Introduction to RNA-seq for Differential Expression Analysis – 3

by Jiajia Li

Biological Data Science Institute

12 May 2025

Australian National University

Learning Objectives of Today

- ERCC Expression Analysis
  - compare the concentration with our expression
  - fit linear model
  - generate plots

- Differential Expression with Ballgown
  - Interpret Ballgown result
  - Filter out low abundance genes/transcripts
  - Run DE on filtered expression
  - Get significant expressed genes/transcripts with p-value

ERCC Expression Analysis

Based on the raw counts result we get from the last session; we will plot the linearity of the ERCC spike-in **read counts observed** in our RNA-seq data versus the **expected concentration** of the ERCC spike-in Mix.

First, let's download the file describing the expected concentrations and fold-change differences for the ERCC spike-in reagent.

```
mkdir ~/RNAseq-Workshop/expression/ercc_spikein_analysis
cd ~/RNAseq-Workshop/expression/ercc_spikein_analysis
wget http://genomedata.org/rnaseq-tutorial/ERCC_Controls_Analysis.txt
less -S ERCC_Controls_Analysis.txt
```

ERCC Expression Analysis

Re-sort ID	ERCC ID	subgroup	concentration in Mix 1 (attomoles/ul)	concentration in Mix 2 (attomoles
1	ERCC-00130	A	30000	7500
2	ERCC-00004	A	7500	1875
3	ERCC-00136	A	1875	468.75
4	ERCC-00108	A	937.5	234.375
5	ERCC-00116	A	468.75	117.1875
6	ERCC-00092	A	234.375	58.59375
7	ERCC-00095	A	117.1875	29.296875
8	ERCC-00131	A	117.1875	29.296875
9	ERCC-00062	A	58.59375	14.6484375

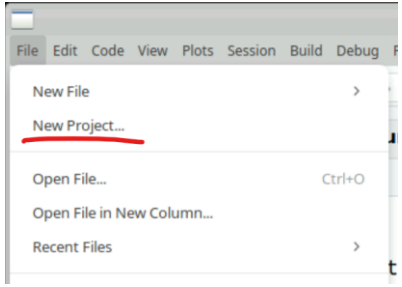
The columns are :

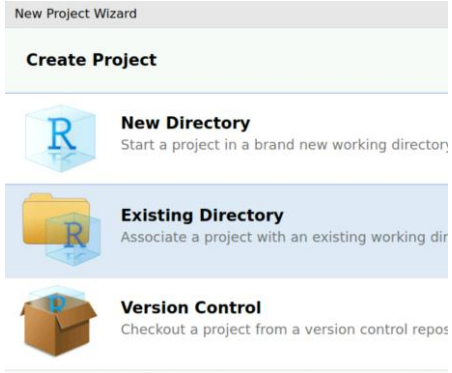
- ID
- ERCC ID
- Subgroup
- Concentration in Mix1 (attomoles/ul)
- Concentration in Mix2 (attomoles/ul)
- Expected fold-change ratio (Mix1/Mix2)
- Log2(Mix1/Mix2)

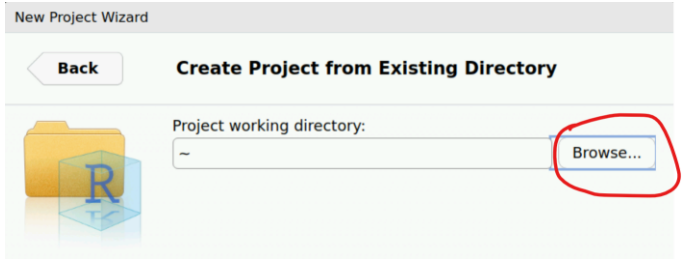
Plot Linearity in Rstudio

We will import both the **htseq-count result of our sample** and the **expected ERCC concentration** to R and generate some plots to compare.

- Open Rstudio.
- Click "File", then "New Project".
- Choose "Existing Directory".
- Click "Browse", then select "RNAseq-Workshop", then create new project.

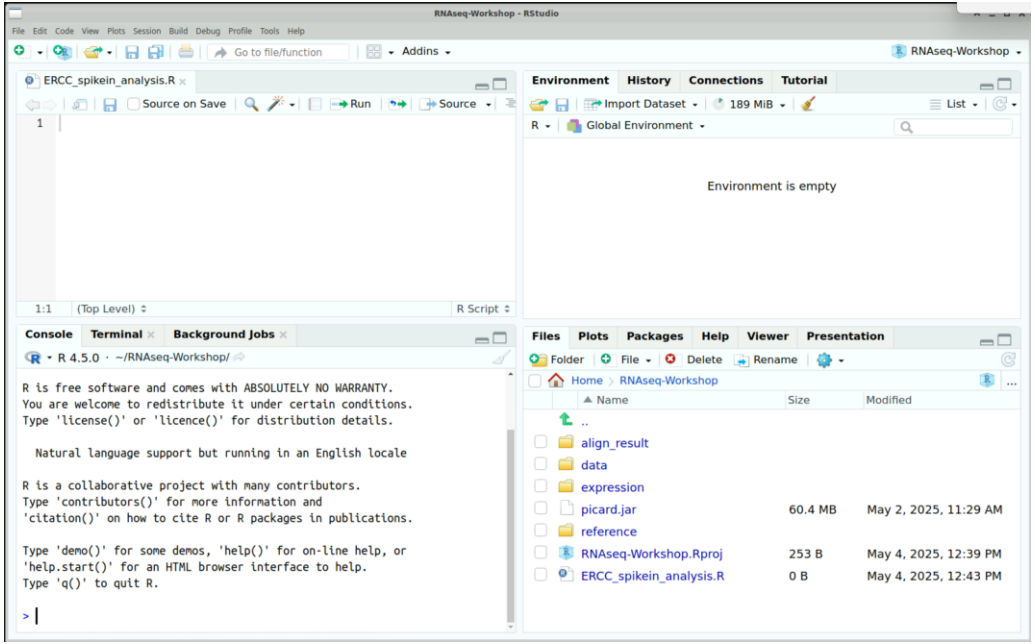






Plot Linearity in Rstudio

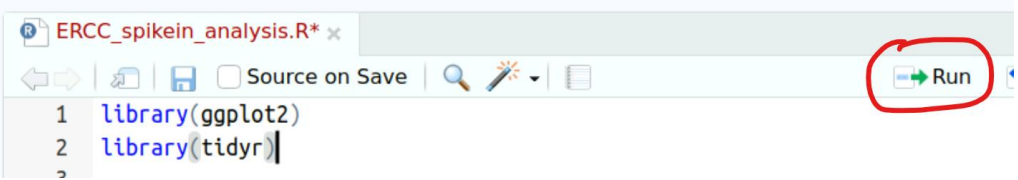
- Click "File", then create a new R script
- Name it "ERCC_spikein_analysis.R"
- Save it under your current directory which is "RNAseq-Workshop"

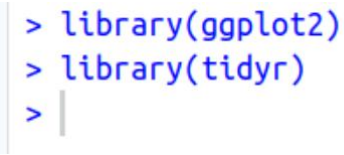


Plot Linearity in Rstudio

From now on, we will write our code in the top-left section, in our R script. Then we can save it for use next time.

First, let's import the packages we are going to use this time.





You can put your cursor in line 1, then click "Run", it will run the first line.
Same for line 2.

You can also select all 2 lines, then click "Run", it will run 2 lines together.

Then, you can see the commands ran in your Console, like shown in the second figure.

Plot Linearity in Rstudio

Import the "ERCC_Control_Analysis.txt" file into R.

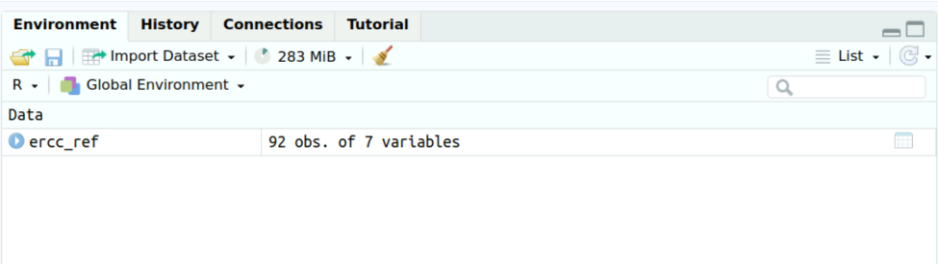
```
# load the expected ERCC concentration and fold-change
ercc_ref <- read.delim("expression/ercc_spikein_analysis/ERCC_Controls_Analysis.txt",
                       header = TRUE, sep = "\t")
```

You can use # to write a comment line, whatever after # won't be run by R.

The `read.delim` function can read a file in table format and creates a data frame from it.

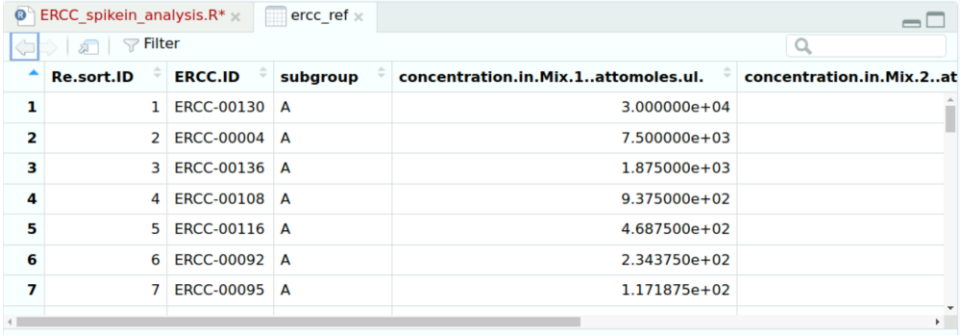
You can run `?read.delim` to see the detailed explanation of this function. You can use ? to get the help of any function.

Plot Linearity in Rstudio



After you run the previous command, you can see a new variable show up in the top-right section.

Here, you can click on this data frame to view it.



	Re.sort.ID	ERCC.ID	subgroup	concentration.in.Mix.1..attomoles.ul.	concentration.in.Mix.2..at
1	1	ERCC-00130	A	3.000000e+04	
2	2	ERCC-00004	A	7.500000e+03	
3	3	ERCC-00136	A	1.875000e+03	
4	4	ERCC-00108	A	9.375000e+02	
5	5	ERCC-00116	A	4.687500e+02	
6	6	ERCC-00092	A	2.343750e+02	
7	7	ERCC-00095	A	1.171875e+02	

Plot Linearity in Rstudio

Now, we are going to change the column names of our data frame because it was too long.

To get your column names, you can run `names(ercc_ref)`

```
> names(ercc_ref)
[1] "Re.sort.ID"                          "ERCC.ID"
[3] "subgroup"                            "concentration.in.Mix.1..attomoles.ul."
[5] "concentration.in.Mix.2..attomoles.ul." "expected.fold.change.ratio"
[7] "log2.Mix.1.Mix.2."
>
```

These dots were automatically added by the program because it used to have spaces in the column names which the R didn't like.

It is also good practice to not include space in your column names, variable names, file names, or any other names you use.

Plot Linearity in Rstudio

Rename the column names:

```
# rename the column names
names(ercc_ref) <- c("id", "ercc_id", "subgroup", "ref_conc_mix_1",
                     "ref_conc_mix_2", "ref_fc_mix1_vs_mix2", "ref_log2_mix1_vs_mix2")
```

The `c()` in R is how you create a list of items, items are separated by a comma.

If the text are wrapped by double quotes, it tells the computer it is a "**string**", which means plain text.

You can also create a list of **integers** or **floating-point numbers**, and all types of other data.

Plot Linearity in Rstudio

Check the dimension of your data frame:

```
# check the dimension of data frame
dim(ercc_ref)

```

```
> dim(ercc_ref)
[1] 92  7
>
```

There are 92 rows and 7 columns in this dataset. Which the 92 rows should be the 92 RNA transcripts in ERCC spike-in control.

Exercise:

- **Import the read counts of our sample, and name it "rna_counts".**
- **Check the dimension of it.**

```
# load the htseq-count result of our sample
rna_counts <- read.delim()
# check dimension of `rna_counts`

```

Plot Linearity in Rstudio

Combine the `ercc_ref` table with our `rna_counts` table.

```
# combine ERCC concentration data with our RNA-seq count data
ercc_ref_counts <- merge(x = ercc_ref, y = rna_counts,
                         by.x = "ercc_id", by.y = "GeneID", all.x = TRUE)
```

The `merge()` function can merge two data frames by common columns or row names.

- `x`: specify first data frame to join
- `y`: second data frame to join
- `by.x`: which column to match
- `by.y`: …
- `all.x = TRUE`: do a left join, keep all rows in `ercc_ref`, even if there is no matching row in `rna_counts`.

Click to view the merged table.

Plot Linearity in Rstudio

Extract only the **Mix1 concentrations** and **UHR counts** data to a new table.

Because Mix1 was added to UHR samples and Mix2 was added to HBR samples.

```
# extract the mix1 concentrations and UHR counts to a new table
uhr_data <- ercc_ref_counts[,c("ercc_id","subgroup","ref_conc_mix_1",
                               "UHR_Rep1","UHR_Rep2","UHR_Rep3")]
```

The `dataframe[,]` in R is a way to select rows and columns. The above command selects all the rows, and columns ercc_id, subgroup, ref_conc_mix1, … etc.

For example, `ercc_ref_counts[1,1]` select the value of first row and first column.

```
> ercc_ref_counts[1,1]
[1] "ERCC-00002"
>
```

Plot Linearity in Rstudio

The `uhr_data` table looks like this:

ERCC_spikein_analysis.R | uhr_data

Filter

	ercc_id	subgroup	ref_conc_mix_1	UHR_Rep1	UHR_Rep2	UHR_Rep3
1	ERCC-00002	D	1.500000e+04	19987	12985	16992
2	ERCC-00003	D	9.375000e+02	1742	1098	1338
3	ERCC-00004	A	7.500000e+03	4590	3332	3690
4	ERCC-00009	B	9.375000e+02	692	477	554
5	ERCC-00012	C	1.144409e-01	1	0	0

Now we need to convert the `uhr_data` to **long format**, to prepare it for later creating plots.

```r
# convert `uhr_data` to long format
uhr_data_long <- pivot_longer(
  data = uhr_data,
  cols = starts_with("UHR_Rep"),
  names_to = "sample",
  values_to = "count"
)
```

- `data =` specify data frame to convert
- `cols =` specify columns to convert to long format
- `names_to` specify the name of new column where it stores the previous column name information
- `values_to` specify new name for the column where it stores values

Plot Linearity in Rstudio

The converted long format table looks like:

	ercc_id	subgroup	ref_conc_mix_1	sample	count
1	ERCC-00002	D	1.500000e+04	UHR_Rep1	19987
2	ERCC-00002	D	1.500000e+04	UHR_Rep2	12985
3	ERCC-00002	D	1.500000e+04	UHR_Rep3	16992
4	ERCC-00003	D	9.375000e+02	UHR_Rep1	1742
5	ERCC-00003	D	9.375000e+02	UHR_Rep2	1098
6	ERCC-00003	D	9.375000e+02	UHR_Rep3	1338
7	ERCC-00004	A	7.500000e+03	UHR_Rep1	4590
8	ERCC-00004	A	7.500000e+03	UHR_Rep2	3332
9	ERCC-00004	A	7.500000e+03	UHR_Rep3	3690

Add a new column "mix" which describe mix information, and fill all the values with 1:

```
# add a new column "mix" and fill with 1
uhr_data_long$mix <- 1
```

Plot Linearity in Rstudio

Then clean up the column names once again:

```
# clean up the column names
names(uhr_data_long) <- c("ercc_id", "subgroup", "concentration", "sample",
                          "count", "mix")
```

Your new table will look like:

	ercc_id	subgroup	concentration	sample	count	mix
1	ERCC-00002	D	1.500000e+04	UHR_Rep1	19987	1
2	ERCC-00002	D	1.500000e+04	UHR_Rep2	12985	1
3	ERCC-00002	D	1.500000e+04	UHR_Rep3	16992	1
4	ERCC-00003	D	9.375000e+02	UHR_Rep1	1742	1
5	ERCC-00003	D	9.375000e+02	UHR_Rep2	1098	1
6	ERCC-00003	D	9.375000e+02	UHR_Rep3	1338	1

Plot Linearity in Rstudio

Exercise: do the same for HBR.

- **Extract columns for HBR.**
- **Convert to long format.**
- **Add another column "mix".**
- **Clean up the column names.**

Plot Linearity in Rstudio

Rejoin two tables.

```
# rejoin UHR and HBR data
ercc_ref_counts_long <- rbind(uhr_data_long, hbr_data_long)

```

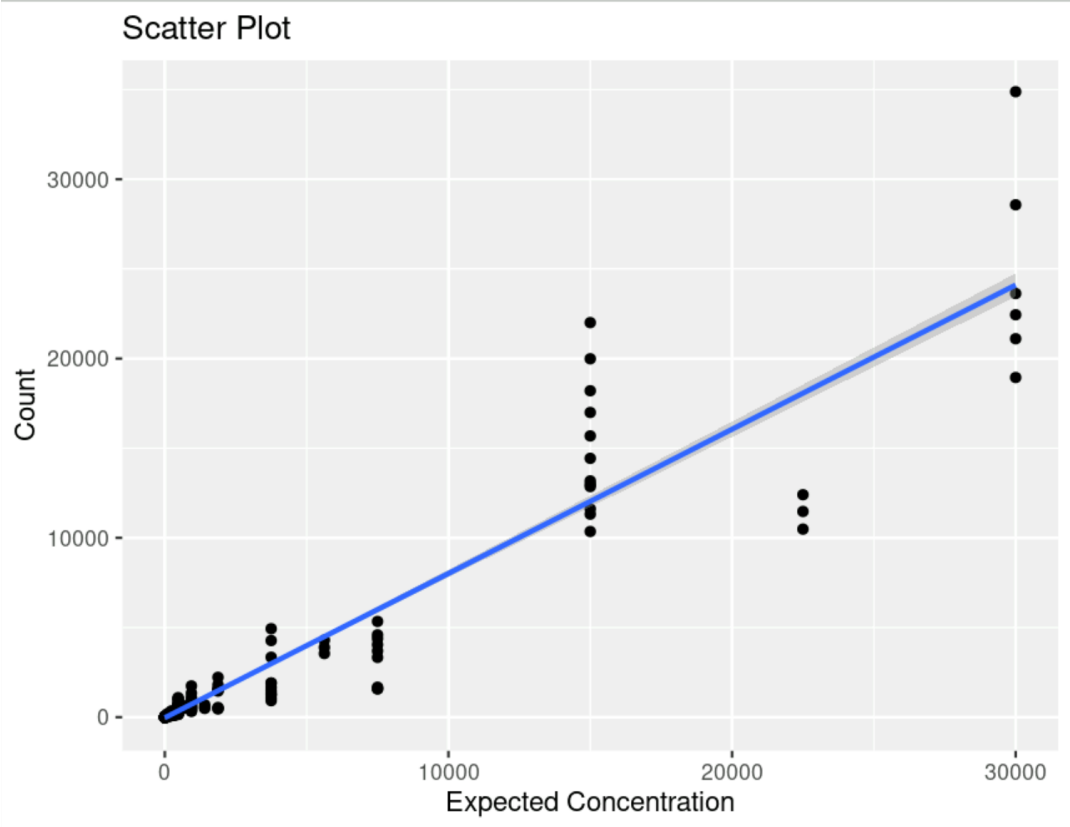
Create a scatter plot with "concentration" on x axis and "count" on y axis to see the trend:

```
# create a scatter plot on "count" by "concentration"
ggplot(ercc_ref_counts_long, aes(x = concentration, y = count)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Scatter Plot", x = "Expected Concentration", y = "Count")
```

`geom_smooth` fits a line on the scatter plot using method "lm".

Plot Linearity in Rstudio

The plot looks like:

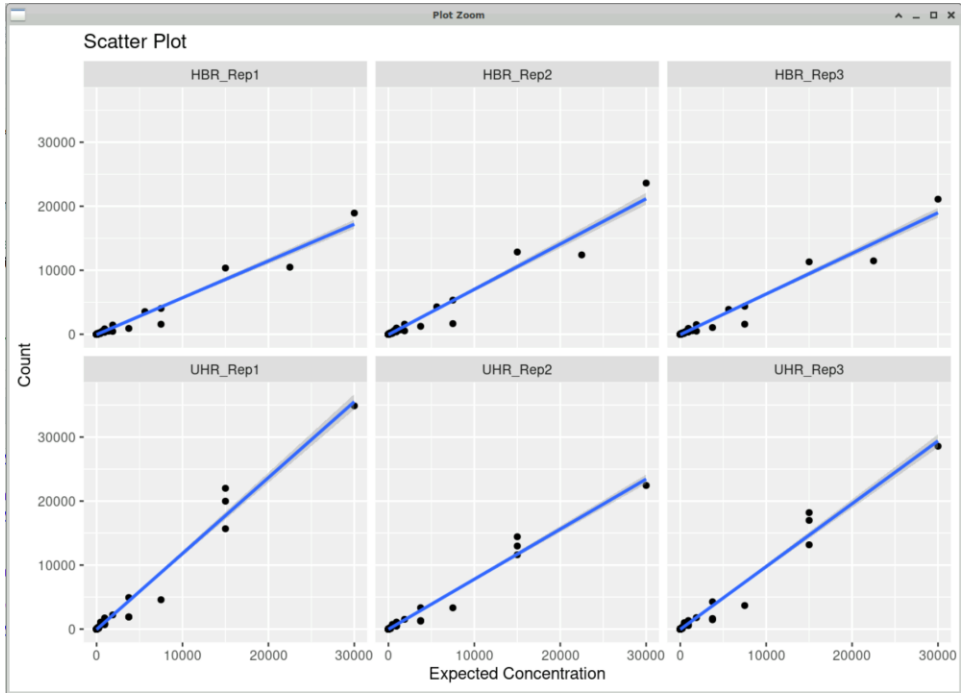


Plot Linearity in Rstudio

If you want to see the trend for each sample, you can add `facet_wrap(~sample)`:

```
# create a scatter plot on "count" by "concentration"
ggplot(ercc_ref_counts_long, aes(x = concentration, y = count)) +
  geom_point() +
  geom_smooth(method = "lm") +
  facet_wrap(~sample) +
  labs(title = "Scatter Plot", x = "Expected Concentration", y = "Count")
```

Try changing `facet_wrap()` to other variables.



Log-transformation

Log-transformation is commonly used in data analysis for a few key reasons, especially when you're dealing with **count data** or **data with skewed distributions**.

Many biological and environmental data (like count data and concentration measurements) tend to be skewed, meaning **most values are clustered near the low end** with **a few outliers on the higher end**.

Log transformation compresses the large values and stretches out the smaller values, making the data distribution more normal (bell-shaped) or symmetric.

This is particularly important if you plan to **fit a linear model**, as linear models assume data is approximately normally distributed.

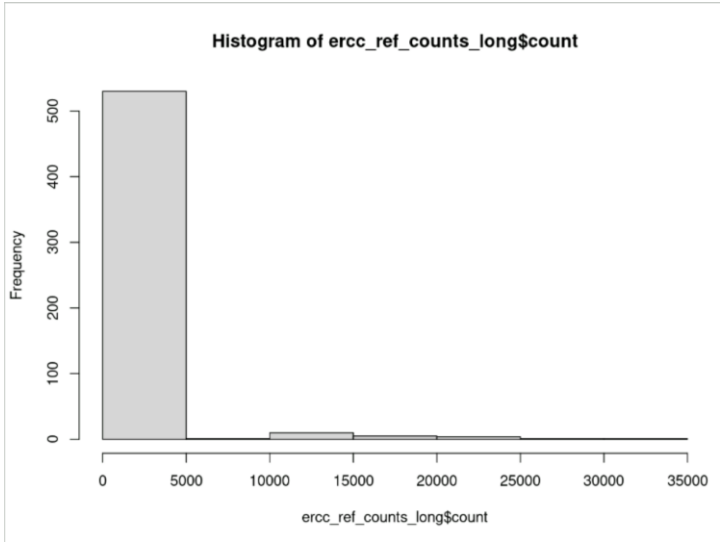
How to plot distribution?

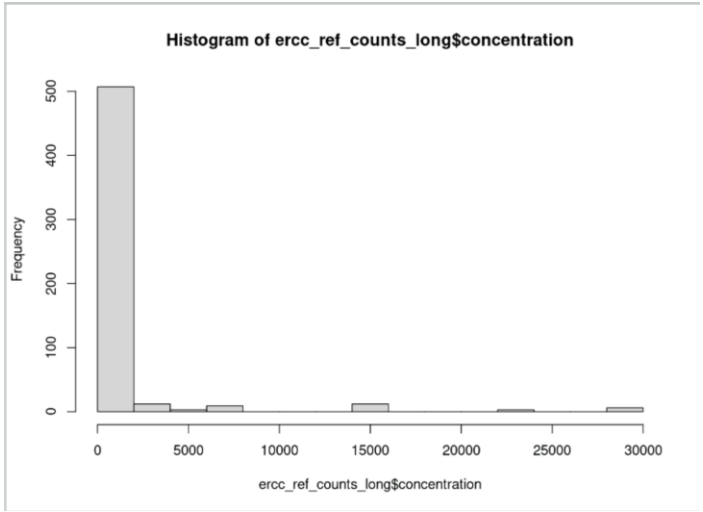
- We can plot a basic histogram for "count" and "concentration".

Log-transformation

```
# plot a histogram for count
hist(ercc_ref_counts_long$count)
# plot a histogram for concentration
hist(ercc_ref_counts_long$concentration)
```

Our "count" and "concentration" data are very skewed. It is necessary we perform log-transformation on it.





Log-transformation

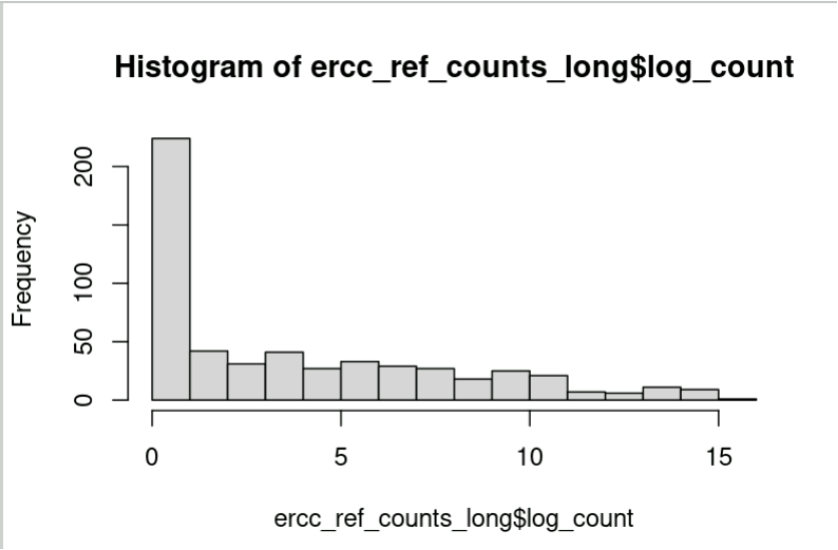
Now, let's perform log transformation on both "count" and "concentration".

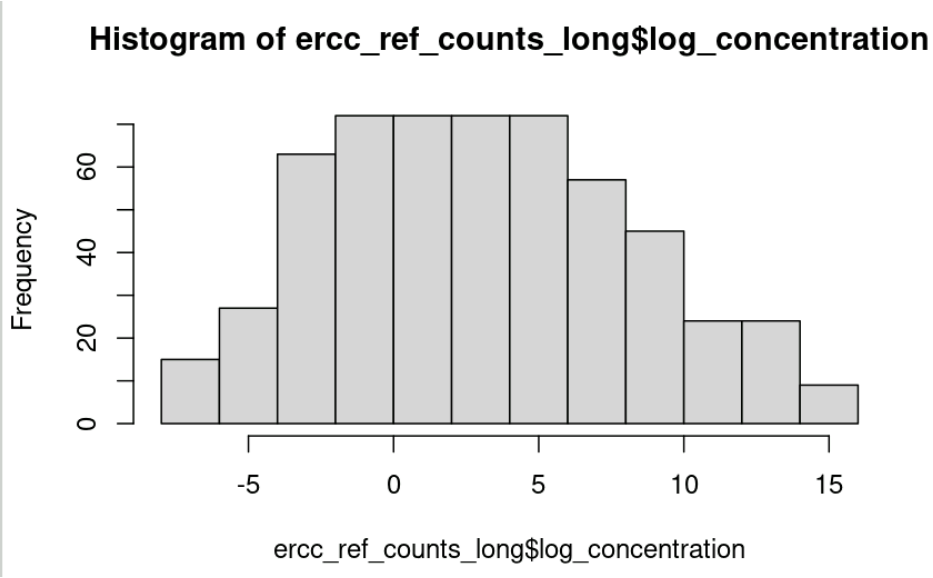
```
# log transform both count and concentration
ercc_ref_counts_long$log_count <- log2(ercc_ref_counts_long$count + 1)
ercc_ref_counts_long$log_concentration <- log2(ercc_ref_counts_long$concentration)
```

Then, we can also plot a histogram to inspect the distribution of log-transformed data.

Exercise: plot histogram for log-transformed count and concentration.

Log-transformation





The log_concentration looks better but log_count still skewed.

Don't worry. The purpose of log transformation is to **reduce variance** and **compress the scale**, not to make the data perfectly normal. This is expected behaviour for RNA-seq data.

Fit to Linear Model

We use the `lm()` function to fit the linear model.

```
# fit a linear model
count_model <- lm(log_count ~ log_concentration, data = ercc_ref_counts_long)
count_model
```

After fitting the model, you can get the details of the model by calling the "count_model".

```
> count_model

Call:
lm(formula = log_count ~ log_concentration, data = ercc_ref_counts_long)

Coefficients:
      (Intercept)  log_concentration
           1.3620             0.7321
```

R-squared and Slope

By calling `summary(count_model)` you can get more detail for the fitted linear model.

```
> summary(count_model)

Call:
lm(formula = log_count ~ log_concentration, data = ercc_ref_counts_long)

Residuals:
    Min      1Q  Median      3Q     Max 
-5.6617 -0.8878  0.0369  0.9277  3.8562 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)        1.36200    0.07338   18.56   <2e-16 ***
log_concentration  0.73215    0.01193   61.38   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.449 on 550 degrees of freedom
Multiple R-squared:  0.8726,	Adjusted R-squared:  0.8724 
F-statistic:  3767 on 1 and 550 DF,  p-value: < 2.2e-16
```

To retrieve a certain value from the summary, we can run:

```
count_r_squared <- summary(count_model)[["r.squared"]]
count_slope <- coef(count_model)["log_concentration"]
```

```
> count_r_squared
[1] 0.872599
> count_slope
log_concentration 
        0.7321468 
>
```

R-squared and Slope Explained

```
> count_r_squared
[1] 0.872599
> count_slope
log_concentration
        0.7321468
>
```

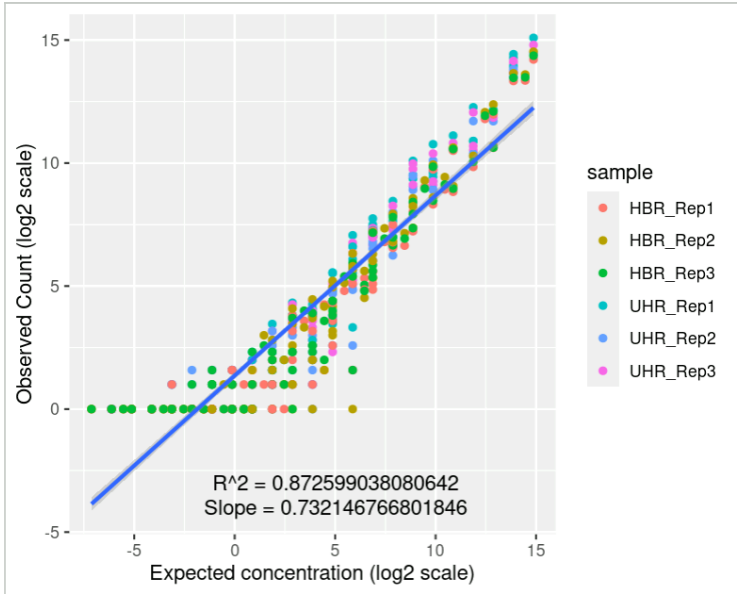
An **R-squared of 0.872599** means a good fit, **counts closely track expected concentrations**, but with a small deviation. Possibly due to noise or low-abundance issues.

A slope of 1 means a perfect doubling of count for every doubling of concentration - ideal, linear, proportional.

A **slope of 0.7321468** means that **counts increases less than expected** - possible saturation, low capture efficiency, or technical loss.

Create a plot for linear model

```
# create a plot
ggplot(ercc_ref_counts_long, aes(x = log_concentration, y = log_count)) +
  geom_point(aes(color = sample)) +
  geom_smooth(method = lm) +
  annotate("text", 5, -3, label = paste("R^2 =", count_r_squared, sep = " ")) +
  annotate("text", 5, -4, label = paste("Slope =", count_slope, sep = " ")) +
  xlab("Expected concentration (log2 scale)") + ylab("Observed Count (log2 scale)")
```



Create a new folder "plots" under "RNAseq-Workshop".

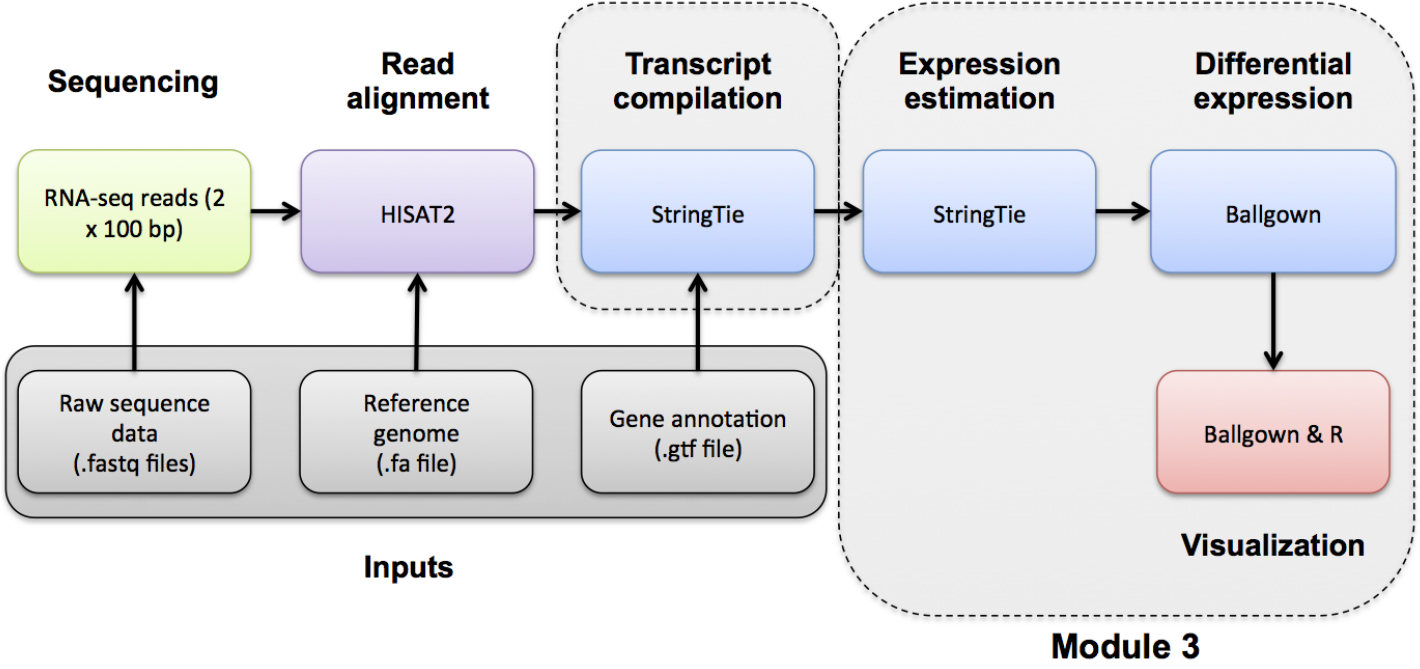
Save the plot as PDF.

You can do this by clicking in the Rstudio.

ERCC Expression Analysis with TPM

Exercise: do the same analysis for the TPM expression results.

Expression Analysis with Ballgown



Expression Analysis with Ballgown

Ballgown is a software package designed to facilitate differential expression analysis of RNA-Seq data. It also provides functions to organise, visualise, and analyse the expression measurements for your transcriptome assembly.

We will use Ballgown to compare the **UHR and HBR** samples.

Ballgown readable expression output:

- "e_data.ctab"
- "i_data.ctab"
- "t_data.ctab"
- "e2t.ctab"
- "i2t.ctab"

We have these files from the StringTie output.

Expression Analysis with Ballgown

First, let's create a new folder to save the Ballgown results.

- Create `~/RNAseq-Workshop/de/ballgown/`
- Open Rstudio.
- Create a new R script "Ballgown_Analysis.R"

Load packages:

```
library(ballgown)
library(genefilter)
library(dplyr)
library(devtools)

```

Expression Analysis with Ballgown

Then we need to create the phenotype data needed for ballgown.

```
# create phenotype data for ballgown
ids <- c("UHR_Rep1", "UHR_Rep2", "UHR_Rep3", "HBR_Rep1", "HBR_Rep2", "HBR_Rep3")
type <- c("UHR", "UHR", "UHR", "HBR", "HBR", "HBR")
inputs <- "/home/vdiuser/RNAseq-Workshop/expression/stringtie/"
path <- paste(inputs, ids, sep="")
pheno_data <- data.frame(ids, type, path)
```

If you print out `path`, it would be the 6 paths to the folder that stores ballgown readable expression data.

```
> path
[1] "/home/vdiuser/RNAseq-Workshop/expression/stringtie/UHR_Rep1"
[2] "/home/vdiuser/RNAseq-Workshop/expression/stringtie/UHR_Rep2"
[3] "/home/vdiuser/RNAseq-Workshop/expression/stringtie/UHR_Rep3"
[4] "/home/vdiuser/RNAseq-Workshop/expression/stringtie/HBR_Rep1"
[5] "/home/vdiuser/RNAseq-Workshop/expression/stringtie/HBR_Rep2"
[6] "/home/vdiuser/RNAseq-Workshop/expression/stringtie/HBR_Rep3"
```

Expression Analysis with Ballgown

And your `pheno_data` would look like:

	ids	type	path
1	UHR_Rep1	UHR	/home/vdiuser/RNAseq-Workshop/expression/stringt...
2	UHR_Rep2	UHR	/home/vdiuser/RNAseq-Workshop/expression/stringt...
3	UHR_Rep3	UHR	/home/vdiuser/RNAseq-Workshop/expression/stringt...
4	HBR_Rep1	HBR	/home/vdiuser/RNAseq-Workshop/expression/stringt...
5	HBR_Rep2	HBR	/home/vdiuser/RNAseq-Workshop/expression/stringt...
6	HBR_Rep3	HBR	/home/vdiuser/RNAseq-Workshop/expression/stringt...

Then, we can load data into R as a **ballgown object**:

```
# load data to ballgown
bg <- ballgown(samples = as.vector(pheno_data$path), pData = pheno_data)
```

```
> bg <- ballgown(samples = as.vector(pheno_data$path), pData = pheno_data)
Wed May  7 12:11:10 2025
Wed May  7 12:11:10 2025: Reading linking tables
Wed May  7 12:11:10 2025: Reading intron data files
Wed May  7 12:11:10 2025: Merging intron data
Wed May  7 12:11:10 2025: Reading exon data files
Wed May  7 12:11:10 2025: Merging exon data
Wed May  7 12:11:10 2025: Reading transcript data files
Wed May  7 12:11:10 2025: Merging transcript data
Wrapping up the results
Wed May  7 12:11:10 2025
```

Use `?ballgown` to see what `samples = ` and `pData = ` means.

Expression Analysis with Ballgown

You can view the ballgown object by calling the object name `bg`:

```
> bg
ballgown instance with 4564 transcripts and 6 samples
>
```

Load all ballgown attributes to a table, then extract unique genes and transcripts:

```
# load all attributes including gene name
bg_table <- texpr(bg, 'all')
bg_gene_names <- unique(bg_table[, 9:10])
bg_transcript_names <- unique(bg_table[, c(1, 6)])
```

?texpr

Save the ballgown object for later use:

```
# save ballgown object for later use
save(bg, file = 'de/ballgown/bg.rda')
```

Expression Analysis with Ballgown

Then, we need to pull the **gene** and **transcript expression** from the ballgown object.

```r
# pull gene and transcript expression from ballgown object
gene_expression <- as.data.frame(gexpr(bg))
transcript_expression <- as.data.frame(texpr(bg))
```

?gexpr
?as.data.frame

Ballgown_Analysis.R* × gene_expression ×

Filter

	FPKM.UHR_Rep1	FPKM.UHR_Rep2	FPKM.UHR_Rep3	FPKM.HBR_Rep1
ENSG00000008735	9.733280	19.257822	6.9885366	565.891856
ENSG00000015475	101.648194	92.512731	118.9674979	77.882924
ENSG00000025708	65.150466	35.587802	39.0739550	32.946374
ENSG00000025770	302.197952	253.872392	272.7187762	
ENSG00000040608	23.674617	18.662807	16.8523781	
ENSG00000054611	60.588021	56.225716	66.8954804	

Ballgown_Analysis.R* × transcript_expression ×

Filter

	FPKM.UHR_Rep1	FPKM.UHR_Rep2	FPKM.UHR_Rep3	FPKM.HBR_Rep1
1	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000	0.000000

DE for transcript

Then, we can perform DE analysis with no filtering, at both gene and transcript level.

```r
# DE analysis - transcript level
results_transcripts <- stattest(bg, feature = "transcript", covariate = "type",
                                getFC = TRUE, meas = "FPKM")
```

	feature	id	fc	pval	qval
9	transcript	9	1.00000000	NaN	NaN
10	transcript	10	1.00000000	NaN	NaN
11	transcript	11	0.88037335	0.883077263	0.95527897
12	transcript	12	1.00000000	NaN	NaN
13	transcript	13	0.81182953	0.883077263	0.95527897
14	transcript	14	1.00000000	NaN	NaN
15	transcript	15	1.00000000	NaN	NaN
16	transcript	16	1.00000000	NaN	NaN

Interpret DE result

The table generated by the ballgown DE analysis consists of several columns:

- feature - transcript or gene
- id
- fc - Log2 fold change between the two groups (UHR vs HBR)
- pval - Raw p-value from the statistical test (t-test or linear model)
- qval - Adjusted p-value (FDR, usually via Benjamin-Hochberg)

Log2 Fold Change

	feature	id	fc	pval	qval
2210	transcript	2210	1.301122e+02	1.172003e-07	0.0003692980
1113	transcript	1113	1.159073e+04	2.602241e-07	0.0004099831
4097	transcript	4097	7.972280e+01	3.252065e-06	0.0034157518

- Positive value: higher expression in the **second group**
- Negative value: higher expression in the **first group**

The fold change of 2.1 means about a 4.3-fold increase ($2^{2.1} \approx 4.3$).

From the above example, you can see that transcript 2210, 1113, 4097 are **upregulated** in group2 which is the HBR samples.

p-value and q-value

	feature	id	fc	pval	qval
2210	transcript	2210	1.301122e+02	1.172003e-07	0.0003692980
1113	transcript	1113	1.159073e+04	2.602241e-07	0.0004099831
4097	transcript	4097	7.972280e+01	3.252065e-06	0.0034157518

qval < 0.05

- statistically significant difference in expression after multiple testing correction (False Discover Rate)

pval < 0.05 but qval > 0.05

- Might be interesting but not statistically reliable after correction

A q-value of 0.01 indicates high statistical confidence.

DE for transcript

Then, let's add names and sample FPKM values:

```
# also add names and FPKM values
results_transcripts <- merge(results_transcripts, bg_transcript_names,
                             by.x = c("id"), by.y = c("t_id"))
results_transcripts <- merge(results_transcripts, transcript_expression,
                             by.x = c("id"), by.y = c("row.names"))
```

	id	feature	fc	pval	qval	t_name	FPKM.UHR_Rep1
1	1	transcript	1.000000e+00	NaN	NaN	ENST00000615943	0.000000
2	10	transcript	1.000000e+00	NaN	NaN	ENST00000448473	0.000000
3	100	transcript	1.000000e+00	NaN	NaN	ENST00000517943	0.000000
4	1000	transcript	1.000000e+00	NaN	NaN	ENST00000403807	0.000000
5	1001	transcript	1.000000e+00	NaN	NaN	ENST00000302273	0.000000
6	1002	transcript	8.877916e-01	8.830773e-01	0.955278968	ENST00000624350	0.000000
7	1003	transcript	3.081885e-01	6.275653e-01	0.918173301	ENST00000610778	2.335612
8	1004	transcript	1.000000e+00	NaN	NaN	ENST00000412149	0.000000
9	1005	transcript	1.177738e+00	7.055053e-01	0.937765470	ENST00000413293	129.974991

DE for gene

Exercise: do the same for gene.

Try ?stattest to find the option for gene.

Save the two tables:

```
# save tables
write.table(results_transcripts, "de/ballgown/UHR_vs_HBR_transcript_results.tsv",
            sep = "\t", quote = FALSE, row.names = FALSE)
write.table(results_genes, "de/ballgown/UHR_vs_HBR_gene_results.tsv",
            sep = "\t", quote = FALSE, row.names = FALSE)
```

Filter low abundance genes

Remove all transcripts with a variance across the samples of less than 1.

```
# remove low-abundance transcripts
bg_filt <- subset(bg, "rowVars(texpr(bg)) > 1", genomesubset = TRUE)
```

Then, we have a new ballgown object.

Filter low abundance genes

Why filter out those with variance less than 1?

Filtering out transcripts with **low variance across samples** is a common preprocessing step in RNA-seq analysis. Low-variance transcript add noise, not signal.

Transcripts that barely change across conditions or samples **aren't biologically informative** - they tend to represent:

- Housekeeping genes (?) or non-regulated transcripts
- Measurement noise, especially at low expression levels
- Unexpressed or universally low-abundance transcripts

They **dilute statistical power** and **increase false positives** in differential expression analysis.

Filter low abundance genes

Improves statistical modelling.

Differential expression tests (like in DESeq2, edgeR, or ballgown) assume:

- There's a meaningful difference between conditions
- The variance reflects biological variability, not just technical noise

By filtering out near-constant transcripts:

- Dispersion estimates (?) are more accurate
- P-value corrections are less conservative (?)
- The multiple testing burden is reduced

Filter low abundance genes

Reduces computational cost.

RNA-seq datasets can contain tens of thousands of transcripts. Filtering helps us:

- Focus on biologically relevant features
- Speed up analysis
- Reduce memory usage

Why not filter too aggressively?

- If your threshold is too high (e.g., var>10), you risk throwing out **real but subtle signals** - especially for low-expressed or tissue-specific genes.
- So, **variance>1 is often used** as a moderate, safe threshold.

DE with filtered result

Exercise:

- **perform DE analysis on the new ballgown object, for both gene and transcript**
- **save the two tables**

Identify Significant Genes/Transcripts

With p-value < 0.05:

```
# identify significant genes/transcripts with p-value < 0.05
sig_transcripts <- subset(results_transcripts, results_transcripts$pval<0.05)
sig_genes <- subset(results_genes, results_genes$pval<0.05)
```

Save the two tables:

```
# save the tables
write.table(sig_transcripts, "de/ballgown/UHR_vs_HBR_transcript_results_sig.tsv",
            sep = "\t", quote = FALSE, row.names = FALSE)
write.table(sig_genes, "de/ballgown/UHR_vs_HBR_gene_results_sig.tsv",
            sep = "\t", quote = FALSE, row.names = FALSE)
```

Save gene names

Open your terminal and run:

`grep -v feature UHR_vs_HBR_gene_results_sig.tsv | cut -f 6 > DE_genes.txt`

This will save only the gene names column to a file.

```
(base) vdiuser@vdj-33xefq:~/RNAseq-Workshop/de/ballgown$ head DE_genes.txt
MAPK8IP2
BID
NCAPH2
RTN4R
CECR5
UFD1L
CLTCL1
DGCR2
PANX2
GTSE1
```

Thank you

Contact us

Jiajia Li
Biological Data Science Institute

RN Robertson Building, 46 Sullivan's Creek Rd
The Australian National University
Canberra ACT 2600

E jiajia.li1@anu.edu.au
W https://bdsi.anu.edu.au/

Australian National University