An example use of linear modeling

```
# 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))
```

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.

First, we need to create some toy data:

In this imaginary 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. The abundance values have already been log2-transformed.

To start, we'll consider the "Total" data

```
total_data <- matrix(c(3.5,5,4.5,4.5,4,4.8), nrow=1)
colnames(total_data) <- paste("Total", rep(c("Control", "Treated"), each=3), 1:6, sep="_")
rownames(total_data) <- "PROTEIN_1"

print(total_data)
```

```
##           Total_Control_1 Total_Control_2 Total_Control_3 Total_Treated_4
## PROTEIN_1             3.5               5             4.5             4.5
##           Total_Treated_5 Total_Treated_6
## PROTEIN_1               4             4.8
```

In order to perform the linear modelling, we first need to reshape the data from "wide" to "long" format

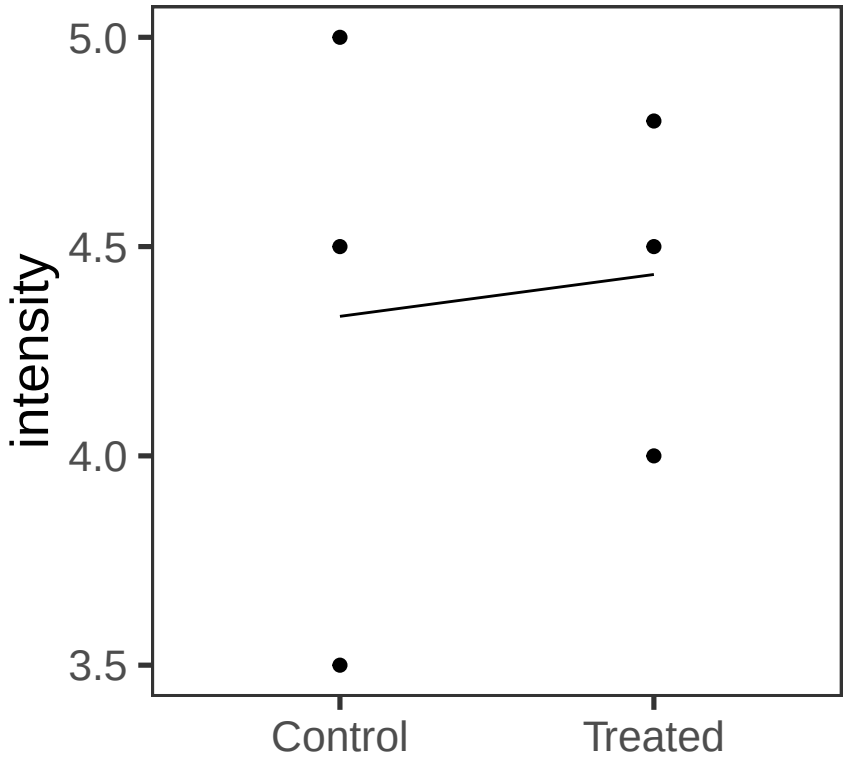
```
total_data_for_lm <- total_data %>%
  data.frame() %>% # make into data.frame() so it can handled by gather()
  gather(key="sample", value="intensity") %>% # convert from wide to long with two columns, "sample" an
  separate(sample, into=c("type", "condition", "tag"), remove=FALSE) %>% # separate the same name into
  mutate(condition=factor(condition, levels=c("Control", "Treated"))) # make "Control" the first condit

print(total_data_for_lm)
```

```
##            sample  type condition tag intensity
## 1 Total_Control_1 Total   Control   1       3.5
## 2 Total_Control_2 Total   Control   2       5.0
## 3 Total_Control_3 Total   Control   3       4.5
## 4 Total_Treated_4 Total   Treated   4       4.5
## 5 Total_Treated_5 Total   Treated   5       4.0
## 6 Total_Treated_6 Total   Treated   6       4.8
```

Now let's visualise these intensity values

```
total_data_for_lm %>%
  ggplot(aes(condition, intensity, group=1)) +
  geom_point(size=2) + # plot a point for each row in the input data
  stat_summary(geom="line") + #  summarise each group (defaults to mean) and plot line
  xlab("")
```

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



OK, so if we were interested in 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=0.1667). However, the Std. Error for the coefficient estimate is very large (0.5270; we only have 3 replicates) and the p-value is therefore large (p=0.77). 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 <- total_data_for_lm %>% lm(formula=intensity~condition)
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
```

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_values <- total_data_for_lm[total_data_for_lm$condition=="Treated", "intensity"]
control_values <- total_data_for_lm[total_data_for_lm$condition=="Control", "intensity"]

t.test(treated_values, control_values, var.equal = TRUE)
```

```
##
##  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, why not just use a t-test? Because it isn't suitable for more complex experimental designs...

Now, let's introduce our toy OOPS data.

```
oops_data <- matrix(c(2,2.1,1.2,3.5,4,4.1), nrow=1)
colnames(oops_data) <- paste("OOPS", rep(c("Control", "Treated"), each=3), 1:6, sep="_")
rownames(oops_data) <- "PROTEIN_1"

print(oops_data)
```

```
##           OOPS_Control_1 OOPS_Control_2 OOPS_Control_3 OOPS_Treated_4
## PROTEIN_1              2            2.1            1.2            3.5
##           OOPS_Treated_5 OOPS_Treated_6
## PROTEIN_1              4            4.1
```

```
oops_data_for_lm <- oops_data %>%
  data.frame() %>% # make into data.frame() so it can handled by gather()
  gather(key="sample", value="intensity") %>% # convert from wide to long with two columns, "sample" an
  separate(sample, into=c("type", "condition", "tag"), remove=FALSE) %>%
  mutate(condition=factor(condition, levels=c("Control", "Treated"))) # make "Control" the first condit

print(oops_data_for_lm)
```

```
##           sample type condition tag intensity
## 1 OOPS_Control_1 OOPS   Control   1       2.0
## 2 OOPS_Control_2 OOPS   Control   2       2.1
## 3 OOPS_Control_3 OOPS   Control   3       1.2
## 4 OOPS_Treated_4 OOPS   Treated   4       3.5
## 5 OOPS_Treated_5 OOPS   Treated   5       4.0
## 6 OOPS_Treated_6 OOPS   Treated   6       4.1
```

Now, we concatenate together the Total and OOPS data. This is possible since both dataframes have the same columns

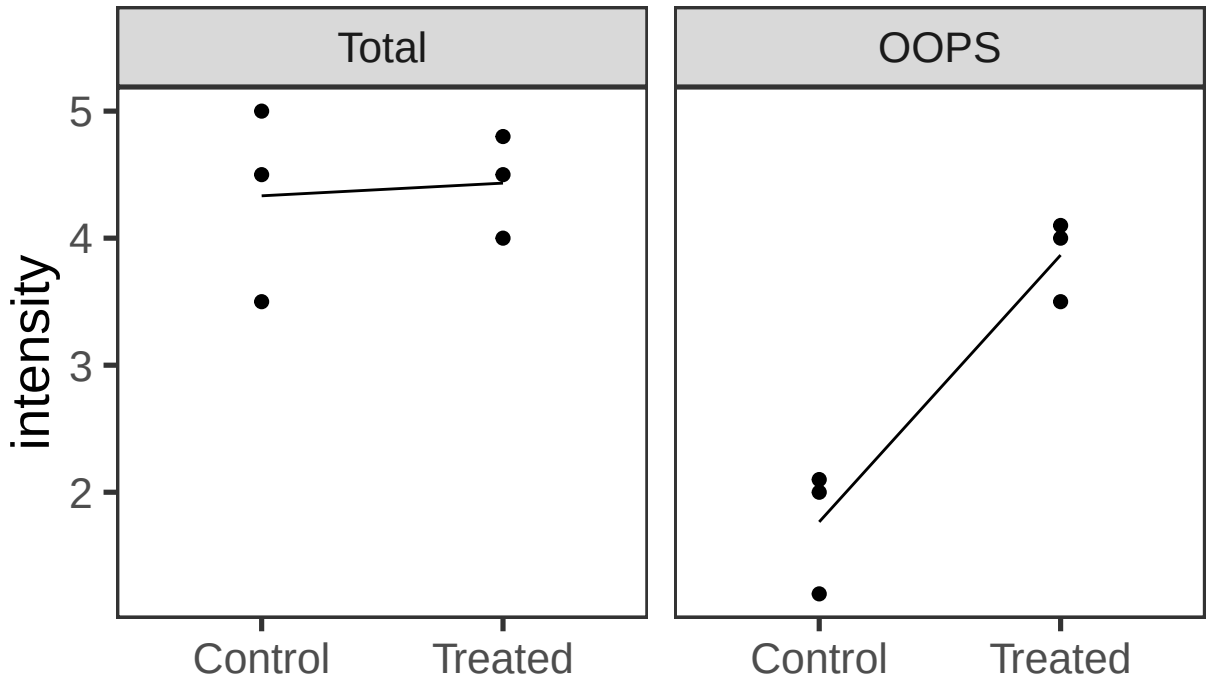
```
combined_data_for_lm <- rbind(total_data_for_lm, oops_data_for_lm) %>%
  mutate(type=factor(type, levels=c("Total", "OOPS"))) %>% # make "Total" the first data type
  mutate(tag=factor(tag, levels=1:6))

print(combined_data_for_lm)
```

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

Now, let's take a look at the intensities across our combined data.

```
combined_data_for_lm %>%
  ggplot(aes(condition, intensity, group=1)) +
  geom_point(size=2) +
  stat_summary(geom="line") +
  xlab("") +
  facet_wrap(~type)
```

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



As we can see above, there is a stronger increase in abundance between Control and Treated in the OOPS samples than in the Total samples.

Question: How would you interpret the intentisy 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 remember) for RNA binding

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

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

print(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
## F-statistic: 17.07 on 3 and 8 DF,  p-value: 0.0007752
```

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

The last thing to consider is that we have actually quantified Total and OOPS from the exact same samples (Organic phase and Interface). Therefore, we can include the sample as a "blocking" factor.

With the toy data, the sample number is indicated by the tag number as we have used the same tag for the Total and OOPS quantification.

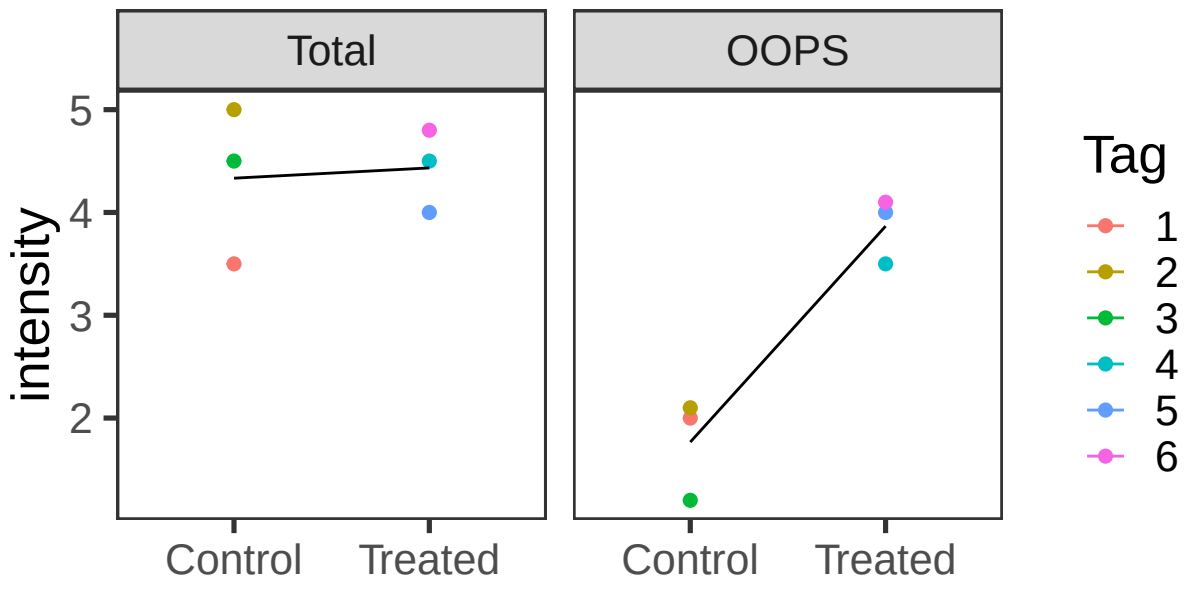
Note that we are not trying to model any systemic differences in tag labelling efficiency here (the data should already have been e.g center median normalised). Rather, we are interested in accounting for relative difference in the distribution of protein abundances within each sample. E.g if a protein was by chance more abundant in the first Control sample we would expect the intensity of the protein to be higher in Total_Control_1 vs Total_Control_2 or Total_Control_3. Importantly, since the OOPS samples originate from the same starting material, we would also expect OOPS_Control_1 to be higher than OOPS_Control_2 or OOPS_Control_3.

By and large, the more blocking factors in your experimental design, the easier it is to identify signficant effects for the biological factors of interest. For example, if possible, it's always better to pair the control/treated samples, especially if you expect the baseline (e.g Control) condition to be relatively variable.

So, first, let's highlight the data by tag to visualise this. Notice that the most abundant sample in Control is tag 2, for both Total and OOPS and the most abundant sample in Treated is tag 6 for both Total and OOPS. The rank isn't exactly the same but there does appear to be some relationship.

```
combined_data_for_lm %>%
  ggplot(aes(condition, intensity, group=1, colour=factor(tag))) +
  geom_point(size=2) +
  stat_summary(geom="line") +
  scale_colour_discrete(name="Tag") +
  xlab("") +
  facet_wrap(~type)
```

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



Now, let's fit a model including the tag as well...

The first thing we notice is that we have a lot more coefficients estimated now. Also, we can now longer estimate a coefficient for the `condition`. This is because some of the independent variables are now co-linear. Essentially, we're getting to the limit of the sensible number of indepedent variables we can include to model the protein intensity. The important thing to notice is that the interaction term is still signficant.

```
fit <- combined_data_for_lm %>%
  lm(formula=intensity ~ tag + condition + type + condition*type )

print(summary(fit))
```

```
##
## Call:
## lm(formula = intensity ~ tag + condition + type + condition *
##     type, data = .)
##
## Residuals:
##        1        2        3        4        5        6        7        8
## -0.53333  0.16667  0.36667  0.21667 -0.28333  0.06667  0.53333 -0.16667
##        9       10       11       12
## -0.36667 -0.21667  0.28333 -0.06667
##
## Coefficients: (1 not defined because of singularities)
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 4.0333     0.4391   9.186  0.00078 ***
```

```
## tag2                         0.8000     0.5377   1.488  0.21104
## tag3                         0.1000     0.5377   0.186  0.86152
## tag4                         0.2500     0.6209   0.403  0.70782
## tag5                         0.2500     0.6209   0.403  0.70782
## tag6                         0.7000     0.6209   1.127  0.32265
## conditionTreated                 NA         NA      NA       NA
## typeOOPS                    -2.5667     0.4391  -5.846  0.00427 **
## conditionTreated:typeOOPS    2.0000     0.6209   3.221  0.03225 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5377 on 4 degrees of freedom
## Multiple R-squared:  0.9285, Adjusted R-squared:  0.8034
## F-statistic: 7.422 on 7 and 4 DF,  p-value: 0.03564
```

The question then is which model is "better"

We can formally compare two models using the Akaike information criterion. AIC measures the "goodness of fit" but with penalises models which use too many parameters to reach the goodness of fit. This is similiar to the adjusted R-Squared above. Lower AIC indicates a "better" model.

So, for our full set of proteins, we will try both models (+/- tag) and take the one with the lowest AIC

```
testModels <- function(obj, coeff_of_interest="conditionTreated:typeOOPS"){

  fit1 <- obj %>% lm(formula=intensity ~ condition + type + condition*type + tag)

  fit2 <- obj %>% lm(formula=intensity ~ condition + type + condition*type)

  cat(sprintf("fit1 (with tag) has AIC of %s\n", AIC(fit1)))
  cat(sprintf("fit2 (without tag) has AIC of %s\n", AIC(fit2)))

  min_AIC <- min(AIC(fit1), AIC(fit2))
  max_AIC <- max(AIC(fit1), AIC(fit2))

  cat(sprintf("The best model is %s times more likely to minimise the information loss\n\n",
              round(1/exp((min_AIC-max_AIC)/2),3)))

  if ( AIC(fit1) < AIC(fit2) ) {
    chosen_fit <- fit1
    fit_name <- "With_tag"

  } else {
    chosen_fit <- fit2
    fit_name <- "Without_tag"
  }

  fit_values <- c(round(coef(summary(chosen_fit))[coeff_of_interest,], 4),
                  round(summary(chosen_fit)$adj.r.squared, 4),
                  fit_name)

  names(fit_values)[4:6] <- c("p_value", "adj_R_squared", "fit")

  return(fit_values)
}
```

```
combined_data_for_lm %>% testModels()
```

```
## fit1 (with tag) has AIC of 23.982152662944
## fit2 (without tag) has AIC of 23.6241847709107
## The best model is 1.196 times more likely to minimise the information loss
```

```
##      Estimate    Std. Error       t value       p_value adj_R_squared
##           "2"      "0.6037"      "3.3129"      "0.0107"      "0.8142"
##           fit
## "Without_tag"
```

So it turns out the model without the `tag` variable is a slightly model and we will therefore use the coefficient estimate (fold-change) and p-value from this model.

Now we've satisfactorily modeled the intensity for a single protein, let's try this with a full data set (see `2_identify_changes_in_RNA_binding.Rmd` notebook)