Quantitative proteomics data analysis

Tom S. Smith, Charlotte Dawson, and Oliver M. Crook

Contents

Learning Outcomes

1. Be able to aggregate peptide level data to obtain protein-level quantification.
2. Be able to normalise proteomics data and understand why we perform normalisation.
3. Perform and understand statistical tests to identify changes in protein abundances.
4. Perform a functional analysis to identify the perturbed biological processes.

---

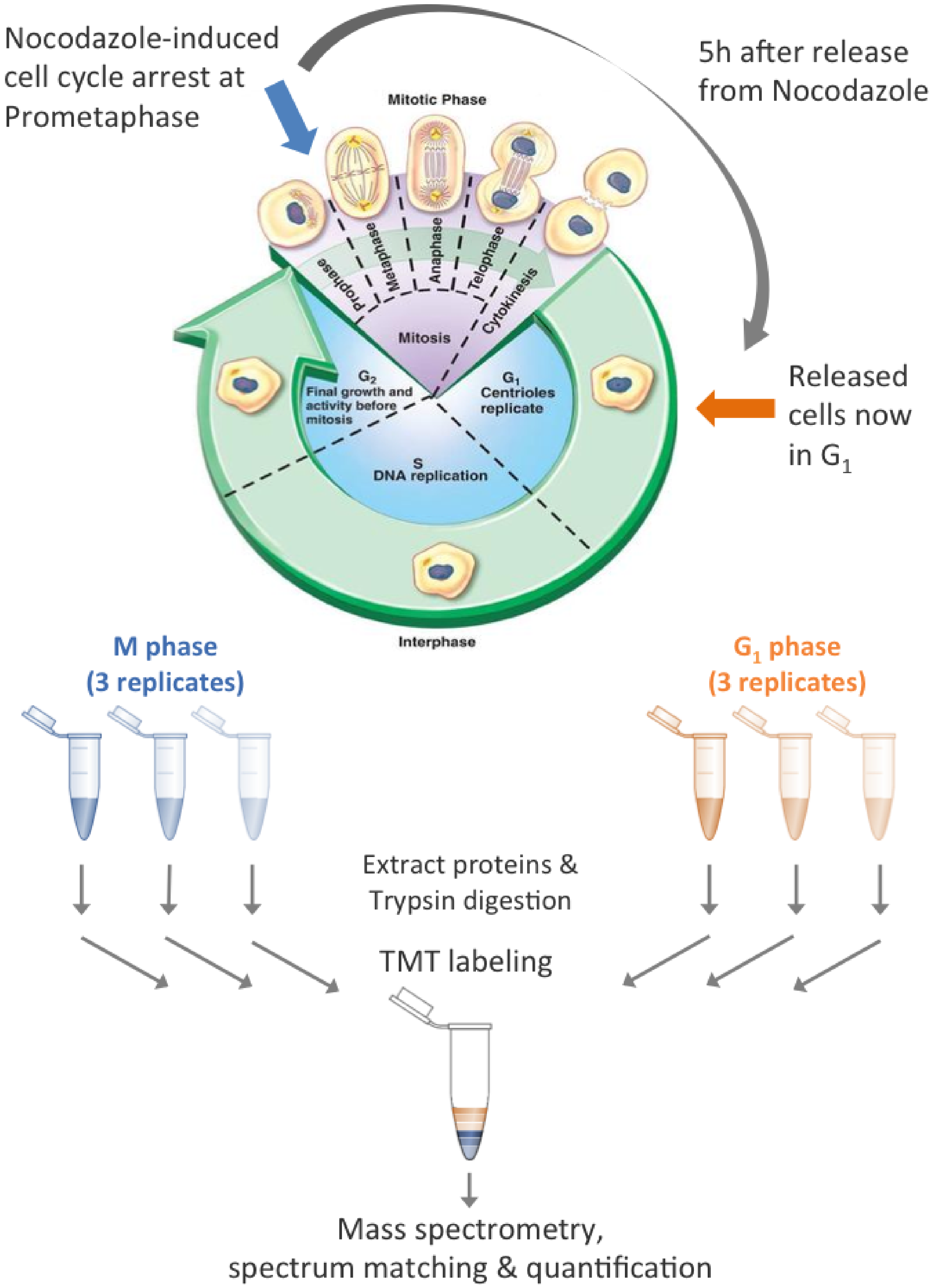
1. Introduction

This practical analyses data from an experiment to examine proteins involved in the cell cycle (See diagram below). Cells from a human cell line are treated with Nocodazole to inhibit polymerisation of microtubules and block transition beyond the prometaphase stage of M-phase.

Cells are then released from M-phase by withdrawing Nocodazole. Samples are taken in triplicate at 0 hours and at 5 hours after release from M-phase. This process generates 3 x M-phase and 3 x G1-phase samples from 6 separate batches of cells.

To measure protein abundances, proteins are first digested to peptides with trypsin. The same quantity of peptides from each sample was then labelled using Tandem Mass Tags (TMT). This allows all the samples to be combined into a single MS run to yield quantification over the same peptides in each sample.

The peptide spectrum matching and quantification steps have already been completed and the peptide-level quantification is provided to you.


Nocodazole-induced cell cycle arrest at Prometaphase
5h after release from Nocodazole
Mitotic Phase
Prophase
Metaphase
Anaphase
Telophase
Cytokinesis
Mitosis
G_2 Final growth and activity before mitosis
G_1 Centrioles replicate
S DNA replication
Interphase
Released cells now in G_1
M phase (3 replicates)
G_1 phase (3 replicates)
Extract proteins & Trypsin digestion
TMT labeling
Mass spectrometry, spectrum matching & quantification

Check your knowledge

1. Why does inhibiting polymerisation of microtubles block transition beyond prometaphase?
2. What else might inhibiting polymerisation of microtubules do to the cells?
3. Why do we take experimental triplicates?
4. Why do we digest using trypsin? At which residues does trypsin cleave?
5. What is a tandem mass tag?
6. Why do we process all samples in the same MS run?
7. What is peptide spectrum matching?

First, we load the required libraries. If you get a message saying `Error in library(sds) : there is no package called ...`, temporarily uncomment the necessary lines to install the package.

```
#install.packages("tidyr")
#install.packages("dplyr")
#install.packages("ggplot2")

library(tidyr)
library(dplyr, warn.conflicts = FALSE)
library(ggplot2)

options(width = 60, dplyr.summarise.inform = FALSE)
```

2. Preparing the data

2a. Loading and reading in the peptide-level quantification

Usually your data is provided to you in some format such as a text file, excel file or some other format. We would like to load this data into R so we can analyse it.

For the purposes of this practical, peptides which come from proteins in the cRAP database of common proteomics contaminants, e.g keratin, have been removed. Peptides which cannot be uniquely assigned to a single protein have also been removed.

The data is provided here in a tab-separated text file `"data/peptide_data.tsv"`, which is located in the repository. We start by reading the peptide-level quantification data into a `data.frame`. The code chunk below loads the data from the .tsv file into an R data frame called `peptides_quant`.

```
peptides_quant <- read.table("data/peptide_data.tsv", sep = "\t", header = TRUE)
```

If we take a look at the column names (R function `colnames()`) of the `peptides_quant` data frame, we can see we have 9 columns. The first 3 columns describe the sequence of the peptide, the modifications on the peptide which were detected and the protein which the peptide has been assigned to. The next 6 columns provide the quantification values for each peptide across the 3 M-phase and 3 G1-phase samples.

```
print(colnames(peptides_quant))
```

```
## [1] "Sequence"       "Modifications"  "master_protein"
## [4] "M_1"            "M_2"            "M_3"
## [7] "G1_1"           "G1_2"           "G1_3"
```

```
print(dim(peptides_quant))
```

```
## [1] 20065     9
```

```
print(head(peptides_quant, 5))
```

```
##          Sequence
## 1     EESTSSGNVSNR
## 2   TSNERPGSGQGQGR
## 3 AQHSNAAQTQTGEANR
## 4     QQQPPGAQGCPR
## 5   TVSTSSQPEENVDR
##                                Modifications
## 1                          1xTMT6plex [N-Term]
## 2                          1xTMT6plex [N-Term]
## 3                          1xTMT6plex [N-Term]
## 4 1xTMT6plex [N-Term]; 1xCarbamidomethyl [C10]
## 5                          1xTMT6plex [N-Term]
##   master_protein M_1 M_2 M_3 G1_1 G1_2 G1_3
## 1         P23193 1.4  NA  NA   NA   NA   NA
## 2         Q8TAD8 2.7  NA  NA  3.5  1.6  3.5
## 3         P34897 1.7  NA 1.7  2.0  1.6  2.0
## 4         A8MYA2 6.8 2.1 7.6  8.5  6.3  8.5
## 5         P42684 2.7  NA 6.2  2.7  1.0  2.7
```

You'll notice that some of the quantification values of the peptides are `NA`.

We can also see that some of the peptides have the same sequence but different modifications. Below, we inspect one such peptide sequence.

```
filter(peptides_quant, Sequence == 'AAAEELQEAAGAGDGATENGVQPPK')
```

```
##                    Sequence
## 1 AAAEELQEAAGAGDGATENGVQPPK
## 2 AAAEELQEAAGAGDGATENGVQPPK
##                                Modifications
## 1 2xTMT6plex [N-Term; K25]; 2xDeamidated [Q/N]
## 2 2xTMT6plex [N-Term; K25]; 1xDeamidated [N19]
##   master_protein  M_1  M_2  M_3 G1_1 G1_2 G1_3
## 1         Q9H9Y2 30.5 21.2 32.6 27.1 29.7 27.1
## 2         Q9H9Y2 23.6 12.3 27.0 30.3 25.4 30.3
```

Check your knowledge

1. Where can I find keratin in daily life?

2. Why is keratin a contaminant?
3. Why do we worry about contaminants?
4. What does if mean to say a peptide is 'assigned' to a protein. Why could a peptide be assigned to multiple proteins?
5. How many peptides were quantified in this experiment?
6. What do you think `NA` means in this context?
7. Why do you think some of the quantification values are `NA`?
8. What do you think we should do with `NA`?
9. Why do peptide 'modifications' represent?
10. Why might we have multiple quantification values for the same peptide but with different modifications?

2b. `magrittr` pipes

The `magrittr` package introduced 'pipe' functions to R. Here, we will be using the `%>%` pipe a few times so the next two cells demonstrates how this works in case you haven't come across it before as the syntax can look a little odd. Feel free to skip these cells if you're confident you understand `%>%`.

The `%>%` pipe allows you to use the output from one function to be the input for the next function, much like the `|` pipe in bash. The main advantage of this is to make the code easier to understand. The pipe can be used to pass any object. However, to keep things simple here, for the example below, we will just consider a simple vector of values, from which we wish to identify the mean of the log2-transformed values.

```
values <- c(1, 4, 100, 21, 34)

# we can wrap the mean and log functions together, but this can get ugly,
# especially if there are lots of functions and arguments.
mean(log(values, base = 2))
```

```
## [1] 3.624727
```

```
# with magrittr pipes, we can see the 'flow' from:
values %>%  # the values vector, through
  log(base = 2) %>%  # the log(base = 2) transformation, and finally,
  mean()  # the mean summarisation
```

```
## [1] 3.624727
```

Magrittr pipes become especially useful for piping together complex functions when re-formating, summarising and plotting data. They also make it much easier to document each step. If you're wondering how the package got its name, see The_Treachery_of_Images.

With that aside done, let's return to the practical.

2c. Aggregating the peptide quantification data to generate protein-level quantification

We can aggregate the quantification values across the multiple instances of the same peptide with difference modifications using the `dplyr` package.

```
peptides_quant_no_mods <- peptides_quant %>%
  select(-Modifications) %>% # exclude the Modifications column
  group_by(Sequence, master_protein) %>% # group by the Sequence and master_protein columns
  summarise(across(everything(), sum)) %>%  # aggregate the quant values using the sum function
  ungroup()

print(head(peptides_quant_no_mods))
```

```
## # A tibble: 6 x 8
##   Sequence      master_protein   M_1   M_2   M_3  G1_1  G1_2
##   <chr>         <chr>          <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 AAAAAAAK      Q9NYV4          92.2  38.3  66.4  78.9  61.6
## 2 AAAAAAALQAK   P36578         372.  182.  384.  394.  315.
## 3 AAAAASAAGPGG~ A6NIH7          35.9  19    36.3  38.8  36.6
## 4 AAAAAWEEPSSG~ Q9P258         103.   46.1  81   111.  109.
## 5 AAAACLDK      Q9H3U1          43.9  18.5  36.1  43.2  39.9
## 6 AAAAGSLDR     Q9Y2U8          59.1  38.3  61.1  97.3  79.9
## # ... with 1 more variable: G1_3 <dbl>
```

2d. Dealing with missing values

We still have peptides with missing values.

```
missing_values_pre_agg <- peptides_quant  %>%
  select(where(is.numeric)) %>%  # select only quantification columns
  is.na() %>% # get a matrix of booleans (TRUE, FALSE), where TRUE = value is NA
  sum() # TRUE=1, FALSE=0 so sum of booleans is the number of TRUEs i.e. number of NA

missing_values_post_agg <- peptides_quant_no_mods %>%
  select(where(is.numeric)) %>%
  is.na() %>%
  sum()

message(
  sprintf("Before aggregation, we had %i peptides with missing values.
After aggregation we have %i peptides with missing values.",
missing_values_pre_agg, missing_values_post_agg)
)
```

```
## Before aggregation, we had 138 peptides with missing values.
## After aggregation we have 137 peptides with missing values.
```

We could impute the missing values but this is beyond the scope of this practical. For our purposes, we will remove any peptide with a missing value.

```
peptides_quant_no_mods_no_na <- peptides_quant_no_mods %>%
  drop_na() # removes any row with at least 1 NA
```

Check your knowledge

1. Why should we remove `NA`?
2. Are there biological reasons for `NA`?
3. What imputation strategies exist?

2e. Normalisation: What is it and why do we do it?

Before we proceed with the aggregation to protein-level quantification, we should consider whether we need to perform any normalisation to the data. The total peptide abundance in each column should be approximately the same. However, we can see below that this is not the case.

Below, we obtain the sum of peptide abundances in each sample (column). Note these are relatively similar but M_2 is ~2-fold less than M_1 or G1_1.

```
pep_tots <- peptides_quant_no_mods_no_na %>%
  select(where(is.numeric)) %>% # select only quantification columns
  colSums() # get the sum of values in each column

# formatC(format='e') formats numeric values into scientific notation.
# See ?formatC for more info
formatC(pep_tots, format = 'e', digits = 1)
```

```
##       M_1       M_2       M_3      G1_1      G1_2      G1_3
## "1.9e+06" "9.9e+05" "1.7e+06" "1.9e+06" "1.6e+06" "1.9e+06"
```

Check your knowledge

1. Why should the total peptide abundance be approximately the same (see experimental design)?
2. What could cause it to be different?
3. What impact could this have on any downstream statistical analysis?

Next, we normalise the peptide-level quantification to adjust for the differences in the total peptide abundance.

```
# make a correction factor using the total peptide abundance
pep_tots_correction_factors <- pep_tots / mean(pep_tots)

pep_tots_correction_factors
```

```
##       M_1       M_2       M_3      G1_1      G1_2      G1_3
## 1.1413016 0.5941574 0.9976013 1.1496041 0.9677315 1.1496041
```

```
# create a copy of data pre-normalisation
peptides_quant_norm <- peptides_quant_no_mods_no_na

# normalise by dividing each numeric column by the corresponding correction factor
quant_norm <- peptides_quant_no_mods_no_na %>%
```

```r
  select(where(is.numeric)) %>% # select only quantification columns
  as.matrix() %>% # sweep() function only works on matrix
  sweep(MARGIN = 2, STATS = pep_tots_correction_factors, FUN = '/') # perform normalisation

# replace pre-normalisation data with post-normalisation data
peptides_quant_norm[colnames(quant_norm)] <- quant_norm
```

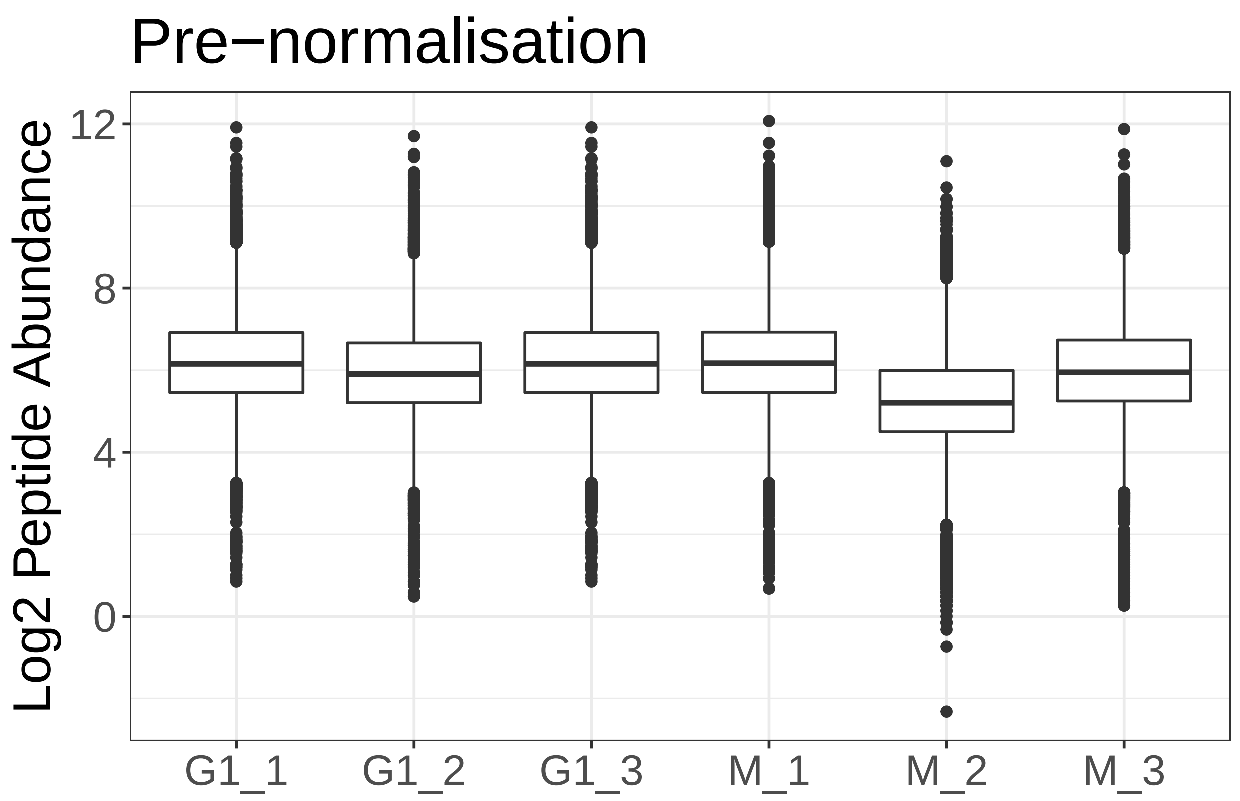
We can check that the total are now the same:

```r
norm_pep_tots <- peptides_quant_norm %>%
  select(where(is.numeric)) %>%
  colSums()

formatC(norm_pep_tots, format = 'e', digits = 1)
```

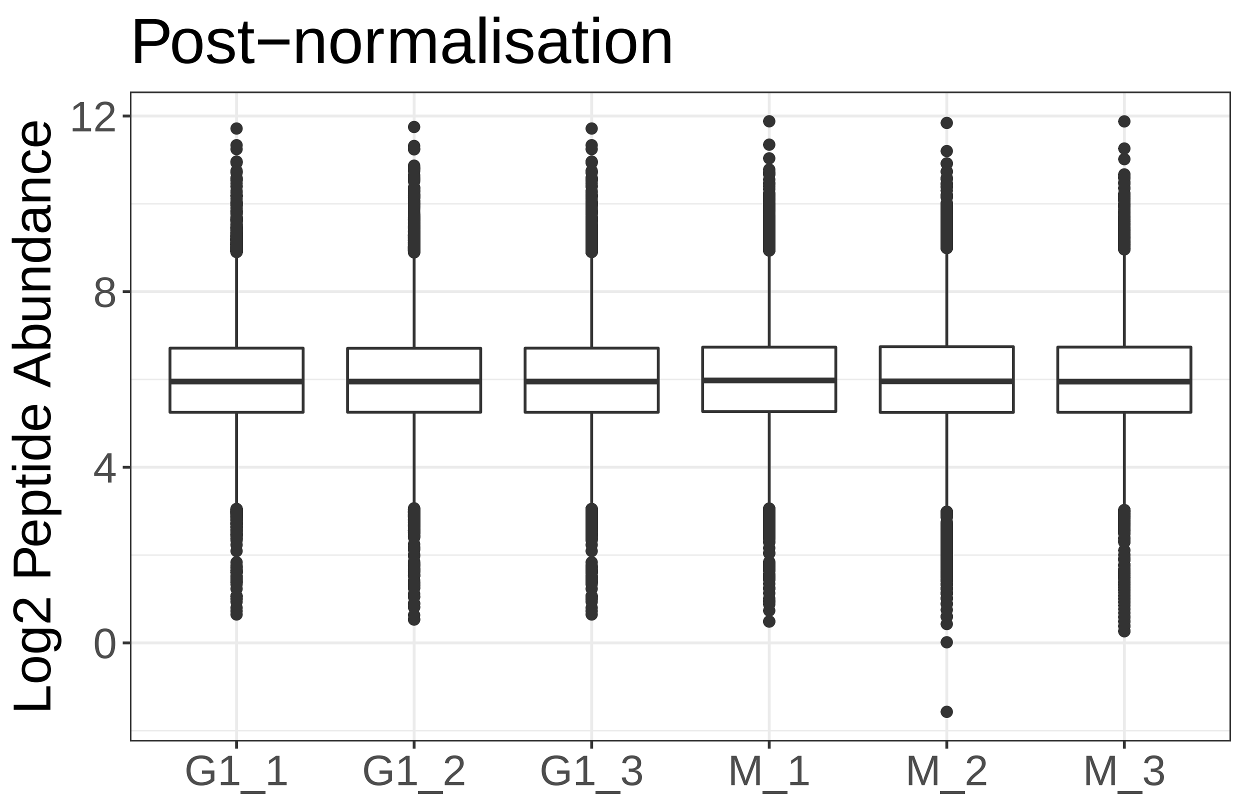
```
##       M_1       M_2       M_3      G1_1      G1_2      G1_3
## "1.7e+06" "1.7e+06" "1.7e+06" "1.7e+06" "1.7e+06" "1.7e+06"
```

We plot the distribution of abundance values before and after normalisation allowing us to see the impact of normalisation. Here we create a small function to perform the plotting to save space from copying code. Note that we are plotting the log of the abundance values. This is because the abundance values extend across 4-orders of magnitude and are not Gaussian (normally) distributed. However, they are approximately log-Gaussian distributed.

```r
plotPeptideAbundance <- function(peptide_data, title = "") {

  p <- peptide_data %>%
    pivot_longer(cols = -c(Sequence, master_protein), # reshape data into 'long' format for ggplot
                 names_to = 'sample', values_to = 'abundance') %>% # set names and values columns
    ggplot(aes(x = sample, y = log(abundance, 2))) +
    geom_boxplot() +
    theme_bw() +
    theme(text = element_text(size = 20)) +
    labs(
      x = "",
      y = "Log2 Peptide Abundance",
      title = title
    )

  print(p)
}


plotPeptideAbundance(peptides_quant_no_mods_no_na, "Pre-normalisation")
```



```
plotPeptideAbundance(peptides_quant_norm, "Post-normalisation")
```



Now we have a single quantification value for each peptide, we can aggregate across the different peptides for the same protein to obtain the protein-level quantification. Here we are using the median.

Check your knowledge

1. Why do you think we are using the median to aggregate across the different peptides?
2. What other mathematical operations could we use and what are the relative merits?
3. What information do we lose by aggregating peptides into peptides?

```
# aggregate peptide quantification values from same protein using median function
protein_quant <- peptides_quant_norm %>%
  group_by(master_protein) %>% # group by master_protein column
  summarise(across(-Sequence, median)) # drop Sequence column and aggregate by median

head(protein_quant)
```

```
## # A tibble: 6 x 7
##   master_protein   M_1   M_2   M_3  G1_1  G1_2  G1_3
##   <chr>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 A0AVT1          75.2  66.5  70.2  76.7  79.0  76.7
## 2 A0MZ66          52.4  53.0  52.1  45.8  49.2  45.8
## 3 A1L390          51.4  51.5  54.3  49.4  57.7  49.4
## 4 A1X283          43.3  42.5  49.4  45.5  45.3  45.5
## 5 A2RTX5          35.0  36.2  33.0  32.4  38.0  32.4
## 6 A3KN83          13.5  15.3  14.9  20.6  18.6  20.6
```

---

3. Performing a statistical test to identify changes in protein abundance

We now want to identify the proteins with significant changes in abundance between M and G1 phases. For the purposes of this practical, we consider hypothesis-based statistics. Note there are other schools of statistics and they might have different tests.

Formally, we state our null hypothesis:

(H0) The change in abundance for a protein is zero.

We want to test this against the alternative hypothesis:

(H1) The change is greater than zero.

For each protein, we conduct an independent statistical test. The important points to consider when choosing the appropriate statistical test are:

- We have independent samples from two populations and wish to test whether the difference in the mean of the two populations is non-zero.
- The samples are not paired, e.g M-phase replicate 1 and G1-phase replicate 1 are not related samples.
- We want to identify changes in both directions.
- The protein abundances are not Gaussian (normal) distributed. However, they are approximately log-Gaussian distributed.
- The cell cycle stage is **not** expected to have a large impact on the biological variability.

This is a very simple experimental design and we recommend that you use a Student's t-test for this practical. However, there are other statistical tests which would also be suitable. For example, if one was concerned about whether the protein abundances are truly approximately Gaussian, a non-parametric test such as the Mann-Whitney U test or Wilcoxon rank-sum tests may be used depending on the exact experimental design. Alternatively, if the experimental design was more complicated and we wished to take into account confounding factors, or investigate interactions between factors, we could use ANOVA or Linear Regression. Choosing the correct test must be based on the assumptions of your experimental design; choosing a test so that you see the results you want is academic fraud.

Check your knowledge

1. Should the t-test be one-tailed or two-tailed?
2. Should you perform a paired t-test?
3. A two-sample equal variance t-test or a two-sample unequal variance t-test?
4. Do you need to perform any transformation on the data before conducting the t-test?
5. What are the assumptions of the t-test and do these assumptions hold for these data?
6. How low does the p-value have to be before you reject H0 (what's your value for alpha)?
7. What effect size do you think would be biologically relevant?
8. Would a 2-fold increase/decrease in abundance be relevant?

9. What about a 0.5-fold or a 0.1-fold increase/decrease?

We perform a log-transform on the data before conducting the t-test. We use `log2()` since it's intuitive to have a single increase by a point in the `log2()` scale indicate a doubling and a two point increase indicate a 4-fold increase.

```
# log2-transform the quantification values
protein_quant_log <- protein_quant %>%
  mutate(across(where(is.numeric), log2))
```

In order to perform the t-test on each row, we create a function which performs a t-test on a vector of six values (where 1-3 = group 1, 4-6 = group 2) and returns the p-value, the difference between the group means, and the 95% confidence interval for the difference.

```
run_single_t_test <- function(quant_values) {

  test_results <- t.test(x = quant_values[4:6],
                         y = quant_values[1:3],
                         paired = FALSE,
                         alternative = "two.sided",
                         conf.int = TRUE,
                         var.equal = TRUE)

  p.value <- test_results$p.value # extract the p-value

  # extract difference between the groups
  difference <- as.numeric(test_results$estimate[1] - test_results$estimate[2])

  ci <- as.numeric(test_results$conf.int) # extract the 95% CI

  return(c("p.value" = p.value, "difference" = difference,
           "CI_diff_low" = ci[1], "CI_diff_high" = ci[2]))
}
```

We now `apply` this function over the rows and reformat the output.

```
# apply run_single_t_test() over each row (MARGIN = 1) for quantification columns
t_test_results <- protein_quant_log %>%
  select(where(is.numeric)) %>% # select only quantification columns
  apply(MARGIN = 1, run_single_t_test) %>% # apply our function
  t() %>%  # transpose output to get correct layout
  cbind(protein_quant_log) # attach results to protein_quant_log data.frame

print(head(t_test_results))
```

```
##       p.value   difference CI_diff_low CI_diff_high
## 1 0.063328583  0.135689357 -0.01206948   0.28344819
## 2 0.009856586 -0.162537648 -0.26014674  -0.06492855
## 3 0.911365390 -0.009368493 -0.22882502   0.21008803
## 4 0.831304335  0.015486687 -0.17364597   0.20461935
## 5 0.828215265 -0.019858827 -0.25793018   0.21821253
## 6 0.003743387  0.452256458  0.24505087   0.65946204
##   master_protein      M_1      M_2      M_3     G1_1
## 1         A0AVT1 6.232226 6.054864 6.132748 6.261570
## 2         A0MZ66 5.711394 5.728363 5.702517 5.517225
## 3         A1L390 5.683379 5.686543 5.762354 5.626682
## 4         A1X283 5.435759 5.409294 5.625517 5.507602
## 5         A2RTX5 5.127637 5.177348 5.043480 5.019967
## 6         A3KN83 3.754178 3.936949 3.900705 4.365678
##       G1_2     G1_3
## 1 6.303766 6.261570
## 2 5.620211 5.517225
## 3 5.850807 5.626682
## 4 5.501826 5.507602
## 5 5.248955 5.019967
## 6 4.217246 4.365678
```

3a. Accounting for multiple hypothesis test

Check your knowledge

1. For your chosen level of alpha, how many proteins would you expect to have a significant change in abundance by chance (false positives; Type I errors) ?

The problem of multiple comparisons is the following. Suppose, I have two groups and I make 100 tests between these two groups. Let us assume I make these tests at the 5% level (alpha = 0.05) and then assume all my 100 test corresponded to support of the null hypothesis. In expectation ("on average"), the number of incorrect rejections would be 5.

Since we have conducted multiple tests, we get many false positives if we use the uncorrected p-values. Here are the two most popular options to deal with the multiple testing. Note that multiple testing is still an active research area.

1. Control the Family-Wise Error Rate (FWER) so that the probability of getting even a single false positive equals our initial alpha.
2. Control the False Discovery Rate (FDR) so that the percentage of false positives among the null hypotheses rejected is no greater than a chosen value.

The first approach is underpowered, since we are trying to avoid identifying even a single false positive across all the tests conducted. This stringently avoids false positives (type I errors), but leads to many false negatives (type II errors). The second approach is powerful, since we allow a certain percentage of our rejected null hypothesis to be incorrect. Thus, the number of type I errors increases, but the number of type II errors also decreases. The approach to take depends on the application.

Here, the downstream analysis focus on groups of proteins with shared functionality, rather than specific proteins which have altered abundance. Therefore, we can accept a low percentage of false positives. We calculate the FDR using the Benjamini-Hochberg method via the `p.adjust` function and reject the null hypothesis where the FDR < 0.01 (1%). This means approximately 1% of the rejected null hypothesis are false positives but we do not know which ones these are.

```
FDR_threshold <- 0.01

t_test_results <- t_test_results %>%
  mutate(FDR = p.adjust(p.value, method = "BH"),
         sig = FDR < FDR_threshold,
         sig_2fold = FDR < FDR_threshold & difference > 1)
```

We can summarise how many proteins have a significant change in abundance using the `table` function.

```
table(t_test_results$sig)
```

```
##
## FALSE  TRUE
##  2731   156
```

Check your knowledge

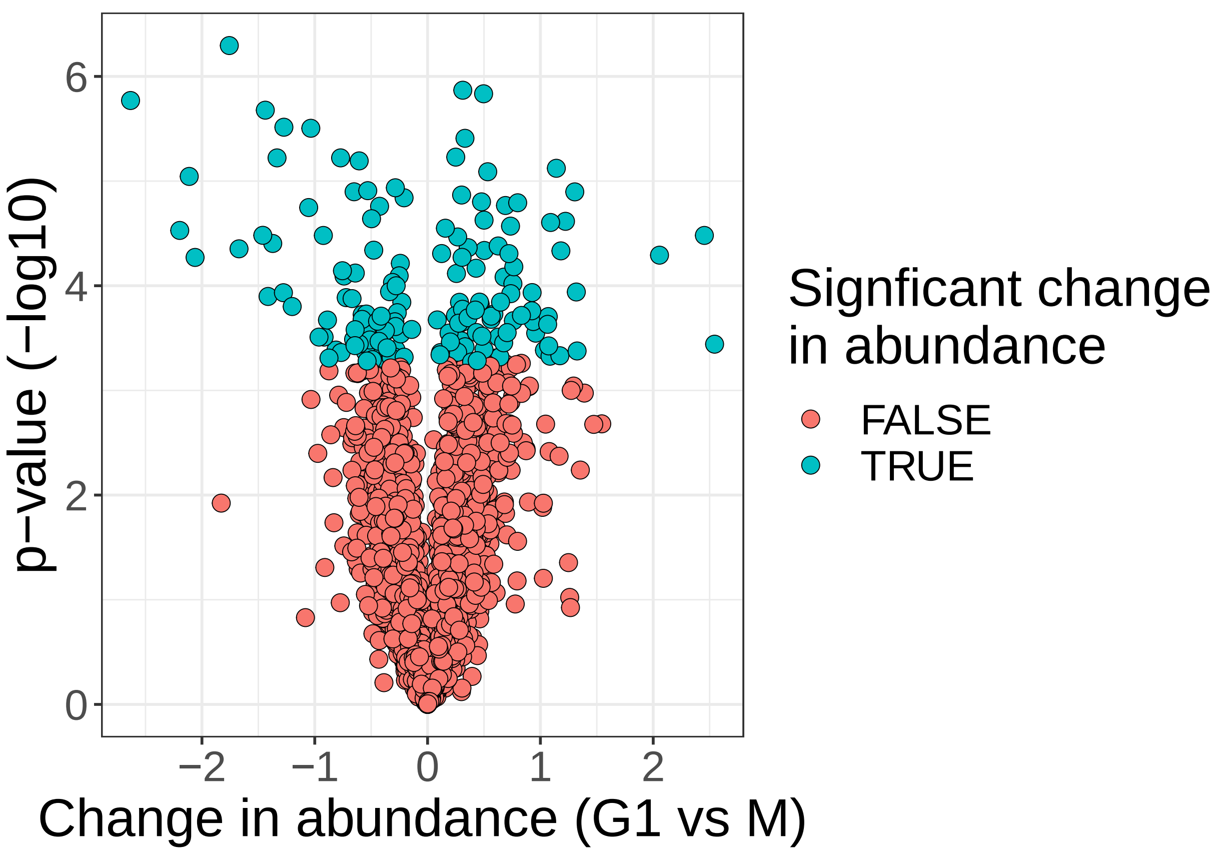
1. Can you add another command in the cell below to work out how many of the proteins have a significant increase in abundance and how many decrease?

We visualise the t-test results in a so-called “Volcano” plot (see below). In the second plot we add the 95% confidence intervals for the change in abundance. Many of the significant changes are small. In fact, just 16/156 of the significant changes in abundance are greater than 2-fold (Remember that we have log base 2 transformed the abundance data so a change in abundance of 1 on the log scale is a 2-fold change).

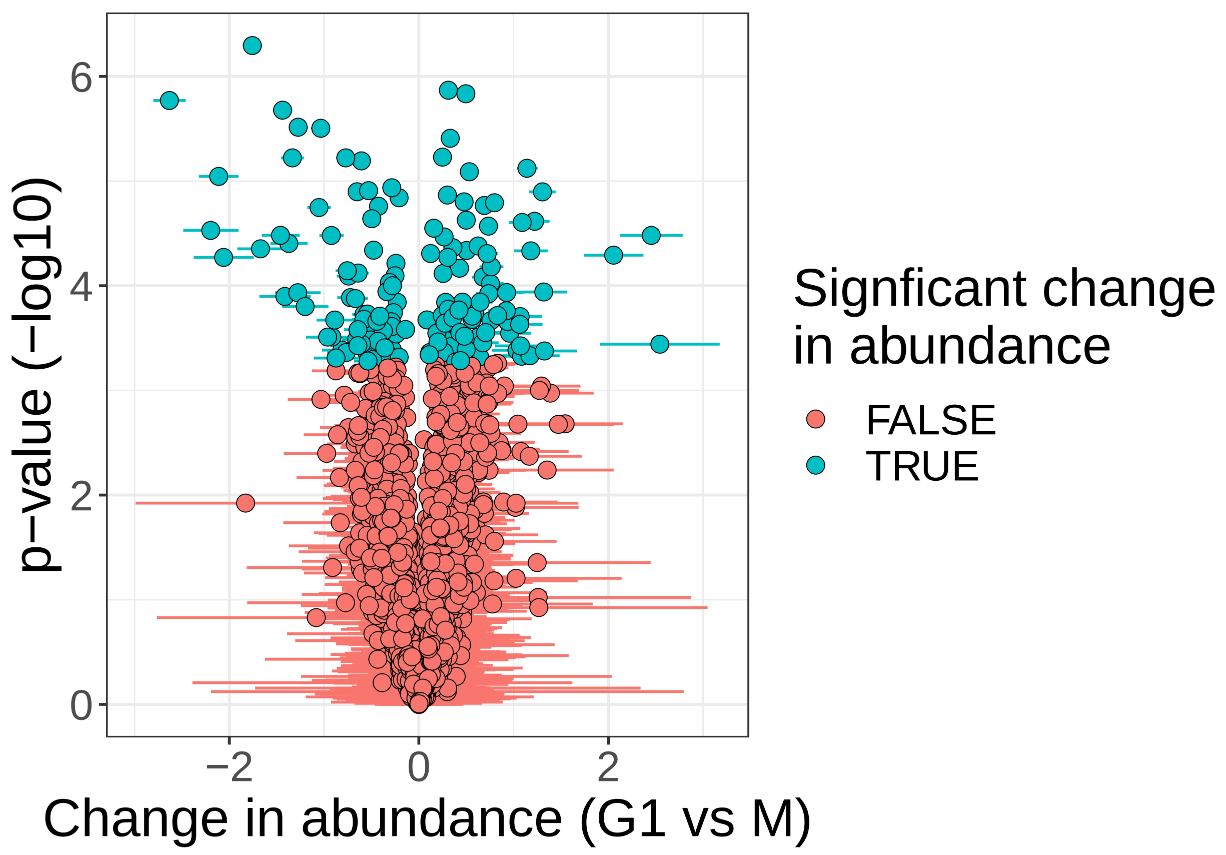
```
message(
  sprintf("%i proteins show significant changes in abundance.
Only %i of these proteins show a change greater than 2-fold.",
sum(t_test_results$sig), sum(t_test_results$sig_2fold))
)
```

```
## 156 proteins show significant changes in abundance.
## Only 16 of these proteins show a change greater than 2-fold.
```

```
p <- ggplot(t_test_results, aes(x = difference, y = -log(p.value, 10), fill = sig)) +
  scale_fill_discrete(name="Signficant change\nin abundance") +
  theme_bw() +
  theme(text = element_text(size = 20)) +
  labs(
    x = "Change in abundance (G1 vs M)",
    y = "p-value (-log10)"
  )

print(p + geom_point(shape = 21, stroke = 0.25, size = 3))
```



```r
p2 <- p +
  geom_errorbarh(aes(xmin = CI_diff_low, xmax = CI_diff_high, colour = sig)) +
  scale_color_discrete(guide = "none")

print(p2 + geom_point(shape = 21, stroke = 0.25, size = 3))
```



4. Functional analysis of proteins with differential abundance

4a. What are the proteins?

Now that we've identified the proteins that have a different abundance in G1-phase vs. M-phase, the next step is to investigate the functions for these proteins. First, let's add the protein names and descriptions (from another data file).

```
human_proteins_ids_df <- read.csv(
  "data/human_protein_ids.tsv", sep = "\t", header = FALSE,
  colClasses = c("character", "character", "character", "NULL", "NULL"),
  col.names = c("UniprotID", "Name", "Description", NA, NA)
)

t_test_results_annot <- merge(human_proteins_ids_df,
                              t_test_results,
                              by.x = "UniprotID",
                              by.y = "master_protein",
                              all.y = TRUE)

# we'll write the results out to file so they can be
# further interrogated outside this notebook
write.table(t_test_results_annot, './data/t_test_results_annotated.tsv',
            row.names = FALSE, sep = '\t')
```

We can inspect the descriptions for the proteins which have a significant decrease in abundance in G1 vs M.

```
t_test_results_annot %>%
  filter(sig == TRUE, # significant changes
         difference < 0) %>% # decrease in abundance
  arrange(p.value) %>% # sort by p-value
  select(Name, Description, p.value, FDR) %>% # select columns of interest
  head(20)
```

```
##           Name
## 1  CENPE_HUMAN
## 2   SGO2_HUMAN
## 3  AURKA_HUMAN
## 4   PRC1_HUMAN
## 5  BOREA_HUMAN
## 6  AKTS1_HUMAN
## 7   KITH_HUMAN
## 8  KIF11_HUMAN
## 9   ANLN_HUMAN
## 10  RT31_HUMAN
## 11 HACD2_HUMAN
## 12   VIR_HUMAN
## 13  ETFB_HUMAN
## 14  RT29_HUMAN
## 15 KIF22_HUMAN
## 16  PREB_HUMAN
## 17 KIFC1_HUMAN
## 18  TPX2_HUMAN
## 19 COCA1_HUMAN
## 20  PLK1_HUMAN
##                                          Description
## 1                      Centromere-associated protein E
## 2                                          Shugoshin 2
## 3                                      Aurora kinase A
## 4                   Protein regulator of cytokinesis 1
## 5                                             Borealin
## 6                        Proline-rich AKT1 substrate 1
## 7                          Thymidine kinase, cytosolic
## 8                          Kinesin-like protein KIF11
## 9                                              Anillin
## 10               28S ribosomal protein S31, mitochondrial
## 11 Very-long-chain (3R)-3-hydroxyacyl-CoA dehydratase 2
## 12                           Protein virilizer homolog
## 13          Electron transfer flavoprotein subunit beta
## 14             28S ribosomal protein S29, mitochondrial
## 15                          Kinesin-like protein KIF22
## 16         Prolactin regulatory element-binding protein
## 17                          Kinesin-like protein KIFC1
## 18                         Targeting protein for Xklp2
## 19                         Collagen alpha-1(XII) chain
## 20                Serine/threonine-protein kinase PLK1
##         p.value         FDR
## 1  5.071636e-07 0.001213524
## 2  1.699184e-06 0.001213524
## 3  2.101705e-06 0.001213524
## 4  3.057858e-06 0.001290723
## 5  3.129568e-06 0.001290723
## 6  6.001327e-06 0.001543256
## 7  6.003272e-06 0.001543256
## 8  6.414643e-06 0.001543256
## 9  9.024699e-06 0.001736954
## 10 1.158814e-05 0.001926388
## 11 1.235089e-05 0.001926388
## 12 1.265304e-05 0.001926388
## 13 1.444128e-05 0.001985333
## 14 1.741950e-05 0.001987746
## 15 1.790142e-05 0.001987746
## 16 2.290124e-05 0.002392901
## 17 2.961931e-05 0.002591241
## 18 3.299144e-05 0.002652798
## 19 3.307957e-05 0.002652798
## 20 3.946135e-05 0.002998024
```

Can you see what functions some of these proteins might have based on their descriptions?

As with most such analyses there are likely be some proteins you have heard of, but many more which you haven't. After all, there are ~20,000 protein-coding genes in the human genome! It's also hard looking at the proteins which have changed to understand what shared functional pathways they may have in common. Many of the protein's functions may not be obvious just from their description. Likewise, the sub-cellular localisation may not be obvious, and it could be that most of the proteins with altered abundance have the same localisation which would be an interesting observation.

Even if you were a fountain of knowledge on human proteins, it's difficult by eye to know if an observation is unexpected. For example, if you see 4 kinases which are all more abundant in G1, is this a relevant observation? To answer this, you would need to know how many kinases *could* have changed abundance in your experiment, from which you can estimate how many kinases you would expect to see *by chance*, given the number of proteins with an altered abundance. In short, just looking at the protein descriptions and using your prior knowledge only gets you so far.

4b. Identifying over-representation of particular protein functions/localisations

Thankfully, the Gene Ontology (GO) consortium have defined a set of descriptions for genes and their protein products, split into 3 categories: Molecular Functions (MF), Biological Processes (BP) and Cellular Components (CC). These terms are hierarchical. For example as shown here, `RNA binding` is a child term of `Nucleic acid binding` and a parent term of e.g `RNA cap binding`.

To understand the biological relevance of the protein abundance changes between M and G1-phase, we can look for GO terms which are over-represented in the proteins with a significant change. A GO over-representation analysis involves asking whether a GO term is more frequent in a selected set of proteins (the `foreground`) relative to all the proteins which could have been selected (the `background`). As with any over-representation analysis, it's important to consider what the `foreground` and `background` should be.

In this case, we could either perform:

1. One test where the foreground is all proteins with a significant change in abundance, or
2. Two tests, one where the foreground is all proteins with a significant increase in abundance, and the other is proteins with a significant decrease in abundance.

Check your knowledge

1. What do you the most suitable background would be for the GO term over-representation analysis?
2. Do you think it makes more sense to perform one or two tests for the GO term over-representation analysis?

To simplify this practical, we use GORILLA to perform the over-representation analysis. But note there are many better tools available and one should always be wary of confounding factors when performing functional enrichment analyses. For example, significant fold changes are usually easier to detect for more abundant proteins.

First we save our lists of proteins which have increased or decreased abundance in G1 vs M, and the background set of proteins.

```
background_proteins <- t_test_results_annot$UniprotID

t_test_results_sig_relevant <- t_test_results_annot %>%
  filter(sig == TRUE) %>% # retains sig changes only
  select(UniprotID, difference) # select required columns

table(t_test_results_sig_relevant$difference > 0)
```

```
##
## FALSE  TRUE
##    76    80
```

```
# identify proteins with increase in abundance
up_proteins <- t_test_results_sig_relevant %>%
  filter(difference > 0) %>% # positive change in abundance
  select(UniprotID)

# identify proteins with decrease in abundance
dw_proteins <- t_test_results_sig_relevant %>%
  filter(difference < 0) %>%
  select(UniprotID)

write(unlist(up_proteins), "data/foreground_up.tsv")
write(unlist(dw_proteins), "data/foreground_dw.tsv")
write(background_proteins, "data/background.tsv")
```

Then go to the web-link: http://cbl-gorilla.cs.technion.ac.il/

To use GORILLA:

- Step 1. Select `Homo sapiens` for the organism
- Step 2: Select `Two unranked lists of genes (target and background lists)`
- Step 3: Below the `Target Set` box, click browse and upload your foreground data, e.g "foreground_dw.tsv". Below the `Background set box`, click browse and upload your background data.
- Step 4: Select "All" ontologies

Leave the advanced parameters as they are and click `Search Enriched GO terms`.

The GORILLA output is split into three sections at the top of the page, for `Processes`, `Function` and `Component`. For each section there is graph depicting the over-represented terms and their relationship to one another. Below this there is a table detailing the GO terms which are over-represented. Each GO term is associated with a p-value for the over-representation and an estimation of the False Discovery Rate (FDR). For this practical, only consider any GO term over-representation where the FDR < 0.01 (1E-2)

Check your knowledge

1. What GO terms are over-represented in the proteins with a change in abundance between M and G1 phase?
2. Why might there only be over-represented GO terms in the proteins with decreased abundance in G1 phase?
3. Why do you think there are so many related GO terms identified?
4. From inspecting the over-represented GO terms, can you say whether the Nocodazole has had the desired effect?
5. What other analyses could you perform to better understand the function of these proteins?

You've now completed the practical - Well done!

If you want to continue, you can investigate the effect of changing the thresholds used above for selecting the proteins with altered abundance. Does this change your biological interpretation of the results? Alternatively, you can have a look for R packages, software, online tools or publicly available data which could help you to perform the analyses you've suggested in the final question above. Finally, if you're interested in a) further exploring how to select more 'biologically relevant' changes in protein abundance, or b) a demonstration of the pitfalls of thresholding on the point estimate of the difference between means, see the notebook entitled "Thresholding_on_point_estimate".
