoR34_5	5383467
ColoR37_1	5045555
ColoR37_5	4578353

In [63]:
```
## Total number of normalized counts per sample
colSums(counts(dds, normalized=T))
```

ColoP14_5	5201405.69723949
ColoP14_6	5375120.04909883
ColoP28_2	5311042.43084255
ColoP28_3	5349372.82191332
ColoR34_3	5098096.23044569
ColoR34_5	5451470.98715706
ColoR37_1	5009463.83739973
ColoR37_5	5070615.18871322

In [64]:
```
## Plot dispersion estimates
plotDispEsts(dds)
```



Building the results table

To build our results table we will use the *results()* function. To tell DESeq2 which groups we wish to compare, we supply the contrasts we would like to make using the *contrast* argument. For this example we will save the unshrunken and shrunken version of results to separate variables. Additionally, we are including the *alpha* argument and setting it to 0.05. This is the significant cutoff used for optimizing the independent filtering (by default it is set to 0.1). If the adjusted p-value cutoff (FDR) will be a value other than 0.1 (for our final list of significant genes), *alpha* should be set to that value.

In [70]:

```r
## Define contrasts, extract results table, and shrink the log2 fold chang

contrast_oe <- c("sampletype", "R", "P")

res_tableOE_unshrunken <- results(dds, contrast=contrast_oe, alpha = 0.05)

res_tableOE <- lfcShrink(dds, contrast=contrast_oe, res=res_tableOE_unshru
```

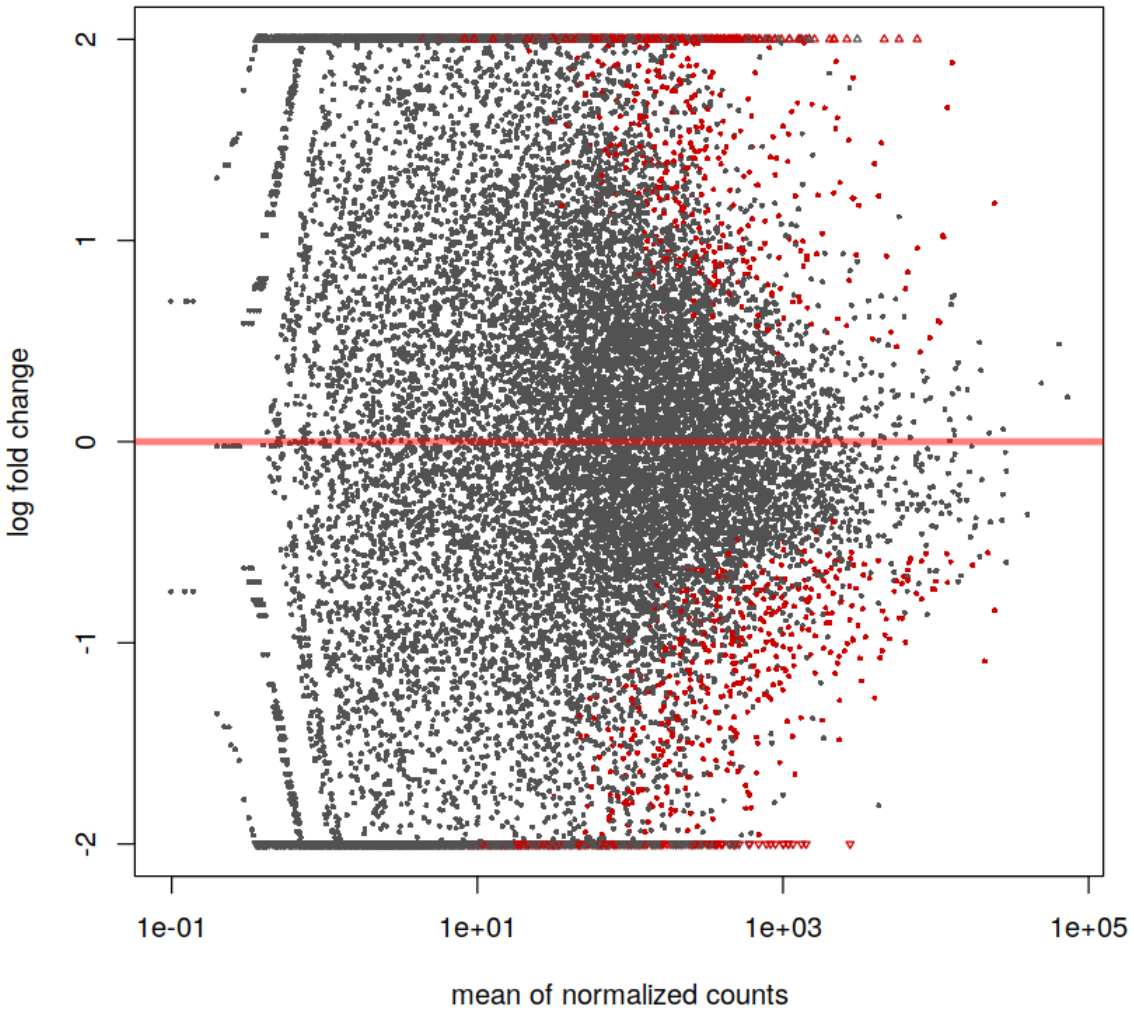
```
using 'normal' for LFC shrinkage, the Normal prior from Love et al (2014).

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)
```

The order of the names determines the direction of fold change that is reported. The name provided in the second element is the level that is used as baseline. So for example, if we observe a log2 fold change of -2 this would mean the gene expression is lower in R relative to the control.

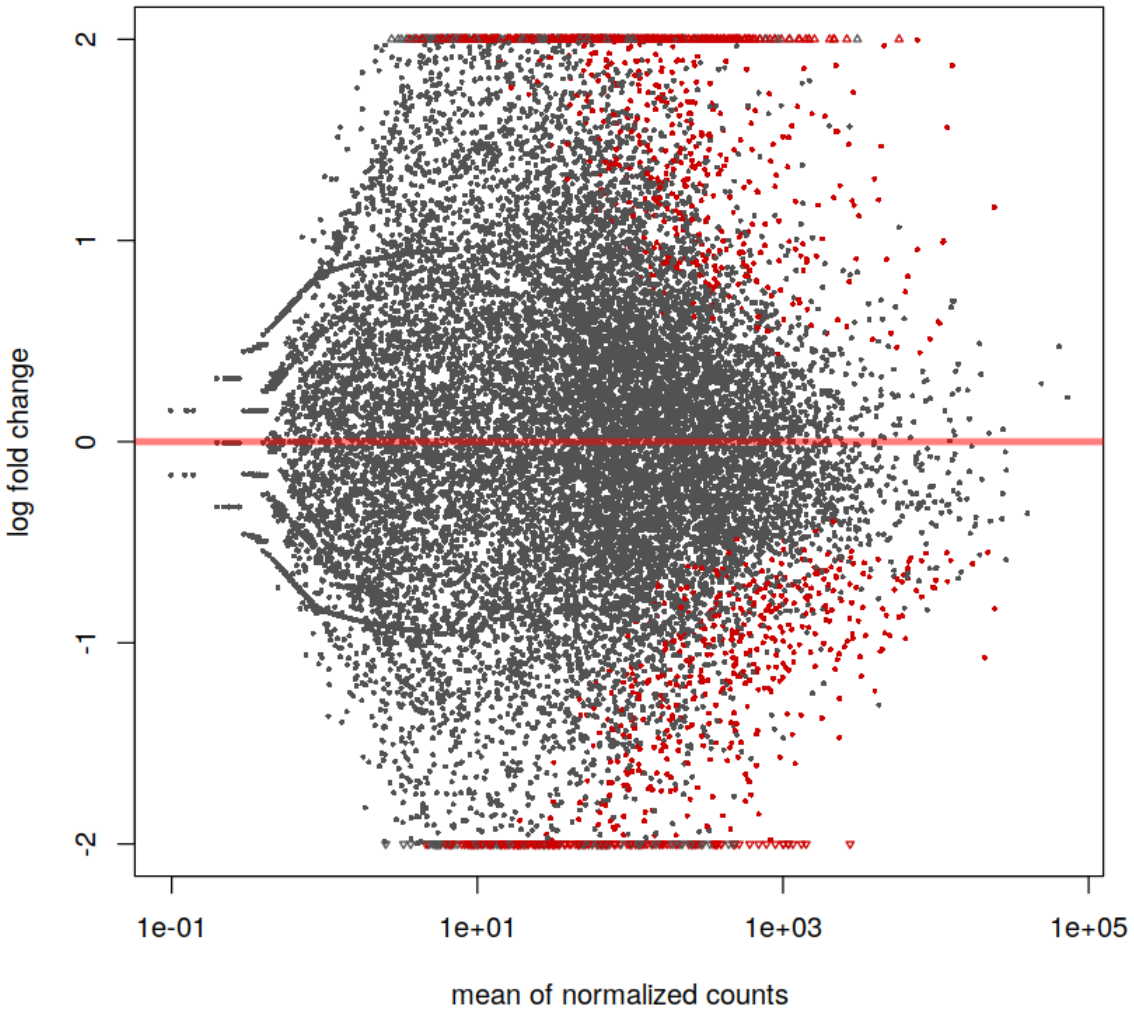
MA Plot

A plot that can be usful to exploring our results is the MA plot. The MA plot shows the mean of the normalized counts versus the log2 foldchanges for all genes tested. The genes that are significantly DE are colored to be easily identified. This is also a great way to illustrate the effect of LFC shrinkage. The DESeq2 package offers a simple function to generate an MA plot.

In [71]:
```
1 plotMA(res_tableOE_unshrunken, ylim=c(-2,2))
```



In [72]:
```
1 plotMA(res_tableOE, ylim=c(-2,2))
```



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.

Results exploration

The results table looks very much like a dataframe and in many ways it can be treated like one.

In [73]:
```
1 class(res_tableOE)
```

'DESeqResults'

Let's go through some of the columns in the results table to get a better idea of what we are looking at. To extract information regarding the meaning of each column we can use *mcols()*:

In [74]:
```
mcols(res_tableOE, use.names=T)
```

```
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 (MAP): sampletype R vs P
lfcSE               results         standard error: sampletype R vs P
stat                results         Wald statistic: sampletype R vs P
pvalue              results      Wald test p-value: sampletype R vs P
padj                results                       BH adjusted p-values
```

In [75]:
```
head(res_tableOE)
```

```
log2 fold change (MAP): sampletype R vs P
Wald test p-value: sampletype R vs P
DataFrame with 6 rows and 6 columns
                        baseMean     log2FoldChange             lfcSE
                       <numeric>          <numeric>         <numeric>
ENSG00000000003   89.146822402102  0.593078344262436 0.473468045151854
ENSG00000000005                 0                 NA                NA
ENSG00000000419  552.183683856479   0.44731542443885 0.216346878754337
ENSG00000000457  16.0077349135033 -0.311340377652959 0.585735341378342
ENSG00000000460  34.9180732079915 -0.283251343748901 0.728431260902359
ENSG00000000938 0.295764339549019  0.448213055161812 0.782146080842955
                             stat            pvalue              padj
                        <numeric>         <numeric>         <numeric>
ENSG00000000003   1.25252749104158 0.210377717426609 0.516064253034451
ENSG00000000005                 NA                NA                NA
ENSG00000000419    2.0675624543363 0.038681185700734  0.19500073001307
ENSG00000000457 -0.531290682962063 0.595217360800855 0.832883219181388
ENSG00000000460 -0.388813234534168 0.697414309932149 0.882014336491528
ENSG00000000938  0.498454004058028 0.618164080995765                NA
```

Summarizing results

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

In [76]:
```
# Summarize results
summary(res_tableOE, alpha = 0.05)
```

```
out of 20029 with nonzero total read count
adjusted p-value < 0.05
LFC > 0 (up)       : 764, 3.8%
LFC < 0 (down)     : 609, 3%
outliers [1]       : 661, 3.3%
low counts [2]     : 5918, 30%
(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 the number of genes not included in multiple test correction due to a low mean count.**

In [78]:
```
# Set thresholds
padj.cutoff <- 0.05
```

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 [77]:

```
res_tableOE_tb <- res_tableOE %>%
  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 threshodlds:

In [79]:

```
sigOE <- res_tableOE_tb %>%
        filter(padj < padj.cutoff)
sigOE
```

A tibble: 1373 × 7

gene	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
ENSG00000001036	846.613414	0.7291648	0.2098064	3.475375	5.101390e-04	9.099960e-03
ENSG00000001497	342.020392	0.7459986	0.2603999	2.864773	4.173079e-03	4.330857e-02
ENSG00000002549	423.870594	-0.6443120	0.1815959	-3.548001	3.881672e-04	7.291688e-03
ENSG00000002586	530.504742	1.0324361	0.2609097	3.956943	7.591502e-05	2.038038e-03
ENSG00000002726	343.693412	2.0887875	0.5904013	3.537895	4.033302e-04	7.534432e-03
ENSG00000003436	160.611814	2.1106567	0.6632581	3.182028	1.462478e-03	2.027870e-02
ENSG00000004779	1548.819903	-0.8370769	0.2637544	-3.173673	1.505233e-03	2.058836e-02
ENSG00000004864	371.123455	-2.8316169	0.8192912	-3.487178	4.881465e-04	8.801958e-03
ENSG00000004866	242.075311	-0.8502915	0.2920264	-2.911564	3.596246e-03	3.892298e-02
ENSG00000005448	385.755470	-1.3859462	0.3018152	-4.591577	4.399092e-06	2.145564e-04
ENSG00000005486	247.771784	1.6300100	0.2318114	7.029964	2.065865e-12	6.777045e-10
ENSG00000005893	880.388777	2.4557824	0.3680240	6.671862	2.525779e-11	6.661124e-09
ENSG00000005981	244.571323	-1.0845491	0.3870826	-2.801595	5.085069e-03	4.988635e-02
ENSG00000006047	71.649593	-1.3815012	0.3419322	-4.037930	5.392500e-05	1.538677e-03
ENSG00000006071	5.199920	2.9767032	0.9365386	3.251490	1.148019e-03	1.678354e-02
ENSG00000006210	16.368436	2.8192911	0.9177540	3.130324	1.746135e-03	2.293507e-02
ENSG00000006712	504.037725	2.2864682	0.6552386	3.491401	4.804953e-04	8.698064e-03
ENSG00000007062	135.737978	1.9973138	0.5301893	3.765589	1.661567e-04	3.781401e-03
ENSG00000007168	157.635499	-1.3771700	0.4914636	-2.801686	5.083636e-03	4.988635e-02
ENSG00000007392	162.786318	1.7271051	0.2693778	6.408603	1.468593e-10	3.347895e-08
ENSG00000007541	394.059063	3.0184434	0.8270791	3.692903	2.217089e-04	4.703446e-03
ENSG00000008130	253.750027	-1.4067874	0.4510508	-3.118538	1.817505e-03	2.366451e-02
ENSG00000008735	7.410318	3.3987605	0.8844821	3.753579	1.743274e-04	3.901338e-03
ENSG00000009413	108.924916	3.3797134	0.7957882	4.266463	1.985961e-05	7.029257e-04
ENSG00000010278	6555.140065	0.8212756	0.2928269	2.804645	5.037201e-03	4.962999e-02
ENSG00000010704	33.372444	-2.1227862	0.6936567	-3.054012	2.258028e-03	2.776095e-02
ENSG00000011275	96.508330	-1.7111291	0.5709646	-2.995344	2.741354e-03	3.234317e-02
ENSG00000011376	78.066070	2.6683594	0.8032766	3.332249	8.614725e-04	1.359954e-02
ENSG00000012061	351.581211	1.2226400	0.2892118	4.227188	2.366300e-05	8.181680e-04
ENSG00000012232	96.411968	2.1095748	0.7172713	2.942482	3.255931e-03	3.652400e-02
⋮	⋮	⋮	⋮	⋮	⋮	⋮
ENSG00000275035	151.403646	2.5296598	0.6689092	3.780756	1.563528e-04	3.582530e-03
ENSG00000275326	18.290380	-3.0254943	0.9233720	-3.349286	8.102019e-04	1.304895e-02
ENSG00000275342	68.909448	2.4639671	0.4949230	4.965511	6.852035e-07	4.888674e-05
ENSG00000275718	329.283091	1.5280712	0.2210552	6.911580	4.792865e-12	1.499164e-09
ENSG00000275903	6855.210394	-0.5775618	0.1449387	-3.984863	6.751904e-05	1.838322e-03
ENSG00000276234	78.982899	-1.9249337	0.6376535	-3.016345	2.558420e-03	3.069648e-02
ENSG00000276345	30.608225	3.0041136	0.8671104	3.476368	5.082550e-04	9.078394e-03

gene	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
ENSG00000276725	5.534469	2.7241789	0.9411331	3.057804	2.229654e-03	2.748749e-02
ENSG00000277147	477.508540	-1.3484141	0.2307095	-5.844217	5.089580e-09	7.606094e-07
ENSG00000277443	283.413053	1.3804298	0.3931938	3.510475	4.473065e-04	8.196556e-03
ENSG00000277594	31.445312	-4.1631390	0.7931012	-5.008863	5.475251e-07	4.160572e-05
ENSG00000277656	999.768169	1.5155819	0.3382370	4.480579	7.444080e-06	3.200663e-04
ENSG00000277998	2128.008083	-0.3962082	0.1224946	-3.234490	1.218600e-03	1.763950e-02
ENSG00000278540	27.112438	-2.5222508	0.9402713	-3.016500	2.557111e-03	3.069648e-02
ENSG00000278550	38.023958	2.3743188	0.5420319	4.362421	1.286309e-05	4.929019e-04
ENSG00000278728	43.830412	2.3329376	0.7225694	3.223908	1.264542e-03	1.820994e-02
ENSG00000278845	609.065134	-1.0388349	0.2768249	-3.752591	1.750164e-04	3.910251e-03
ENSG00000279392	376.068740	-0.6605732	0.2338639	-2.824540	4.734854e-03	4.718074e-02
ENSG00000279968	7.867257	2.2173794	0.9364488	2.850490	4.365187e-03	4.451233e-02
ENSG00000280071	335.232213	-1.5801636	0.3948371	-4.001426	6.296194e-05	1.738611e-03
ENSG00000281348	72.414503	-1.6947798	0.5594046	-3.027465	2.466143e-03	2.982880e-02
ENSG00000281406	9.486457	3.4008819	0.8594481	3.639583	2.730798e-04	5.531510e-03
ENSG00000281618	4.359853	2.6050145	0.9415617	2.873415	4.060604e-03	4.245097e-02
ENSG00000282076	21.123320	3.6370239	0.7379827	4.736306	2.176494e-06	1.258869e-04
ENSG00000283050	12.683236	3.4462572	0.7846510	4.137204	3.515629e-05	1.102483e-03
ENSG00000283082	127.613069	2.3800076	0.6391609	3.721627	1.979435e-04	4.321981e-03
ENSG00000283149	323.282115	-1.1866361	0.4149914	-2.859127	4.248085e-03	4.364915e-02
ENSG00000283189	8.256915	3.4257001	0.9269325	3.792150	1.493486e-04	3.454264e-03
ENSG00000284308	25.388585	-2.0005824	0.6848791	-2.913554	3.573398e-03	3.882245e-02
ENSG00000285250	56.710098	1.6438922	0.5641262	2.912052	3.590630e-03	3.892298e-02

Visualizing the results

When wotking with large amounts of data it can be useful to display that information graphically to gain more insight.

Three different data object will be used for this:

- Metadata for our samples (a dataframe): *meta*
- Normalized expression data for every gene in each of our samples (a matrix): normalized_counts
- tibble versions of the DESeq2 results we generated above: res_tableOE_tb

First create a metadata tibble from the data frame.

In [80]:
```
mov10_meta <- meta %>%
  rownames_to_column(var="samplename") %>%
  as_tibble()
```

Next, let's bring in a column with gene symbols to the *normalized_counts* object, so we can use them to label our plots. Ensembl IDs are great for many things, but as biologists the gene symbol are much more recognizable.

In [81]:

```r
# DESeq2 creates a matrix when you use the counts() function
## First convert normalized_counts to a data frame and transfer the row na
normalized_counts <- counts(dds, normalized=T) %>%
  data.frame() %>%
  rownames_to_column(var="gene")

# Next, merge together (ensembl IDs) the normalized counts data frame with
grch38annot <- tx2gene %>%
  dplyr::select(ensgene, symbol) %>%
  dplyr::distinct()

## This will bring in a column of gene symbols
normalized_counts <- merge(normalized_counts, grch38annot[, c("ensgene", "
```

Once you have the gene names column, you can now convert *normalized_counts* into a tibble.

In [82]:

```
# Now create a tibble for the normalized counts
normalized_counts <- normalized_counts %>%
  as_tibble()

normalized_counts
```

A tibble: 37896 × 10

gene	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<db
ENSG00000000003	62.935029	89.060209	31.19157	95.8619839	106.4751622	50.631600	133.0414
ENSG00000000005	0.000000	0.000000	0.00000	0.0000000	0.0000000	0.000000	0.0000
ENSG00000000419	454.898806	389.226098	483.46935	536.3517614	613.6124165	763.524532	650.3147
ENSG00000000457	15.457726	9.895579	23.39368	22.9751862	16.5628030	7.088424	23.8283
ENSG00000000460	12.145356	56.074946	56.53472	31.6899120	36.2804257	72.909504	5.9570
ENSG00000000938	0.000000	0.000000	0.00000	0.0000000	2.3661147	0.000000	0.0000
ENSG00000000971	0.000000	4.398035	19.49473	2.3767434	3.1548196	11.138952	31.7711
ENSG00000001036	568.623508	538.759288	773.94086	655.9811794	925.9395590	1117.945735	997.8111
ENSG00000001084	896.548133	372.733466	825.60190	401.6696352	358.0720271	440.494923	345.5107
ENSG00000001167	164.514374	101.154805	71.15577	88.7317537	85.1801298	88.098985	36.7353
ENSG00000001460	64.039152	20.890666	68.23156	15.8449560	10.2531638	18.227376	48.6495
ENSG00000001461	68.455646	43.980350	101.37261	197.2697025	110.4186868	28.353696	89.3562
ENSG00000001497	323.508132	260.583574	219.31573	212.3224107	317.0593720	465.810723	464.6523
ENSG00000001561	4.416493	21.990175	20.46947	9.5069736	21.2950324	11.138952	23.8283
ENSG00000001617	2.208247	0.000000	0.00000	0.0000000	0.7887049	0.000000	2.9785
ENSG00000001626	0.000000	0.000000	0.00000	0.7922478	0.0000000	0.000000	1.9856
ENSG00000001629	54.102042	162.727295	15.59579	15.0527082	59.9415728	7.088424	114.1773
ENSG00000001630	454.898806	362.837888	344.08202	299.4696689	423.5345342	411.128594	608.6151
ENSG00000001631	13.249480	85.761683	10.72210	28.5209208	24.4498521	2.025264	270.0543
ENSG00000002016	71.768016	0.000000	5.84842	0.0000000	34.7030158	9.113688	2.9785
ENSG00000002079	0.000000	0.000000	0.00000	3.1689912	0.0000000	0.000000	0.0000
ENSG00000002330	418.462737	425.509886	378.19780	423.0603258	274.4693071	435.431763	232.3261
ENSG00000002549	558.686398	480.485324	490.29251	541.1052482	287.0885856	368.598050	314.7324
ENSG00000002586	393.067901	351.842800	320.68834	314.5223771	500.0389101	651.122380	791.2990
ENSG00000002587	80.601002	23.089684	14.62105	11.0914692	40.2239502	4.050528	132.0486
ENSG00000002726	94.954605	96.756770	149.13470	110.1224444	157.7409811	162.021121	894.5550
ENSG00000002745	0.000000	0.000000	0.00000	0.0000000	0.0000000	0.000000	0.0000
ENSG00000002746	0.000000	28.587228	0.00000	5.5457346	1.5774098	0.000000	0.0000
ENSG00000002822	566.415261	608.028339	605.31143	676.5796222	832.8723802	909.343541	667.1931
ENSG00000002834	1247.659347	876.308474	999.10501	897.6167588	825.7740360	702.766612	837.9628
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
ENSG00000285862	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285868	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285880	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285891	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285897	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285901	91.64224	0.00000	136.463124	45.15812	0.00000	0.00000	0.0000

gene	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<db
ENSG00000285912	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285913	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285920	108.20408	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285927	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285932	0.00000	0.00000	3.898946	0.00000	0.00000	0.00000	0.0000
ENSG00000285937	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285938	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285942	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285943	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285944	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285946	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285947	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285950	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285953	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285956	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285962	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285975	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285976	0.00000	21.99018	28.267361	0.00000	14.19669	139.74322	5.9570
ENSG00000285978	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285982	0.00000	0.00000	0.000000	0.00000	0.00000	12.15158	0.0000
ENSG00000285986	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285989	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285990	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000
ENSG00000285991	0.00000	0.00000	0.000000	0.00000	0.00000	0.00000	0.0000

Plotting significant DE genes

One way to visualize results would be to simply plot the expression data for a handful of genes. We could do that by picking out specific genes of interest or selecting a range of genes.

Using DESeq2 *plotCounts()* to plot expression of a single gene

To pick out a specific gene of interest to plot, for example GAPDH, we can use the *plotCounts()* from DESeq2. *plotCounts()* requires that the gene specified matches the original input to DESeq2, which in our case was Ensembl IDs.

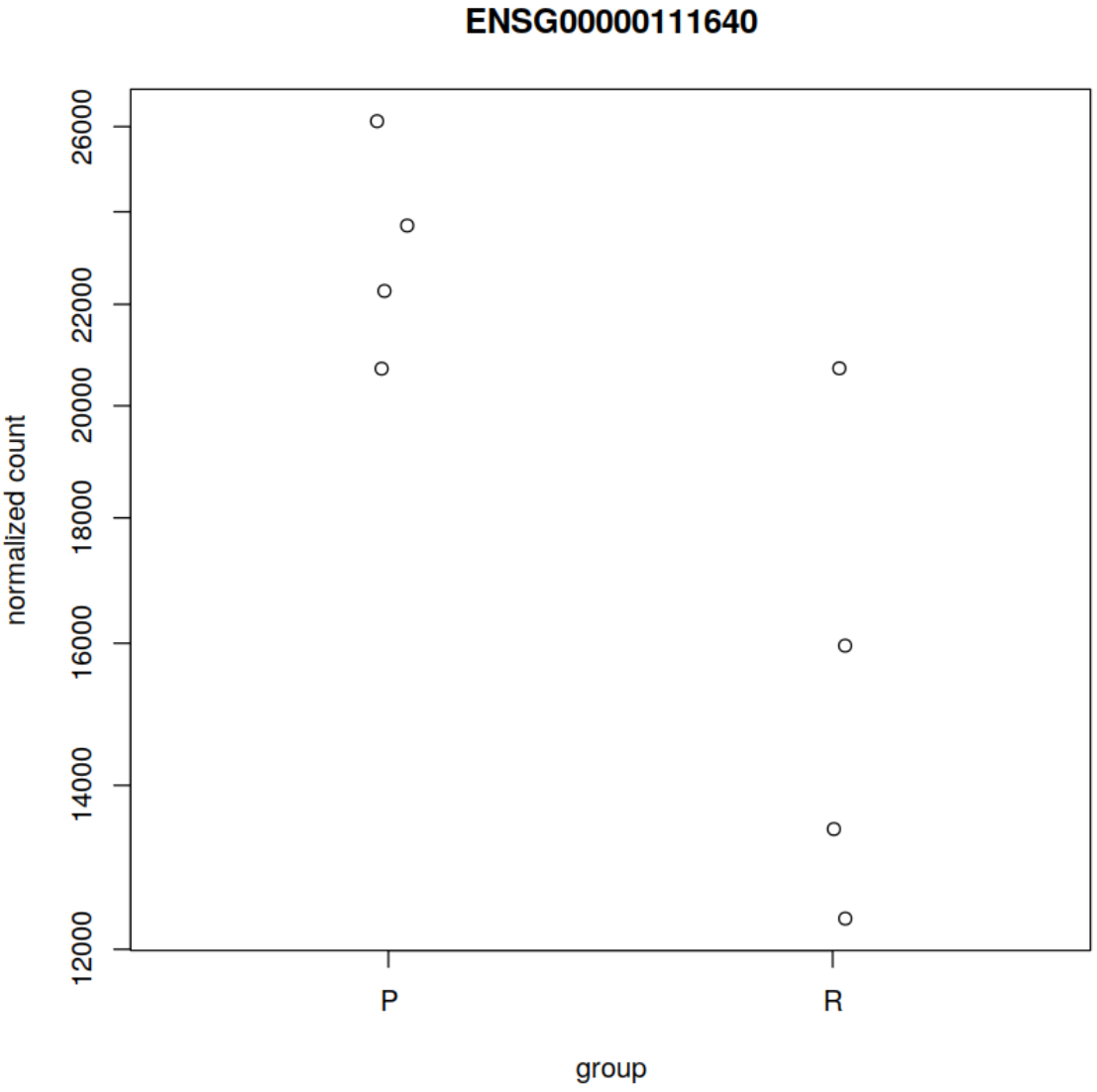
In [83]:
```
# Find the Ensembl ID
grch38annot[grch38annot$symbol == "GAPDH", "ensgene"]
```

ENSG00000111640

▶ **Levels**:

In [84]:

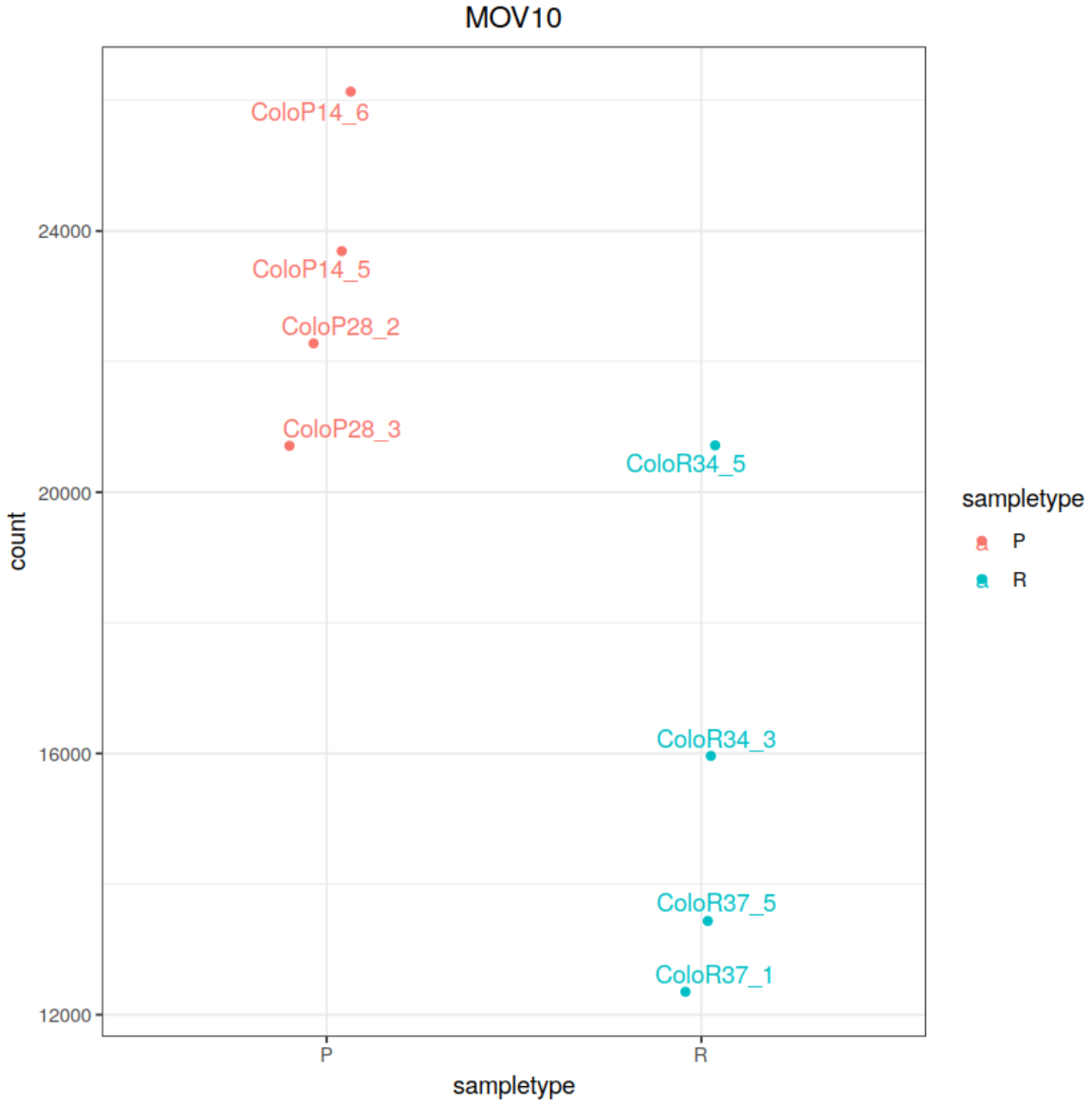
```
# Plot expression for single gene
plotCounts(dds, gene="ENSG00000111640", intgroup="sampletype")
```



Using ggplot2 to plot expression of a single gene If you wish to change the appearance of this plot , we can save the output of *plotCounts()* to a variable specifying the *returnData=TRUE* argument, then use *ggplot()*:

In [85]:

```
# Save plotcounts to a data frame object
d <- plotCounts(dds, gene="ENSG00000111640", intgroup="sampletype", return

# Plotting the Colo normalized counts, using the samplenames (rownames of
ggplot(d, aes(x = sampletype, y = count, color = sampletype)) +
  geom_point(position=position_jitter(w = 0.1,h = 0)) +
  geom_text_repel(aes(label = rownames(d))) +
  theme_bw() +
  ggtitle("Colo") +
  theme(plot.title = element_text(hjust = 0.5))
```



Using *ggplot()* to plot multiple genes (e.g. top 20). Often it is helpful to check the expression of multiple genes of interst at the same time. This often first requires some data wrangling.

We are going to plot the normalized count values for the **top 20 differentially expressed genes (by padj values)**.

To do this, we 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 [86]:

```
# 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
```

Then, we can extract the normalized count values for these top 20 genes:

In [87]:
```
# normalized counts for top 20 significant genes
top20_sigOE_norm <- normalized_counts %>%
        filter(gene %in% top20_sigOE_genes)
```

Now that we have the normalized ciunts for each of the top 20 genes for all 8 samples, to plot using *ggplot()*, we need to gather the counts for all samples into a single column to allow us to give *ggplot* the one column with the values we want it to plot. The *gather()* function in the **tidyr** package will perform this operation and will output the normalized counts for all genes listed in the first 20 rows, followed by the normalized counts for int the next 20 rows.

In [89]:
```
# 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

# check the column header in the "gathered" data frame
head(gathered_top20_sigOE)
```

A tibble: 6 × 4

gene	symbol	samplename	normalized_counts
<chr>	<fct>	<chr>	<dbl>
ENSG00000044574	HSPA5	ColoP14_5	3725.3121
ENSG00000088992	TESC	ColoP14_5	214.1999
ENSG00000101558	VAPA	ColoP14_5	1274.1583
ENSG00000101608	MYL12A	ColoP14_5	2960.1546
ENSG00000102007	PLP2	ColoP14_5	351.1112
ENSG00000110651	CD81	ColoP14_5	234.0741

Now, if we want our counts colored by sample group, then we need to combine the metadata information with the melted normalized counts data into the same data frame for input to *ggplot()*:

In [92]:
```
gathered_top20_sigOE <- inner_join(mov10_meta, gathered_top20_sigOE)
head(gathered_top20_sigOE)
```

```
Joining, by = c("samplename", "sampletype")
```

A tibble: 6 × 5

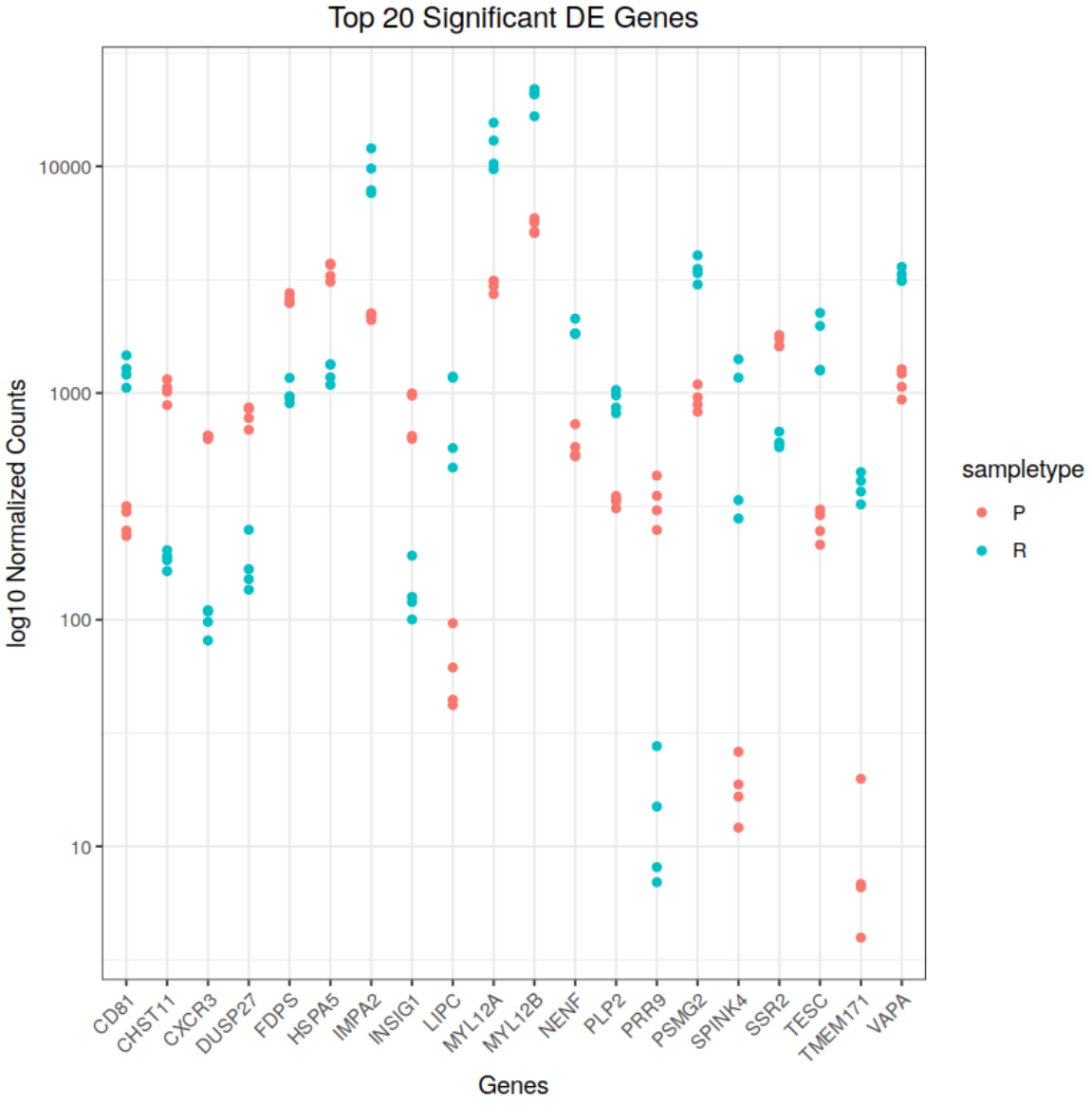
samplename	sampletype	gene	symbol	normalized_counts
<chr>	<fct>	<chr>	<fct>	<dbl>
ColoP14_5	P	ENSG00000044574	HSPA5	3725.3121
ColoP14_5	P	ENSG00000088992	TESC	214.1999
ColoP14_5	P	ENSG00000101558	VAPA	1274.1583
ColoP14_5	P	ENSG00000101608	MYL12A	2960.1546
ColoP14_5	P	ENSG00000102007	PLP2	351.1112
ColoP14_5	P	ENSG00000110651	CD81	234.0741

The *inner_join()* will merge 2 data frames with respect to the "samplename" column, i.e. a column with the same column name in both data frames.

Now we have a data frame in a format that can be utilised by ggplot easily.

In [91]:

```
## plot using ggplot2
ggplot(gathered_top20_sigOE) +
        geom_point(aes(x = symbol, y = normalized_counts, color = samplety
        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))
```



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()*.

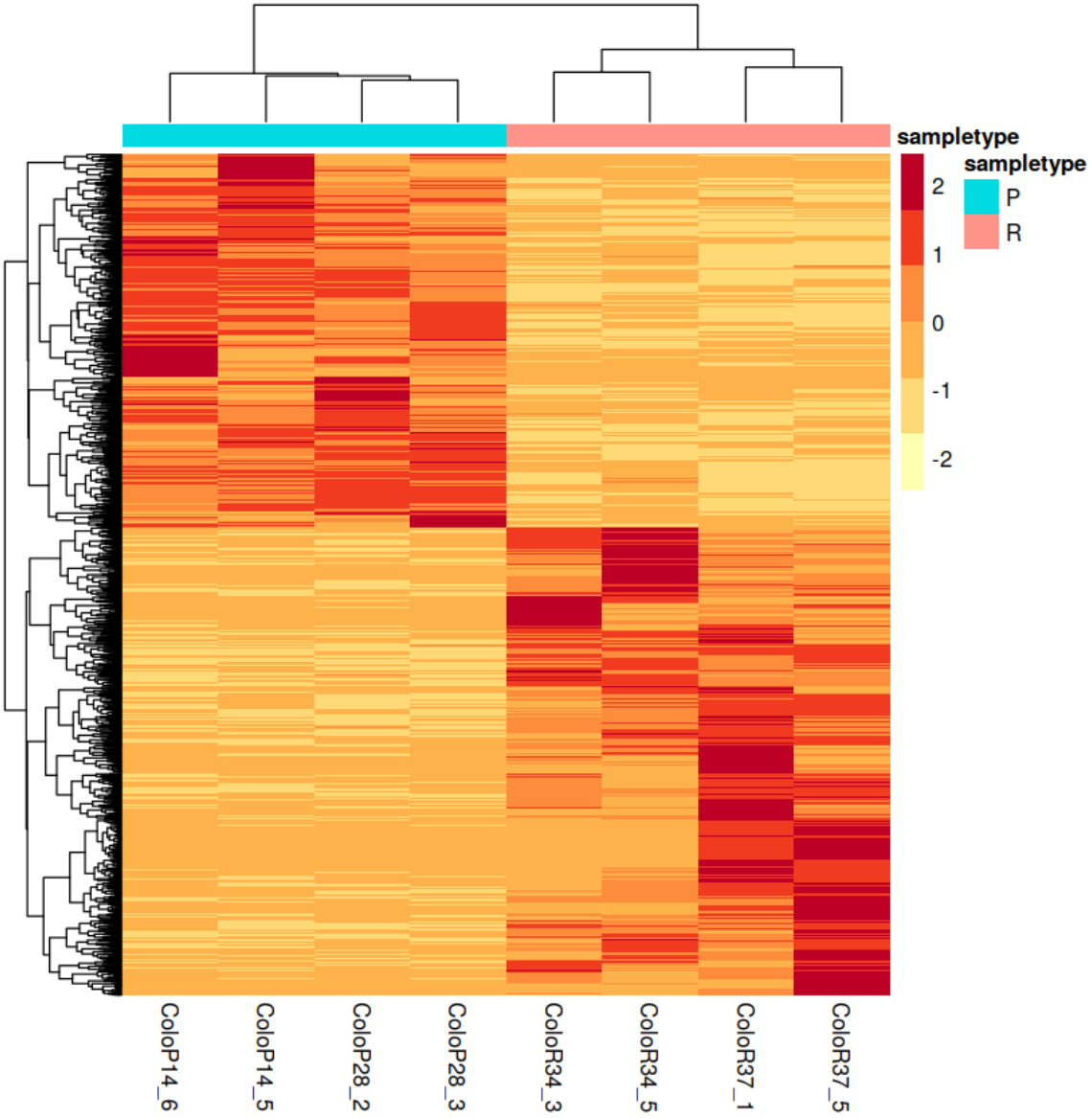
In [93]:

```
# Extract normalized expression for significant genes from the OE and cont
norm_OEsig <- normalized_counts[,c(1:9)] %>%
              filter(gene %in% sigOE$gene)
```

Now lets draw the heatmap using *pheatmap*.

In [95]:

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

### Run pheatmap using the metadata data frame for the annotation
pheatmap(norm_OEsig[2:9],
         color = heat_colors,
         cluster_rows = T,
         show_rownames = F,
         annotation = meta,
         border_color = NA,
         fontsize = 10,
         scale = "row",
         fontsize_row = 10,
         height = 20)
```



!Note There are several additional arguments we have included in the function for aesthetics. One important one is *scale="row"*, in which Z-scores are plotted, rather than the actual normalized count value. Z-scores are computed on a gene-by-gene basis by subtractin the mean and then dividing by standard deviation. The Z-scores are computed **after the clustering**, so that in only affects the graphical aesthetics and the color visualization is improved.

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 indicating whether or not the gene is considered differentially expressed based on p-adjusted values and we will include a log2fold change here.

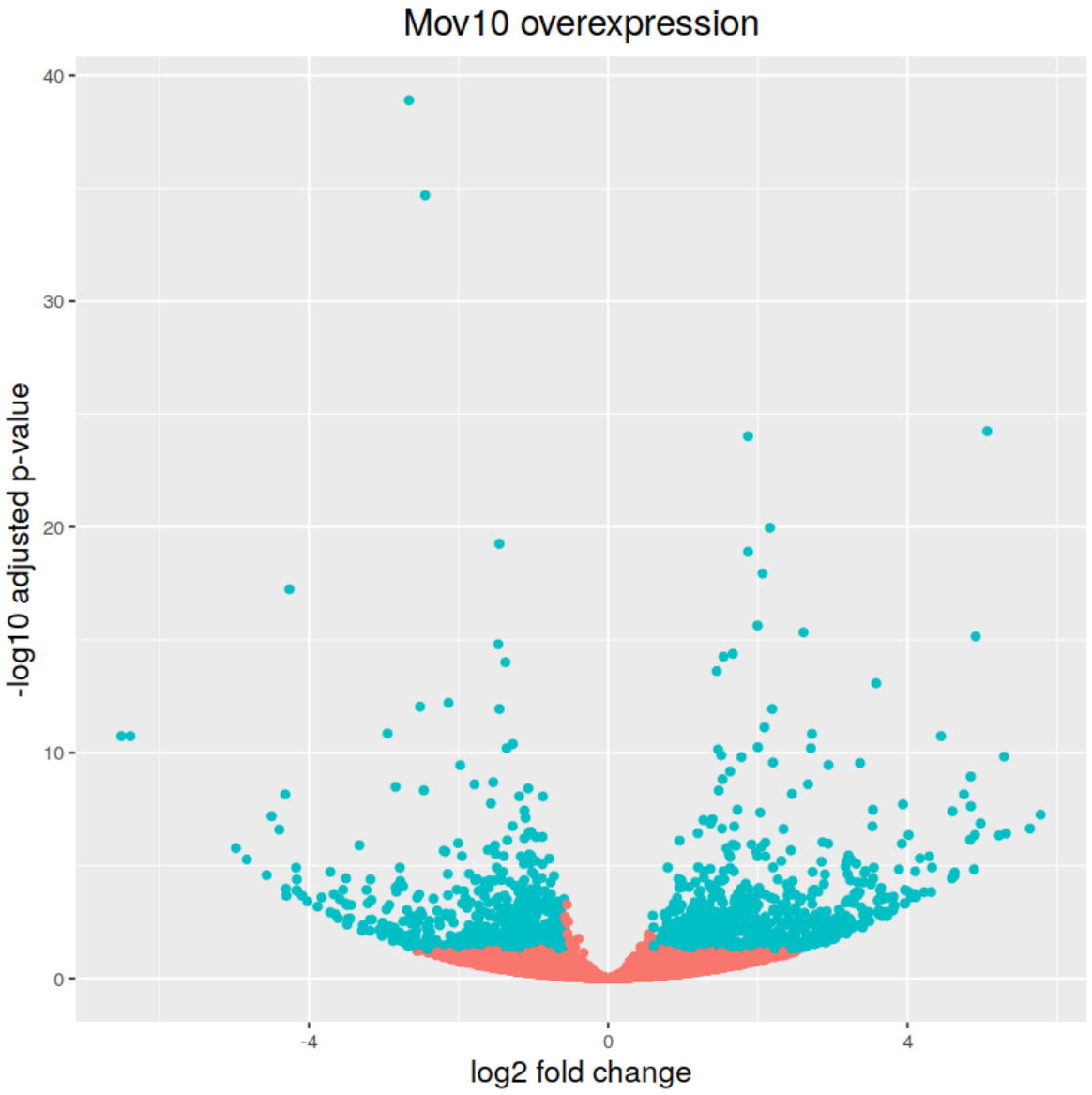
In [96]:
```
# Obtain logical vector where TRUE values denote padj values < 0.05 and fo

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

Now we can start plotting. The *geom_point* object is most applicable, as this is essentially a scatter plot:

In [97]:

```
# Volcano plot
ggplot(res_tableOE_tb) +
        geom_point(aes(x = log2FoldChange, y = -log10(padj), colour = thre
        ggtitle("Colo 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 24446 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 (lowest padj) in our DE list are located on this plot? We could label those dots with the gene name on the Volcano plot usin *geom_text_repel()*.

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

In [99]:

```
## Sort the results tibble by padj values and create a column to indicate
res_tableOE_tb <- res_tableOE_tb %>% arrange(padj) %>% mutate(genelabels =

## Add the gene symbols as a column to the res_tableOE tibble from the grc
res_tableOE_tb <- bind_cols(res_tableOE_tb, symbol=grch38annot$symbol[matc

### In the line above, you could have also used the merge() function as we

## Populate the genelables column with information from the new symbol col
res_tableOE_tb$genelabels[1:10] <- as.character(res_tableOE_tb$symbol[1:10

head(res_tableOE_tb)
```

A tibble: 6 × 11

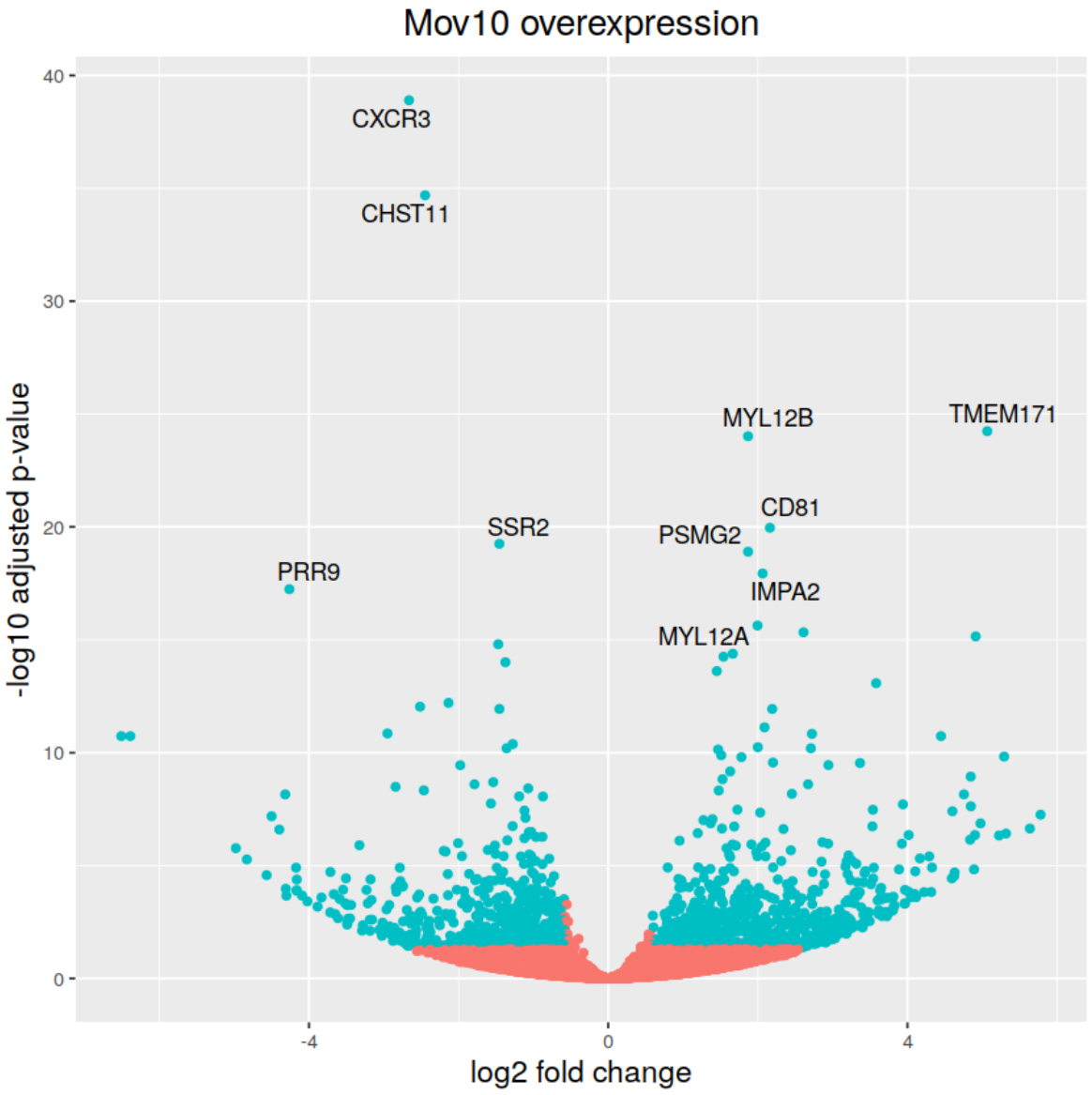
gene	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj	threshold_(
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<lg
ENSG00000186810	370.4062	-2.662834	0.1918363	-13.872054	9.355919e-44	1.258371e-39	TRU
ENSG00000171310	605.4285	-2.447494	0.1866875	-13.106387	3.026770e-39	2.035503e-35	TRU
ENSG00000157111	198.0894	5.067342	0.4501828	11.098127	1.280998e-28	5.743142e-25	TRU
ENSG00000118680	12787.6380	1.869004	0.1695049	11.026187	2.857236e-28	9.607455e-25	TRU
ENSG00000110651	763.0351	2.161753	0.2133574	10.130497	4.045883e-24	1.088343e-20	TRU
ENSG00000163479	1151.6058	-1.455216	0.1462316	-9.951072	2.494809e-23	5.592530e-20	TRU

Next, we plot as before with an additional layer for *geom_text_repel()* wherein we can specify the column of gene labels we just created.

In [100]:

```
ggplot(res_tableOE_tb, aes(x = log2FoldChange, y = -log10(padj))) +
        geom_point(aes(colour = threshold_OE)) +
        geom_text_repel(aes(label = genelabels)) +
        ggtitle("Colo  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 24446 rows containing missing values (geom_point).”Warning message:
“Removed 24446 rows containing missing values (geom_text_repel).”
```



Genomic annotations

The analysis of next-generation sequencing results requires associating genes, transcripts, proteins, etc. with functional or regulatory information. To perform functional analysis on gene lists, we often need to obtain gene identifiers that are compatible with the tools we wish to use and this is not always trivial. Here, we discuss **ways in which you can obtain gene annotation information an some of advantages and disadvantages of each method.**

Genome builds

When a new genome build is acquired, the names and/or coordinate location of genomic features (gene, transcript, exon, etc.) may change. Therefor, the annotations regarding genome features (gene, transcript, exon, etc.) is genome-build specific and we need to make sure that our annotations are obtained from the appropriate resource. You should know which **build of the genome** was used to generate your gene list and make sure you use the **same build for the annotations** during functional analysis.

For example, if we used the GRCh38 build of the human genome to quantify gene expresson used for differential expression analysis, then we should use the **same GRCh38 build** of the genome to convert between IDs and to identify annotations for each of the genes.

Databases

We retrieve information on the processes, pathways, etc. for which a gene is involved, from the necessary database where the information is stored. The database you choose will be dependent on what type of information you are trying to obtain. Examples of databases that are often queried, include:

General databases

Offer comprehensive information on genome features, feature coordinates, homology, variant information, phenotypes, protein domain/family information, associated biological processes/pathways, associated microRNAs, etc.:

- **Ensembl** (use Ensembl gene IDs)
- **NCBI** (use Entrey gene IDs)
- **UCSC**
- **EMBL-EBI**

Annotation-specific databases

Provide annotations related to a specific topic:

- **Gene Ontology(GO)**: database of gene ontology biological processes, cellular components and molecular functions based on Ensembl or Entrez gene IDs or official gene symbols
- **KEGG**: database of biological pathways based on Entrez IDs
- **MSigDB**: database of gene sets
- **Reactome**: database of biological pathways
- **Human Phenotype Ontology**: database of genes associated with human disease
- **CORUM**: database of protein complexes for human, mouse, rat

This is by no means an exhaustive list, there are many other databases available that are not listed here.

Tools

When performing functional analysis, the tools will take the list of genes you provide and retrieve information for each gene using one or more of these databases. Within R, there are many popular packages used for gene/transcript-level annotation:

Interface tools: for accessing/querying annotations from multiple different annotations databases

- **AnnotationDBi**: queries the OrgDb, TxDb, Go.db, EnsDb, and BioMart annotations.
- **AnnotationsHub**: queries large collection of whole genome resources, including ENSEMBL, UCSC, ENCODE, Broad Institute, KEGG, NIH Pathway Interaction Database, etc.

!Note: These are both packages that can be used to create the *tx2gene* files which were used in the beginning.

Annotation tools: for accessing/querying annotations from a specific database

- **org.Gs.eg.db**: these OrgDb annotation tools query gene feature information for the organism of interest, including gene IDs and associated GO and KEGG IDs, but unable to get previous gene build information easily

- **EnsDb.Gspecies.v86**: Ensembl database for transcript and gene-level information directly fetched from Ensembl API (similar to TxDb, but with filtering ability and versioned by Ensembl release) or can create using the ensembldb package
- **TxDb.Gspecies.UCSC.HG19.knownGene**: UCSC database for transcript and gene-level information or can create own TxtDb from an SQLite database file using the *GenomicFeatures* package.
- **annotables:** easy-to-use package making gene-level feature information immediately available for the current and most recent genome builds for human/mouse.
- **biomaRt**: wealth of information available by querying Ensembl's database using their BioMart online tool (http://www.ensembl.org/biomart/martview/52696953d67820bc310d9572b735e6ad) all previous genome builds and gene feature, structure, homolgy, variant, and sequence information available and connects to external databases

AnnotationDbi

AnnotationDbi is an R package that provides an interface for connecting and querying various annotation databases using SQLite data storage. The AnnotationDbi packages can query the *OrgDb, TxDb, EnsDb, Go.db,* and *BioMart* annotations. There is helpful documentation (https://bioconductor.org/packages/release/bioc/vignettes/AnnotationDbi/inst/doc/IntroToAnnotationPackag available to reference when extracting data from any of these databases.

org.Hs.eg.db

There are a plethora of organism-specific *orgDb* packages, such as *org.Hs.eg.db* for human *org.Mm.eg.db* for mouse. These databases are best for converting gene IDs or obtaining GO information for current genome builds, but not for older genome builds. These packages provide the current builds corresponding to the release data of the package, and update every 6 months. If a package is not available for your organism of interest, you can create you own using *AnnnotationsHub*.

In [103]:

```
# Load libraries
library(org.Hs.eg.db)
library(AnnotationDbi)

# Check object metadata
org.Hs.eg.db
```

```
Loading required package: AnnotationDbi

Attaching package: ‘AnnotationDbi’

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

    select


Please see: help('select') for usage information
```

```
OrgDb object:
| DBSCHEMAVERSION: 2.1
| Db type: OrgDb
| Supporting package: AnnotationDbi
| DBSCHEMA: HUMAN_DB
| ORGANISM: Homo sapiens
| SPECIES: Human
| EGSOURCEDATE: 2019-Apr26
| EGSOURCENAME: Entrez Gene
| EGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA (ftp://ftp.ncbi.nlm.nih.gov/gene/DATA)
| CENTRALID: EG
| TAXID: 9606
| GOSOURCENAME: Gene Ontology
| GOSOURCEURL: ftp://ftp.geneontology.org/pub/go/godatabase/archive/latest-lite/ (ftp://ftp.geneontology.org/pub/go/godatabase/archive/latest-lite/)
| GOSOURCEDATE: 2019-Apr24
| GOEGSOURCEDATE: 2019-Apr26
| GOEGSOURCENAME: Entrez Gene
| GOEGSOURCEURL: ftp://ftp.ncbi.nlm.nih.gov/gene/DATA (ftp://ftp.ncbi.nlm.nih.gov/gene/DATA)
| KEGGSOURCENAME: KEGG GENOME
| KEGGSOURCEURL: ftp://ftp.genome.jp/pub/kegg/genomes (ftp://ftp.genome.jp/pub/kegg/genomes)
| KEGGSOURCEDATE: 2011-Mar15
| GPSOURCENAME: UCSC Genome Bioinformatics (Homo sapiens)
| GPSOURCEURL:
| GPSOURCEDATE: 2018-Dec3
| ENSOURCEDATE: 2019-Apr08
| ENSOURCENAME: Ensembl
| ENSOURCEURL: ftp://ftp.ensembl.org/pub/current_fasta (ftp://ftp.ensembl.org/pub/current_fasta)
| UPSOURCENAME: Uniprot
| UPSOURCEURL: http://www.UniProt.org/ (http://www.UniProt.org/)
| UPSOURCEDATE: Fri Apr 26 20:12:58 2019
```

We can see the metadata for the database by just typing the name of the database, including the species, last updates for the different source information, and the source urls. Note the KEGG data from this database was last updated in 2011, so may not be the best site for that information.

We can easily extract information from this database using *AnnotationDbi* with the methods: *columns, keys, keytypes,* and *select*. For example, we will use our *org.Hs.eg.db* database to acquire information, but know that the same methods work for the *TxDb, Go.db, EnsDb, BioMart* annotations.

In [104]:
```
# Return the Ensembl IDs for a set of genes
annotations_orgDb <- AnnotationDbi::select(org.Hs.eg.db, # database
                                           keys = res_tableOE_tb$gene,  # data t
                                           columns = c("SYMBOL", "ENTREZID","GEN
                                           keytype = "ENSEMBL") # type of data g
```

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

This easily retuned to us the information that we desired, but note the *warning* returned. This is always going to happen with converting between different gene IDs. Unless we would like to keep multiple mappings for a single gene, then we probably want to de-duplicate our data before using it.

In [105]:
```
# Determine the indices for the non-duplicated genes
non_duplicates_idx <- which(duplicated(annotations_orgDb$SYMBOL) == FALSE)

# Return only the non-duplicated genes using indices
annotations_orgDb <- annotations_orgDb[non_duplicates_idx, ]
```

Note that if your analysis was conducted using an older genome (i.e hg19) some genes maybe found to be not annotated (NA), since orgDB is always the most recent release. It is likely that some of the genes have changed names in between versions (due to updates and patches), so may not be present in this version of the database. Our dataset was created based on the GRCH38 build of the human genome, using a recent release of Ensembl as our reference and so we should not see much of a discrepancy.

EnsDb.Hsapiens.v86

To generate the Ensembl annotations, the *EnsDb* database can also be easily queried using AnnotationsDbi. You will need to decide the release of Ensembl you would like to query. All Ensembl releases are listed here (http://www.ensembl.org/info/website/archives/index.html). We know that our data is for GRCh38, and the most current release for GRCh38 in Bioconductor is release 86, so we can install this release of the *EnsDb* database. **Note: this is not the most current release, yet it is the latest release available through AnnotationDbi.**

Since we are using *AnnotationsDbi* to query the database, we can use the same functions that we used previously:

In [108]:
```
# Load the library
library(EnsDb.Hsapiens.v86)

# Check object metadata
EnsDb.Hsapiens.v86

# Explore the fields that can be used as keys
keytypes(EnsDb.Hsapiens.v86)
```

```
Loading required package: ensembldb
Loading required package: GenomicFeatures
Loading required package: AnnotationFilter

Attaching package: 'ensembldb'

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

    filter

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

    filter


EnsDb for Ensembl:
|Backend: SQLite
|Db type: EnsDb
|Type of Gene ID: Ensembl Gene ID
|Supporting package: ensembldb
|Db created by: ensembldb package from Bioconductor
|script_version: 0.3.0
|Creation time: Thu May 18 16:32:27 2017
|ensembl_version: 86
|ensembl_host: localhost
|Organism: homo_sapiens
|taxonomy_id: 9606
|genome_build: GRCh38
|DBSCHEMAVERSION: 2.0
| No. of genes: 63970.
| No. of transcripts: 216741.
|Protein data available.
```

'ENTREZID' 'EXONID' 'GENEBIOTYPE' 'GENEID' 'GENENAME' 'PROTDOMID' 'PROTEINDOMAINID' 'PROTEINDOMAINSOURCE' 'PROTEINID' 'SEQNAME' 'SEQSTRAND' 'SYMBOL' 'TXBIOTYPE' 'TXID' 'TXNAME' 'UNIPROTID'

Now we can return all gene IDs for our gene list:

In [109]:
```
# Return the Ensembl IDs for a set of genes
annotations_edb <- AnnotationDbi::select(EnsDb.Hsapiens.v86,
                                         keys = res_tableOE_tb$gene,
                                         columns = c("SYMBOL", "ENTREZID
                                         keytype = "GENEID")
```

Then we can again deduplicate:

In [110]:
```
# Determine the indices for the non-duplicated genes
non_duplicates_idx <- which(duplicated(annotations_edb$SYMBOL) == FALSE)

# Return only the non-duplicated genes using indices
annotations_edb <- annotations_edb[non_duplicates_idx, ]
```

AnnotationsHub

AnnotationsHub is a resource for accessing genomic data or querying large collection of whole genome resources, including ENSEMBL, UCSC, ENCODE, Broad Institute, KEGG, NIH Pathway Interaction Database, etc. All of this information is stored and easily accessible by directly connecting to the database.

To get started with AnnotationsHub, we first load the libray and connect to the database:

In [111]:
```
# Load libraries
library(AnnotationHub)
library(ensembldb)

# Connect to AnnotationHub
ah <- AnnotationHub()
```

```
Loading required package: BiocFileCache
Loading required package: dbplyr

Attaching package: 'dbplyr'

The following objects are masked from 'package:dplyr':

    ident, sql


Attaching package: 'AnnotationHub'

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

    cache

Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
snapshotDate(): 2019-05-02
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
```

To see the types of information stored inside, we can just type the name of the object:

In [112]:
```
# Explore the AnnotationHub object
ah
```

```
AnnotationHub with 44904 records
# snapshotDate(): 2019-05-02
# $dataprovider: BroadInstitute, Ensembl, UCSC, ftp://ftp.ncbi.nlm.nih.gov/
g... (ftp://ftp.ncbi.nlm.nih.gov/g...)
# $species: Homo sapiens, Mus musculus, Drosophila melanogaster, Bos tauru
s,...
# $rdataclass: GRanges, BigWigFile, TwoBitFile, Rle, OrgDb, EnsDb, ChainFil
e...
# additional mcols(): taxonomyid, genome, description,
#   coordinate_1_based, maintainer, rdatadateadded, preparerclass, tags,
#   rdatapath, sourceurl, sourcetype
# retrieve records with, e.g., 'object[["AH5012"]]'

           title
  AH5012  | Chromosome Band
  AH5013  | STS Markers
  AH5014  | FISH Clones
  AH5015  | Recomb Rate
  AH5016  | ENCODE Pilot
  ...       ...
  AH73982 | Ensembl 97 EnsDb for Xiphophorus couchianus
  AH73983 | Ensembl 97 EnsDb for Xiphophorus maculatus
  AH73984 | Ensembl 97 EnsDb for Xenopus tropicalis
  AH73985 | Ensembl 97 EnsDb for Zonotrichia albicollis
  AH73986 | Ensembl 79 EnsDb for Homo sapiens
```

Using the output, you can get an idea of the information that you can query within the AnnotationHub object. Notice the nore on retrieving records with *object[[AH2]]* this will be how we can extract a single record from the AnnotationHub object.

If you would like to see more information about any of these classes of data you can extract that information as well. For example, if you wanted to determine all species information available, you could subset the AnnotationsHub object:

In [113]:
```
# Explore all species information available
unique(ah$species)
```

```
'Homo sapiens' 'Vicugna pacos' 'Dasypus novemcinctus' 'Otolemur garnettii' 'Papio hamadryas'
'Papio anubis' 'Felis catus' 'Pan troglodytes' 'Bos taurus' 'Canis familiaris' 'Tursiops truncatus'
'Loxodonta africana' 'Nomascus leucogenys' 'Gorilla gorilla' 'Cavia porcellus' 'Equus caballus'
'Dipodomys ordii' 'Trichechus manatus' 'Callithrix jacchus' 'Pteropus vampyrus' 'Myotis lucifugus'
'Mus musculus' 'Microcebus murinus' 'Heterocephalus glaber' 'Monodelphis domestica'
'Pongo abelii' 'Ailuropoda melanoleuca' 'Sus scrofa' 'Ochotona princeps'
'Ornithorhynchus anatinus' 'Oryctolagus cuniculus' 'Rattus norvegicus' 'Macaca mulatta'
'Procavia capensis' 'Ovis aries' 'Sorex araneus' 'Petromyzon marinus' 'Anolis carolinensis'
'Oryzias latipes' 'Geospiza fortis' 'Oreochromis niloticus' 'Chrysemys picta'
'Gasterosteus aculeatus' 'Tetraodon nigroviridis' 'Meleagris gallopavo' 'Xenopus tropicalis'
'Taeniopygia guttata' 'Danio rerio' 'Ciona intestinalis' 'Branchiostoma floridae'
'Strongylocentrotus purpuratus' 'Apis mellifera' 'Anopheles gambiae' 'Drosophila ananassae'
'Drosophila erecta' 'Drosophila grimshawi' 'Drosophila melanogaster' 'Drosophila mojavensis'
'Drosophila persimilis' 'Drosophila pseudoobscura' 'Drosophila sechellia' 'Drosophila simulans'
'Drosophila virilis' 'Drosophila yakuba' 'Caenorhabditis brenneri' 'Caenorhabditis briggsae'
'Caenorhabditis elegans' 'Caenorhabditis japonica' 'Caenorhabditis remanei'
'Pristionchus pacificus' 'Aplysia californica' 'Saccharomyces cerevisiae'
```

Now that we know the types of information available from AnnotationsHub we can query it for the information we want using the *query()* function. Let's say we would like to return the Ensembl *EnsDb* information for human. To return the records available, we need to use the terms as they are output from the *ah* object to extract the desired data.

In [114]:
```
# Query AnnotationHub
human_ens <- query(ah, c("Homo sapiens", "EnsDb"))
```

```
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
```

The output for *EnsDb* objects is much more recent than what we encountered with AnnotationsDbi (most current release is Ensembl 94), however for Home sapiens the releases only go back as far as Ensembl87. This is fine if you are using GRCh38, however if you were using an older genome build like hg19, you would need to load the *EnsDb* package if available for that release or you might need to build your own with *ensembldb.*

In our case, we are looking for the latest Ensembl release so that the annotations are most up-to-date. To extract this information from AnnotationsHub, we can use the AnnotationsHub ID to subset the object:

In [115]:
```
# Extract annotations of interest
human_ens <- human_ens[["AH64923"]]
```

```
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
downloading 1 resources
retrieving 1 resource
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
loading from cache
    'AH64923 : 71669'
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
Using temporary cache /tmp/RtmptlNrt1/BiocFileCache
```

Now we can use *ensembldb* functions to extract the information at the gene, transcript, or exon levels. We are interested in the gene-level annotatons, so we can extract that information as follows:

In [117]:
```
# Extract gene-level information
genes(human_ens, return.type = "data.frame")
head(genes)
```

A data.frame: 65687 × 12

	gene_id	gene_name	gene_biotype	gene_seq_start	gene_seq_
	<chr>	<chr>	<chr>	<int>	<i
1	ENSG00000223972	DDX11L1	transcribed_unprocessed_pseudogene	11869	14
2	ENSG00000227232	WASH7P	unprocessed_pseudogene	14404	29
3	ENSG00000278267	MIR6859-1	miRNA	17369	17
4	ENSG00000243485	MIR1302-2HG	lincRNA	29554	31

But note that it is just as easy to get the transcript- or exon-level information:

In [118]:

```
# Extract transcript-level information
transcripts(human_ens, return.type = "data.frame")
head(transcripts)
```

A data.frame: 228432 × 10

tx_id	tx_biotype	tx_seq_start	tx_seq_end	tx_cds_seq_start	tx_cds_seq_end
<chr>	<chr>	<int>	<int>	<int>	<int>
ENST00000387314	Mt_tRNA	577	647	NA	NA
ENST00000389680	Mt_rRNA	648	1601	NA	NA
ENST00000387342	Mt_tRNA	1602	1670	NA	NA
ENST00000387347	Mt_rRNA	1671	3229	NA	NA
ENST00000612848	protein_coding	2585	11802	2676	11615
ENST00000386347	Mt_tRNA	3230	3304	NA	NA
ENST00000361390	protein_coding	3307	4262	3307	4262
ENST00000387365	Mt_tRNA	4263	4331	NA	NA
ENST00000387372	Mt_tRNA	4329	4400	NA	NA
ENST00000387377	Mt_tRNA	4402	4469	NA	NA
ENST00000361453	protein_coding	4470	5511	4470	5511
ENST00000614336	protein_coding	4612	24650	6127	21127
LRG_93t1	LRG_gene	4981	20466	5174	20069
LRG_183t1	LRG_gene	5001	16567	5097	11556
LRG_186t1	LRG_gene	5001	46136	18905	45680
LRG_186t3	LRG_gene	5001	42019	18905	41936
LRG_187t1	LRG_gene	5001	11555	6538	10996
LRG_311t1	LRG_gene	5001	110337	6032	107034
LRG_721t1	LRG_gene	5001	31396	8102	30404
LRG_741t1	LRG_gene	5001	229167	5112	228860
LRG_763t1	LRG_gene	5001	174286	5146	170385
ENST00000387382	Mt_tRNA	5512	5579	NA	NA
ENST00000387392	Mt_tRNA	5587	5655	NA	NA
ENST00000387400	Mt_tRNA	5657	5729	NA	NA
ENST00000387405	Mt_tRNA	5761	5826	NA	NA
ENST00000387409	Mt_tRNA	5826	5891	NA	NA
ENST00000361624	protein_coding	5904	7445	5904	7445
ENST00000612640	protein_coding	6101	29626	6127	29453
ENST00000612721	protein_coding	6101	29626	9514	23107
ENST00000616361	protein_coding	6102	29626	7097	29453
⋮	⋮	⋮	⋮	⋮	⋮
ENST00000482023	nonsense_mediated_decay	248850045	248855458	248850904	248855458
ENST00000412341	nonsense_mediated_decay	248850045	248859068	248857330	248858309
ENST00000451251	protein_coding	248850066	248858860	248850210	248858514
ENST00000470787	retained_intron	248850282	248859065	NA	NA
ENST00000462037	retained_intron	248850377	248854523	NA	NA
ENST00000474351	retained_intron	248850384	248856086	NA	NA
ENST00000478107	retained_intron	248855384	248858119	NA	NA

tx_id	tx_biotype	tx_seq_start	tx_seq_end	tx_cds_seq_start	tx_cds_seq_end
<chr>	<chr>	<int>	<int>	<int>	<int>
ENST00000491971	retained_intron	248855595	248857637	NA	NA
ENST00000496411	nonsense_mediated_decay	248855728	248859018	248857954	248858309
ENST00000468455	processed_transcript	248855782	248857772	NA	NA
ENST00000483223	nonsense_mediated_decay	248856515	248858529	248857954	248858309
ENST00000533647	nonsense_mediated_decay	248857273	248858324	248857954	248858309
ENST00000496231	protein_coding	248857313	248859067	248857313	248858309
ENST00000495731	processed_transcript	248857365	248859083	NA	NA
ENST00000528141	nonsense_mediated_decay	248857391	248859085	248858004	248858309
ENST00000497847	retained_intron	248857440	248859066	NA	NA
ENST00000530986	nonsense_mediated_decay	248857469	248859085	248858004	248858309
ENST00000534660	retained_intron	248857557	248859072	NA	NA
ENST00000483791	retained_intron	248857569	248859116	NA	NA
ENST00000530699	retained_intron	248857610	248858695	NA	NA
ENST00000533614	processed_transcript	248857652	248859085	NA	NA
ENST00000496053	retained_intron	248857669	248859071	NA	NA
ENST00000534456	retained_intron	248857865	248859144	NA	NA
ENST00000533976	retained_intron	248857975	248859033	NA	NA
ENST00000417047	antisense	248859164	248864796	NA	NA
ENST00000355360	protein_coding	248906196	248919946	248917338	248918363
ENST00000329291	protein_coding	248906243	248918573	248913863	248918363
ENST00000462488	processed_transcript	248906372	248917401	NA	NA
ENST00000363625	snRNA	248912690	248912795	NA	NA
ENST00000430973	processed_pseudogene	248936581	248937043	NA	NA

```
1 new("standardGeneric", .Data = function (x, ...)
2 standardGeneric("transcripts"), generic = structure("transcripts", package
= "GenomicFeatures"),
3     package = "GenomicFeatures", group = list(), valueClass = character(0),
4     signature = "x", default = NULL, skeleton = (function (x,
5         ...)
6     stop("invalid call in method dispatch to 'transcripts' (no default meth
od)",
```

In [119]:

```
# Extract exon-level information
exons(human_ens, return.type = "data.frame")
head(exons)
```

A data.frame: 783995 × 3

exon_id	exon_seq_start	exon_seq_end
<chr>	<int>	<int>
ENSE00001544501	577	647
ENSE00001544499	648	1601
ENSE00001544498	1602	1670
ENSE00001544497	1671	3229
ENSE00003754502	2585	2692
ENSE00002006242	3230	3304
ENSE00001435714	3307	4262
ENSE00001993597	4263	4331
ENSE00001544494	4329	4400
ENSE00001544493	4402	4469
ENSE00001435686	4470	5511
ENSE00003731717	4612	6370
LRG_93t1e1	4981	5476
LRG_183t1e1	5001	5324
LRG_186t1e1	5001	5080
LRG_187t1e1	5001	5058
LRG_311t1e1	5001	6110
LRG_721t1e1	5001	5261
LRG_741t1e1	5001	5438
LRG_763t1e1	5001	5414
LRG_187t1e2	5172	5675
ENSE00001544492	5512	5579
ENSE00001544491	5587	5655
ENSE00001544490	5657	5729
LRG_183t1e2	5743	6595
ENSE00001544489	5761	5826
ENSE00001544488	5826	5891
ENSE00001435647	5904	7445
ENSE00003717024	6094	6216
ENSE00003721193	6101	6370
⋮	⋮	⋮
ENSE00003615063	248858131	248858321
ENSE00001638872	248858512	248858860
ENSE00002165900	248858512	248858695
ENSE00001274045	248858918	248859072
ENSE00001812574	248858918	248859068
ENSE00001825273	248858918	248859071
ENSE00001826181	248858918	248859067

exon_id	exon_seq_start	exon_seq_end
<chr>	<int>	<int>
ENSE00001927639	248858918	248859018
ENSE00001940006	248858918	248859083
ENSE00001958349	248858918	248859066
ENSE00002166615	248858918	248859116
ENSE00002168387	248858918	248859085
ENSE00002188413	248858918	248859033
ENSE00002200033	248858918	248859065
ENSE00001433276	248859015	248859085
ENSE00002142255	248859015	248859144
ENSE00002150127	248859015	248859072
ENSE00001697027	248859164	248859246
ENSE00001763498	248862629	248862787
ENSE00001779720	248864577	248864796
ENSE00001441802	248906196	248906342
ENSE00001637224	248906243	248906342
ENSE00001845503	248906372	248906466
ENSE00001438388	248912690	248912795
ENSE00003603972	248913816	248913879
ENSE00003610993	248913816	248913879
ENSE00001312561	248916602	248918573
ENSE00001400348	248917279	248919946
ENSE00001956622	248917279	248917401
ENSE00001666859	248936581	248937043

```
1 new("standardGeneric", .Data = function (x, ...)
2 standardGeneric("exons"), generic = structure("exons", package = "GenomicFe
atures"),
3     package = "GenomicFeatures", group = list(), valueClass = character(0),
4     signature = "x", default = NULL, skeleton = (function (x,
5         ...)
6     stop("invalid call in method dispatch to 'exons' (no default method)",
```

Using AnnotationHub to create our tx2gene file

To create our *tx2gene* file, we would need to use a combination of the methods above and merge two dataframes together. For example:

In [120]:
```r
# Create a transcript dataframe
 txdb <- transcripts(human_ens, return.type = "data.frame") %>%
   dplyr::select(tx_id, gene_id)
 txdb <- txdb[grep("ENST", txdb$tx_id),]

 # Create a gene-level dataframe
 genedb <- genes(human_ens, return.type = "data.frame")  %>%
   dplyr::select(gene_id, symbol)

 # Merge the two dataframes together
 annotations <- inner_join(txdb, genedb)
```

```
Joining, by = "gene_id"
```

In [121]:
```r
head(annotations)
```

A data.frame: 6 × 3

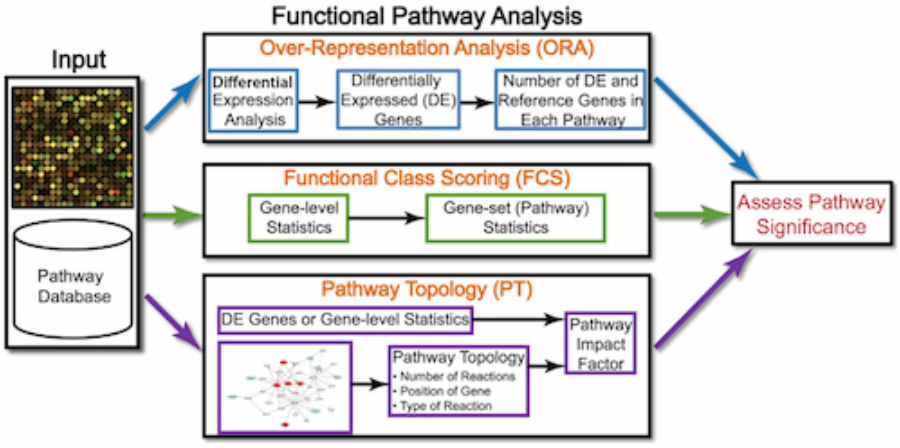
tx_id	gene_id	symbol
<chr>	<chr>	<chr>
ENST00000387314	ENSG00000210049	MT-TF
ENST00000389680	ENSG00000211459	MT-RNR1
ENST00000387342	ENSG00000210077	MT-TV
ENST00000387347	ENSG00000210082	MT-RNR2
ENST00000612848	ENSG00000276345	AC004556.1
ENST00000386347	ENSG00000209082	MT-TL1

Functional analysis

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

- determine whether there is enrichment of known biological functions, interactions, or pathways
- identify genes involvement in novel pathways or networks by grouping genes together based on similar trends
- use global changes in gene expression by visualizing all genes being significantly up- or down regulated in the context of external interacton data

Generally for any differential expression analysis, it is useful to interpret the resulting gene list 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.



The goal of functional analysis is provide biological insight, so it's necessary to analyze our results in the context of our experimental hypothesis: Parental and Resistent cells should have a different translation of a subset of RNAs. Therefore, based on the hypothesis, we may expect the enrichment of processes/pathways related to **translation, splicing, and the regulation of mRNAs**, which would need to validate experimentally.

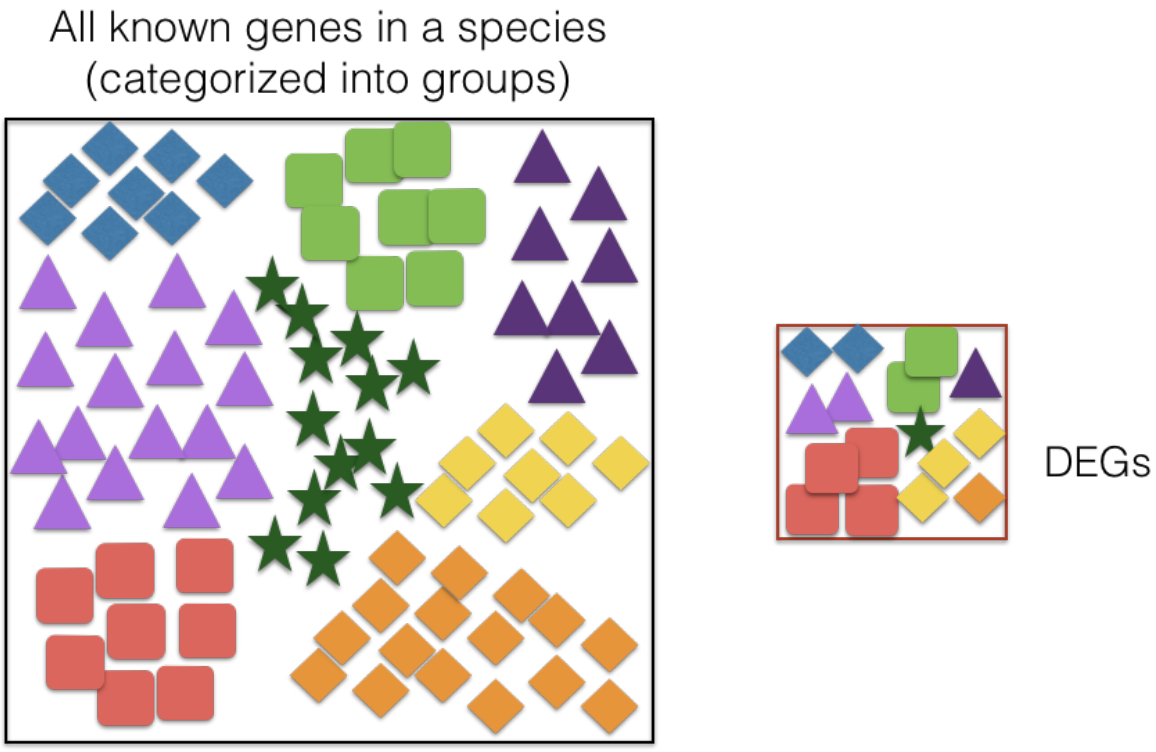
Note that all tools decribed below are great tools to validate experimental results and to make hypotheses. These tools suggest genes/pathways that may be involved with your condition of interest; however, you should NOT use these tools to make conclusions about the pathways involved in your experimental process. You will need to perform experimental validation of any suggested pathways.

Over-representation analysis

There are a plethora of functional enrichment tools that perform some type of "over-representation" analysis by querying databases contatining information about gene function and interactions.

These databases typically **categorize genes into groups (gene sets)** based on shared function, or involvement in a pathway, or presence in a specific cellular location, or other categorizations, e.g. functional pathways, etc. Essentially, known genes are binned into categories that have been consistently named (controlled vocabulary) based on how the gene has been annotated functionally. These categories are independent of any organism, however each organism has distinct categorizations available.

To determine whether any categories are over-represented, you can determine the probability of having the observed proportion of genes associated with a specific category in your gene list based on the proportion of genes associated with the same category in the background set (gene categorizations for the appropriate organism).



Genes categories	Organism-specific Background	DE results	Over-represented?
Functional category 1	35/13000	25/1000	Likely
Functional category 2	56/13000	4/1000	Unlikely
Functional category 3	90/13000	8/1000	Unlikely
Functional category 4	15/13000	10/1000	Likely
…			
…			

The statistical test will determine whether something is actually over-represented is the *Hypergeometric test*.

Hypergeometric testing

Using the example of the first functional category above, hypergeometric distribution is a probability distribution that describes the probability that describes the probability of 25 genes (k) being associated with "Functional category 1", for all genes in our gene list (n=1000), from a population of all the genes in entire genome (N=13000) which contains 35 gene (K) associated with "Functional category 1".

The calculation of probability of k successes follows the formula:

$$P(X = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$

This test will result in an asjusted p-value (after multiple test correction) for each category tested.

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".

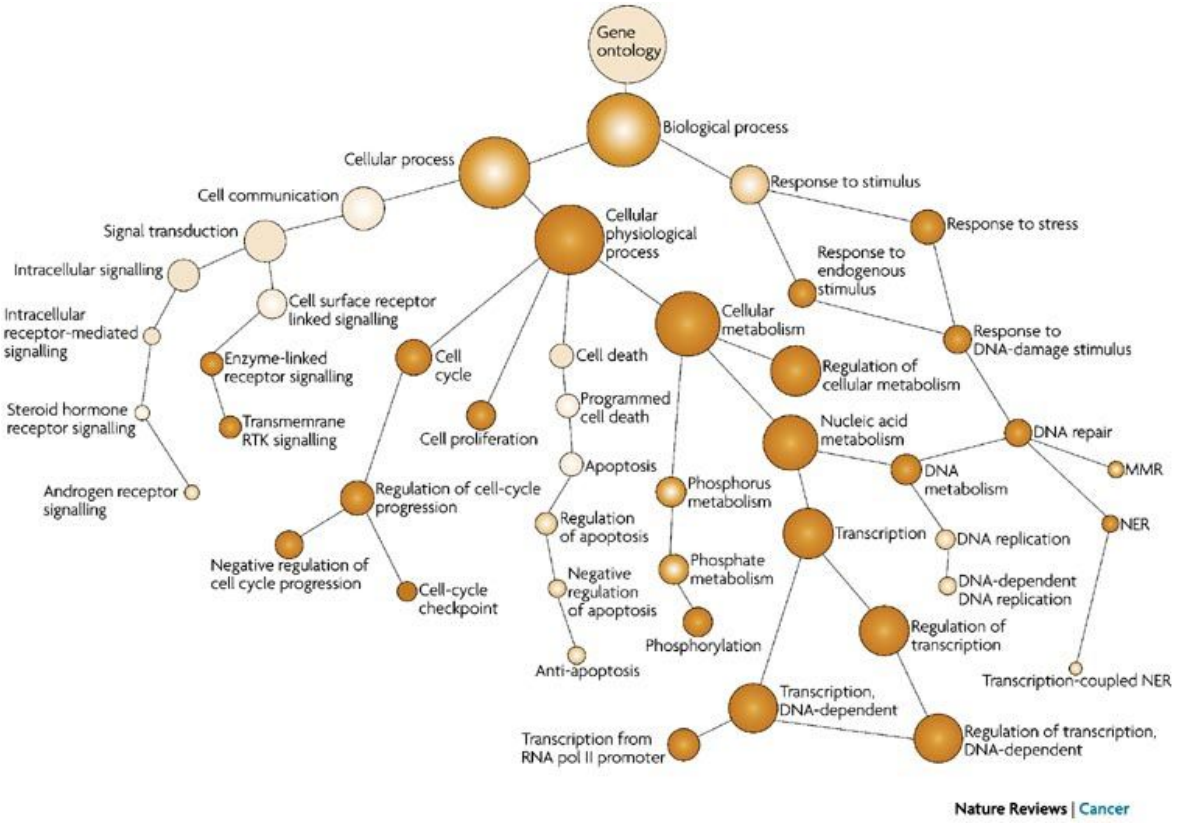
Each GO term has a term name (e.g. **DNA repair**) and a unique term accession number (**GO:0005125**), and a single gene product can be associated with many GO terms, since a single gene product "may function in several processes, contain domains that cary out diverse molecular functions, and participate 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.



clusterProfiler

We will be using clusterProfiler (http://bioconductor.org/packages/release/bioc/html/clusterProfiler.html) 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 ontology (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 [122]:

```r
# Load libraries
library(DOSE)
library(pathview)
library(clusterProfiler)
```

```
DOSE v3.10.2  For help: https://guangchuangyu.github.io/DOSE (https://guangch
uangyu.github.io/DOSE)

If you use DOSE in published research, please cite:
Guangchuang Yu, Li-Gen Wang, Guang-Rong Yan, Qing-Yu He. DOSE: an R/Bioconduc
tor package for Disease Ontology Semantic and Enrichment analysis. Bioinforma
tics 2015, 31(4):608-609

#####################################################################################
#
Pathview is an open source software package distributed under GNU General
Public License version 3 (GPLv3). Details of GPLv3 is available at
http://www.gnu.org/licenses/gpl-3.0.html. (http://www.gnu.org/licenses/gpl-3.
0.html.) Particullary, users are required to
formally cite the original Pathview paper (not just mention it) in publicatio
ns
or products. For details, do citation("pathview") within R.

The pathview downloads and uses KEGG data. Non-academic uses may require a KE
GG
license agreement (details at http://www.kegg.jp/kegg/legal.html). (http://ww
w.kegg.jp/kegg/legal.html).)
#####################################################################################
#
Registered S3 method overwritten by 'enrichplot':
  method               from
  fortify.enrichResult DOSE
clusterProfiler v3.12.0  For help: https://guangchuangyu.github.io/software/c
lusterProfiler (https://guangchuangyu.github.io/software/clusterProfiler)

If you use clusterProfiler in published research, please cite:
Guangchuang Yu, Li-Gen Wang, Yanyan Han, Qing-Yu He. clusterProfiler: an R pa
ckage for comparing biological themes among gene clusters. OMICS: A Journal o
f Integrative Biology. 2012, 16(5):284-287.

Attaching package: 'clusterProfiler'

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

    simplify

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

    simplify
```

For the different steps in the functional analysis, we require Ensembl and Entrez IDs. We will use the gene annotations that we generated previously to merge with our differential expression results.

In [123]:

```r
## Merge the annotations with the results
res_ids <- inner_join(res_tableOE_tb, annotations_edb, by=c("gene"="GENEID
```

To perform the over-representation analysis, we need a list of background genes and a list of significant genes. For our background 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 [124]:
```r
## Create background dataset for hypergeometric testing using all genes te
allOE_genes <- as.character(res_ids$gene)

## Extract significant results
sigOE <- dplyr::filter(res_ids, padj < 0.05)

sigOE_genes <- as.character(sigOE$gene)
```

Now we can perform the GO enrichment analysis and save the results:

In [125]:
```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)

write.csv(cluster_summary, "clusterProfiler.csv")
```

Visualizing clusterProfiler results

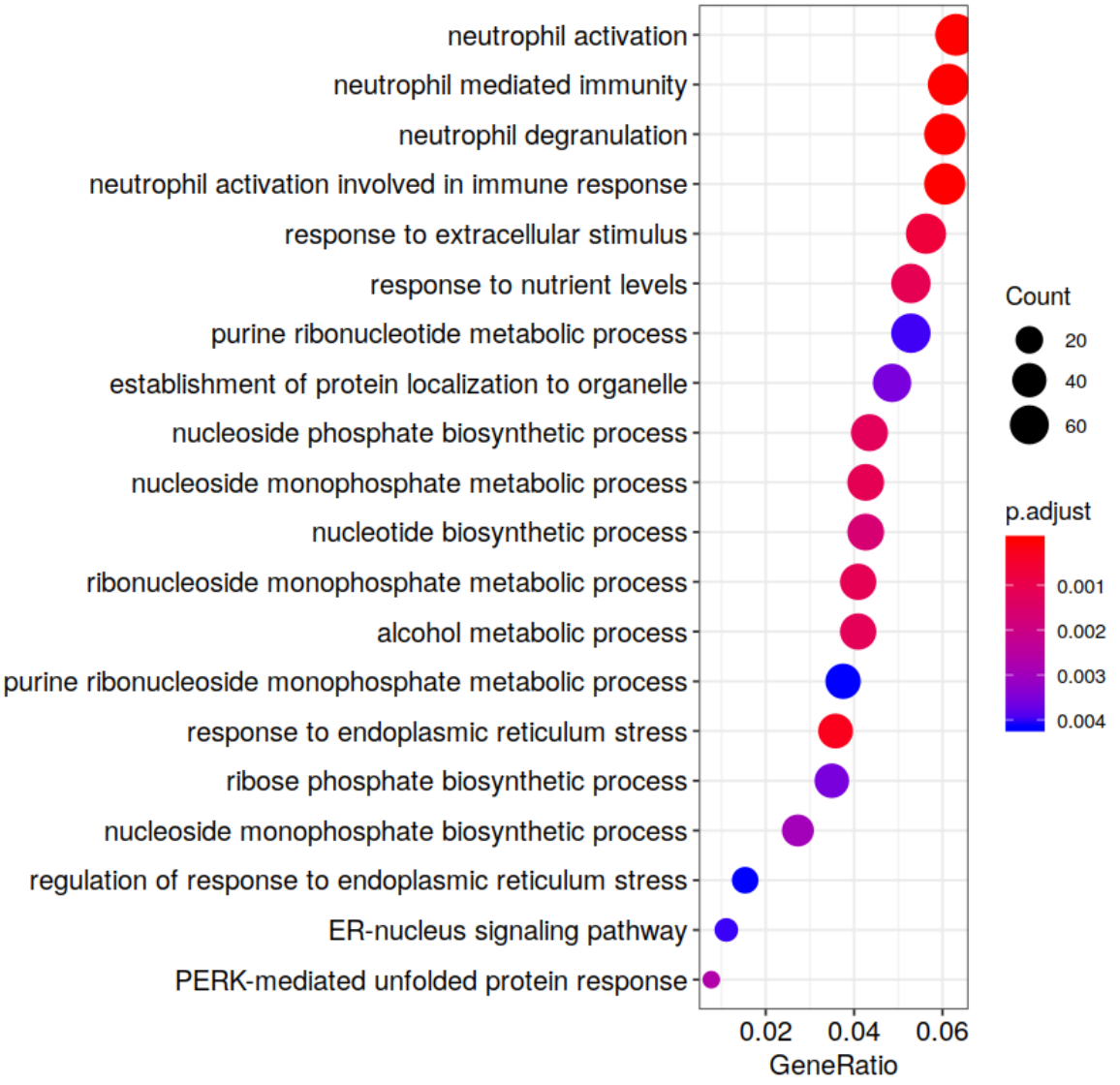
clusterProfiler has a variety of options for viewing the over-represented GO terms. We will explore the dotplot, enrichment plot, and the category netplot.

The **dotplot** shows the number of genes associated with the first 50 terms (size) and the p-adjusted values for these terms (color). This plot displays the top 50 genes by gene ratio (# genes related to GO term/total number of sig genes), not p-adjusted value.

In [132]:

```
## Dotplot
dotplot(ego, showCategory=20)
```

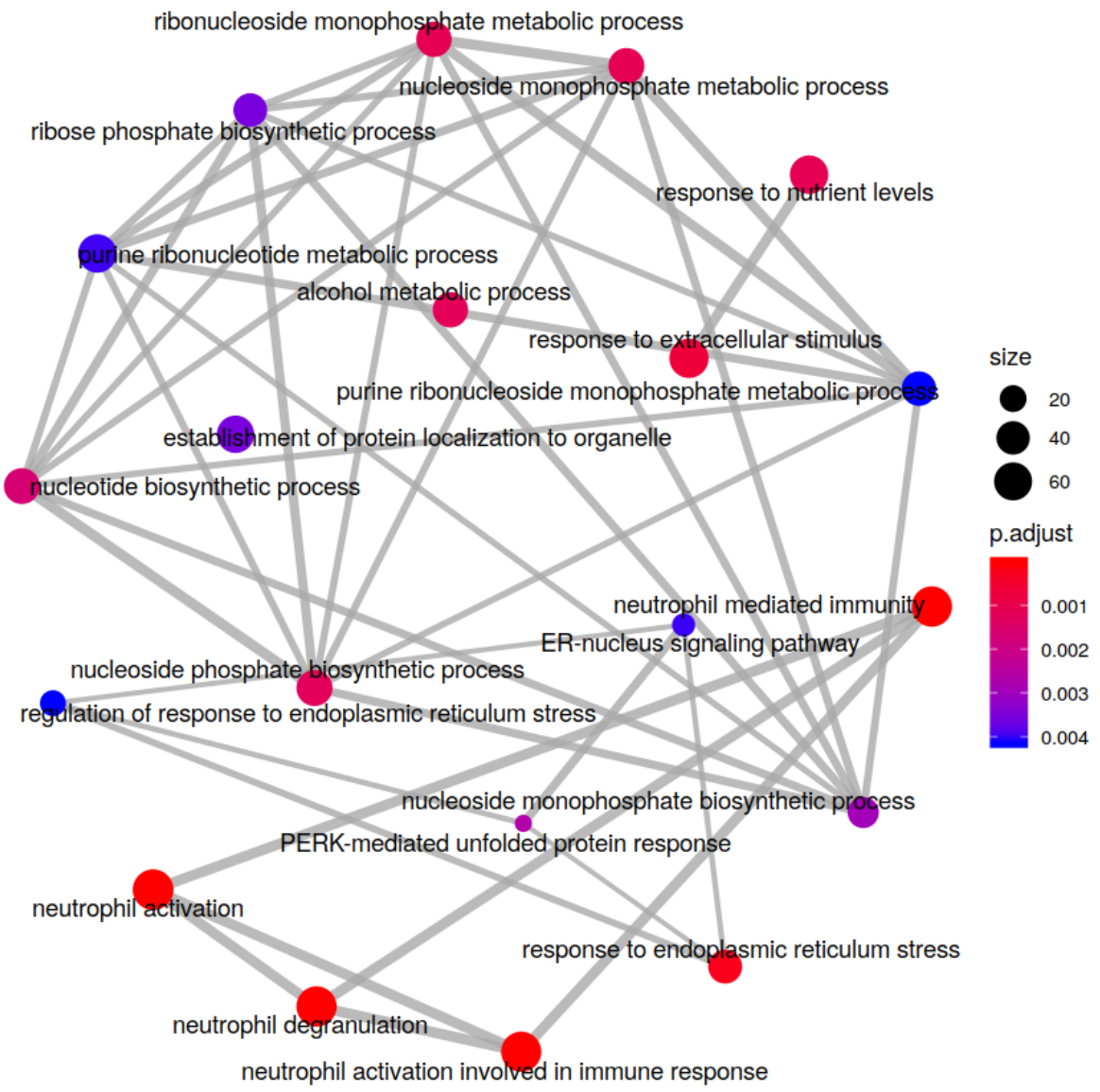
```
wrong orderBy parameter; set to default `orderBy = "x"`
```



The next plot is the **enrichment GO plot**, which shows the relationship between the top 50 most significantly enriched GO terms (padj.), by grouping similar term together. The color represents the p-values relative to the other displayed terms (brighter red is more significant) and the size of the terms represents the number of genes that are significant from our list.

In [129]:

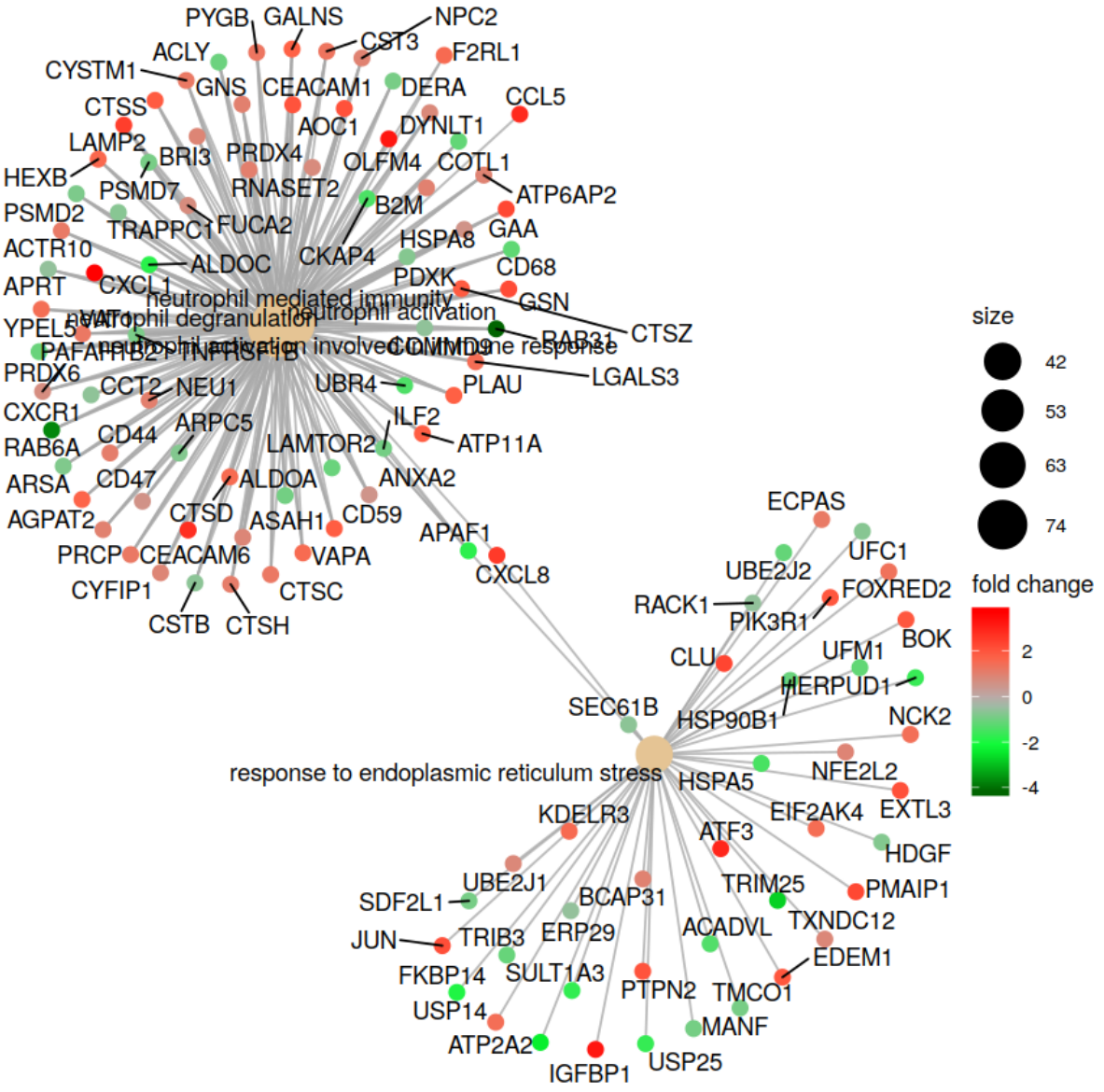
```
## Enrichmap clusters the 50 most significant (by padj) GO terms to visual
emapplot(ego, showCategory = 20)
```

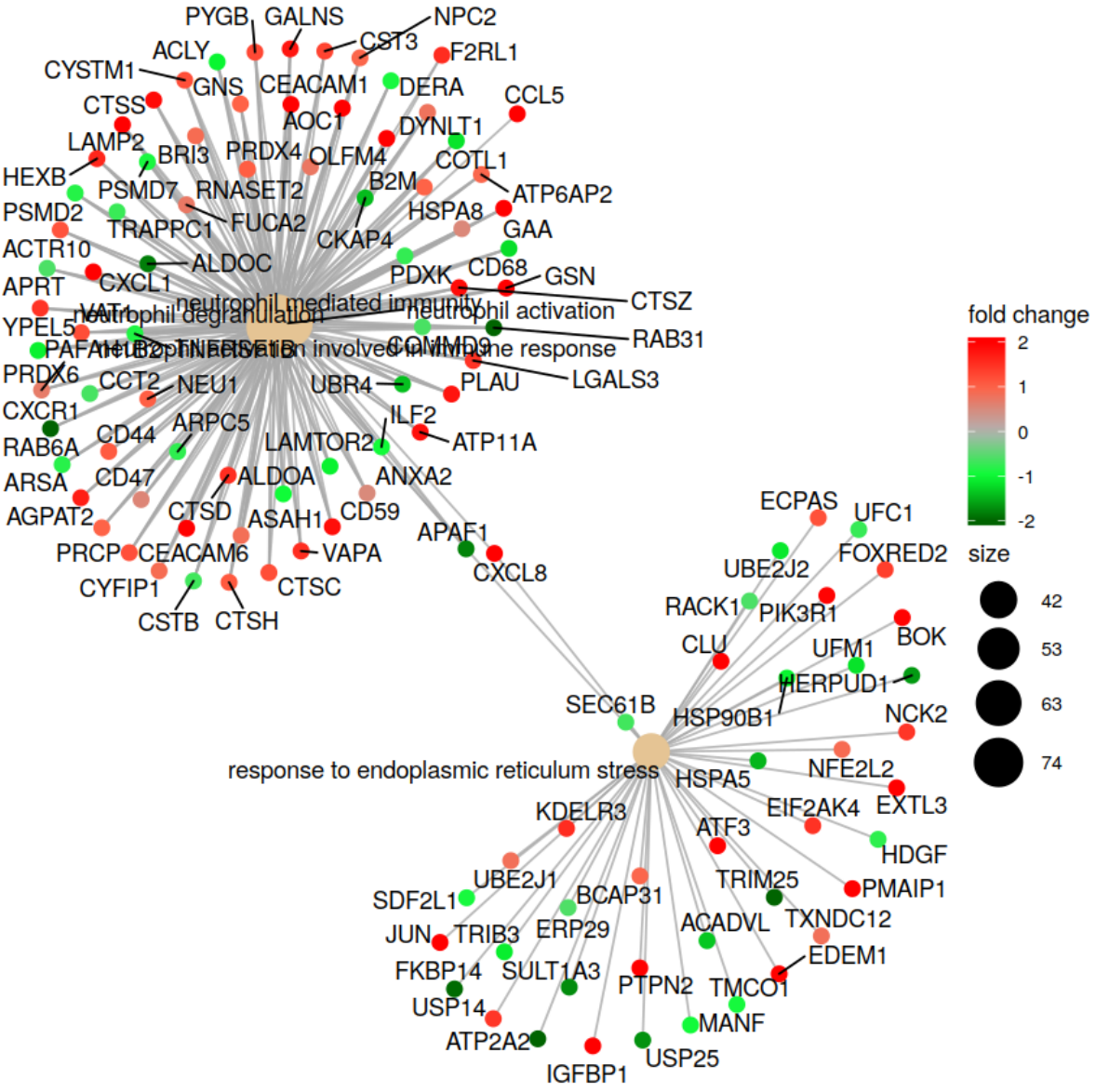


Finally, the **category netplot** shows the relationships between the genes associated with the top five most significant GO terms and the fold changes of the significant genes associated with these terms (color). The size of the GO terms reflects the p-values of the terms, with the more significant terms being larger. This plot is particular useful for hypothesis generaton in identifying genes that may be important to several of the most affected processes.

In [130]:

```
## To color genes by log2 fold changes, we need to extract the log2 fold c
OE_foldchanges <- sigOE$log2FoldChange

names(OE_foldchanges) <- sigOE$gene

## Cnetplot details the genes associated with one or more terms - by defau
cnetplot(ego,
         categorySize="pvalue",
         showCategory = 5,
         foldChange=OE_foldchanges,
         vertex.label.font=6)

## 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)
```

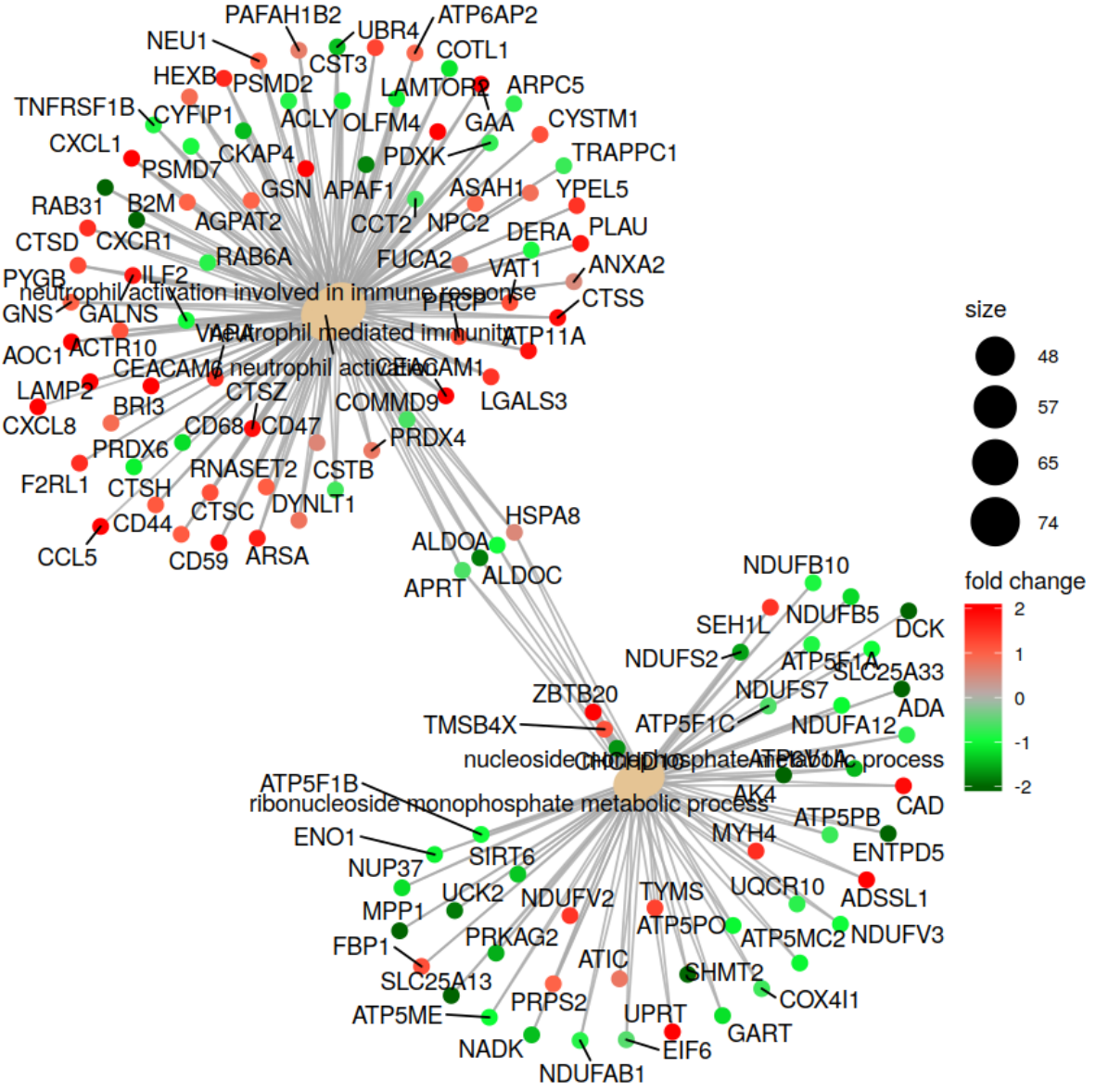




If you are interested in significant processes that are **not** among the top five, you can subset your *ego* dataset to only display these processes:

In [131]:

```
## 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)
```



Functional class scoring tools

Functional class scoring (FCS) tools, such as GSEA (http://software.broadinstitute.org/gsea/index.jsp), most often use the gene-level statistics or log2 fold changes for all genes form the differnetial 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 although large changes in individual genes have significant effects on pathways(and will be detected via ORA methods), 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 statistic and statistical significance of each pathway is reported. This type of analysis can be particularly helpful if the differential expression analysis only outputs a samll list of significant DE genes.

Gene set enrichment analysis using clusterProfiler and Pathview

Using the log2 fold changes obtained from the differential expression analysis for every gene, gene set enrichment analysis and pathway analysis can be performed using clusterProfiler and Pathview tools.

For gene set or pathway analysis using clusterProfiler, coordinated differential expression over gene sets is tested instead of changes of individual genes. "Gene sets are pre-defined groups of genes, which are functionally related. Commonly used gene sets include those derived from KEGG pathways, Gene Ontolgy terms, MSigDB, Reactome, or gene groups that share some other functional annotations, etc. Consistent perturbations over such gene sets frquently suggest mechanistic changes".

To perform GSEA analysis of KEGG gene sets, clusterProfiler requires the genes to be indentified using Entrez IDs for all genes in our results dataset. We also need to remove the NA values and duplicates (due to gene ID conversion) prior to the analysis:

In [133]:
```
# Remove any NA values
res_entrez <- dplyr::filter(res_ids, ENTREZID != "NA")

# Remove any Entrez duplicates
res_entrez <- res_entrez[which(duplicated(res_entrez$ENTREZID) == F), ]

```

Finally, extract and name the fold changes:

In [134]:
```
# Extract the foldchanges
foldchanges <- res_entrez$log2FoldChange

# Name each fold change with the corresponding Entrez ID
names(foldchanges) <- res_entrez$ENTREZID
```

Next we need to order the fold changes in decreasing order. To do this we'll use the *sort()* function, which takes a vector as input. This is in contrast to Tidyverse's *arrange()*, which requires a data frame.

In [136]:
```
## Sort fold changes in decreasing order
foldchanges <- sort(foldchanges, decreasing = TRUE)

head(foldchanges)
```

55600	5.78009007440368
260293	5.63712050013105
2766	5.31794822923993
221458	5.29291655829906
57161	5.22478232741358
134285	5.06734213604332

Perform the GSEA using KEGG gene sets:

In [149]:

```
# GSEA using gene sets from KEGG pathways
gseaKEGG <- gseKEGG(geneList = foldchanges, # ordered named vector of fold
              organism = "hsa", # supported organisms listed below
              nPerm = 1000, # default number permutations
              minGSSize = 20, # minimum gene set size (# genes in set) - c
              pvalueCutoff = 0.08, # padj cutoff value
              verbose = FALSE)

# Extract the GSEA results
gseaKEGG_results <- gseaKEGG@result
```

```
Warning message in fgsea(pathways = geneSets, stats = geneList, nperm = nPer
m, minSize = minGSSize, :
"There are ties in the preranked stats (6.77% of the list).
The order of those tied genes will be arbitrary, which may produce unexpected
results."
```

In [150]:

```
head(gseaKEGG_results)
```

A data.frame: 6 × 11

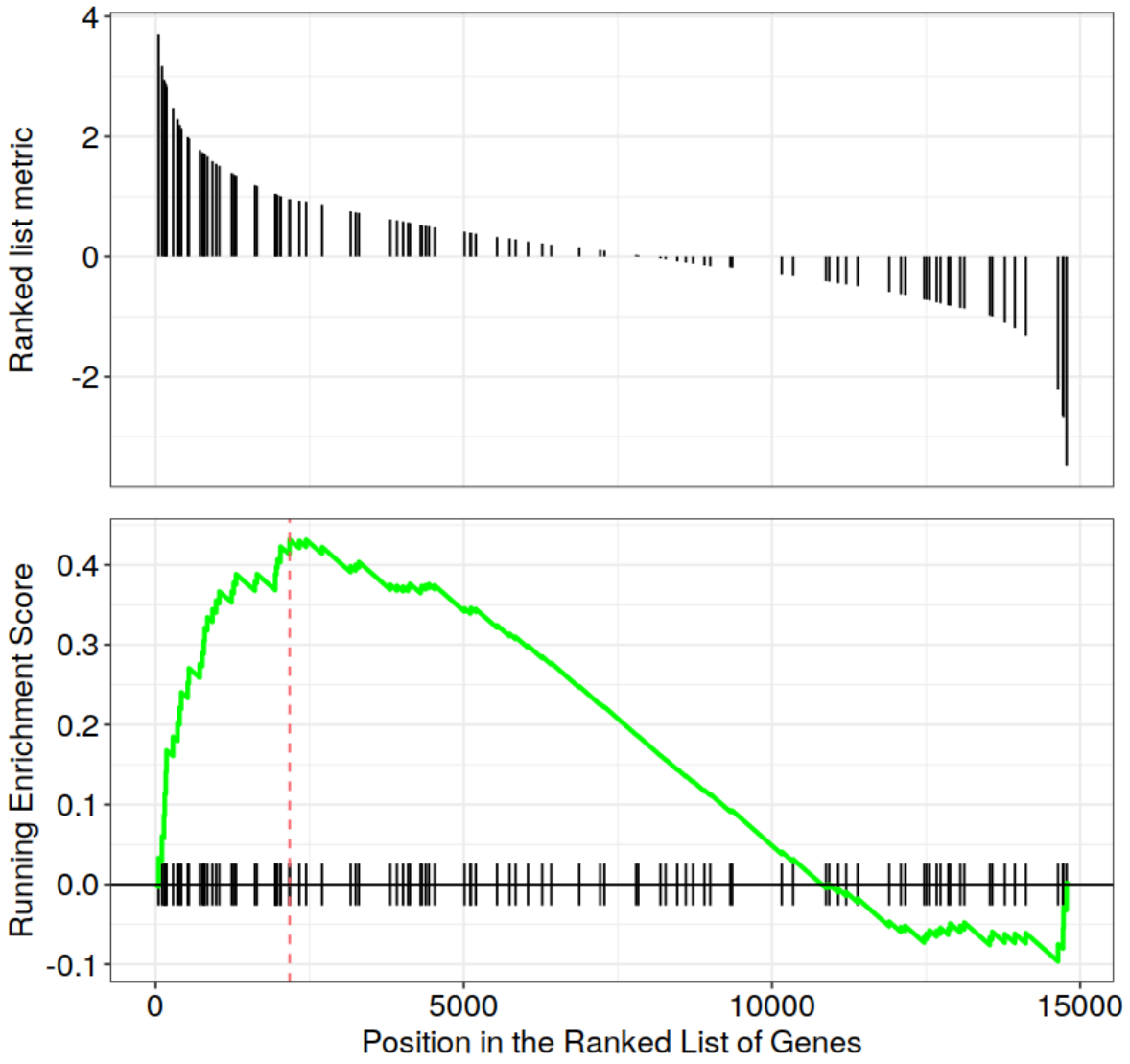
	ID	Description	setSize	enrichmentScore	NES	pvalue	p.adjust	qvalue
	<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
hsa05167	hsa05167	Kaposi sarcoma-associated herpesvirus infection	151	0.3904335	1.570906	0.001200480	0.05542769	0.0454242
hsa04668	hsa04668	TNF signaling pathway	101	0.4318232	1.660873	0.001298701	0.05542769	0.0454242
hsa00562	hsa00562	Inositol phosphate metabolism	70	0.4637802	1.680340	0.001321004	0.05542769	0.0454242
hsa04657	hsa04657	IL-17 signaling pathway	72	0.4650036	1.695436	0.001322751	0.05542769	0.0454242
hsa04350	hsa04350	TGF-beta signaling pathway	76	0.4415554	1.621011	0.001324503	0.05542769	0.0454242
hsa04514	hsa04514	Cell adhesion molecules (CAMs)	82	0.4743704	1.747887	0.001355014	0.05542769	0.0454242

In [151]:

```
# write GSEA results to file
write.csv(gseaKEGG_results, "gseaOE_kegg.csv", quote=F)
```

Explore the GSEA plot of enrichment of one of the pathways in the ranked list:

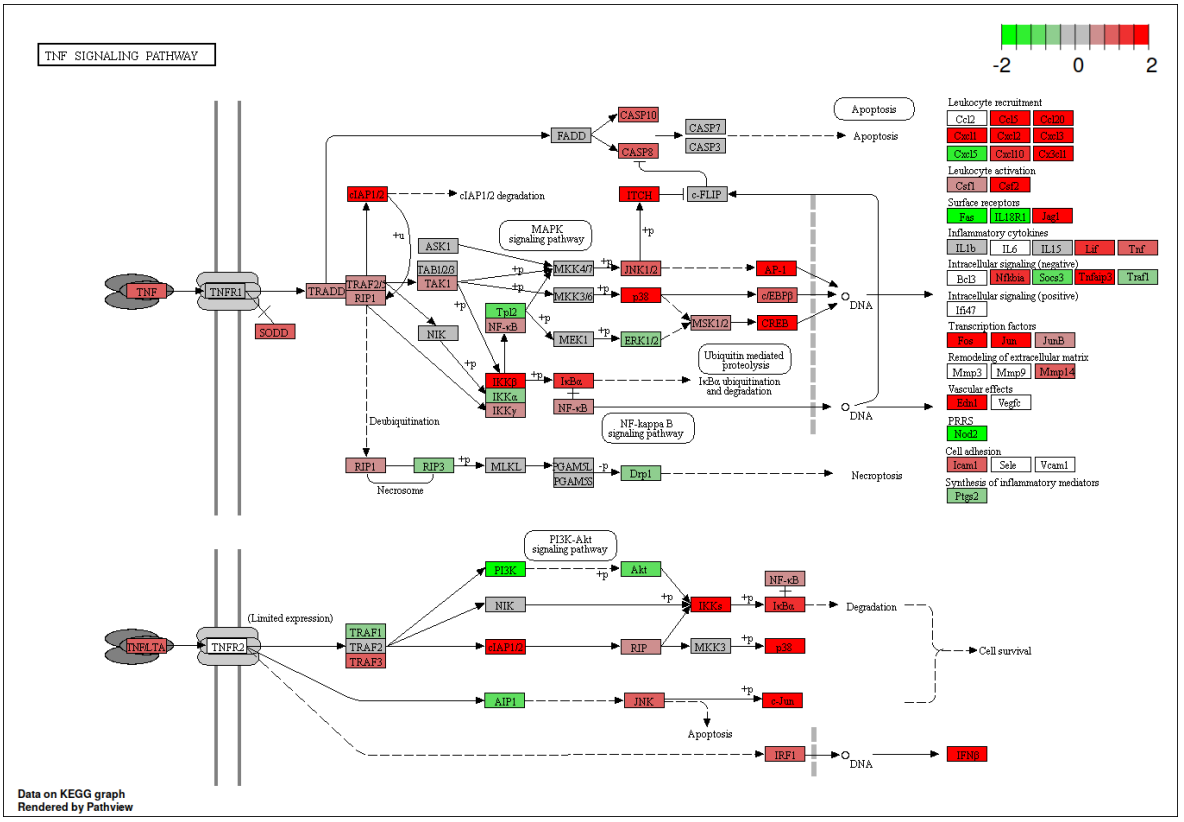
In [152]:
```
## Plot the GSEA plot for a single enriched pathway, `hsa03040`
gseaplot(gseaKEGG, geneSetID = 'hsa04668')
```



Use the Pathview R package (http://bioconductor.org/packages/release/bioc/html/pathview.html) to integrate the KEGG pathway data from clusterProfiler into pathway images:

In [153]:
```
detach("package:dplyr", unload=TRUE) # first unload dplyr to avoid conflic

## Output images for a single significant KEGG pathway
pathview(gene.data = foldchanges,
              pathway.id = "hsa04668",
              species = "hsa",
              limit = list(gene = 2, # value gives the max/min limit for f
              cpd = 1))
```

```
Warning message:
"'dplyr' namespace cannot be unloaded:
  namespace 'dplyr' is imported by 'AnnotationHub', 'DEGreport', 'dbplyr', 'b
room', 'europepmc', 'ggraph', 'BiocFileCache' so cannot be unloaded"Info: Dow
nloading 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 /mnt/data/Salmon/DEanalysis
Info: Writing image file hsa04668.pathview.png
```



Printing out Pathview images for all significant (0.07) pathways:

In [154]:

```
# Output images for all significant KEGG pathways
get_kegg_plots <- function(x) {
   pathview(gene.data = foldchanges, pathway.id = gseaKEGG_results$ID[x],
       limit = list(gene = 2, cpd = 1))
}

purrr::map(1:length(gseaKEGG_results$ID), get_kegg_plots)
```

```
Info: Downloading xml files for hsa05167, 1/1 pathways..
Info: Downloading png files for hsa05167, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /mnt/data/Salmon/DEanalysis
Info: Writing image file hsa05167.pathview.png
Info: some node width is different from others, and hence adjusted!
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /mnt/data/Salmon/DEanalysis
Info: Writing image file hsa04668.pathview.png
Info: Downloading xml files for hsa00562, 1/1 pathways..
Info: Downloading png files for hsa00562, 1/1 pathways..
Warning message in structure(x$children, class = "XMLNodeList"):
"Calling 'structure(NULL, *)' is deprecated, as NULL cannot have attribute
s.
  Consider 'structure(list(), *)' instead."Warning message in structure(x$c
hildren, class = "XMLNodeList"):
"Calling 'structure(NULL, *)' is deprecated, as NULL cannot have attribute
s.
  Consider 'structure(list(), *)' instead."Warning message in structure(x$c
```

Instead of exploring enrichment of KEGG gene sets, we can also explore the enrichment of BP Gene Ontology terms using gene set enrichment analysis:

In [160]:

```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
```

```
Warning message in fgsea(pathways = geneSets, stats = geneList, nperm = nPer
m, minSize = minGSSize, :
"There are ties in the preranked stats (6.77% of the list).
The order of those tied genes will be arbitrary, which may produce unexpected
results."
```

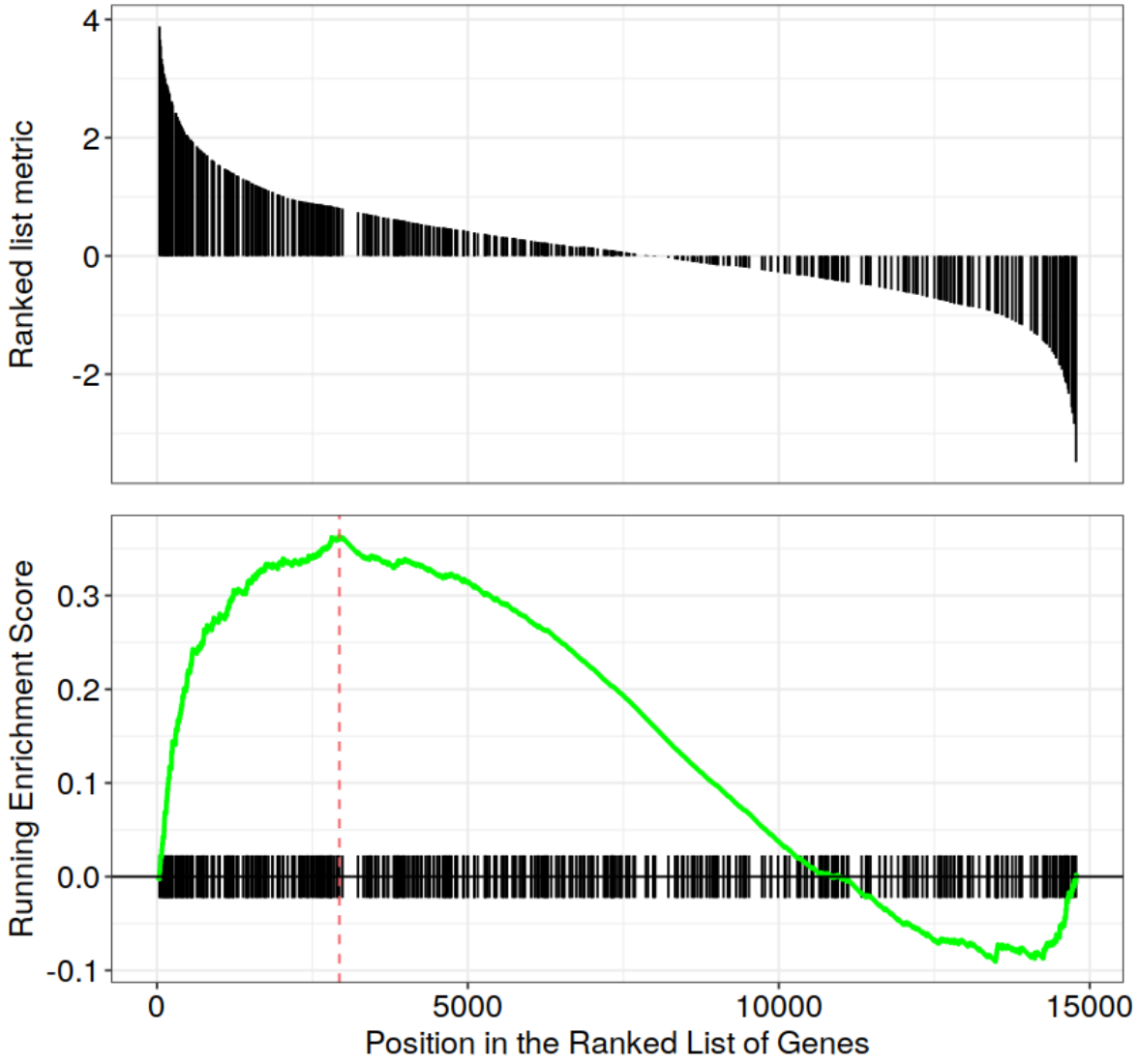
In [163]:

```r
head(gseaGO_results)
```

A data.frame: 6 × 11

	ID	Description	setSize	enrichmentScore	NES	pvalue	p.adjust	
	<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	
GO:0001568	GO:0001568	blood vessel development	498	0.3294808	1.447485	0.001036269	0.06498701	0.
GO:0030155	GO:0030155	regulation of cell adhesion	496	0.3633737	1.596087	0.001036269	0.06498701	0.
GO:0031400	GO:0031400	negative regulation of protein modification process	484	0.3356374	1.471668	0.001037344	0.06498701	0.
GO:0009611	GO:0009611	response to wounding	483	0.3451167	1.511993	0.001038422	0.06498701	0.
GO:0045596	GO:0045596	negative regulation of cell differentiation	491	0.3283326	1.441451	0.001038422	0.06498701	0.
GO:0000904	GO:0000904	cell morphogenesis involved in differentiation	482	0.3278515	1.435267	0.001042753	0.06498701	0.

In [164]:
```
1 gseaplot(gseaGO, geneSetID = 'GO:0030155')
```



Pathway topology tools

The last main type of 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 (http://bioconductor.org/packages/release/bioc/html/SPIA.html) tool can be used to integrate the lists of differentially expressed genes, their fold changes, and pathway topology to identify affected pathways.

In [165]:

```
library(SPIA)

## Significant genes is a vector of fold changes where the names are ENTRE

background_entrez <- res_entrez$ENTREZID

sig_res_entrez <- res_entrez[which(res_entrez$padj < 0.05), ]

sig_entrez <- sig_res_entrez$log2FoldChange

names(sig_entrez) <- sig_res_entrez$ENTREZID

head(sig_entrez)
```

```
Loading required package: KEGGgraph

Attaching package: 'KEGGgraph'

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

    plot
```

2833	-2.66283359916744
50515	-2.44749416906583
134285	5.06734213604332
103910	1.86900426702373
975	2.16175271471528
6746	-1.45521630048128

Now that we have our background and significant genes in the appropriate format, we can run SPIA:

In [166]:

```
spia_result <- spia(de=sig_entrez, all=background_entrez, organism="hsa")

head(spia_result, n=20)
```

```
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
```

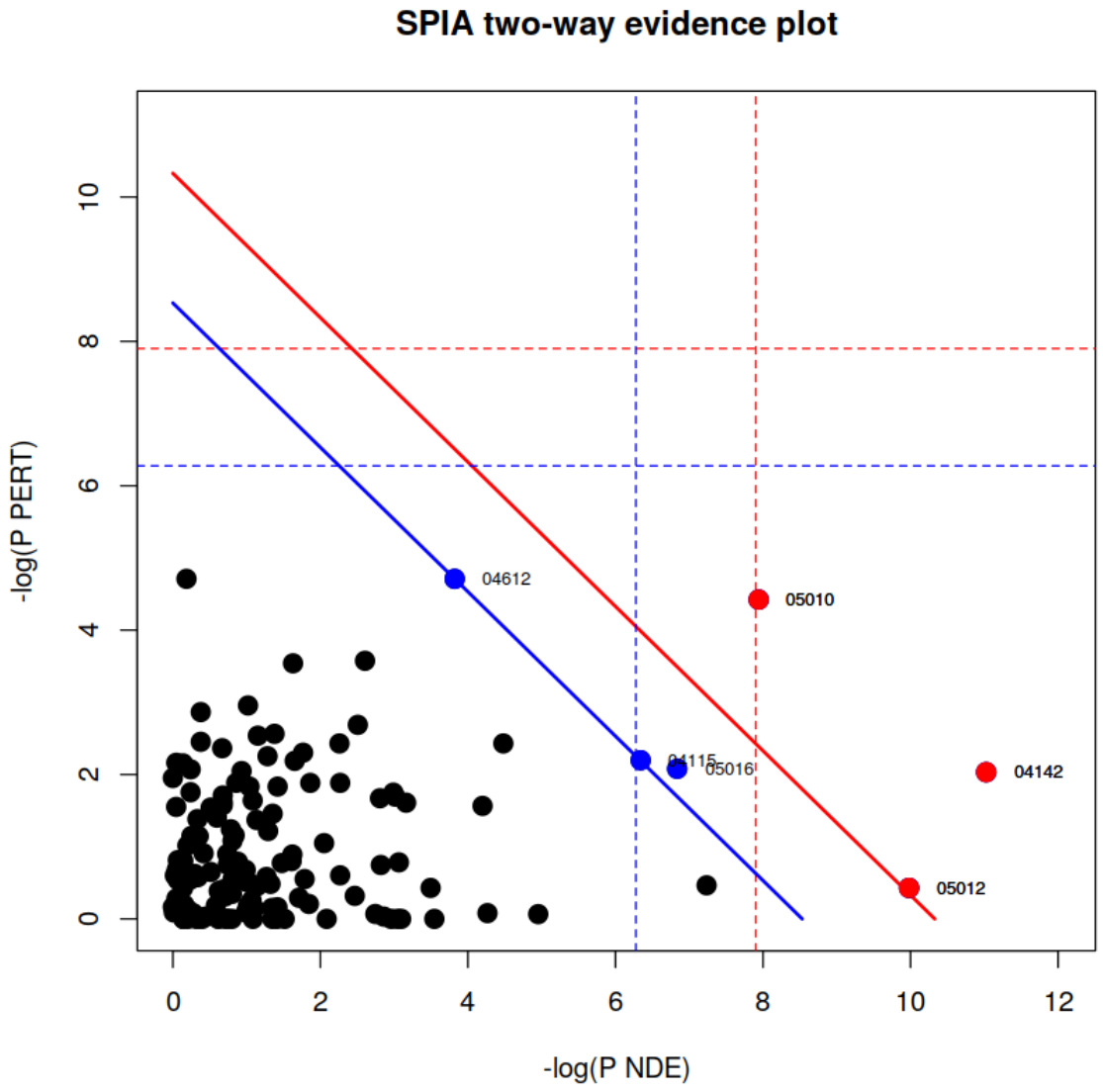
SPIA outputs a table showing significantly dysregulated pathways based on over-representation and signaling perturbations accumulation. The table shows the following information:

- 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 a 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 the KEGG website that displays the pathway image with the differentially expressed genes highlighted in red

We can view the significantly dysregulated pathways by viewing the over-representation and perturbations for each pathway.

In [167]:
```
1 plotP(spia_result, threshold=0.05)
```



In this plot, each pathway is a point and the coordinates are the log of pNDE (using a hypergeometric model) and the p-value from perturbations, pPERT. The oblique lines in the plot show the significance regions based on the combined evidence.

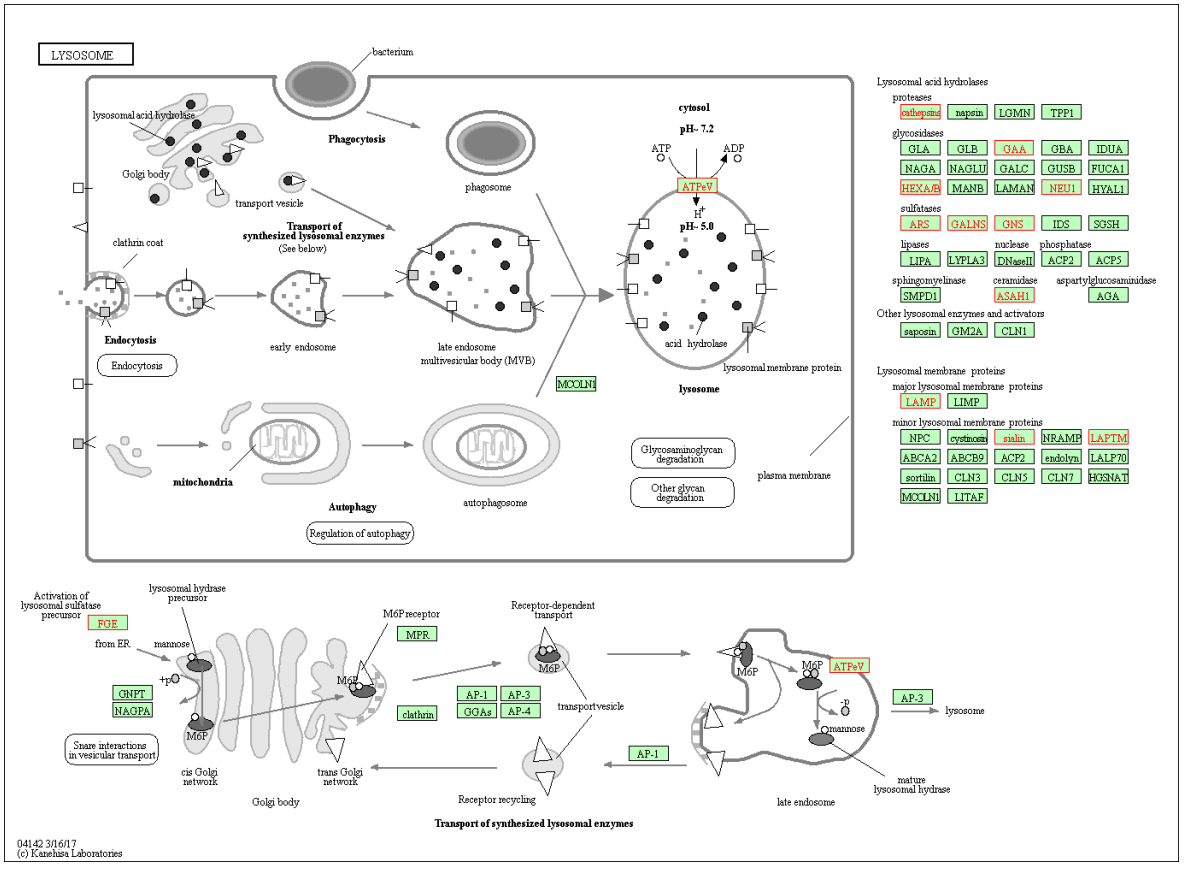
If we choose to explore the significant gene from our dataset occuring in these pathways, we can subset our SPIA results:

In [170]:

```
# Look at pathway 00510 and view kegglink
subset(spia_result, ID == "04142")

```

A data.frame: 1 × 12

Name	ID	pSize	NDE	pNDE	tA	pPERT	pG	pGFdr	pGFWER	Stat
<chr>	<chr>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<ch
Lysosome	04142	119	22	1.627676e-05	1.673526	0.131	2.997595e-05	0.00385034	0.004046753	Activat



Sources

- https://combine-lab.github.io/salmon/ (https://combine-lab.github.io/salmon/)
- https://salmon.readthedocs.io/en/latest/ (https://salmon.readthedocs.io/en/latest/)
- https://galaxyproject.org/tutorials/rb_rnaseq/ (https://galaxyproject.org/tutorials/rb_rnaseq/)
- https://hbctraining.github.io/DGE_workshop_salmon/schedule/ (https://hbctraining.github.io/DGE_workshop_salmon/schedule/)
- https://www.biorxiv.org/content/biorxiv/early/2015/06/27/021592.full.pdf (https://www.biorxiv.org/content/biorxiv/early/2015/06/27/021592.full.pdf)
- https://europepmc.org/abstract/ppr/ppr81065 (https://europepmc.org/abstract/ppr/ppr81065)
- http://bioconductor.org/packages/release/bioc/html/clusterProfiler.html (http://bioconductor.org/packages/release/bioc/html/clusterProfiler.html)
- http://software.broadinstitute.org/gsea/index.jsp (http://software.broadinstitute.org/gsea/index.jsp)

In [171]:

```
# 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] SPIA_2.36.0                 KEGGgraph_1.44.0
 [3] clusterProfiler_3.12.0      pathview_1.24.0
 [5] DOSE_3.10.2                 AnnotationHub_2.16.0
 [7] BiocFileCache_1.8.0         dbplyr_1.4.2
 [9] EnsDb.Hsapiens.v86_2.99.0   ensembldb_2.8.0
[11] AnnotationFilter_1.8.0      GenomicFeatures_1.36.4
[13] org.Hs.eg.db_3.8.2          AnnotationDbi_1.46.0
[15] vsn_3.52.0                  ggrepel_0.8.1
[17] tximport_1.12.3             DEGreport_1.20.0
[19] pheatmap_1.0.12             RColorBrewer_1.1-2
[21] forcats_0.4.0               stringr_1.4.0
[23] purrr_0.3.2                 readr_1.3.1
[25] tidyr_0.8.3                 tibble_2.1.3
[27] ggplot2_3.2.0               tidyverse_1.2.1
[29] DESeq2_1.24.0               SummarizedExperiment_1.14.0
[31] DelayedArray_0.10.0         BiocParallel_1.18.0
[33] matrixStats_0.54.0          Biobase_2.44.0
[35] GenomicRanges_1.36.0        GenomeInfoDb_1.20.0
[37] IRanges_2.18.1              S4Vectors_0.22.0
[39] BiocGenerics_0.30.0

loaded via a namespace (and not attached):
  [1] tidyselect_0.2.5              RSQLite_2.1.1
  [3] htmlwidgets_1.3               grid_3.6.1
  [5] munsell_0.5.0                 preprocessCore_1.46.0
  [7] pbdZMQ_0.3-3                  withr_2.1.2
  [9] colorspace_1.4-1              GOSemSim_2.10.0
 [11] knitr_1.23                    uuid_0.1-2
 [13] rstudioapi_0.10               labeling_0.3
 [15] lasso2_1.2-20                 urltools_1.7.3
 [17] repr_1.0.1                    GenomeInfoDbData_1.2.1
 [19] polyclip_1.10-0               mnormt_1.5-5
 [21] farver_1.1.0                  bit64_0.9-7
 [23] vctrs_0.2.0                   generics_0.0.2
 [25] xfun_0.8                      R6_2.4.0
 [27] clue_0.3-57                   locfit_1.5-9.1
 [29] gridGraphics_0.4-1            bitops_1.0-6
 [31] reshape_0.8.8                 fgsea_1.10.0
 [33] assertthat_0.2.1              promises_1.0.1
 [35] scales_1.0.0                  ggraph_1.0.2
 [37] enrichplot_1.4.0              nnet_7.3-12
 [39] gtable_0.3.0                  affy_1.62.0
 [41] rlang_0.4.0                   zeallot_0.1.0
```

```
 [43] genefilter_1.66.0             GlobalOptions_0.1.0
 [45] splines_3.6.1                 rtracklayer_1.44.0
 [47] lazyeval_0.2.2                acepack_1.4.1
 [49] hexbin_1.27.3                 europepmc_0.3
 [51] broom_0.5.2                   checkmate_1.9.4
 [53] BiocManager_1.30.4            yaml_2.2.0
 [55] reshape2_1.4.3                modelr_0.1.5
 [57] backports_1.1.4               httpuv_1.5.1
 [59] qvalue_2.16.0                 Hmisc_4.2-0
 [61] tools_3.6.1                   psych_1.8.12
 [63] ggplotify_0.0.3               logging_0.10-108
 [65] affyio_1.54.0                 ggdendro_0.1-20
 [67] ggridges_0.5.1                Rcpp_1.0.1
 [69] plyr_1.8.4                    base64enc_0.1-3
 [71] progress_1.2.2                zlibbioc_1.30.0
 [73] RCurl_1.95-4.12               prettyunits_1.0.2
 [75] rpart_4.1-15                  viridis_0.5.1
 [77] GetoptLong_0.1.7              cowplot_1.0.0
 [79] haven_2.1.1                   cluster_2.1.0
 [81] magrittr_1.5                  data.table_1.12.2
 [83] DO.db_2.9                     circlize_0.4.6
 [85] triebeard_0.3.0               ProtGenerics_1.16.0
 [87] hms_0.5.0                     mime_0.7
 [89] evaluate_0.14                 xtable_1.8-4
 [91] XML_3.98-1.20                 readxl_1.3.1
 [93] gridExtra_2.3                 shape_1.4.4
 [95] compiler_3.6.1                biomaRt_2.40.4
 [97] crayon_1.3.4                  htmltools_0.3.6
 [99] later_0.8.0                   Formula_1.2-3
[101] geneplotter_1.62.0            lubridate_1.7.4
[103] DBI_1.0.0                     tweenr_1.0.1
[105] ComplexHeatmap_2.0.0          MASS_7.3-51.4
[107] rappdirs_0.3.1                Matrix_1.2-17
[109] cli_1.1.0                     igraph_1.2.4.1
[111] pkgconfig_2.0.2               rvcheck_0.1.3
[113] GenomicAlignments_1.20.1      foreign_0.8-72
[115] IRdisplay_0.7.0               xml2_1.2.0
[117] annotate_1.62.0               XVector_0.24.0
[119] rvest_0.3.4                   digest_0.6.20
[121] ConsensusClusterPlus_1.48.0   graph_1.62.0
[123] Biostrings_2.52.0             cellranger_1.1.0
[125] fastmatch_1.1-0               htmlTable_1.13.1
[127] edgeR_3.26.5                  curl_3.3
[129] shiny_1.3.2                   Rsamtools_2.0.0
[131] rjson_0.2.20                  nlme_3.1-140
[133] jsonlite_1.6                  viridisLite_0.3.0
[135] limma_3.40.2                  pillar_1.4.2
[137] lattice_0.20-38               Nozzle.R1_1.1-1
[139] KEGGREST_1.24.0               httr_1.4.0
[141] survival_2.44-1.1             GO.db_3.8.2
[143] interactiveDisplayBase_1.22.0 glue_1.3.1
[145] UpSetR_1.4.0                  png_0.1-7
[147] Rgraphviz_2.28.0              bit_1.1-14
[149] ggforce_0.2.2                 stringi_1.4.3
[151] blob_1.2.0                    latticeExtra_0.6-28
[153] memoise_1.1.0                 IRkernel_1.0.2
[155] dplyr_0.8.3
```

This document was saved on:

In [173]:
```
1 Sys.Date()
```

```
2019-09-06
```