Quantitative proteomics data analysis

Tom S. Smith and Oliver M. Crook

Contents

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.

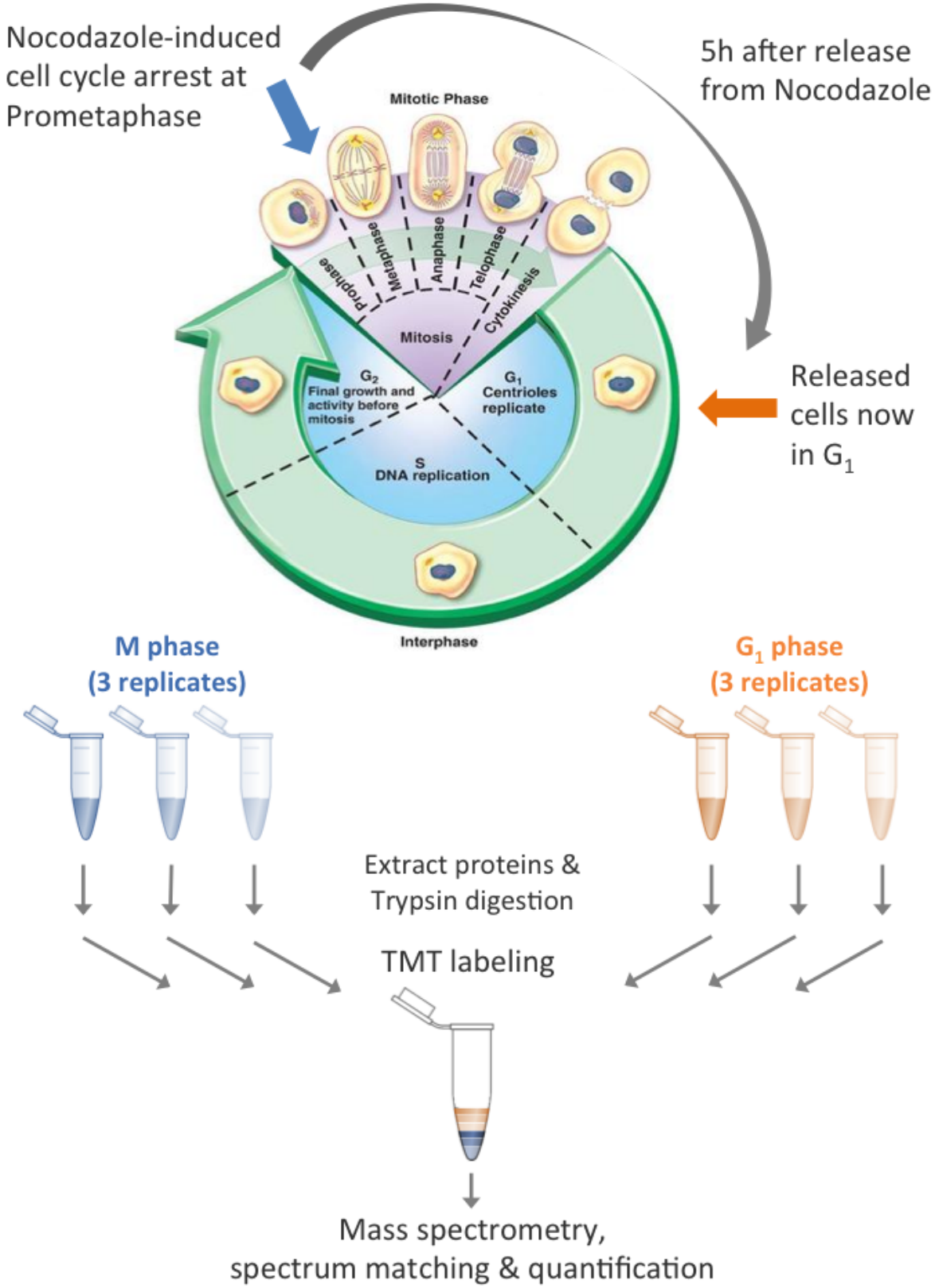
Cells are then released from M-phase by withdrawing Nocodazole. Samples are taken in triplicate at 0 hours and at 5 hours after release from M-phase. This process generates 3 x M-phase and 3 x G1-phase samples from 6 separate batches of cells.

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 6 plex tag. 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.

Check your knowledge

1. Can you remember the stages of the cell-cyle? What happens at each stage?
2. Why does inhibiting polymerisation of microtubles block transition beyond prometaphase?
3. What other
4. Why do we take experimental triplicates?
5. Why do we digest using trypsin? At which residues does trypsin cleave?
6. What is a tandem mass tag?
7. Why do we process all samples in the same MS run?
8. What is peptide spectrum matching


Nocodazole-induced cell cycle arrest at Prometaphase
5h after release from Nocodazole
Mitotic Phase
Prophase
Metaphase
Anaphase
Telophase
Cytokinesis
Mitosis
G2 Final growth and activity before mitosis
G1 Centrioles replicate
S DNA replication
Interphase
Released cells now in G1
M phase (3 replicates)
G1 phase (3 replicates)
Extract proteins & Trypsin digestion
TMT labeling
Mass spectrometry, spectrum matching & quantification

Learning outcomes

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

---

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

temporarily uncomment the neccessary lines to install the package

```
#install.packages("tidyr")
#install.packages("dplyr")
#install.packages("reshape2")
#install.packages("ggplot2")

suppressMessages(library(tidyr))
suppressMessages(library(dplyr))
suppressMessages(library(reshape2))
suppressMessages(library(ggplot2))
```

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 cannot be uniquely assigned to a single protein have been removed. Peptides which come from proteins in the cRAP database of common proteomics contaminates, e.g keratin, 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 dataframe. The code chunk below loads the data from the .tsv file into an R dataframe called `peptides_quant`.

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

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

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

```
## [1] "Sequence"       "Modifications"  "master_protein" "M_1"
## [5] "M_2"            "M_3"            "G1_1"           "G1_2"
## [9] "G1_3"
```

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

```
##           Sequence                          Modifications master_protein
## 1     EESTSSGNVSNR                1xTMT6plex [N-Term]         P23193
## 2   TSNERPGSGQGQGR                1xTMT6plex [N-Term]         Q8TAD8
## 3 AQHSNAAQTQTGEANR                1xTMT6plex [N-Term]         P34897
```

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

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

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

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

If we re-order the dataframe, we can see that some of the peptides have the same sequence but different modifications. Take some time to understand what each of the following lines of code is doing.

```
isDuplicated <- duplicated(peptides_quant$Sequence) # Type ?duplicated to see what duplicated does

duplicated_peptides <- unique(peptides_quant$Sequence[isDuplicated]) # identify the duplicated peptides

peptides_quant %>% # this is a 'pipe'. See the next section if you haven't come across this before
  filter(Sequence %in% duplicated_peptides) %>% # filter to only duplicated peptides
  arrange(Sequence) %>% # sort by peptide sequence
  head(4) # view the "head" of the data
```

```
##                    Sequence                                    Modifications
## 1 AAAEELQEAAGAGDGATENGVQPPK 2xTMT6plex [N-Term; K25]; 2xDeamidated [Q/N]
## 2 AAAEELQEAAGAGDGATENGVQPPK 2xTMT6plex [N-Term; K25]; 1xDeamidated [N19]
## 3           AAAFEEQENETVVVK 2xTMT6plex [N-Term; K15]; 1xDeamidated [N/Q]
## 4           AAAFEEQENETVVVK                     2xTMT6plex [N-Term; K15]
##   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
## 3         Q9Y490  14.3 12.0  10.6  16.0 15.8  16.0
## 4         Q9Y490 126.1 74.5 123.9 122.2 93.9 122.2
```

Check your knowledge

1. Where can I find keratin in daily life?
2. Why is keratin a contaminate?
3. Why do we worry about contaminates?
4. How many peptides were quantified in this experiment?
5. What do you think NA means in this context?
6. Why do you think some of the quantification values are NA?
7. What do you think we should do with NA?
8. Why might we have multiple quantification values for the same peptide but with different modifications?
9. What does if mean to say a peptide is 'assigned' to a protein. Could a peptide be assigned to multiple proteins?

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 unix. The main advantage of this is to simply reading the code.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 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
```

Normally, we want to pass the object before the pipe into the first argument of the function after the pipe. However, since the object is just passed to the first unnamed argument, we can pass it to any argument by providing the values for all prior arguments, By way of a simple example, not the first `setdiff` call below is equivalent to `setdiff(a, b)`, whereas the second `setdiff` call is equivalent to `setdiff(b, a)`.

```
a <- c(1, 2, 3)
b <- c(3, 4, 5)
a %>% setdiff(b)
```

```
## [1] 1 2
```

```
setdiff(a, b)
```

```
## [1] 1 2
```

```
a %>% setdiff(x=b)
```

```
## [1] 4 5
```

```
setdiff(b, a)
```

```
## [1] 4 5
```

Magrittr pipes become especially useful for piping together complex functions when reformating, 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 `dyplr` 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_all(funs(sum(.))) %>% # aggregate the quantification values using the sum function
  data.frame() # convert back to a data.frame
```

```
## Warning: `funs()` is deprecated as of dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
print(head(peptides_quant_no_mods))
```

```
##               Sequence master_protein   M_1   M_2   M_3  G1_1  G1_2  G1_3
## 1             AAAAAAAK         Q9NYV4  92.2  38.3  66.4  78.9  61.6  78.9
## 2          AAAAAAALQAK         P36578 371.7 182.3 383.8 394.4 315.2 394.4
## 3 AAAAASAAGPGGLVAGKEEK         A6NIH7  35.9  19.0  36.3  38.8  36.6  38.8
## 4    AAAAAWEEPSSGNGTAR         Q9P258 103.1  46.1  81.0 110.6 108.8 110.6
## 5             AAAACLDK         Q9H3U1  43.9  18.5  36.1  43.2  39.9  43.2
## 6            AAAAGSLDR         Q9Y2U8  59.1  38.3  61.1  97.3  79.9  97.3
```

2d. Dealing with missing values

We still have peptides with missing values

```
quant_columns <- colnames(peptides_quant)[4:9]

missing_values_pre_agg <- peptides_quant  %>%
  select(one_of(quant_columns)) %>% # only select the quantification columns
  is.na() %>% # returns a matrix of booleans (TRUE, FALSE), where TRUE = values is NA
  sum() # TRUE=1, FALSE=0 so sum of booleans is the number of TRUEs

missing_values_post_agg <- peptides_quant_no_mods %>%
  select(one_of(quant_columns)) %>%
  is.na() %>%
  sum()

message(sprintf("Before aggegation, we had %i peptides with missing values, after aggregation we have %i
```

```
## Before aggegation, we had 138 peptides with missing values, after aggregation we have 137 peptides wi
```

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

```
peptides_quant_no_mods_no_na <- peptides_quant_no_mods %>% na.omit()
```

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.

Check your knowledge

1. Why should we remove NA?
2. Are there biological reasons for NA?
3. How should we impute?
4. What imputation stratagies exist?

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

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

```
pep_tots <- peptides_quant_no_mods_no_na %>%
  select(one_of(quant_columns)) %>%
  colSums()

# formatC(format='e') formats numeric values into scientific notation. See ?formatC
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

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

peptides_quant_norm <- peptides_quant_no_mods_no_na

# the divide operation works on each cell in the dataframe by column so we need to transpose (t)
# the dataframe so columns = peptides and rows = samples before performing the divide operation.
peptides_quant_transposed <- t(peptides_quant_norm[, quant_columns])

#Perform data normalisation
peptides_quant_norm_transposed <- peptides_quant_transposed/pep_tots_correction_factors

# then re-transform back to the initial layout
peptides_quant_norm[, quant_columns] <- t(peptides_quant_norm_transposed)
```

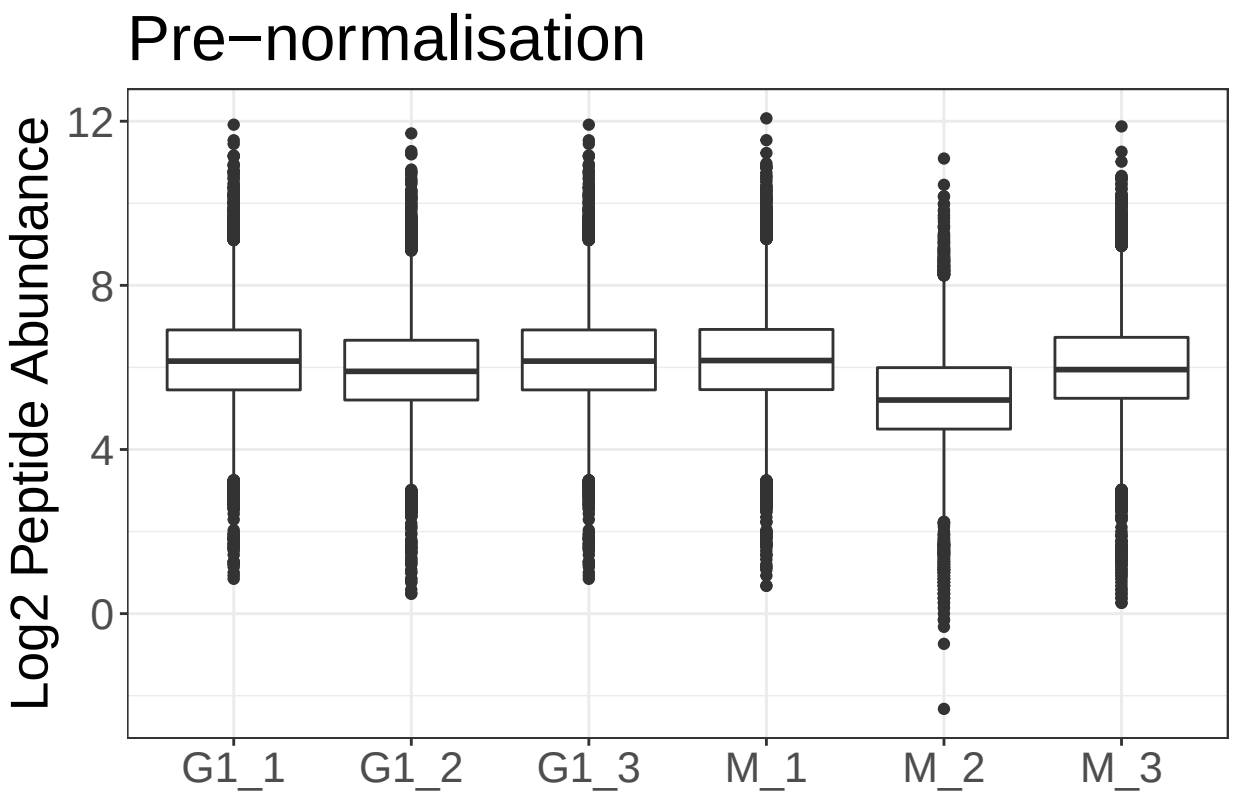
We can check that the total are now the same:

```
norm_pep_tots <- peptides_quant_norm %>%
  select(one_of(quant_columns)) %>%
  colSums()

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

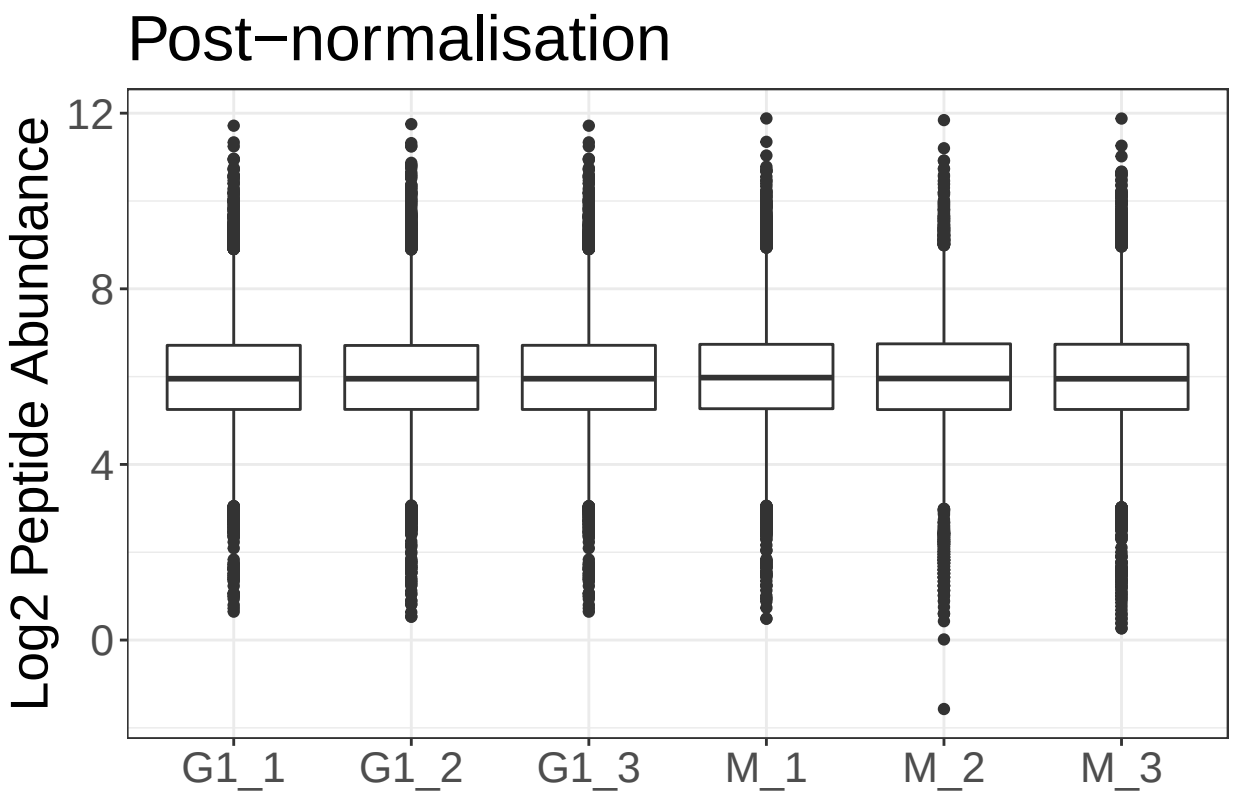
```
##       M_1       M_2       M_3      G1_1      G1_2      G1_3
## "1.7e+06" "1.7e+06" "1.7e+06" "1.7e+06" "1.7e+06" "1.7e+06"
```

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 (normal) distributed. However, they are approximately log-Gaussian distributed.

```
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(sample, log(abundance, 2)) +
    geom_boxplot() +
    xlab("") + ylab("Log2 Peptide Abundance") +
    theme_bw() +
    theme(text=element_text(size=20)) +
    ggtitle(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?

```
protein_quant <- peptides_quant_norm  %>%
  select(-one_of(c("Sequence"))) %>% # exclude the Sequence column
  group_by(master_protein) %>% # group by the master_protein column
  summarise_all(funs(median(.))) %>% # aggregate the quantification values using the median function
  tibble::column_to_rownames('master_protein') # indexes dataframe with master_protein column

head(protein_quant)
```

```
##             M_1      M_2      M_3     G1_1     G1_2     G1_3
## A0AVT1 75.17733 66.48070 70.16831 76.72206 78.99918 76.72206
## A0MZ66 52.39632 53.01625 52.07491 45.79838 49.18720 45.79838
## A1L390 51.38870 51.50150 54.28020 49.40831 57.71229 49.40831
## A1X283 43.28391 42.49715 49.36842 45.49392 45.31215 45.49392
## A2RTX5 34.96009 36.18570 32.97911 32.44595 38.02708 32.44595
## A3KN83 13.49337 15.31581 14.93583 20.61579 18.60020 20.61579
```

---

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 Guassian, 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 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.

```
protein_quant_log <- protein_quant
protein_quant_log[,quant_columns] <- log2(protein_quant_log[,quant_columns])
```

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.

```
runsingleTTest <- 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, difference, ci))
}
```

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

```
TTest_results <- protein_quant_log

# apply the runSingleTTest function over each row ("MARGIN=1") for the quantification columns
# We need to transform (t) the values returned by apply to get the correct layout
TTest_results[c("p.value", "difference", "CI_diff_low", "CI_diff_high")] <- t(apply(
  TTest_results[, quant_columns], MARGIN = 1, runsingleTTest))
```

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 correspondend 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.

```
TTest_results$FDR <- p.adjust(TTest_results$p.value, method = "BH")

FDR_threshold <- 0.01
TTest_results$sig <- TTest_results$FDR < FDR_threshold
```

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

```
table(TTest_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 signficant changes are small. In fact, just 16/156 of the significant changes in abundance are greater than 2-fold (Note 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).

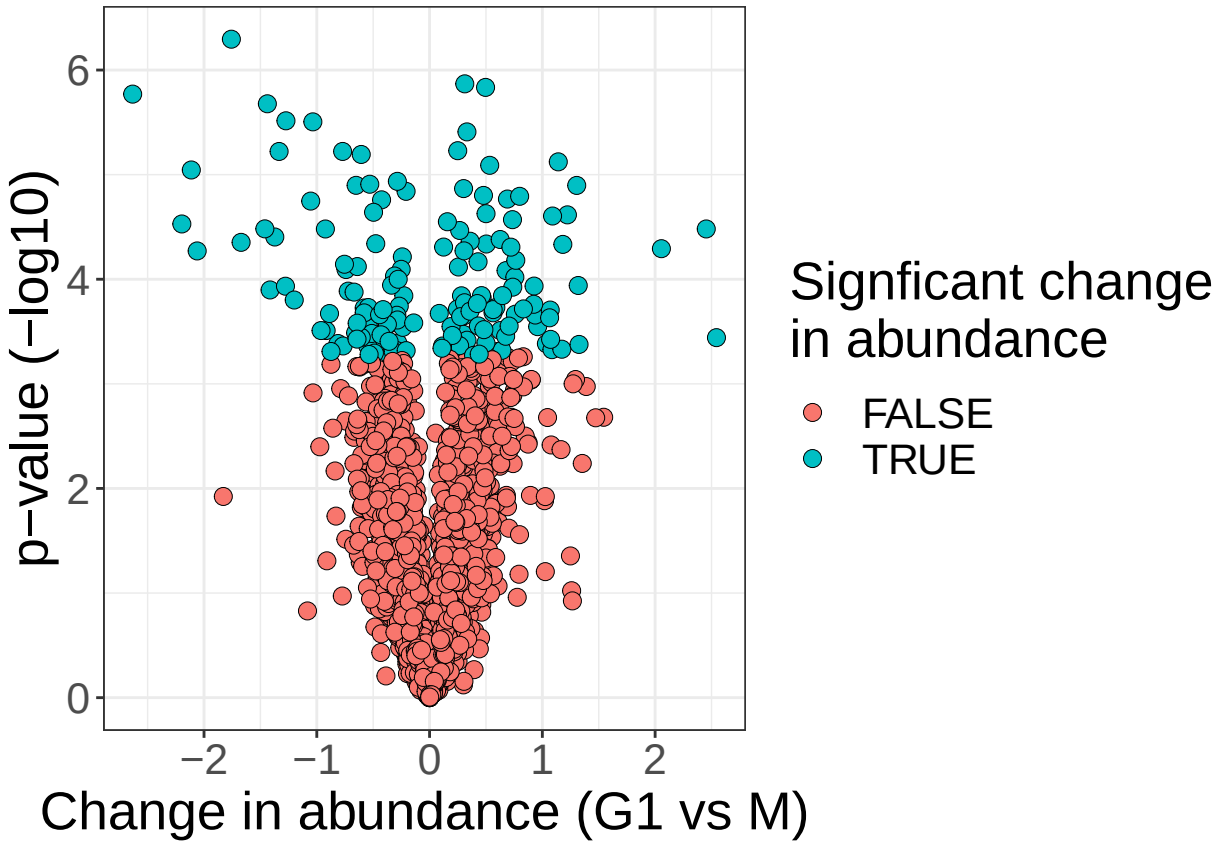
```
cat("Tally of changes > 2-fold for proteins where null hypothesis rejected")
```

```
## Tally of changes > 2-fold for proteins where null hypothesis rejected
```

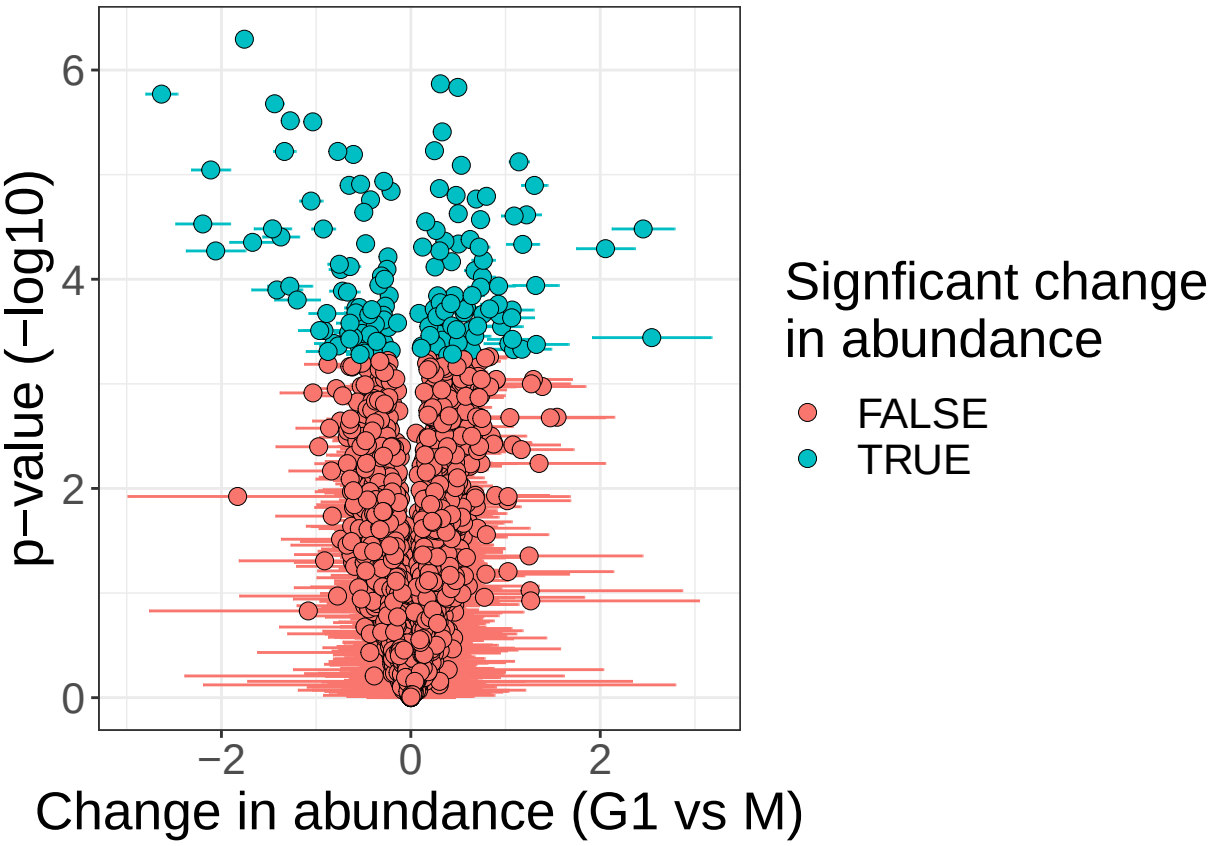
```
print(table(abs(TTest_results[TTest_results$sig == T, ]$difference > 1)))
```

```
##
##   0   1
## 140  16
```

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

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




```
p2 <- p + geom_errorbarh(aes(xmin=CI_diff_low, xmax=CI_diff_high, colour=sig)) +
  geom_point(pch=21, stroke=0.25, size=3) +
  scale_color_discrete(guide=F)

print(p2)
```



3b. Selecting the most relevant changes

You may not be concerned that many of the changes are relatively small. However, it's reasonable to assume a very small change in abundance is unlikely to be biologically relevant. Therefore, it's beneficial to also threshold on the scale of the change, the so called "effect size". We could simply apply a threshold on the point estimate of the difference between the two populations. However, this would not take account of the confidence we have about the true difference between the two populations. For a more explanation of the pitfalls of thresholding on the point estimate, see the example at the bottom of this notebook.

For this reason, we apply a threshold on the 95% confidence interval of the difference between the means. In the cell below we apply a filter that change is at least 1.5-fold

```
# Function to find the minimum distance between the CI and zero
GetCIMinDiff <- function(row){

  if(sign(row[['CI_diff_low']]) != sign(row[['CI_diff_high']]) | # if low and high CI have diff. signs,
     row[['CI_diff_high']]==0 | row[['CI_diff_low']] == 0){ # either is zero
    return(0) # then the CI overlaps zero
  }

  else{

    if(abs(row[['CI_diff_high']])<(abs(row[['CI_diff_low']]))){ # if abs. values of high CI is lower
      return(row[['CI_diff_high']]) # return the high CI
    }

    else{
      return(row[['CI_diff_low']]) # otherwise, return the low CI
```

```
    }
  }
}

# apply GetCIMinDiff to each protein
TTest_results$abs_CI_diff <- apply(TTest_results, MARGIN=1, GetCIMinDiff)

# 50% difference = 2^0.55
TTest_results$relevant_change <- abs(TTest_results$abs_CI_diff)> ((2^0.5)-1)

cat("Cross-tabulation of significant differences (columns) and differences where CI indicates differenc
```

```
## Cross-tabulation of significant differences (columns) and differences where CI indicates difference >
```

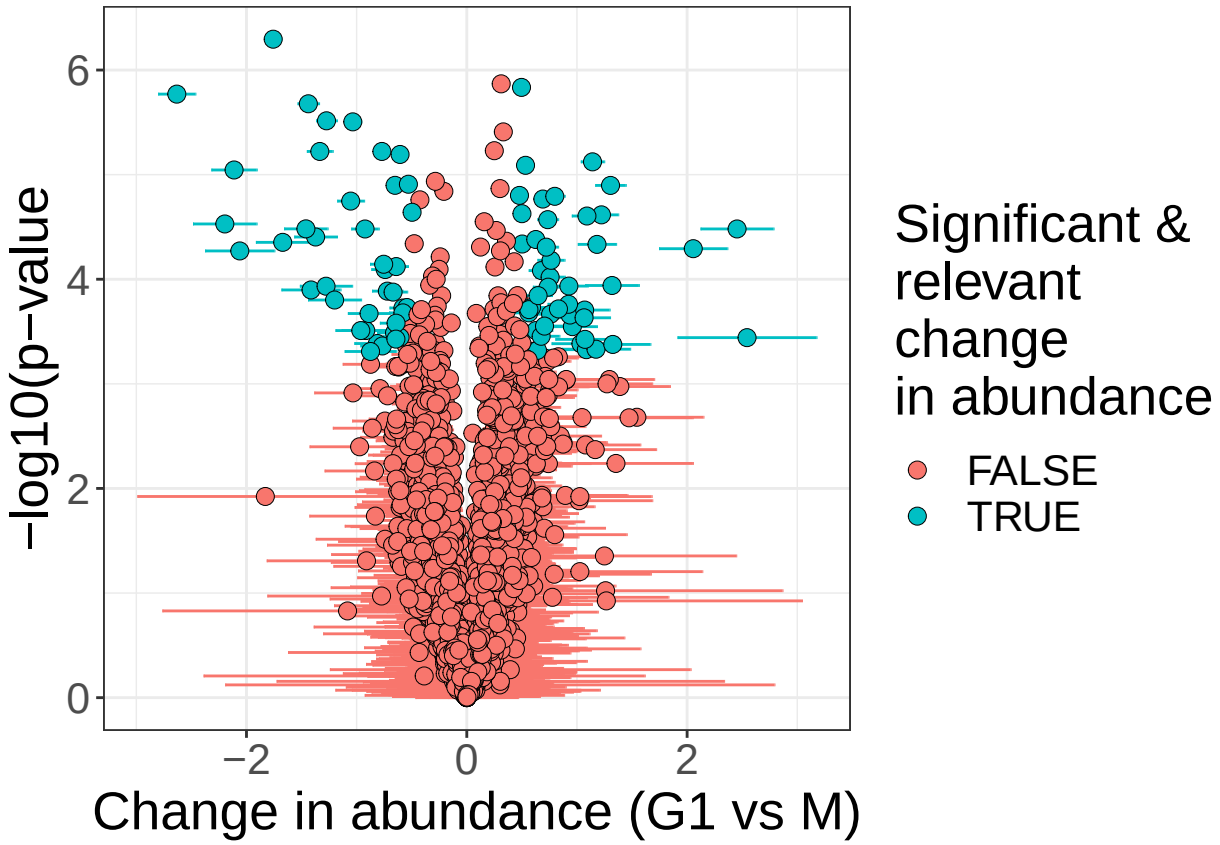
```
print(table(TTest_results$relevant_change, TTest_results$sig))
```

```
##
##         FALSE TRUE
##   FALSE  2690   70
##   TRUE     41   86
```

So we can see that for 86/156 proteins with a significant change in abundance, the 95 % CI for the differece suggests the change is $> 50\%$.

We can now make a new volcano plot to show which proteins are identified as having a statistically significant and biologically relevant change in abundance.

```
p <- ggplot(TTest_results, aes(x=difference, y=-log(p.value,10), fill=(sig & relevant_change))) +
  xlab("Change in abundance (G1 vs M)") + ylab("-log10(p-value") +
  scale_fill_discrete(name="Significant &\nrelevant\nchange\nin abundance") +
  geom_errorbarh(aes(xmin=CI_diff_low, xmax=CI_diff_high, colour=(sig & relevant_change))) +
  geom_point(pch=21, stroke=0.25, size=3) +
  scale_color_discrete(guide=F) +
  theme_bw() +
  theme(text=element_text(size=20))

print(p)
```



Check your knowledge

1. Suppose you screened many patients for their levels of a single known disease-associated protein with a blood test and , would you try and control the FWER or the FDR? Why?
2. Suppose you quantified all proteins in the blood of patients with a disease and compared to matched healthy patients. How would I decide what significant changes in abundance are clinically relevant?

---

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=F,
    colClasses=c("character", "character", "character", "NULL", "NULL"),
  col.names = c("UniprotID", "Name", "Description", NA, NA))

TTest_results_annotated <- merge(human_proteins_ids_df, TTest_results, by.x="UniprotID", by.y="row.name
```

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

```r
TTest_results_annotated %>%
  filter(sig==T) %>% # significant changes
  filter(relevant_change==T) %>% # bioligically relevant changes
  filter(difference<0) %>% # decrease in abundance
  arrange(abs_CI_diff) %>% # sort by CI
  select(Name, Description, abs_CI_diff, FDR) %>% # select columns of interest
  head(20)
```

```
##           Name                                                Description
## 1   SGO2_HUMAN                                                Shugoshin 2
## 2  KIFC1_HUMAN                                 Kinesin-like protein KIFC1
## 3   ANLN_HUMAN                                                    Anillin
## 4   TSP1_HUMAN                                              Thrombospondin-1
## 5  CENPE_HUMAN                             Centromere-associated protein E
## 6  PAF15_HUMAN                                      PCNA-associated factor
## 7  AURKA_HUMAN                                             Aurora kinase A
## 8   TPX2_HUMAN                                 Targeting protein for Xklp2
## 9   KITH_HUMAN                                 Thymidine kinase, cytosolic
## 10  PRC1_HUMAN                               Protein regulator of cytokinesis 1
## 11  PLK1_HUMAN                             Serine/threonine-protein kinase PLK1
## 12 CYR61_HUMAN                                                  Protein CYR61
## 13 CENPF_HUMAN                                       Centromere protein F
## 14 NUSAP_HUMAN                     Nucleolar and spindle-associated protein 1
## 15 BOREA_HUMAN                                                    Borealin
## 16 KIF22_HUMAN                                 Kinesin-like protein KIF22
## 17 COCA1_HUMAN                                 Collagen alpha-1(XII) chain
## 18  SPRC_HUMAN                                                      SPARC
## 19 AKTS1_HUMAN                                 Proline-rich AKT1 substrate 1
## 20 BUB1B_HUMAN Mitotic checkpoint serine/threonine-protein kinase BUB1 beta
##    abs_CI_diff         FDR
## 1   -2.4638786 0.001213524
## 2   -1.9081326 0.002591241
## 3   -1.9068350 0.001736954
## 4   -1.7462992 0.003160560
## 5   -1.6742648 0.001213524
## 6   -1.4281338 0.003046632
## 7   -1.3412110 0.001213524
## 8   -1.2633973 0.002652798
## 9   -1.2170169 0.001543256
## 10  -1.1790351 0.001290723
## 11  -1.1785805 0.002998024
## 12  -1.1459756 0.005455782
## 13  -1.0405826 0.005177838
## 14  -0.9592887 0.006081122
## 15  -0.9577150 0.001290723
## 16  -0.9319832 0.001987746
## 17  -0.7990069 0.002652798
## 18  -0.7331496 0.007701726
## 19  -0.7031621 0.001543256
## 20  -0.6988267 0.007701726
```

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 proteins 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 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, Biological Processes and Cellular Localisations. 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-representated 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 `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-represenatation analysis?
2. Do you think it makes more sense to perform one or two tests for the GO term over-represenatation analysis?

To simplify this practical, we use GORILLA but note there are many better tools available.

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 <- TTest_results_annotated$UniprotID

TTest_results_sig_relevant <- TTest_results_annotated %>%
  filter(sig==T) %>% # retains sig changes only
  filter(relevant_change==T) %>% # retain relevant changes only
  select(UniprotID, difference, abs_CI_diff) # select required columns

table(TTest_results_sig_relevant$difference>0)
```

```
##
## FALSE  TRUE
##    41    45
```

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

```
dw_proteins <- TTest_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 for `Processes`, `Function` and `Component`. For each section there is graph depiciting the over-representated terms and their relationship to one another. Below this there is a table detailing the GO terms which are over-represented. Each GO terms 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?
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 affect 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 publically available data which could help you to perform the analyses you've suggested. Finally, if you're interested in a demonstration of the pitfalls of thresholding on the point estimate of the difference between means, see the notebook entitled "Thresholding_on_point_estimate.R".
