Introduction to Bulk RNAseq data analysis

Linear Models in R and DESeq2

Last modified: 30 Jun 2021

Contents

Summary

QUESTIONS

- How do I define a statistical model in R and DESeq2?
- How do I interpret the results from these models and define hypothesis to test?
- How can I define comparisons of interest between different groups of samples?

We will use the following packages for this lesson:

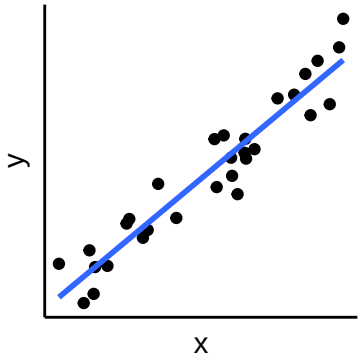
```
# load packages for this session
library(DESeq2)
library(tidyverse)
theme_set(theme_classic())
```

Linear Models in R

Although this course is focused on analysis of RNA-seq data with *DESeq2*, much of the machinery used by this program is based on a general class of statistical models called **linear models**. The way *DESeq2* defines these models is very similar to how it's done in base R, and so we will cover both how defining linear models works in R, and then more specifically how to do it in *DESeq2*.

A model is a simplified representation of how we think different variables relate to each other. Typically we have an observed variable (e.g. gene expression) and we want to understand how it is influenced by other variables, which we often control or vary (e.g. a treatment or a genotype).

One of the most well-known examples of a linear model is the *linear regression*:



In this example, we say that the variable "y" is the **outcome or dependent variable** and its value is influenced by the **predictor or independent variable** "x". We can express this relationship as:

$y = \beta_0 + \beta_1 x$

Those β_i terms are numbers that we want to estimate:

- β_0 is the **intercept**, i.e. the value of y when $x = 0$
- β_1 is the **slope**, i.e. how much y changes for every unit increase of x

For example, if $y = 0.5 - 4.2 \times x$ we have that $\beta_0 = 0.5$ and $\beta_1 = -4.2$.

TODO

Make a note that this is a simplified way to write our model. More correct might be:

$y_i \sim N(\mu_i, \sigma^2)$

$\mu_i = \beta_0 + \beta_1 x_i$

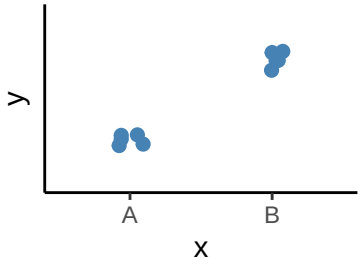
Or, for gene expression:

$y_i \sim NB(\mu_i, \phi)$

$log(\mu_i) = \beta_0 + \beta_1 x_i$

Categorical Variables

In linear regression the predictor variable x is *continuous*. However, in RNA-seq experiments our predictors are more often **categorical variables** (aka **factors**). For example:

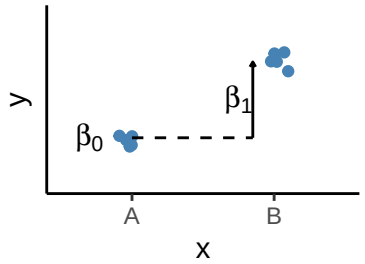


Although this looks a little different from a linear regression, it can also be expressed as a linear model in the following way:

$y = \beta_0 + \beta_1 x_B$

Where:

- β_0 is our *intercept*, which in this case is the average of a **reference group** (in this example "A").
- β_1 is the **difference** between the *reference group* and the other group ("B").



The trick here is to turn x into an **indicator variable** (aka **dummy variable**), which will have values 0 or 1 depending on whether a given observation is in group B.

sample	x	x_b
sample1	A	0
sample2	A	0
sample3	A	0
sample4	B	1
sample5	B	1
sample6	B	1

For example, if $y = 2.1 + 0.3 \times x_B$, we have that $\beta_0 = 2.1$ and $\beta_1 = 0.3$. And because x_B is an indicator 0/1 variable, the average of each group is:

- if "A" then $y = 2.1 + 0.3 \times 0 = 2.1$
- if "B" then $y = 2.1 + 0.3 \times 1 = 2.1 + 0.3 = 2.4$

We don't have to worry about creating these *indicator variables*, since R takes care of creating them for us, as we will see below. However, it is important to understand how these models are specified, so that we can interpret our results.

Glossary of Statistical Terms

- **variable** - something that was measured or controlled for, essentially a column in our table.
- **outcome** variable - the variable that you're trying to make inferences on. For example, gene expression.
- **predictor** variable - a variable that we think influences the outcome variable in some way. For example, genotype or treatment.
- **factor** - a predictor variable that is categorical (rather than continuous).
- **level** - the different values that a factor can take. For example, the genotype variable with two levels: "wt" and "mutant".
- **indicator variables** (aka **dummy variables**) - variables that can take the value 0 or 1 to indicate whether an observation belongs to a given group.
- model **coefficient** - those β_i terms are called the coefficients of the model. They are what we want to estimate from our data.
- model **intercept** - in models with factors (categorical X) this is the average of a reference level. For example, for a genotype with two levels - "wt" and "mutant" -, we may set "wt" as the reference level, so so that the other coefficients are defined in relation to it.

Null Hypothesis Testing

Often, when we fit a model to our data we want to infer whether a particular variable does indeed influence our outcome (and possibly by how much). This is usually achieved using **null hypothesis testing**, where

we ask the question: how compatible are my data with a "boring" hypothesis? If the data are extremely incompatible with this null hypothesis (low p-value), we reject that hypothesis in favour of an alternative hypothesis.

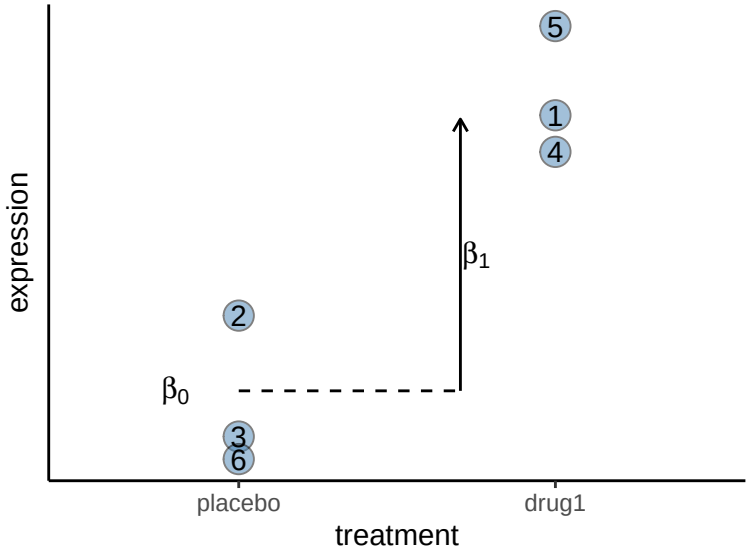
Typically the null hypothesis is that our predictor variable has no effect on the outcome. Taking the examples above, our null hypothesis would be that $\beta_1 = 0$.

To answer this question we need two things:

- Our best estimate for β_1 given the data
- The *uncertainty* around that estimate, which we can denote as σ_{β_1}

Once we have those two pieces of information we can calculate a *test statistic* as $\frac{\beta_1}{\sigma_{\beta_1}}$. We can then compare this distribution to a standard distribution, and calculate a **p-value**. The p-value indicates the probability of observing a test statistic at least as extreme as ours, assuming that nothing is going on. Or, in other words, it tells us how compatible our data are with $\beta_1 = 0$. If the p-value is very low, that tells us that our data are not very compatible with that hypothesis, and so we may reject it in favour of assuming that $\beta_1 \neq 0$.

Note that a high p-value doesn't mean that $\beta_1 = 0$, it just means that our data are compatible with that hypothesis, but it may also be equally compatible with other hypothesis. For example, if we have a lot of noise in our data (or very few replicates), then it's hard to reject any hypothesis, because we cannot be very precise in our estimate of β_1 (i.e. σ_{β_1} is too high).



- Write a model compatible with the plot above, which shows the effect of a drug on gene expression.
- Biologically speaking, what do each of the β coefficients in your model mean?
- What is the null hypothesis for testing for an effect of the treatment on the gene's expression?
- Write the values for the indicator variable for each sample (the numbers in the points are sample ID).

Answer

We could write the model as follows:

$expression = \beta_0 + \beta_1 treatment_{drug1}$

The meaning of the coefficients is:

- β_0 is the average expression of the reference group, the placebo in this case.
- β_1 is the average difference in expression between drug1 and placebo.

The null hypothesis that drug1 has no effect on our gene's expression is $\beta_1 = 0$.

The indicator variable for $treatment_{drug1}$ is:

```
sample1  1
sample2  0
sample3  0
sample4  1
sample5  1
sample6  0
```

Model Specification

To build our model, we start by specifying the **model design**, which is an expression that indicates what factors we expect to influence our outcome variable. In R, we use a *formula* syntax to specify our models, for example:

```
y ~ x
```

We can read this as: the values of the outcome variable "y" are dependent on the predictor variable "x". In RNA-seq, "y" is the expression of a gene, and the variables "x" are typically things that we experimentally manipulated or controlled for (e.g. a treatment, or a genotype).

When we specify a model design (in *DESeq2* or in standard functions in R such as `lm()`, used to fit linear models), *R* takes care of creating those indicator variables that we talked about earlier. These are encoded in a so-called **model matrix**.

Let's see how this works with an example. Say we had the following information about some samples in a RNA-seq experiment:

```
example_samples <- tibble(
  sample = paste0("sample", 1:6),
  treatment = rep(c("A", "B"), each = 3)
)
example_samples
```

```
## # A tibble: 6 x 2
##   sample  treatment
##   <chr>   <chr>
## 1 sample1 A
## 2 sample2 A
## 3 sample3 A
## 4 sample4 B
## 5 sample5 B
## 6 sample6 B
```

First let's define our model formula for the predictor variables:

```
temperature_model <- formula(~ treatment)
```

To find our model matrix we can do:

```
model.matrix(temperature_model, data = example_samples)
```

```
##   (Intercept) treatmentB
## 1           1          0
## 2           1          0
## 3           1          0
## 4           1          1
## 5           1          1
```

```
## 6           1          1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$treatment
## [1] "contr.treatment"
```

We can see that our variable `treatment` was turned into two *indicator variables* called `(Intercept)` and `treatmentB`. These match what we talked about earlier:

- `(Intercept)` is the indicator variable for β_0, which corresponds to the reference group (in this case "A").
- `treatmentB` is the indicator variable for β_1, and tells us whether samples belong to the "B" group (it takes the value 1 only for samples 4-6).

- Using the `read_tsv()` function, read in the table with sample information from our RNA-seq experiment located in `data/samplesheet_corrected.tsv`. Save it in an object called `sample_info`.
- Create a model formula to investigate the effect of "Status" on gene expression.
- Look at the model matrix and identify which is the reference group in your model.

Answer

We can read the table as follows:

```
sample_info <- read_tsv("data/samplesheet_corrected.tsv")

# print it's contents
sample_info
```

```
## # A tibble: 12 x 4
##    SampleName Replicate Status     TimePoint
##    <chr>          <dbl> <chr>      <chr>
##  1 SRR7657878         1 Infected   d11
##  2 SRR7657881         2 Infected   d11
##  3 SRR7657880         3 Infected   d11
##  4 SRR7657874         1 Infected   d33
##  5 SRR7657882         2 Uninfected d33
##  6 SRR7657872         3 Infected   d33
##  7 SRR7657877         1 Uninfected d11
##  8 SRR7657876         2 Uninfected d11
##  9 SRR7657879         3 Uninfected d11
## 10 SRR7657883         1 Uninfected d33
## 11 SRR7657873         2 Infected   d33
## 12 SRR7657875         3 Uninfected d33
```

Our model formula is:

```
status_model <- formula(~ Status)
```

To investigate which is our reference group, we can look at the model matrix:

```
model.matrix(status_model, data = sample_info)
```

```
##    (Intercept) StatusUninfected
## 1            1                0
## 2            1                0
## 3            1                0
## 4            1                0
## 5            1                1
```

```
## 6            1                0
## 7            1                1
## 8            1                1
## 9            1                1
## 10           1                1
## 11           1                0
## 12           1                1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$Status
## [1] "contr.treatment"
```

From the column names of this matrix, we can see that we have `StatusUninfected` as one of our indicator variables. Therefore, by exclusion, "Infected" was set as the reference group.

Although this is OK, it might make more intuitive sense to set "Uninfected" as the reference group. How to achieve this is detailed after this exercise.

You will have noticed that, in the previous exercise, the reference level was set simply taking into account alphabetical order. Whichever value comes first (alphabetically) is set as the reference level in the model.

To change this default behaviour, we need to turn our categorical variables into *factors*, with a specific level set as the reference. We can achieve this with the `fct_relevel()` function:

```
fct_relevel(sample_info$Status, "Uninfected")
```

```
##  [1] Infected   Infected   Infected   Infected   Uninfected Infected
##  [7] Uninfected Uninfected Uninfected Uninfected Infected   Uninfected
## Levels: Uninfected Infected
```

If we modify our `sample_info` table in this way, we can see that the reference level in the model matrix changes to have "Uninfected" as the reference:

```
sample_info$Status <- fct_relevel(sample_info$Status, "Uninfected")

model.matrix(status_model, data = sample_info)
```

```
##    (Intercept) StatusInfected
## 1            1              1
## 2            1              1
## 3            1              1
## 4            1              1
## 5            1              0
## 6            1              1
## 7            1              0
## 8            1              0
## 9            1              0
## 10           1              0
## 11           1              1
## 12           1              0
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$Status
## [1] "contr.treatment"
```

Why do we use model matrices?

TODO Make a note about matrix multiplication.

Common Designs

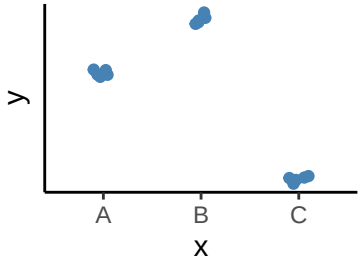
So far we've been showing examples of a simple model design: a single factor with two levels. Examples of this type of design are:

- Treatment: "control", "treated"
- Genotype: "wt", "mutant"
- Timepoint: "t0", "t1"
- Infection: "uninfected", "infected"
- Disease: "healthy", "sick"

However, very often we have other kinds of experimental designs, and we cover some of the more common ones in this section.

One Factor, Multiple Levels

What if we had a factor with three levels?

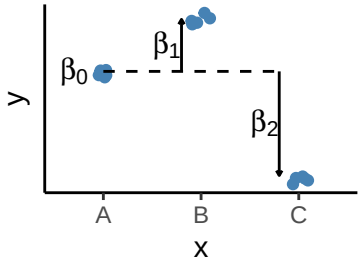


The same concept as we covered before applies here, except that we have to create more *indicator variables*. The linear model for this illustration would be:

$y = \beta_0 + \beta_1 x_B + \beta_2 x_C$

Where:

- β_0 is the average of the *reference group* (in this example "A")
- β_1 is the *difference* between "B" and the reference group "A".
- β_2 is the *difference* between "C" and the reference group "A".
- x_B indicates whether a sample belongs to group B or not.
- x_C indicates whether a sample belongs to group C or not.



For example, if $y = 2.1 + 0.3 \times x_B - 5 \times x_C$, we have that $\beta_0 = 2.1$, $\beta_1 = 0.3$ and $\beta_2 = -5$. And because x_B and x_C are indicator 0/1 variables, the average of each group is:

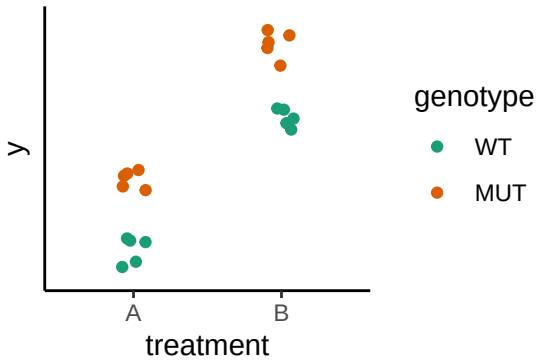
- **If "A"** then $y = 2.1 + 0.3 \times 0 - 5 \times 0 = 2.1$

- **If "B"** then $y = 2.1 + 0.3 \times 1 - 5 \times 0 = 2.1 + 0.3 = 2.4$
- **If "C"** then $y = 2.1 + 0.3 \times 0 - 5 \times 1 = 2.1 - 5 = -2.9$

Two Factors, Additive Model

So far, we've only talked about models with a *single factor* (with either two or three levels). However, many experiments involve two or more factors (e.g. a genotype *and* a treatment). In these cases the design can get a little more complicated, especially due to the fact that there may be *interactions* between the factors.

A two-factor design might look like this:



We have **two factors, each with two levels**:

- treatment with levels "A" and "B"
- genotype with levels "WT" and "MUT"

In this case we can see that there is both an effect of treatment ("B" is on average higher than "A") and also genotype ("WT" is on average higher than "MUT").

Defining a model formula for this relationship is similar to what we did before. Let's see an example:

```
# example data
two_factor <- tibble(genotype = rep(c("WT", "MUT"), 4),
                     treatment = rep(c("A", "B"), each = 4)) %>%
  arrange(genotype, treatment) %>%
  mutate(genotype = fct_relevel(genotype, "WT"))

two_factor
```

```
## # A tibble: 8 x 2
##   genotype treatment
##   <fct>    <chr>
## 1 MUT      A
## 2 MUT      A
## 3 MUT      B
## 4 MUT      B
## 5 WT       A
## 6 WT       A
## 7 WT       B
## 8 WT       B
```

To specify our model, we now have to add both variables to our formula:

```
two_factor_additive <- formula(~ genotype +  treatment)
```

```r
# check the model matrix with dummy/indicator variables
model.matrix(two_factor_additive, data = two_factor)
```

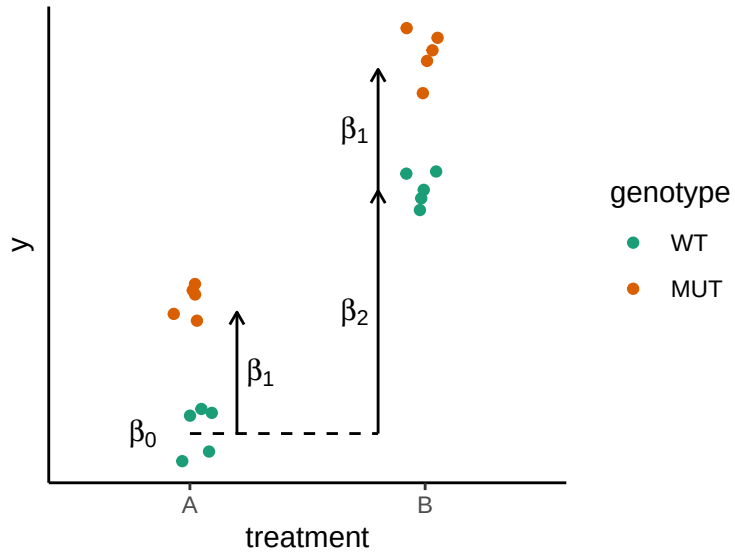
```
##   (Intercept) genotypeMUT treatmentB
## 1           1           1          0
## 2           1           1          0
## 3           1           1          1
## 4           1           1          1
## 5           1           0          0
## 6           1           0          0
## 7           1           0          1
## 8           1           0          1
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$genotype
## [1] "contr.treatment"
##
## attr(,"contrasts")$treatment
## [1] "contr.treatment"
```

Looking at our model matrix, we can see that it looks somewhat similar to what we had before, except that now we have indicator variables for both genotype and treatment We can deduce that our intercept refers to treatment "A", genotype "WT" (this is our reference group).

Our model could therefore be written as:

$y = \beta_0 + \beta_1.genotype_{MUT} + \beta_2.treatment_B$

And illustrated as follows:



Again, everything is defined relative to the reference group, so for example if we had that $y = 1 + 1.5 \times genotype_{MUT} + 2 \times treatment_B$, then the average prediction for each of our groups would be:

- if "A" and "WT": $y = 1 + 1.5 \times 0 + 2 \times 0 = 1$ (only the intercept contributes since this is the reference group)
- if "B" and "WT": $y = 1 + 1.5 \times 0 + 2 \times 1 = 3$ (the intercept and the coefficient for treatment contributes, but not genotype since "WT" is the reference level for genotype)

- if "A" and "MUT": $y = 1 + 1.5 \times 1 + 2 \times 0 = 2.5$ (the intercept and the coefficient for genotype contribute, but not treatment since "A" is the reference level for treatment)
- if "B" and "MUT": $y = 1 + 1.5 \times 1 + 2 \times 1 = 4.5$ (all terms contribute)

- Using our `sample_info` object, create a new design formula specifying an additive model between "Status" and "TimePoint".
- How many coefficients do you have with this model?
  HintLook at the model matrix.
- What is your reference group?

Answer

The model is:

```
additive_model <- formula(~ Status + TimePoint)
```

We can check how many coefficients we will with this mode, by examining the model matrix:

```
model.matrix(additive_model, data = sample_info)
```

```
##    (Intercept) StatusInfected TimePointd33
## 1            1               1            0
## 2            1               1            0
## 3            1               1            0
## 4            1               1            1
## 5            1               0            1
## 6            1               1            1
## 7            1               0            0
## 8            1               0            0
## 9            1               0            0
## 10           1               0            1
## 11           1               1            1
## 12           1               0            1
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$Status
## [1] "contr.treatment"
##
## attr(,"contrasts")$TimePoint
## [1] "contr.treatment"
```

We have 3 coefficients, and from their names we can see that this corresponds to:

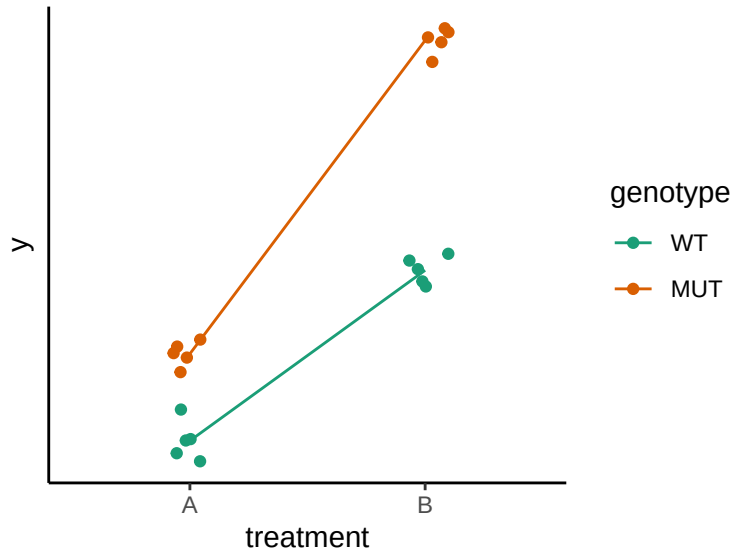
$Expression = \beta_0 + \beta_1.Status_{Infected} + \beta_2.TimePoint_{d33}$

Where:

- β_0 is the intercept, i.e. the average of the reference group (Uninfected, d11).
- β_1 is the difference between Infected and Uninfected.
- β_2 is the difference between d33 and d11.

Two Factors, Interaction Model

One major issue that we need to consider when going from one to two factors is that there might be an **interaction** between them. Take the following example:



Now we can see that the way each genotype responds to the treatment is different: the WT has a bigger difference between "A -> B" compared to the mutant (the "slopes" of those lines are different). This is what we mean by an interaction: the effect of one variable depends on the value of the other variable. In other words, the effect of the treatment depends on the individual's genotype.

We can model interactions between factors by using the following formula syntax:

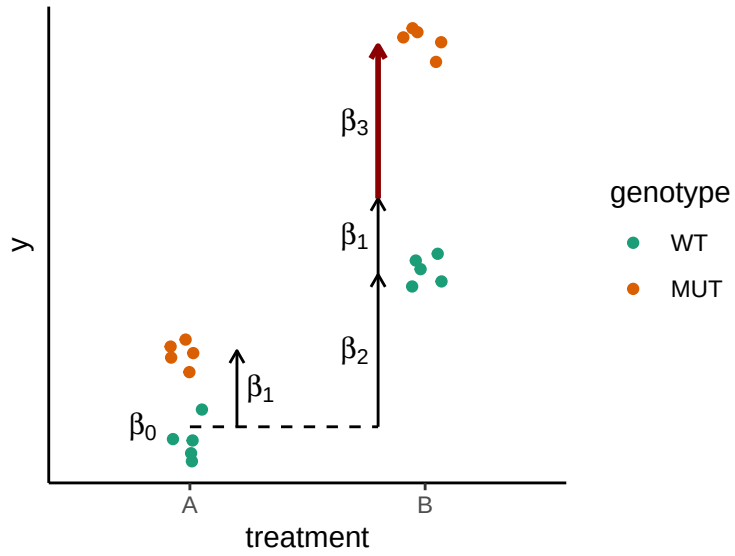
```
# specify our model formula, with both variables added as well as their interaction
two_factor_interaction <- formula(~ genotype + treatment + genotype:treatment)

# check the model matrix with dummy/indicator variables
model.matrix(two_factor_interaction, data = two_factor)
```

```
##   (Intercept) genotypeMUT treatmentB genotypeMUT:treatmentB
## 1           1           1          0                       0
## 2           1           1          0                       0
## 3           1           1          1                       1
## 4           1           1          1                       1
## 5           1           0          0                       0
## 6           1           0          0                       0
## 7           1           0          1                       0
## 8           1           0          1                       0
## attr(,"assign")
## [1] 0 1 2 3
## attr(,"contrasts")
## attr(,"contrasts")$genotype
## [1] "contr.treatment"
##
## attr(,"contrasts")$treatment
## [1] "contr.treatment"
```

This look similar to what we had before, except now we have a third indicator variable, and this tells us which samples are *both* "MUT" genotype and treatment "B". Writing our model, we have:

$y = \beta_0 + \beta_1 . genotype_{MUT} + \beta_2\ treatment_B + \beta_3\ genotype_{MUT} \& treatment_B$

Crucially, β_3 is the interaction term. This term reflects the "extra nudge" that the sample gets for being a MUT in treatment B specifically:



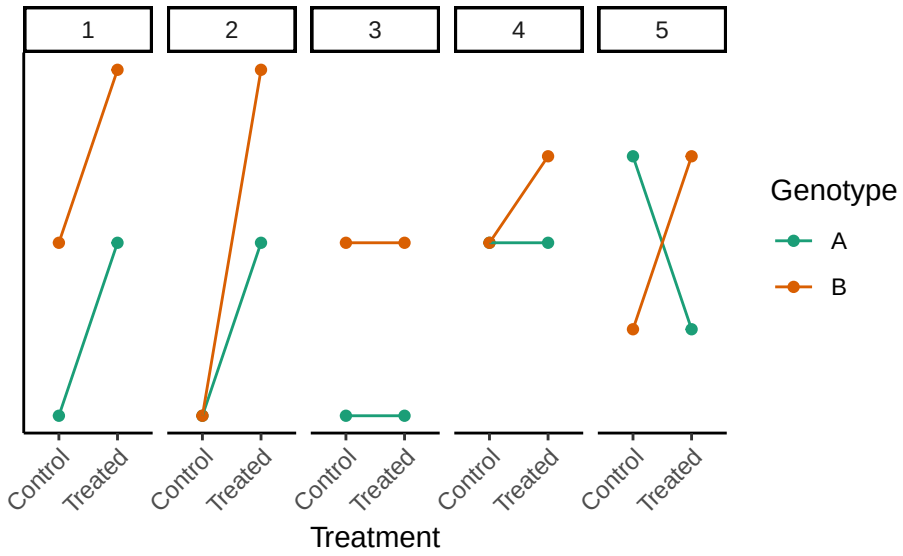
As before, everything is defined relative to the reference group, so for example if we had that $y = 1 + 1.5 \times genotype_{MUT} + 2 \times treatment_B + 1 \times genotype_{MUT}\&treatment_B$, then the average prediction for each of our groups would be:

- if "A" and "WT": $y = 1 + 1.5 \times 0 + 2 \times 0 + 1 \times 0 = 1$ (only the intercept contributes since this is the reference group)
- if "B" and "WT": $y = 1 + 1.5 \times 0 + 2 \times 1 + 1 \times 0 = 3$ (the intercept and the coefficient for treatment contributes, but not genotype since "WT" is the reference level for genotype)
- if "A" and "MUT": $y = 1 + 1.5 \times 1 + 2 \times 0 + 1 \times 0 = 2.5$ (the intercept and the coefficient for genotype contribute, but not treatment since "A" is the reference level for it)
- if "B" and "MUT": $y = 1 + 1.5 \times 1 + 2 \times 1 + 1 \times 1 = 5.5$ (all terms contribute)

Should I always include an interaction in my model?

The answer is probably yes. Unless you have a strong reason to believe that the responses to your factors are purely additive, then you should include an interaction in your model.

- Which of these cases show an interaction between the Treatment and Genotype variables?
- And which ones differ between genotypes under control conditions?



Answer

- Panels 2, 4 and 5 show an interaction between the two variables. In panel 1 there is an effect of both genotype and treatment, but both genotypes respond the same to the treatment (the lines are parallel to each other). In panel 3 there's no effect of treatment for either genotype (the lines are "flat"), although the two genotypes differ from each other.
- Panels 1, 3, 5 show a different of genotypes in control conditions. By contrast, in panels 2 and 4 the points for the "Control" treatment are overlapping.

Other Designs

There are more complicated experimental designs, for example including more factors (not just two), and more levels within each factor. You can also have so-called *nested designs*, where the levels of the factors do not occur in all combinations. For example, if you have samples from patients in different hospitals, each patient only occurs in a single hospital (you can't have a patient in two hospitals at once).

We will not cover these more complicated designs in this course. In those instances it may be helpful to try and get in touch with a local bioinformatician/statistician to help you. However, always try and do a web-search first, as there are many questions and answers posted on the Bioconductor Support Site.

Model Specification in DESeq2

Having covered how models are specified in R, how factors are encoded as *indicator variables* and how we can interpret the coefficients of our models and build hypothesis from them, we are now ready to see how these things apply to DESeq2.

DESeq2 uses a special kind of object called `DESeqDataSet`, which stores a large matrix of gene counts (genes as rows and samples as columns) and also information about each sample in that matrix. The information about the samples is stored in the `colData` part of the object (since samples are the *columns* of the gene counts matrix). This is one of the most relevant parts of the object for us, since it contains the variables that we use to build our model.

We will see how to create a `DESeqDataSet` object from our data in the next lesson. For now, let's work with simulated data just to illustrate how we specify models and extract results of interest.

```r
# create a simulated DESeqDataSeq object
dds <- makeExampleDESeqDataSet(n = 100, m = 9)
colData(dds) <- DataFrame(treatment = factor(rep(c("A", "B", "C"), each = 3)),
                          row.names = paste0("sample", 1:9))

# check the sample information
colData(dds)
```

```
## DataFrame with 9 rows and 1 column
##         treatment
##          <factor>
## sample1         A
## sample2         A
## sample3         A
## sample4         B
## sample5         B
## sample6         B
## sample7         C
## sample8         C
## sample9         C
```

In this example we have a single factor ("treatment") with three levels ("A", "B" and "C"). We can define our model formula using the `design()` function:

```r
design(dds) <- formula(~ treatment)
```

Just as before, we can check our model matrix by pulling the formula directly from the `design` and `colData` information of our object:

```r
model.matrix(design(dds), data = colData(dds))
```

```
##         (Intercept) treatmentB treatmentC
## sample1           1          0          0
## sample2           1          0          0
## sample3           1          0          0
## sample4           1          1          0
## sample5           1          1          0
## sample6           1          1          0
## sample7           1          0          1
## sample8           1          0          1
## sample9           1          0          1
## attr(,"assign")
## [1] 0 1 1
## attr(,"contrasts")
## attr(,"contrasts")$treatment
## [1] "contr.treatment"
```

This looks just like what we did before, except that now everything is stored together in our `DESeqDataSet` object.

Our model for gene expression can therefore be written as:

$expression = \beta_0 + \beta_1 treatment_B + \beta_2 treatment_C$

With this information, *DESeq* is now ready to fit its in-built statistical model to the data (modelling our gene counts using a *negative binomial* distribution). To do this, we update our object by running the `DESeq()` function:

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

Now, for the part that interests us in this session, we want to look at the results of our differential expression test. We can do this with the `results()` function:

```
results(dds)
```

```
## log2 fold change (MLE): treatment C vs A
## Wald test p-value: treatment C vs A
## DataFrame with 100 rows and 6 columns
##          baseMean log2FoldChange     lfcSE        stat    pvalue      padj
##         <numeric>      <numeric> <numeric>   <numeric> <numeric> <numeric>
## gene1    16.93716      0.0554784  0.675141    0.082173  0.934509  0.998924
## gene2     5.75715     -1.8104928  1.151139   -1.572784  0.115769  0.716320
## gene3    13.77509      0.4575277  0.837984    0.545986  0.585075  0.970291
## gene4    14.59073     -0.8588416  0.811190   -1.058743  0.289717  0.911449
## gene5     4.22101      1.3699354  1.602147    0.855062  0.392517  0.970291
## ...           ...            ...       ...         ...       ...       ...
## gene96    4.69107    -0.00396555  1.374596 -0.00288488  0.997698  0.998924
## gene97    1.84454    -2.85379286  2.480231 -1.15061557  0.249890  0.853074
## gene98  143.01302    -0.36519748  0.444417 -0.82174501  0.411222  0.970291
## gene99    3.03927    -1.99952588  1.601656 -1.24841148  0.211880  0.788871
## gene100   1.42573     0.89939505  2.577874  0.34889025  0.727172  0.970291
```

We can see at the top of the printed table that these results are for a test between treatments C vs A. But how did the `results()` function determine this? Well, in this case it picked one of the possible comparisons for us, but we can define which comparison we want.

We can see the estimated coefficients from our model using the `resultsNames()` function:

```
resultsNames(dds)
```

```
## [1] "Intercept"        "treatment_B_vs_A" "treatment_C_vs_A"
```

Linking this back to our model specification:

- `Intercept` is what we've called β_0; typically this is not something we're interested in.
- `treatment_B_vs_A` is what we've called β_1, but DESeq nicely names it with an intuitive name, suggesting that this term is effectively comparing treatment "B" against the reference level "A".
- `treatment_C_vs_A` is what we've called β_2, again named intuitively to indicate this is equivalent to comparing treatment "C" against the reference level "A".

To specify a different comparison from the default, we can use the `contrast` option when getting our results:

```
# compare B vs A
results(dds, contrast = list("condition_B_vs_A"))
```

Comparing "A" vs the others is relatively easy, because "A" is the reference level. But what if we wanted to compare B vs C? As we saw in a previous exercise, this is equivalent to asking whether the two coefficients are different from each other (the null hypothesis is: $\beta_2 - \beta_1 = 0$). We can get this comparison by passing two values to the list:

```
# compare C vs B
results(dds, contrast = list("treatment_C_vs_A", "treatment_B_vs_A"))
```

```
## log2 fold change (MLE): treatment_C_vs_A vs treatment_B_vs_A
## Wald test p-value: treatment_C_vs_A vs treatment_B_vs_A
## DataFrame with 100 rows and 6 columns
##          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
##         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
```

```
## gene1    16.93716      -1.203794  0.648955 -1.854973 0.0636001  0.503459
## gene2     5.75715       0.531701  1.250190  0.425297 0.6706204  0.893284
## gene3    13.77509       0.832356  0.847033  0.982672 0.3257689  0.844011
## gene4    14.59073      -1.918993  0.792785 -2.420571 0.0154961  0.498633
## gene5     4.22101       1.720095  1.621988  1.060485 0.2889238  0.844011
## ...           ...            ...       ...       ...       ...       ...
## gene96    4.69107      -2.353426  1.270122 -1.852914 0.0638947  0.503459
## gene97    1.84454      -2.880422  2.479146 -1.161861 0.2452921  0.783352
## gene98  143.01302       0.226056  0.446606  0.506163 0.6127419  0.893284
## gene99    3.03927      -2.123250  1.597064 -1.329471 0.1836926  0.732852
## gene100   1.42573      -2.321028  2.337808 -0.992822 0.3207966  0.844011
```

Interpreting the Results

When obtaining the `results()` from our model, we get several columns in the output. Their meaning is:

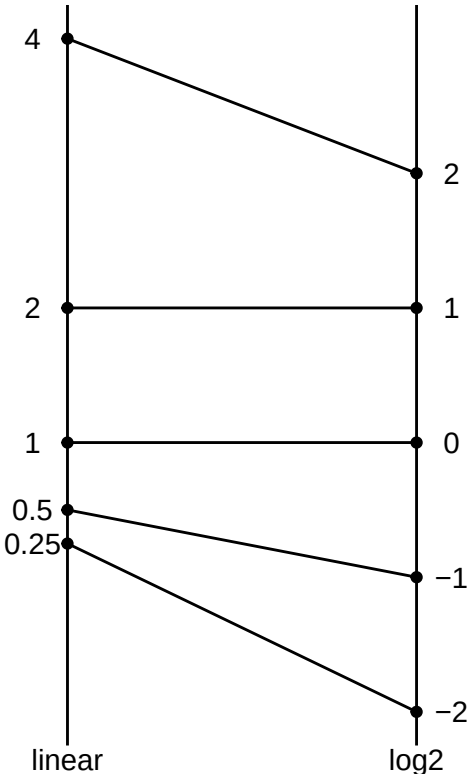
- `baseMean` is the overall mean expression across all the samples (regardless of which group they belong to). It's a useful column to investigate which genes are overall highly or lowly expressed.
- `log2FoldChange` is the estimated difference between the two conditions being compared. It's the value for what we called β in our models. We'll come back to this column below.
- `lfcSE` is the standard error (SE) for the log-fold-change (lfc). This is the uncertainty of our β estimate, what earlier we referred to as σ_{β_i}.
- `stat` is our test statistic (called Z-statistic), corresponding to $Z = \frac{\beta_i}{\sigma_{\beta_i}}$, you can check this by dividing the `log2FoldChange` column by the `lfcSE` column.
- `pvalue` this is the p-value for the test implemented in DESeq2, which by default is the "Wald Z-test", which assumes that $Z \sim N(0, 1)$. The p-value indicates the probability of getting a test statistic, Z, at least as extreme as ours assuming that $\beta_i = 0$ (or, in other words, `log2FoldChange` $= 0$).
- `padj` is the p-value adjusted for multiple testing. In this case, DESeq2 uses a **false discovery rate** correction.

Let's focus a but more on the `log2FoldChange` column. So far, we've been expressing comparisons between groups as *differences*, but DESeq reports the results as fold-change, which is a *ratio*. For example, for the comparison between "A" and "C" that we did above, the meaning of that column is:

$log_2(\frac{\text{Average Expression in C}}{\text{Average Expression in A}})$

Expressing our results as a ratio is very useful when we're interested in *relative* changes. In this case it is more interpretable to say "gene X is 2 times more highly expressed in WT compared to Mutant" than to say "gene X is 20 units expression higher in WT compared to Mutant".

The reason that relative changes are expressed on a log-scale has to do with having a symetric representation of the differences between the numerator and denominator. The issue is that when the numerator > denominator values can go from 1 to infinity. But when the numerator < denominator they can only vary between 0 and 1. Expressing things on a log scale solves this problem:



In the example above, notice that a relative change of 4 times is the same magnitude as a change of 0.25 (i.e. 1/4). On a log-scale these take the same magnitude (2), just with a different sign.

In summary, the values on the `log2FoldChange` column represent the log-ratio of our two groups being compared. Positive values mean that the group in the numerator had higher expression, whereas negative values indicate that the group in the denominator had higher expression.

There is another (more technical) reason why the results are expressed on a log scale. In the negative binomial model, the average expression is modelled on a log-scale. So, if you recall the logarithmic rule that $log(b) - log(a) = log(\frac{b}{a})$, then you can see that on a log-scale the model really considers things as a *difference* between groups. But when we interpret our results, because of the log-scale, we know it's meaning is really a ratio of our original expression values.

Two-Factor Models

As before, let's consider dataset with two varying factors (similar to what we have in our course data).

```
# create a simulated DESeqDataSeq object
dds <- makeExampleDESeqDataSet(n = 100, m = 12)
colData(dds) <- DataFrame(treatment = factor(rep(c("A", "B"), each = 6)),
                          genotype = factor(rep(c("WT", "MUT"), 6),
                                            levels = c("WT", "MUT")),
                          row.names = paste0("sample", 1:12))

# check the sample information
colData(dds)
```

```
## DataFrame with 12 rows and 2 columns
##          treatment genotype
##           <factor> <factor>
```

```
## sample1          A      WT
## sample2          A      MUT
## sample3          A      WT
## sample4          A      MUT
## sample5          A      WT
## ...            ...      ...
## sample8          B      MUT
## sample9          B      WT
## sample10         B      MUT
## sample11         B      WT
## sample12         B      MUT
```

As we've done before for the simpler model, fix the code below to model the effects of genotype, treatment and their interaction.

```
# update model design
design(dds) <- formula(~ FIXME)

# fit the model
dds <- DESeq(dds)
```

After you fit the model, use `resultsNames()` to see the coefficients that DESeq created. Relate them to the β parameters in this notation and interpret their meaning:

$y = \beta_0 + \beta_1\ genotype_{MUT} + \beta_2\ treatment_B + \beta_3\ genotype_{MUT}\&treatment_B$

Answer

The fixed code is:

```
design(dds) <- formula(~ genotype + treatment + genotype:treatment)
dds <- DESeq(dds)
```

After we fit the model, we can extract the results coefficients:

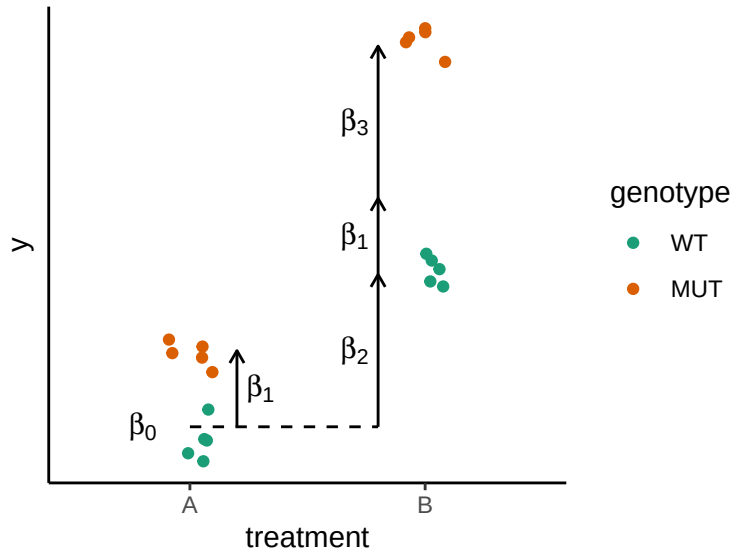
```
resultsNames(dds)
```

```
## [1] "Intercept"              "genotype_MUT_vs_WT"     "treatment_B_vs_A"
## [4] "genotypeMUT.treatmentB"
```

We have four coefficients, which we can relate to that model notation as meaning:

- β_0 = `Intercept` is the average expression of the reference group. In this case the reference group is "MUT" genotype in condition "A".
- β_1 = `genotype_MUT_vs_WT` is the average difference of "MUT" genotype versus the "WT", both in condition "A".
- β_2 = `treatment_B_vs_A` is the average difference of condition "B" versus "A", both for "MUT" genotype.
- β_3 = `genotypeMUT.treatmentB` is the "difference of differences", i.e. whether "WT" and "MUT" respond differently to the treatment.

Because now we have a model with an interaction term, to obtain results of interest we need to consider which genotype or treatment we want to consider. Let's remind ourselves of what these terms mean with a schematic:



If we match these β coefficients with how DESeq names them, we have:

- β_0 = `Intercept`

- β_1 = `genotype_MUT_vs_WT`
- β_2 = `treatment_B_vs_A`
- β_3 = `genotypeMUT.treatmentB`

Here is a summary of all the comparisons we may want to make:

Null Hypothesis	DESeq contrast	Question Answered
$\beta_1 = 0$	`list("genotype_MUT_vs_WT")`	Are "WT" and "MUT" different in condition "A"?
$\beta_2 = 0$	`list("treatment_B_vs_A")`	Are conditions "A" and "B" different for "WT"?
$\beta_3 = 0$	`list("genotypeMUT.treatmentB")`	Do the genotypes respond differently to the treatment?
$\beta_1 + \beta_3 = 0$	`list(c("genotype_MUT_vs_WT", "genotypeMUT.treatmentB"))`	Are "WT" and "MUT" different in condition "B"?

For example to test for the interaction, we would do:

```
results(dds, contrast = list("genotypeWT.conditionB"))
```

Extra: Numeric Contrasts

So far, specifying our contrasts or comparisons of interest relies on a clear interpretation of each coefficient, and can sometimes get a little confusing to know what is what (especially as our designs become more complex). There is an alternative way of defining contrasts, which is less standard and not as thoroughly documented, but offers greater flexibility and, for some people, might feel more intuitive.

This relies on defining a numeric vector of weights for the terms that we want to test for. Taking our current object with these four coefficients:

```
resultsNames(dds)
```

```
## [1] "Intercept"              "genotype_MUT_vs_WT"     "treatment_B_vs_A"
## [4] "genotypeMUT.treatmentB"
```

If we wanted to test for the second coefficient (`genotype_MUT_vs_WT`), with a numeric contrast we would do:

```
results(dds, contrast = c(0, 1, 0, 0))
```

The way to construct these numeric contrasts is to first make a vector of coefficient weights for each of our groups. We can do this based on looking at the model matrix, and checking which samples take the value 1 for each of the coefficients:

```
mod_mat <- model.matrix(design(dds), data = colData(dds))
mod_mat
```

```
##          (Intercept) genotypeMUT treatmentB genotypeMUT:treatmentB
## sample1            1           0          0                       0
## sample2            1           1          0                       0
## sample3            1           0          0                       0
## sample4            1           1          0                       0
## sample5            1           0          0                       0
## sample6            1           1          0                       0
## sample7            1           0          1                       0
## sample8            1           1          1                       1
## sample9            1           0          1                       0
## sample10           1           1          1                       1
## sample11           1           0          1                       0
## sample12           1           1          1                       1
## attr(,"assign")
## [1] 0 1 2 3
## attr(,"contrasts")
## attr(,"contrasts")$genotype
## [1] "contr.treatment"
##
## attr(,"contrasts")$treatment
## [1] "contr.treatment"
```

From this, we can define the following coefficient weights for our four groups of samples:

```
# coefficient weights
wt_a <- c(1, 0, 0, 0)
mut_a <- c(1, 1, 0, 0)
wt_b <- c(1, 0, 1, 0)
mut_b <- c(1, 1, 1, 1)
```

Now, to define our contrasts we can express things as differences between the groups we want to compare. Here are all the contrasts we mentioned earlier:

```
# difference between mutant and wild-type
mut_a - wt_a # under condition A
mut_b - wt_b # under condition B

# difference between conditions
wt_b - wt_a   # for wild-type
mut_b - mut_a # for mutant

# difference of differences (interaction)
```

```
# does the mutant's response to treatment different from the wild-type's response?
(mut_b - mut_a) - (wt_b - wt_a)
```

With numeric contrasts you can even define more unusual contrasts. For example, let's say we were interested in whether the average of the samples in condition "B" was different from the average of the samples in condition "A". This would be the contrast for that question:

```
# average of b_mut and b_wt minus average of a_mut and a_wt
(mut_b + wt_b)/2 - (mut_a + wt_a)/2
```

Summary

KEY POINTS

- Differential expression tests are based on linear models, where the gene expression is modelled as an outcome of several variables of interest (e.g. treatment, genotype, infection status, etc.).
- Linear models use *indicator or dummy variables* to encode categorical variables in a model matrix.
- To define models in R/DESeq2 we use the formula `~` syntax.
- Some common models are:
  - Single factor: `~ variable1`
  - Two factor, additive: `~ variable1 + variable2`
  - Two factor, interaction: `~ variable1 + variable2 + variable1:variable2`
- Interpreting our model coefficients allows us to define hypothesis/comparisons of interest.
- In DESeq2 we use the `results()` function to obtain the log2 fold-change in gene expression between groups of interest ("contrast").