An example use of linear modeling

In this notebook, we will explore the basics of applying linear modelling to identify changes in protein abundance. Ultimately, we'll build towards identifying changes in RNA binding but first we start with the simplest possible experimental design

```
# load packages
library(tidyverse)

# set up standardised plotting scheme
theme_set(theme_bw(base_size = 20) +
            theme(panel.grid.major=element_blank(),
                  panel.grid.minor=element_blank(),
                  aspect.ratio=1))

# Utility function to print to output of lm
print_output <- function(output, cex = 0.7) {
  tmp <- capture.output(output)
  plot.new()
  suppressWarnings(text(0, 1, paste(tmp, collapse='\n'), adj = c(0,1), family = 'mono', cex = cex))
}
```

Treatment vs Control

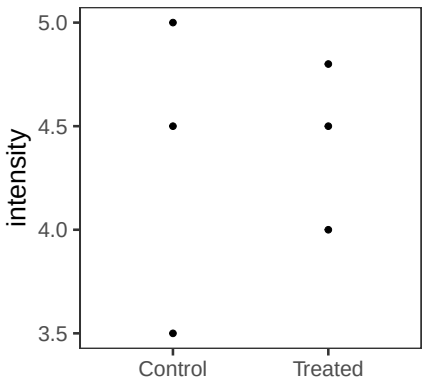
To start, we'll consider a simple experimental design of "Treated" vs "Control" samples. The abundance values have already been log2-transformed so that it's reasonable to assume they are normally distributed.

```
protein_abundance <- read.delim("../raw/simple_example_protein_abundance.tsv", sep="\t")
```

First, let's visualise these intensity values

```
p <- protein_abundance %>%
  ggplot(aes(x=condition, y=intensity)) +
  geom_point(size=2) +
  xlab("")

print(p)
```

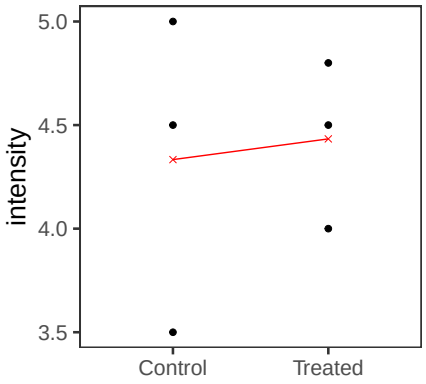


We can make this plot a bit more intuitive if we include a some `stat_summary` objects to show the difference

in the mean intensity

```
 #  summarise each group (defaults to mean) and plot line
p2 <- p +
  aes(group=protein_id) +
  stat_summary(geom="line", fun.y=mean, colour="red") +
  stat_summary(geom="point", pch=4, size=2, colour="red", fun.y=mean)

print(p2)
```



If we want to identify significant changes in protein abundance upon treatment, we can simply use a linear model where the intensity values are assumed to be drawn from a normal distribution described by a constant factor (Intercept) and the condition (Control or Treatment). We want to estimate the value for the Intercept and condition coefficients. If the condition coefficient is significant different from 0, we can state that the condition appears to have a signficant affect on the intensity.

Below, we peform this linear modelling using the base R function `lm` and the formula `intensity~condition`, e.g the dependent variable is `intensity` and the indepdent variable is '. When we summarise the fit, we can see that the coefficient for the condition ("conditionTreated") is estimated to be positive (Estimate=0.1). However, the Std. Error for the coefficient estimate is very large (0.4989; we only have 3 replicates) and the p-value is therefore large (p=0.85). Thus, the coefficient cannot be said to be signficantly different from zero and the treated appears not to affect the abundance of the protein.

```
fit <- protein_abundance %>% lm(formula=intensity~condition)
print_output(summary(fit))
```

```
Call:
lm(formula = intensity ~ condition, data = .)

Residuals:
       1        2        3        4        5        6
−0.83333  0.66667  0.16667  0.06667 −0.43333  0.36667

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)        4.3333     0.3528   12.28 0.000252 ***
conditionTreated   0.1000     0.4989    0.20 0.850911
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.611 on 4 degrees of freedom
Multiple R−squared:  0.009945, Adjusted R−squared:  −0.2376
F−statistic: 0.04018 on 1 and 4 DF,  p−value: 0.8509
```

1.1 Question: What does the Intercept represent? Is the estimate and/or p-value of interest to us?

Note that the above linear model is exactly the same as a two-sided student's t-test (assuming equal variance; shown below) or a one way Analysis of Variance (ANOVA; not shown).

```
# extract intensity values as vectors
treated <- protein_abundance$condition=="Treated"

treated_values <- protein_abundance$intensity[treated]
control_values <- protein_abundance$intensity[!treated]

ttest_results <- t.test(treated_values, control_values, var.equal = TRUE)
print_output(ttest_results)
```

```
 Two Sample t-test

data:  treated_values and control_values
t = 0.20045, df = 4, p-value = 0.8509
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.285134  1.485134
sample estimates:
mean of x mean of y
 4.433333  4.333333
```

So in this case, we could just use a t-test? However, often our experimental design is slightly more complicated. Also, later we will see how to use `limma` to adjust the statistical test for even a simple design such as this when we have a small number of replicates.

Accounting for other factors

In the following experiment, we have quantified protein abundance in mouse blood samples *before* and *after* treatment.

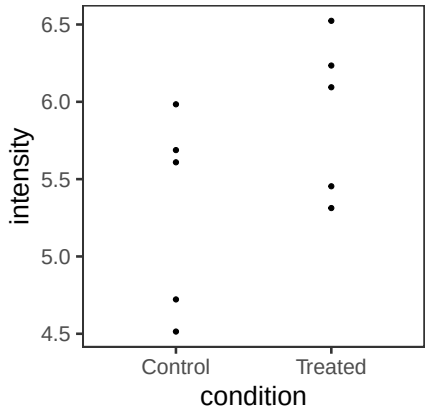
1.2 Question: How would you describe an experimental design such as this?

Since the mice are very expensive, our PI has only allowed us to use 5 mice in this experiment. Note that performing such an experiment without first performing at least a rudimentary power test to see if this experiment is sufficiently powered to detect changes in protein abundance might be considered unethical. In this case, you can rest assured that these *in silico* mice have not been sacrificied in vain!

Let's take a look at the quantification values. Again, we just have a single protein here.

```
mouse_treatment <- read.delim("../raw/mouse_treatment.tsv", sep="\t")
```

```
mouse_treatment %>%
  ggplot(aes(x=condition, y=intensity)) +
  geom_point()
```



1.3 Task: Modify the code so that the plot to show the change between Control and Treated for each mouse. How would you interpret this?

First, let's model the intensity as being dependent on the condition

```
fit <- lm(mouse_treatment, formula=intensity~condition)
print_output(summary(fit))
```

```
Call:
lm(formula = intensity ~ condition, data = mouse_treatment)

Residuals:
    Min      1Q  Median      3Q     Max
-0.7891 -0.5535  0.2378  0.3660  0.6805

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)        5.3034     0.2619  20.247  3.7e-08 ***
conditionTreated   0.6205     0.3704   1.675    0.132
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5857 on 8 degrees of freedom
Multiple R-squared:  0.2596, Adjusted R-squared:  0.1671
F-statistic: 2.805 on 1 and 8 DF,  p-value: 0.1325
```

1.4 Question: How would you interpret the summary above?

Something the above model does consider is that each mouse might have a different "baseline" expression of the protein. To account for this, we can include the mouse ID in the model. This is the standard way to deal with "paired" designs in a linear model.

```
fit <- lm(mouse_treatment, formula=intensity~mouse+condition)
print_output(summary(fit), cex=0.65)
```

```
Call:
lm(formula = intensity ~ mouse + condition, data = mouse_treatment)

Residuals:
    Min      1Q  Median      3Q     Max
−0.8344 −0.4629  0.1867  0.3408  0.6805

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       5.16736    0.50103  10.314 1.74e−05 ***
mouse             0.04534    0.13896   0.326    0.754
conditionTreated  0.62046    0.39304   1.579    0.158
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6214 on 7 degrees of freedom
Multiple R−squared:  0.2707, Adjusted R−squared:  0.06235
F−statistic: 1.299 on 2 and 7 DF,  p−value: 0.3312
```

1.5 Question: Why do we only have one coefficient for the mouse ID, even though there are 5 different mice? Is this what we want?

OK, so we can resolve this by making the `mouse` column a factor and repeating the test

```
mouse_treatment$mouse <- factor(mouse_treatment$mouse)

fit <- lm(mouse_treatment, formula=intensity~mouse+condition)
print_output(summary(fit), cex=0.5)
```

```
Call:
lm(formula = intensity ~ mouse + condition, data = mouse_treatment)

Residuals:
       1        2        3        4        5        6        7        8
−0.05566 −0.14725  0.18488 −0.08909  0.10712  0.05566  0.14725 −0.18488
       9       10
 0.08909 −0.10712

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       4.7777     0.1533  31.158 6.32e−06 ***
mouse2            0.9784     0.1980   4.942  0.00780 **
mouse3            1.0214     0.1980   5.160  0.00670 **
mouse4           −0.1743     0.1980  −0.880  0.42838
mouse5            0.8030     0.1980   4.057  0.01539 *
conditionTreated  0.6205     0.1252   4.956  0.00773 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.198 on 4 degrees of freedom
Multiple R−squared:  0.9577, Adjusted R−squared:  0.9049
F−statistic: 18.12 on 5 and 4 DF,  p−value: 0.007495
```

1.6 Question: Are we interested in the coefficients and p-values for the mouse variables?

Note that we're dealing here with an experimental design that could be handled with a (paired) t-test and yield the exact same result:

```
mouse_treatment_control <- mouse_treatment$intensity[mouse_treatment$condition=="Control"]
mouse_treatment_treatment <- mouse_treatment$intensity[mouse_treatment$condition=="Treated"]

# paired t-test
ttest_results <- t.test(mouse_treatment_treatment, mouse_treatment_control, var.equal = TRUE, paired=TRU
print_output(ttest_results)
```

```
 Paired t-test

data:  mouse_treatment_treatment and mouse_treatment_control
t = 4.9558, df = 4, p-value = 0.007729
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.2728482 0.9680678
sample estimates:
mean of the differences
              0.620458
```

Using interaction terms

OK, so now we're going to finally apply a linear model in a situation where a t-test would be inadequate.

In this dataset, we've quantified "Total" protein abundance and RNA-bound protein using OOPS across a "Control/Treatment" experiment (n=3). For both total and OOPS, we have performed a 6-plex TMT experiment such that we have separately obtained protein-level abundance for "Total" and "OOPS" across the same samples. Again, we will just consider a single protein here.

Here, we read in the data which has been stored in a simple "table"" structure

```
simple_example_rb <- read.delim("../raw/simple_example_RNA_binding.tsv", sep="\t")

print(table(simple_example_rb$condition, simple_example_rb$type))
```

```
##
##           OOPS Total
##   Control    3     3
##   Treated    3     3
```

```
print(simple_example_rb)
```

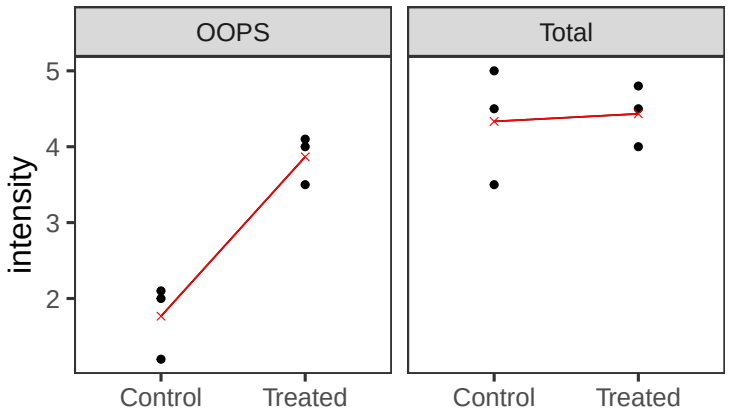
```
##             sample  type condition intensity protein_id
## 1  Total_Control_1 Total   Control       3.5   protein1
## 2  Total_Control_2 Total   Control       5.0   protein1
## 3  Total_Control_3 Total   Control       4.5   protein1
## 4  Total_Treated_4 Total   Treated       4.5   protein1
## 5  Total_Treated_5 Total   Treated       4.0   protein1
## 6  Total_Treated_6 Total   Treated       4.8   protein1
## 7   OOPS_Control_1  OOPS   Control       2.0   protein1
## 8   OOPS_Control_2  OOPS   Control       2.1   protein1
## 9   OOPS_Control_3  OOPS   Control       1.2   protein1
## 10  OOPS_Treated_4  OOPS   Treated       3.5   protein1
## 11  OOPS_Treated_5  OOPS   Treated       4.0   protein1
## 12  OOPS_Treated_6  OOPS   Treated       4.1   protein1
```

Now, let's take a look at the intensities across our dataset

```
plot_all <- simple_example_rb %>%
  ggplot(aes(condition, intensity, group=1)) +
  geom_point(size=2) +
  stat_summary(geom="line") +
  xlab("") +
  stat_summary(geom="line", fun.y=mean, colour="red") +
  stat_summary(geom="point", pch=4, size=2, colour="red", fun.y=mean) +
  facet_wrap(~type)

print(plot_all)
```

```
## No summary function supplied, defaulting to `mean_se()
## No summary function supplied, defaulting to `mean_se()
```



1.7 Question: How would you interpret the intensity values for this protein?

We can test for a significant change in RNA binding by modeling the protein abundance across both TMT plexes and looking for a significant interaction between the condition (Control/Treated) and the type (Total/OOPS). In essense, this is equivalent to estimating the difference between the gradients for the two lines above. The formula for this model is therefore `Intensity ~ condition + type + condition:type`,

where the `condition:type` estimates the fold-change (in log2-space) for RNA binding

Below, we can see that the coefficient for the interaction term is estimated as 2 +/- 0.75 (p=0.03).

```
fit <- simple_example_rb %>%
  # note that one can also use the condition*type shorthand which expands to condition + type + conditi
  lm(formula=intensity ~ condition + type + condition:type)

print_output(summary(fit))
```

```
Call:
lm(formula = intensity ~ condition + type + condition:type, data = .)

Residuals:
    Min      1Q  Median      3Q     Max
-0.8333 -0.3833  0.1500  0.2583  0.6667

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                  1.7667     0.3018   5.853 0.000382 ***
conditionTreated             2.1000     0.4269   4.919 0.001165 **
typeTotal                    2.5667     0.4269   6.013 0.000319 ***
conditionTreated:typeTotal  -2.0000     0.6037  -3.313 0.010651 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5228 on 8 degrees of freedom
Multiple R-squared:  0.8649,  Adjusted R-squared:  0.8142
```

1.8 Question: What do the other coefficients represent and are we interested in the estimates and p-values?

1.9 Question: Note that the interaction term is described as "conditionTreated:typeTotal". What does this mean with respect to how you interpret the coefficient estimate?

R will order factors lexicographically, so type “OOPS” come before type “Total”. We actually want “Total” to be first since this is our “reference level”. We can manually re-level the `type` to acheive this.

```
simple_example_rb$type <- factor(simple_example_rb$type, levels=c("Total", "OOPS"))
```

Question: What effect will this have on our linear model?

```
fit <- simple_example_rb %>%
  # note that one can also use the condition*type shorthand which expands to condition + type + conditi
  lm(formula=intensity ~ condition + type + condition:type)

print_output(summary(fit))
```

```
Call:
lm(formula = intensity ~ condition + type + condition:type, data = .)

Residuals:
    Min      1Q  Median      3Q     Max
−0.8333 −0.3833  0.1500  0.2583  0.6667

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                 4.3333     0.3019  14.356 5.41e−07 ***
conditionTreated            0.1000     0.4269   0.234 0.820667
typeOOPS                   −2.5667     0.4269  −6.013 0.000319 ***
conditionTreated:typeOOPS   2.0000     0.6037   3.313 0.010651 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5228 on 8 degrees of freedom
Multiple R−squared:  0.8649,  Adjusted R−squared:  0.8142
```

Let's explore a bit further why changing the order of the factors changes the results. First we create factors for the condition and type. For the type factor, we create factors with different level orders

```
cond <- factor(simple_example_rb$condition, levels=c("Control", "Treated"))

type_1 <- factor(simple_example_rb$type, levels=c("OOPS", "Total"))
type_2 <- factor(simple_example_rb$type, levels=c("Total", "OOPS"))

intensity <- simple_example_rb$intensity
```

Below, we print out the model matrix where the type factor is ordered with the first level being "OOPS".

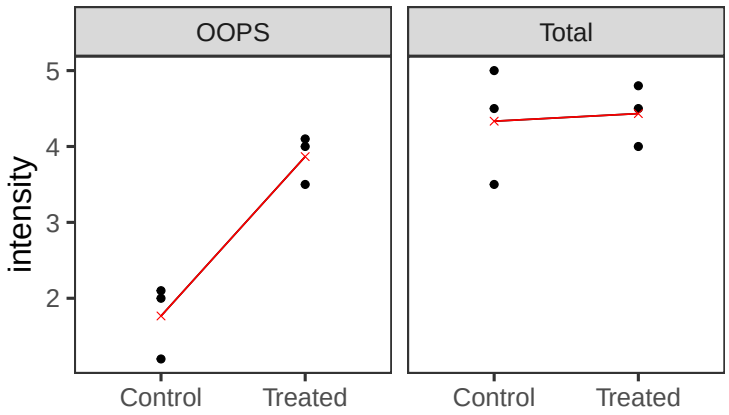
```
model1 <- model.matrix(intensity~cond*type_1)
fit1 <- lm(intensity~cond*type_1)

print(simple_example_rb$sample)
```

```
##  [1] Total_Control_1 Total_Control_2 Total_Control_3 Total_Treated_4
##  [5] Total_Treated_5 Total_Treated_6 OOPS_Control_1  OOPS_Control_2
##  [9] OOPS_Control_3  OOPS_Treated_4  OOPS_Treated_5  OOPS_Treated_6
## 12 Levels: OOPS_Control_1 OOPS_Control_2 OOPS_Control_3 ... Total_Treated_6
```

```
print(plot_all)
```

```
## No summary function supplied, defaulting to `mean_se()
## No summary function supplied, defaulting to `mean_se()
```




```
print(summary(fit1))
```

```
##
## Call:
## lm(formula = intensity ~ cond * type_1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.8333 -0.3833  0.1500  0.2583  0.6667
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)                1.7667     0.3018   5.853 0.000382 ***
## condTreated                2.1000     0.4269   4.919 0.001165 **
## type_1Total                2.5667     0.4269   6.013 0.000319 ***
## condTreated:type_1Total   -2.0000     0.6037  -3.313 0.010651 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5228 on 8 degrees of freedom
## Multiple R-squared:  0.8649, Adjusted R-squared:  0.8142
## F-statistic: 17.07 on 3 and 8 DF,  p-value: 0.0007752
```

```
print(model1)
```

```
##    (Intercept) condTreated type_1Total condTreated:type_1Total
## 1            1           0           1                        0
## 2            1           0           1                        0
## 3            1           0           1                        0
## 4            1           1           1                        1
## 5            1           1           1                        1
## 6            1           1           1                        1
## 7            1           0           0                        0
## 8            1           0           0                        0
## 9            1           0           0                        0
## 10           1           1           0                        0
## 11           1           1           0                        0
## 12           1           1           0                        0
```

```
## attr(,"assign")
## [1] 0 1 2 3
## attr(,"contrasts")
## attr(,"contrasts")$cond
## [1] "contr.treatment"
##
## attr(,"contrasts")$type_1
## [1] "contr.treatment"
```

1.10 Task: By inspecting the model matrix, identify which samples are modelled by the intercept only. Does the intercept value make sense considering the intensity values for these samples?

Next, we do the same using the type factor with the first level being "Total". Note that this neccessarily changes the model matrix and therefore some of the coefficient estimates change.

```
model2 <- model.matrix(intensity~cond*type_2)
fit2 <- lm(intensity~cond*type_2)

print(simple_example_rb$sample)
```

```
##  [1] Total_Control_1 Total_Control_2 Total_Control_3 Total_Treated_4
##  [5] Total_Treated_5 Total_Treated_6 OOPS_Control_1  OOPS_Control_2
##  [9] OOPS_Control_3  OOPS_Treated_4  OOPS_Treated_5  OOPS_Treated_6
## 12 Levels: OOPS_Control_1 OOPS_Control_2 OOPS_Control_3 ... Total_Treated_6
```

```
print(summary(fit2))
```

```
##
## Call:
## lm(formula = intensity ~ cond * type_2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.8333 -0.3833  0.1500  0.2583  0.6667
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              4.3333     0.3019  14.356 5.41e-07 ***
## condTreated              0.1000     0.4269   0.234 0.820667
## type_2OOPS              -2.5667     0.4269  -6.013 0.000319 ***
## condTreated:type_2OOPS   2.0000     0.6037   3.313 0.010651 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5228 on 8 degrees of freedom
## Multiple R-squared:  0.8649, Adjusted R-squared:  0.8142
## F-statistic: 17.07 on 3 and 8 DF,  p-value: 0.0007752
```

```
print(model2)
```

```
##    (Intercept) condTreated type_2OOPS condTreated:type_2OOPS
## 1            1           0          0                      0
## 2            1           0          0                      0
## 3            1           0          0                      0
## 4            1           1          0                      0
## 5            1           1          0                      0
```

```
## 6            1           1          0                       0
## 7            1           0          1                       0
## 8            1           0          1                       0
## 9            1           0          1                       0
## 10           1           1          1                       1
## 11           1           1          1                       1
## 12           1           1          1                       1
## attr(,"assign")
## [1] 0 1 2 3
## attr(,"contrasts")
## attr(,"contrasts")$cond
## [1] "contr.treatment"
##
## attr(,"contrasts")$type_2
## [1] "contr.treatment"
```

Finally, note that if we are only considering the main affects, whilst the sign of the coefficient is affected by the factor levels, the absolute values and t value are exactly the same. Interpreting main affects when your model includes an interaction term is generally not recommended!

```
fit1 <- fit <- lm(intensity ~ cond + type_1)
print_output(summary(fit1))
```

```
Call:
lm(formula = intensity ~ cond + type_1)

Residuals:
    Min      1Q  Median      3Q     Max
-1.0667 -0.3583 -0.1500  0.6417  1.1667

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.2667     0.3796   5.972  0.00021 ***
condTreated   1.1000     0.4383   2.510  0.03332 *
type_1Total   1.5667     0.4383   3.574  0.00598 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7591 on 9 degrees of freedom
Multiple R-squared:  0.6794, Adjusted R-squared:  0.6082
F-statistic: 9.538 on 2 and 9 DF,  p-value: 0.005979
```

```
fit1 <- fit <- lm(intensity ~ cond + type_2)
print_output(summary(fit1))
```

```
Call:
lm(formula = intensity ~ cond + type_2)

Residuals:
    Min      1Q  Median      3Q     Max
−1.0667 −0.3583 −0.1500  0.6417  1.1667

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.8333     0.3796  10.099  3.3e−06 ***
condTreated   1.1000     0.4383   2.510  0.03332 *
type_2OOPS   −1.5667     0.4383  −3.574  0.00598 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7591 on 9 degrees of freedom
Multiple R−squared:  0.6794, Adjusted R−squared:  0.6082
F−statistic: 9.538 on 2 and 9 DF,  p−value: 0.005979
```

OK, so now we can model RNA binding changes with a Total + OOPS TMT experiment. Next we apply this to some real data!