Data Science Workshop

British Society for Proteomic Research Meeting 2018

Alistair Bailey

May 29 2018

Contents

Overview

These lessons cover:

1. An introduction to R and RStudio
2. An introduction to the tidyverse
3. Importing and transforming proteomics data
4. Visualisation of proteomics analysis

Requirements

Up to date version of R and Rstudio

The following R packages:

```
install.packages(c("tidyverse","janitor"))
```

Chapter 1

Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 1. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 1.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 1.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2018) in this sample book, which was built on top of R Markdown and **knitr** (**?**).

Table 1.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

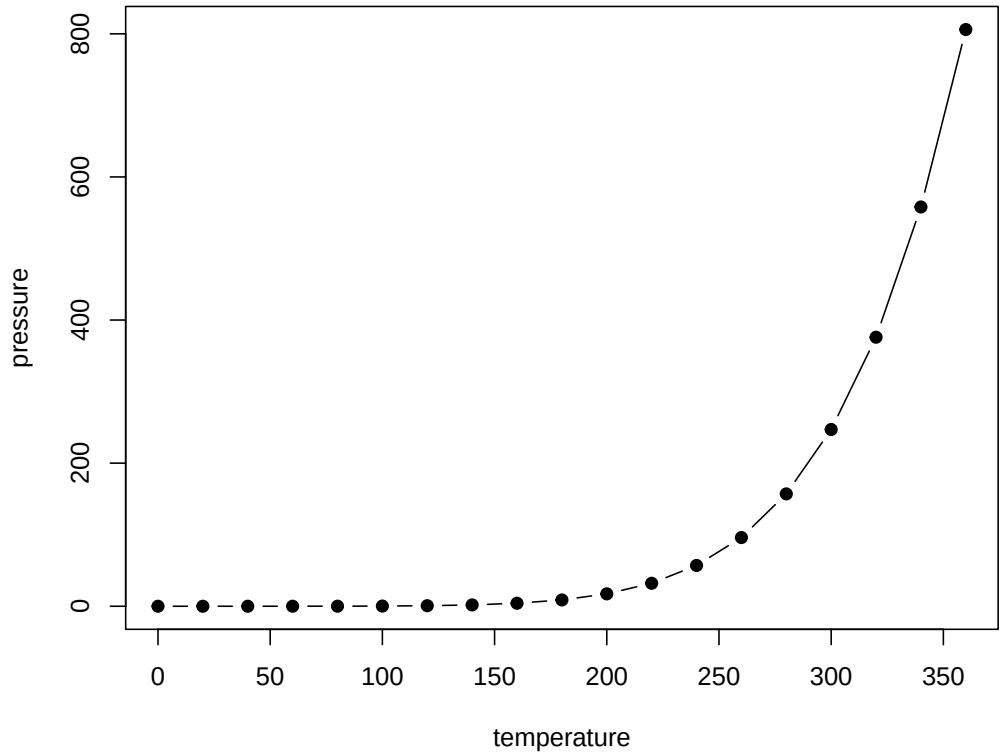


Figure 1.1: Here is a nice figure!

Chapter 2

The tidyverse

An introduction to the tidyverse.

Chapter 3

Importing and transforming proteomics data

3.1 Importing flat files

3.2 Calculating fold change and enrichment

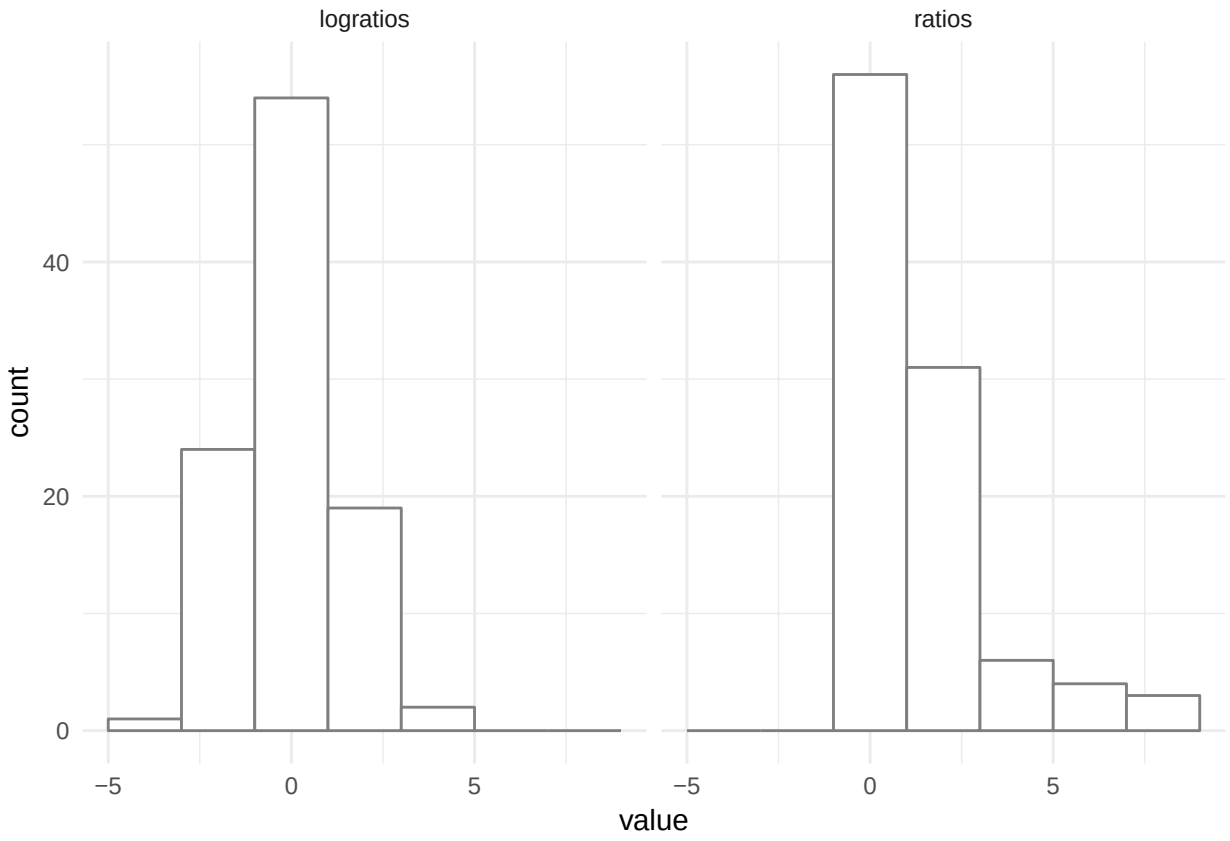
3.2.1 Fold change and log-fold change

Fold changes are ratios, the ratio of say protein expression before and after treatment, where a value larger than 1 for a protein implies that protein expression was greater after the treatment.

In life sciences, fold change is often reported as log-fold change. Why is that? There are at least two reasons which can be shown by plotting.

One is that ratios are not symmetrical around 1, so it's difficult to observe both changes in the forwards and backwards direcion i.e. proteins where expression went up and proteins where expression went down due to treatment. When we transform ratios on a log scale, the scale becomes symmetric around 0 and thus we can now observe the distribution of ratios in terms of positive, negative or no change.

```
set.seed(10)
x <- 2^(rnorm(100))
y <- 2^(rnorm(100))
ratios <- tibble(value = x/y, label = "ratios")
logratios <- tibble(value = log2(ratios$value), label = "logratios")

bind_rows(ratios,logratios) %>%
  ggplot(aes(value)) +
  geom_histogram(binwidth = 2, colour = "grey50", fill = "white") +
  ggplot2::facet_wrap(~ label) +
  theme_minimal()
```



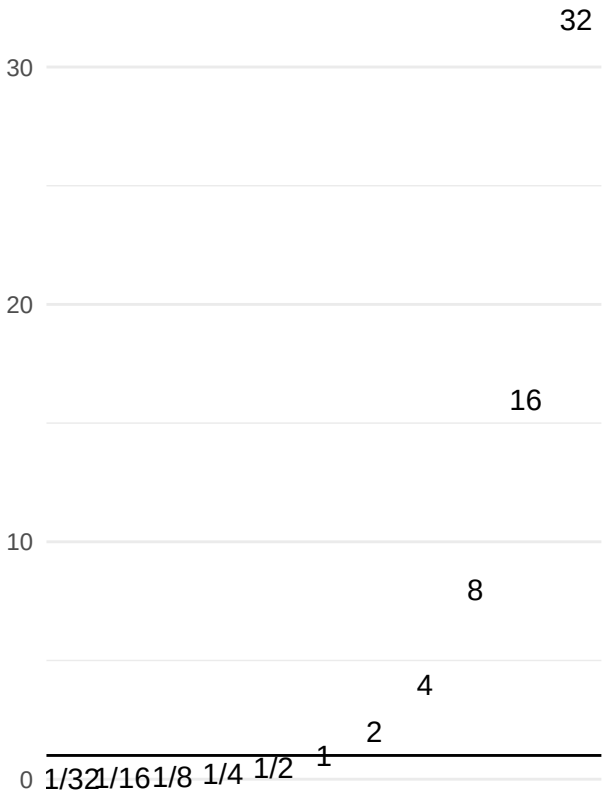
A second reason is that transforming values onto a log scale changes where the numbers actually occur when plotted on that scale. If we consider the log scale to represent magnitudes, then we can more easily see changes of small and large magnitudes when

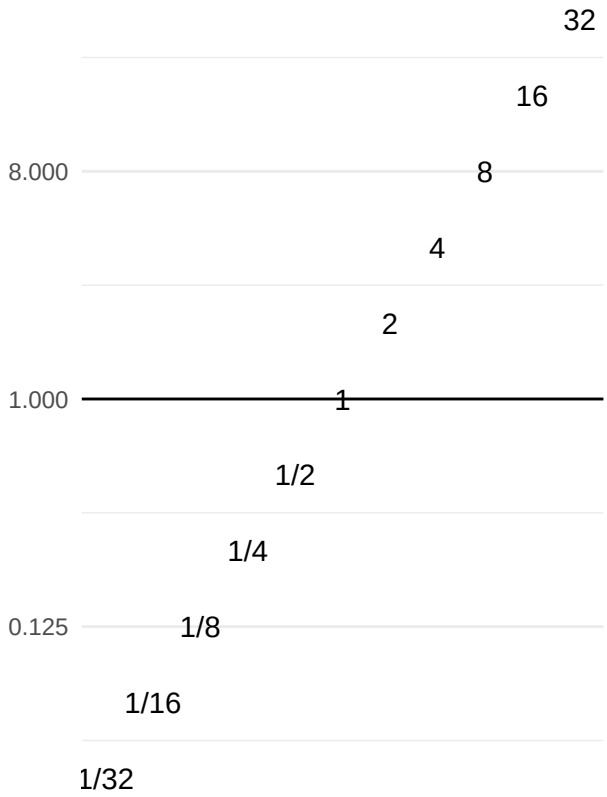
we plot the data.

For example, a fold change of 32 times can be either a ratio 1/32 or 32/1.

1/32 is much closer to 1 than 32/1, but transformed to a log scale we see that in terms of magnitude of difference it is the same as 32/1.

```
x2 <- 2^seq(1,5)
y_vals <- c(rev(1/x2),1,x2)
names <- c(paste0("1/",rev(x2)),1,x2)
x_vals <- seq(along=y_vals)

dat <- tibble(x_vals,y_vals,names)

p1 <- ggplot(dat,aes(x_vals,y_vals, label = names)) +
  geom_text() +
  geom_hline(yintercept = 1) +
  theme_minimal() +
  labs(x = NULL, y = NULL) +
  scale_x_continuous(breaks = NULL)



p2 <- ggplot(dat,aes(x_vals,y_vals, label = names)) +
  geom_text() +
  geom_hline(yintercept = 1) +
  scale_y_continuous(trans = "log2") +
  theme_minimal() +
  labs(x = NULL, y = NULL) +
  scale_x_continuous(breaks = NULL)

plot_grid(p1,p2)
```


32
30
20
16
10
8
4
2
1
0
1/32
1/16
1/8
1/4
1/2


32
16
8.000
8
4
2
1.000
1
1/2
1/4
0.125
1/8
1/16
1/32

Chapter 4

Fold change and t-test

1. Load the data, we need multiple replicates for each condition. To be tidy each protein is a set of observations (rows) of the variables, which are the values recorded for each replicate (columns).

```
## Parsed with column specification:
## cols(
##   protein.key_protein.Accession = col_character(),
##   protein.Description = col_character(),
##   `protein.ngramOnColumn -> EmillyBowler_KOGSK_Rep1_001_Merged_Spectrum_IA_final_prot
##   `protein.ngramOnColumn -> EmillyBowler_KOGSK_Rep2_Merged_IA_final_protein` = col_do
##   `protein.ngramOnColumn -> EmillyBowler_KOGSK_Rep3_001_Merged_Spectrum_IA_final_prot
##   `protein.ngramOnColumn -> EmillyBowler_WT_Rep1_001_Merged_Spectrum_IA_final_protein
##   `protein.ngramOnColumn -> EmillyBowler_WT_Rep2_001_Merged_Spectrum_IA_final_protein
##   `protein.ngramOnColumn -> EmillyBowler_WT_Rep3_001_Merged_Spectrum_IA_final_protein
##   Present_NumFiles = col_integer()
## )

## Observations: 6,150
## Variables: 9
## $ protein_key_protein_accession
```

```
## $ protein_description
## $ protein_ngram_on_column_emilly_bowler_kogsk_rep1_001_merged_spectrum_ia_final_prote
## $ protein_ngram_on_column_emilly_bowler_kogsk_rep2_merged_ia_final_protein
## $ protein_ngram_on_column_emilly_bowler_kogsk_rep3_001_merged_spectrum_ia_final_prote
## $ protein_ngram_on_column_emilly_bowler_wt_rep1_001_merged_spectrum_ia_final_protein
## $ protein_ngram_on_column_emilly_bowler_wt_rep2_001_merged_spectrum_ia_final_protein
## $ protein_ngram_on_column_emilly_bowler_wt_rep3_001_merged_spectrum_ia_final_protein
## $ present_num_files
```

2. Tidy up and deal with missing values, either impute or exclude missing values and normalise.

Let's consider our proteomics data as a distribution of values, one value for each protein in our experiment that together form a distribution. If we have replicate experiments we'll therefore have multiple distributions.

A quantile represents a region of distribution, for example the 0.95 quantile is the value such that 95% of the data lies below it. To normalise two or more distributions with each other without recourse to a reference distribution we:

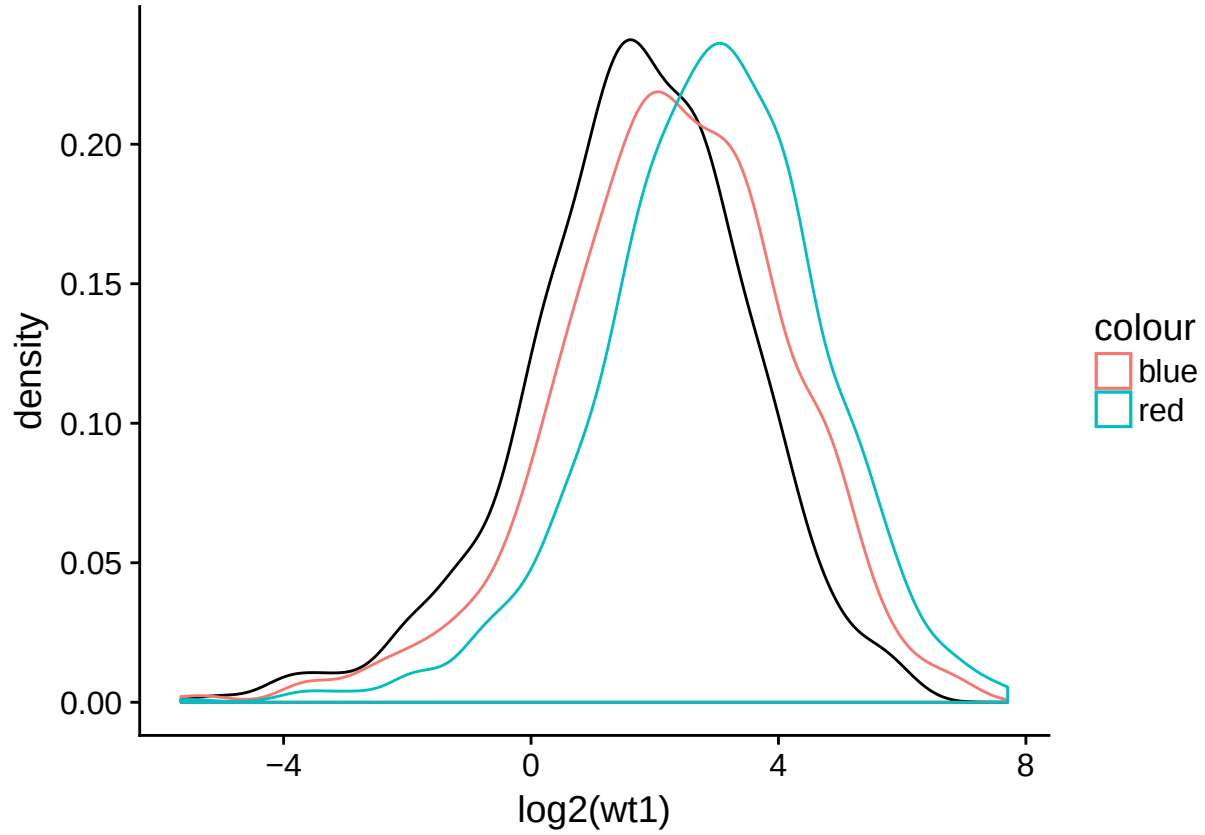
(i) Rank (quantile) the value in each experiment (column) from lowest to highest.
(ii) Sort each experiment (column) from lowest to highest value.
(iii) Calculate the mean across the experiments (rows) on the sorted values.
(iv) Substitute the mean values according to rank for each experiment to restore the original order.

Dave Tang's Blog : Quantile Normalisation in R

.**?** Neither corresponds to the **?** implementation of quantile norm (est 2001 and explained below) pic.twitter.com/lCyy6YyNB8

— Rafael Irizarry (**?**) December 18, 2014

```r
# Tidy data up
dat_tidy <- dat_em %>%
  # Remove missing values
  drop_na() %>%
  # Select and rename columns
  select(protein_accession = protein_key_protein_accession,
         protein_description,
         wt1 = protein_ngram_on_column_emilly_bowler_wt_rep1_001_merged_spectrum_ia_fina
         wt2 = protein_ngram_on_column_emilly_bowler_wt_rep2_001_merged_spectrum_ia_fina
         wt3 = protein_ngram_on_column_emilly_bowler_wt_rep3_001_merged_spectrum_ia_fina
         kog1 = protein_ngram_on_column_emilly_bowler_kogsk_rep1_001_merged_spectrum_ia_
         kog2 = protein_ngram_on_column_emilly_bowler_kogsk_rep2_merged_ia_final_protein
         kog3 = protein_ngram_on_column_emilly_bowler_kogsk_rep3_001_merged_spectrum_ia_

# Plot data
dat_tidy %>%
  ggplot(aes(log2(wt1))) +
  geom_density() +
  geom_density(aes(log2(wt2), colour = "blue")) +
  geom_density(aes(log2(wt3), colour = "red"))
```




```
# Normalise to maximum column value
dat_norm_max <- dat_tidy %>%
  mutate(wt1 = wt1/max(wt1),
         wt2 = wt2/max(wt3),
         wt3 = wt3/max(wt3),
         kog1 = kog1 / max(kog1),
         kog2 = kog2 / max(kog2),
         kog3 = kog3 / max(kog3)
         )

# Normalise to median column value
dat_norm_med <- dat_tidy %>%
  mutate(wt1 = wt1/median(wt1),
         wt2 = wt2/median(wt3),
```

```
          wt3 = wt3/median(wt3),
          kog1 = kog1 / median(kog1),
          kog2 = kog2 / median(kog2),
          kog3 = kog3 / median(kog3)
          )

dat_rank <- dat_tidy %>% select(-c(1:2,6:8)) %>%
  apply(., 2, rank, ties.method="min") %>% as.data.frame()

dat_sort <- dat_tidy %>% select(-c(1:2,6:8)) %>%
  apply(., 2, sort) %>% as.data.frame()

dat_mean <- dat_sort %>% apply(., 1, mean)

index_mean <- function(my_idx, my_mean){
  return(my_mean[my_idx])
}

dat_norm <- dat_rank %>%
  apply(.,2,index_mean, my_mean = dat_mean) %>%
  as.data.frame()

# dat_wt %>%
#   ggplot(aes(log2(wt1))) +
#   geom_density() +
#   geom_density(aes(log2(wt2), colour = "blue")) +
#   geom_density(aes(log2(wt3), colour = "red"))

# Quantile normalisation : the aim is to give different distributions the
```

```
# same statistical properties
quantile_normalisation <- function(df){
  #
  df_rank <- apply(df,2,rank,ties.method="average")
  df_sorted <- data.frame(apply(df, 2, sort))
  df_mean <- apply(df_sorted, 1, mean)

  index_to_mean <- function(my_index, my_mean){
    return(my_mean[my_index])
  }

  df_final <- apply(df_rank, 2, index_to_mean, my_mean=df_mean) %>% as.data.frame()
  #rownames(df_final) <- rownames(df)
  return(df_final)
}

dt_wt <- dat_tidy %>% select(-c(1:2,6:8))
dt_kog <- dat_tidy %>% select(-c(1:5))

dt_wt_norm <- quantile_normalisation(dt_wt)
dt_kog_norm <- quantile_normalisation(dt_kog)

dat_norm <- bind_cols(dat_tidy[,1:2],dt_wt_norm,dt_kog_norm)
# Have a look at the median normalised data
glimpse(dat_norm_med)
```

```
## Observations: 1,095
## Variables: 8
## $ protein_accession   <chr> "4562_DHE3_HUMAN", "14948_RS2_HUMAN", "143...
## $ protein_description <chr> "Glutamate dehydrogenase 1_ mitochondrial ...
```

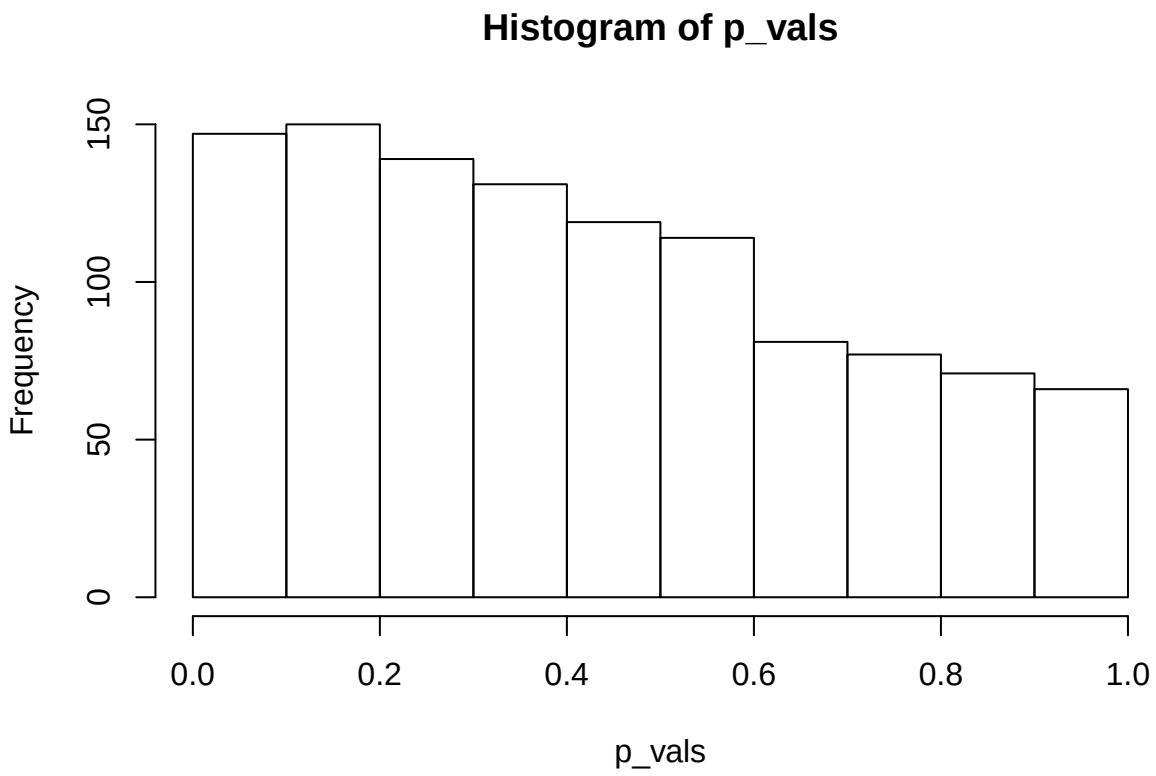
```
## $ wt1                <dbl> 1.46455644, 3.78274902, 0.10257018, 1.5593...
## $ wt2                <dbl> 1.29015126, 3.06162560, 0.06677263, 1.2956...
## $ wt3                <dbl> 1.3898245, 4.6907618, 0.1136384, 0.7625680...
## $ kog1               <dbl> 2.1246421, 2.2498354, 0.2868330, 0.9172013...
## $ kog2               <dbl> 0.06248655, 3.78198563, 0.08551018, 0.5341...
## $ kog3               <dbl> 1.55811005, 2.16717459, 0.32849794, 0.8012...
```

```
# Save the median normalised data
write_excel_csv(dat_norm,"data/example_median_nomralised_proteomics_data.csv")
```

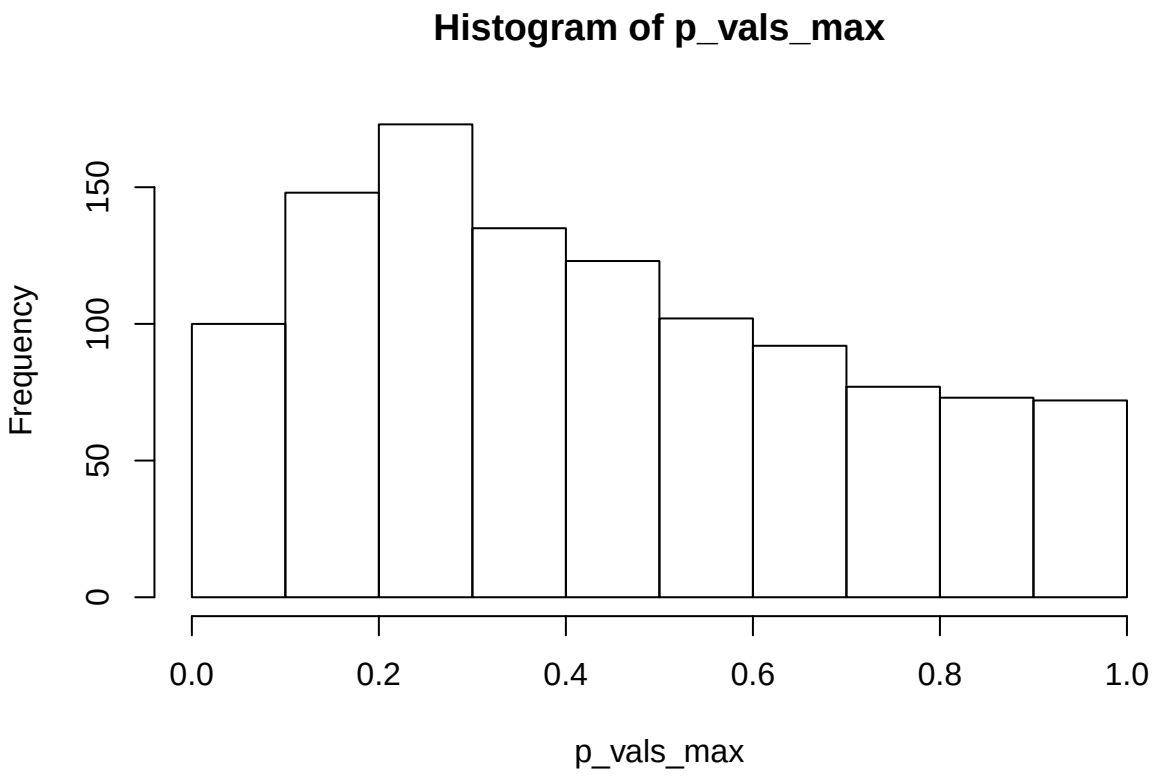
```
# Impute NA with lowest value, need to change this!
control <- dat_select %>% filter(!is.na(control_1) |
                                 !is.na(control_2) |
                                 !is.na(control_3)) %>% select(3,4,5)

treatment <- dat_select %>% filter(!is.na(ras_1) |
                                 !is.na(ras_2) |
                                 !is.na(ras_3)) %>% select(6,7,8)

# Impute minimum
row_min <- function(x) apply(x,1,min,na.rm = TRUE)
impute_min <- function(x) replace(x, is.na(x), row_min(x))

#control_na <- control %>% filter(!complete.cases(.))
impute_min(control)



control %>% replace_na(apply(.,1,min))
```

3. Use `t.test` to perform Welch Two Sample t-test on untransformed data. This outputs the p-values we need for each protein.

```r
# T-test function for multiple experiments
expriments_ttest <- function(dt,grp1,grp2){
  # Subset control group and convert to numeric
  x <- dt[grp1] %>% unlist %>% as.numeric()
  # Subset treatment group and convert to numeric
  y <- dt[grp2] %>% unlist %>% as.numeric()
  # Perform t-test
  result <- t.test(x, y)
  # Return p-values
  return(result$p.value)
}

# Apply t-test function to data
# array = dat, 1 = rows, FUN = expriments_ttest, and arguements
# For median normalised data
p_vals <- apply(dat_norm,1,expriments_ttest, grp1 = c(3:5), grp2 = c(6:8))
# For maximum normalised data
p_vals_max <- apply(dat_norm_max,1,expriments_ttest, grp1 = c(3:5), grp2 = c(6:8))

# Plot histograms
hist(p_vals)
```




```
hist(p_vals_max)
```



4. Perform log transformation of the observations for each protein.

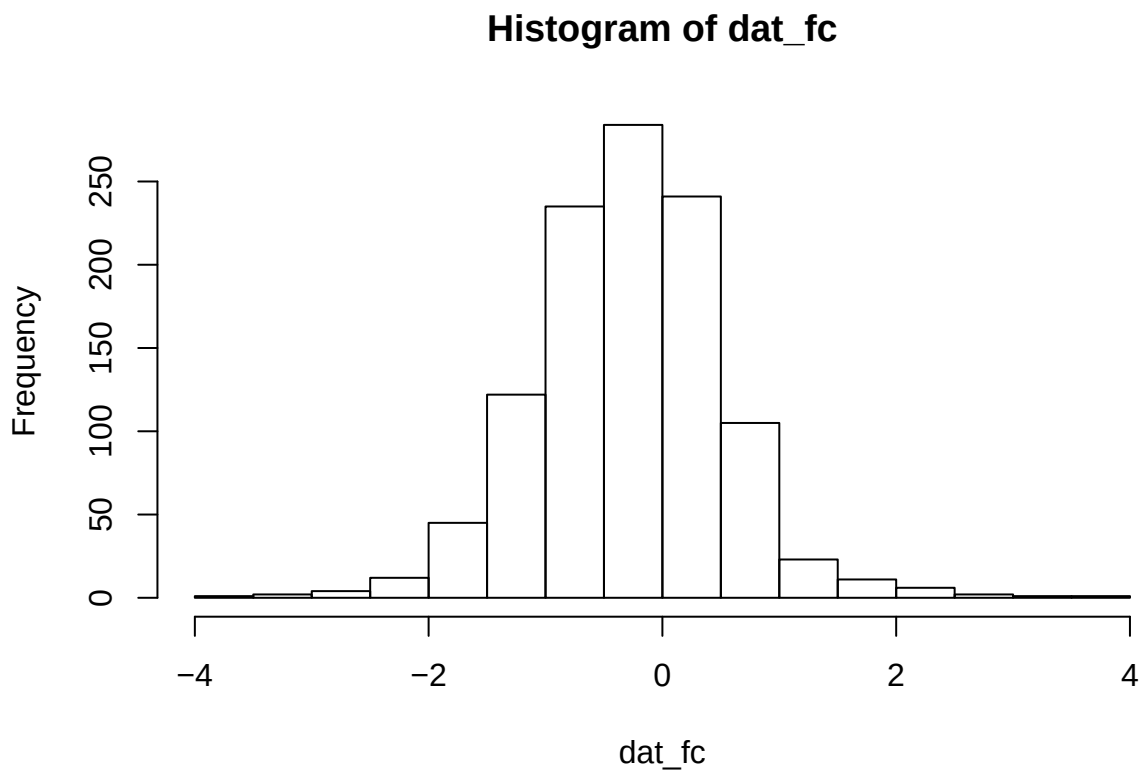
```
# Select columns and log data
dat_log <- dat_norm %>%
  select(-c(protein_accession,protein_description)) %>% log2()
dat_max_log <- dat_norm_max %>%
  select(-c(protein_accession,protein_description)) %>% log2()
```

5. Calculate the mean observation for each protein under each condition.

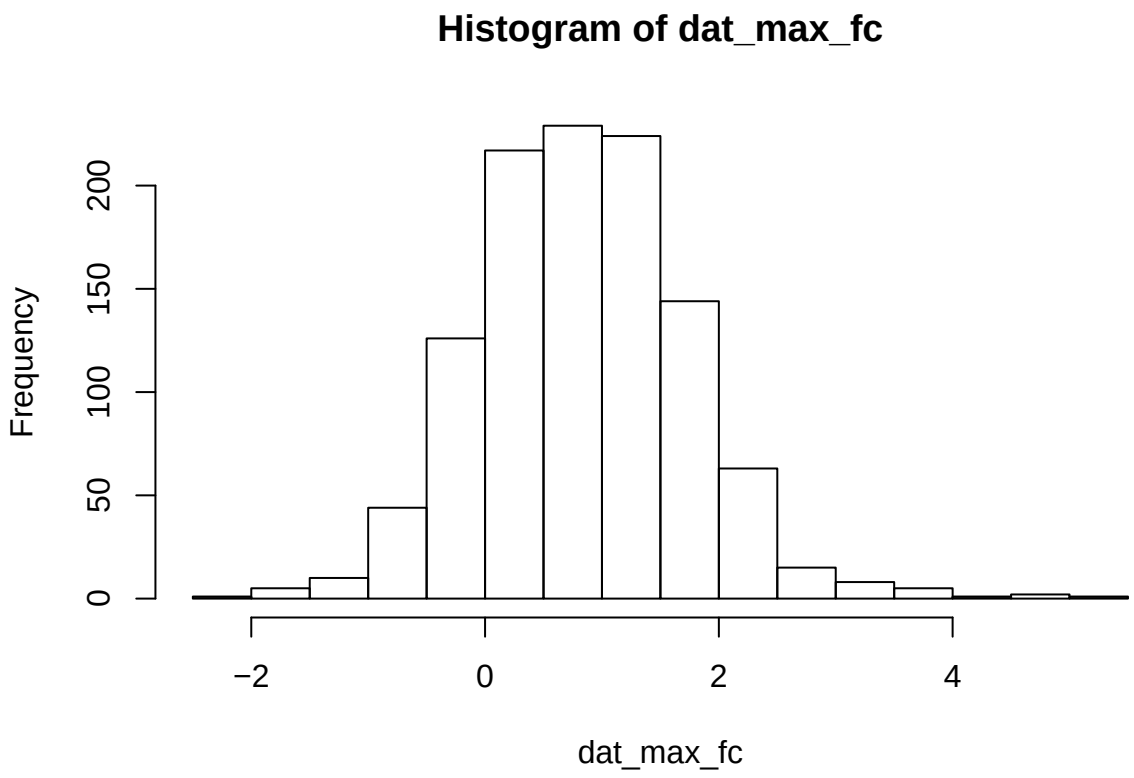
```
con <- apply(dat_log[,1:3],1,mean)
trt <- apply(dat_log[,4:6],1,mean)

con_max <- apply(dat_max_log[,1:3],1,mean)
trt_max <- apply(dat_max_log[,4:6],1,mean)
```

6. The log fold change is then the difference between condition 1 and condition 2. Plot a histogram to look at the distribution.

```
# Calculate fold change
dat_fc <- con - trt
dat_max_fc <- con_max - trt_max

# Plot histograms
hist(dat_fc)
```

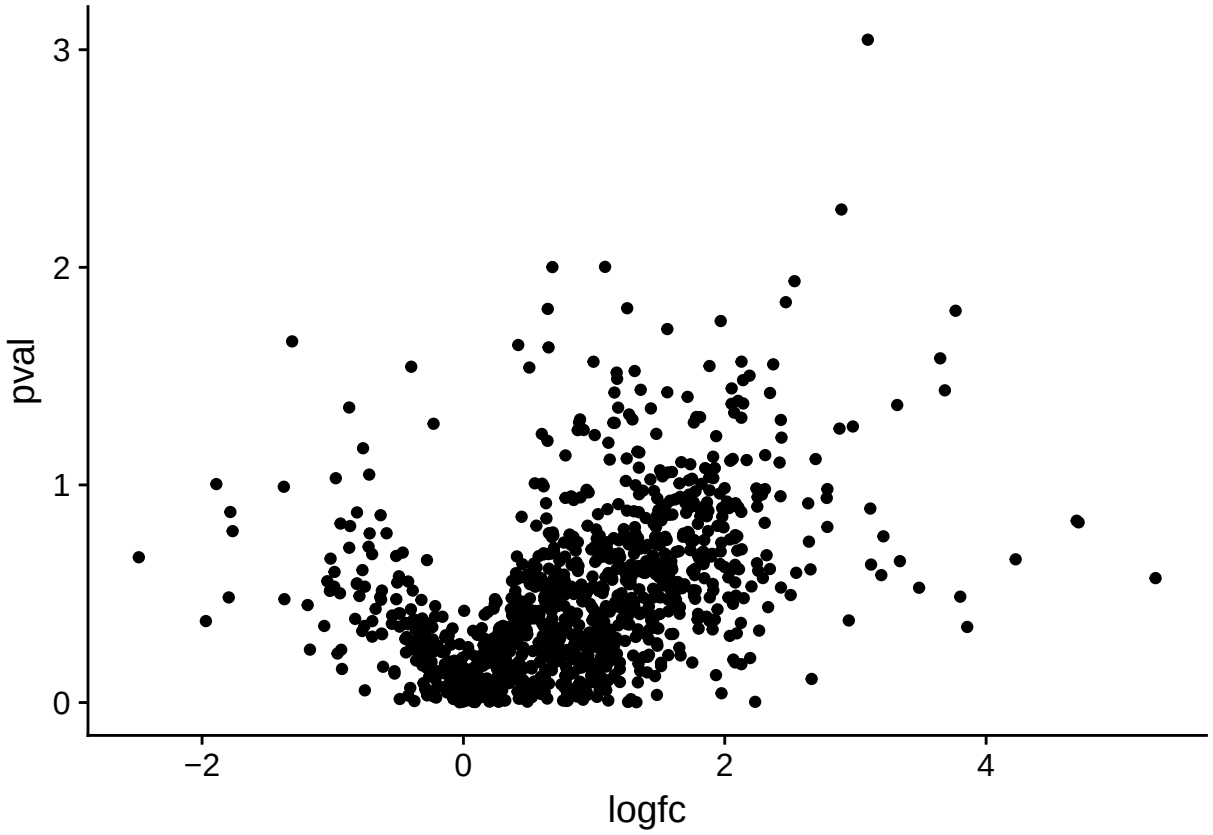


```
hist(dat_max_fc)
```

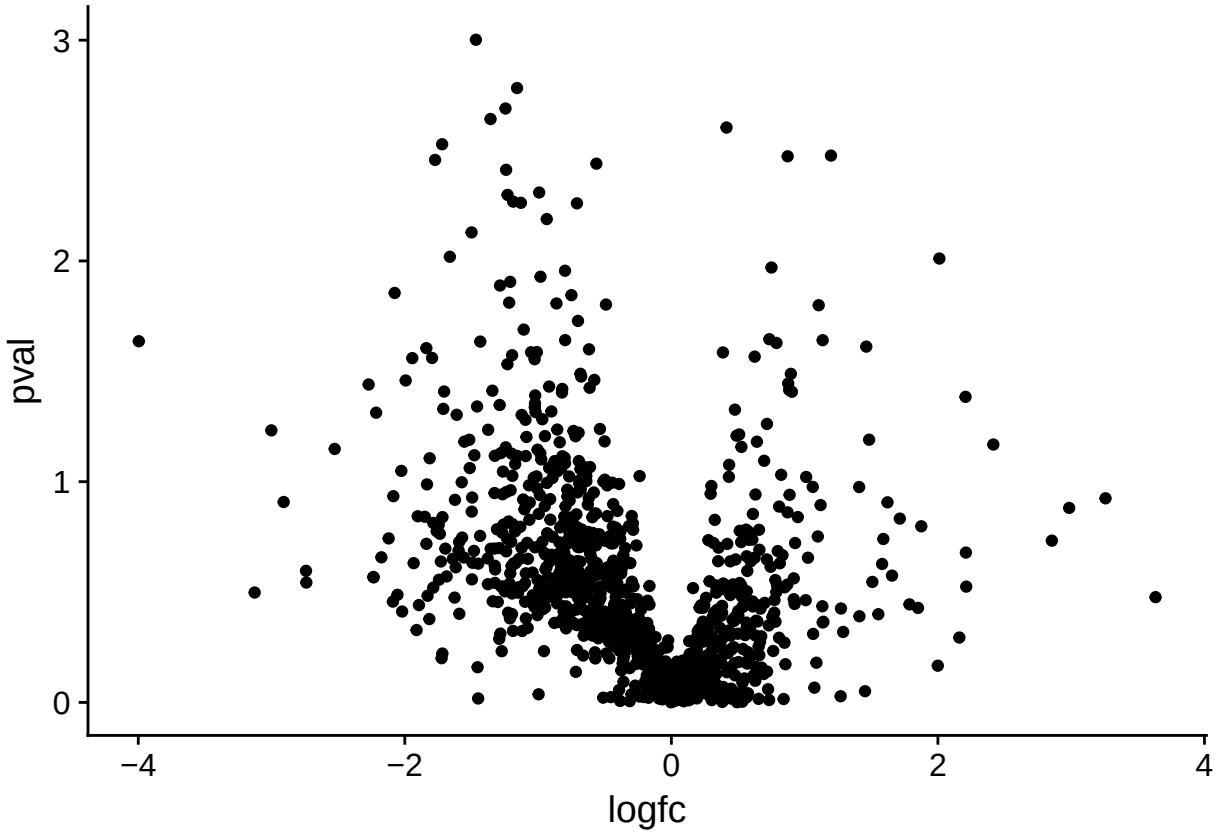


7. Create a combined table of log fold change and p-values for all the proteins for plotting a volcano plot.

```
dat <- data.frame(prots= dat_norm$protein_accession,
                  logfc = dat_fc,
                  pval = -1*log10(p_vals))

dat_max <- data.frame(prots= dat_norm_max$protein_accession,
                  logfc = dat_max_fc,
                  pval = -1*log10(p_vals_max))

dat_max %>% ggplot(aes(logfc,pval)) + geom_point()
```




```
dat %>% ggplot(aes(logfc,pval)) + geom_point()
```


3
2
1
0
pval
−4
−2
0
2
4
logfc

Chapter 5

Visualising proteomics data

Based on empircal research, there are some general rules on visulisations that are worth bearing in mind:

1. Plot

5.1 Creating a volcano plot

VOlcano plot is a plot of the log fold change in the observation between two conditions on the x-axis, for example the protein expression between treatment and control conditions. And on the y-axis is the corresponding p-value for each observation. The p-value representing the likelihood that an observed change is due to the different conditions rather than arising from natural variation in the fold change that might be observed if we performed many replications of the experiment.

The aim of a volcano plot is to enable the viewer to quickly see the effect (if any) of an experiment with two conditions on many species i.e. proteins in terms of both increased and decreased expression.

Like all plots it has it's good and bad points, namely it's good that we can visualise a lot of complex information in one plot. However this is also it's main weakness, it's rather complicated to understand in one glance.

However, volcano plots are widely used in the literature, so there may be an amount of social proof giving rise to their popularity as opposed to their utility.

5.2 Creating a heatmap

A

Chapter 6

Going further

6.1 Getting help and joining the R community

6.2 Communication: creating reports, presentations and websites

References

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.7.