ImpactTB/BAA: Standard Operating Procedures for Data Analysis

Colorado State University Coding Team

2024-06-21

Contents

Chapter 1

Overview

1.1 About the project

The objective of the Immune Mechanisms of Protection against Mycobacterium tuberculosis (IMPAc-TB) program is to get a thorough understanding of the immune responses necessary to avoid initial infection with *Mycobacterium tuberculosis (Mtb)*, formation of latent infection, and progression to active TB illness. To achieve these goals, the National Institute of Allergy and Infectious Diseases awarded substantial funding and established multidisciplinary research teams that will analyze immune responses against *Mtb* in animal models (mice, guinea pigs, and non-human primates) and humans, as well as immune responses elicited by promising vaccine candidates. The contract awards establish and give up to seven years of assistance for IMPAc-TB Centers to explain the immune responses required for *Mtb* infection protection.

The seven centers that are part of the study are (in alphabetical order):

1. Colorado State University
2. Harvard T.H. Chan School of Public Health
3. Seattle Children Hospital
4. [more]

Colorado State University Team and role of each member:

- Dr. Marcela Henao-Tamayo: Principal Investigator
- Dr. Brendan Podell: Principal Investigator
- Dr. Andres Obregon-Henao: Research Scientist-III
- Dr. Taru S. Dutt: Research Scientist-I
- [more]

1.2 About this book

The aim of this book is to provide data protocols and data collection templates for key types of data that are collected over the course of this project. By using standard templates to record data, as well as starting from defined pipelines to process and analyze the data, we aim to standardize the collection and processing of data across this project.

Here, we have built a comprehensive guide to wet lab data collection, sample processing, and computational tool creation for robust and efficient data analysis and dissemination.

Chapter 2

Experimental metadata

Metadata for an experiment:

- species
- start_date
- end_date
- experimental_groups

Chapter 3

Animal initial conditions and weekly weights

3.0.1 Downloads

The downloads for this chapter are:

- Data collection template for collecting initial information about the experimental animals and regular weight measurements, cage changes, and adverse events throughout the experiment
- Report template to process data collected with the template (when you go to this link, go to the "File" bar in your browser's menu bar, chose "Save As", then save the file as "animal_weights.Rmd")
- Example output from the report template

3.0.2 Overview

We use the template in this section to record information about each animal used in the experiment. This includes the species, sex, and experimental group. It also includes some information to identify the animal, which in the case of mice includes a code describing the pattern of notches put in the mouse's ear and the cage that the animal is assigned to at the beginning of the experiment. These are all values that can be determined at the start of the experiment, when the mice are first assigned to groups.

This template is also used to record some data over the course of the experiment. This includes adverse events and cases where an animal is moved from one cage to another during the experiment.

In addition, in our experiments, we are measuring the mice every week to record their weight over the course of the experiment. This weight measuring begins before the first vaccination and continues through until the last mouse is sacrificed. We have used ear notches to identify each mouse, and between the ear notch and the mouse's cage number, we can uniquely track each mouse in the study.

There are a few reasons that we are measuring these mouse weights. The first is to help us manage the mice, particularly in terms of animal welfare. If there are mice that are losing a lot of weight, that can be an indication that they may need to be euthanized. For example, some animal care standards consider that an adult animal that has lost 20% or more of its weight compared to its baseline weight is indicating a clear sign of morbidity or suffering.

A second reason is that the weight measure might provide a record of each mouse's general health over the course of the study. In the study, mice are weighed in grams weekly to monitor clinical status, as one potential sign of tuberculosis infection and severity is weight loss.

In humans, tuberculosis patients frequently display weight loss as a clinical symptom associated with disease progression. In particular, extreme weight loss and loss of muscle mass, also known as cachexia, can present as a result of chronic inflammatory illnesses like tuberculosis (Baazim et al., 2022). This cachexia is part of a systemic response to inflammation, and in humans has been linked to upregulation of pro-inflammatory cytokines including tumor necrosis factor, interleukin-6, and interferon-gamma (Baazim et al., 2022). Additionally, studies support a role in cachexia of key immune cell populations such as cytotoxic T-cells which, when depleted, counteract muscle and fat deterioration (Baazim et al., 2019), suggest that thsi type of T-cells may metabolically reprogram adipose tissue.

Given these relationships between weight loss, diseases, and immune processes, it is possible that mouse weight might provide a regularly measurable insight into the severity of disease in each animal. While many of data points are collected to measure the final disease state of each animal, fewer are available before the animal is sacrificed. We are hoping that mouse weights will provide one measure that, while it may not perfectly capture disease severity, may provide some information throughout the experiment that is correlated to disease severity at regular time intervals.

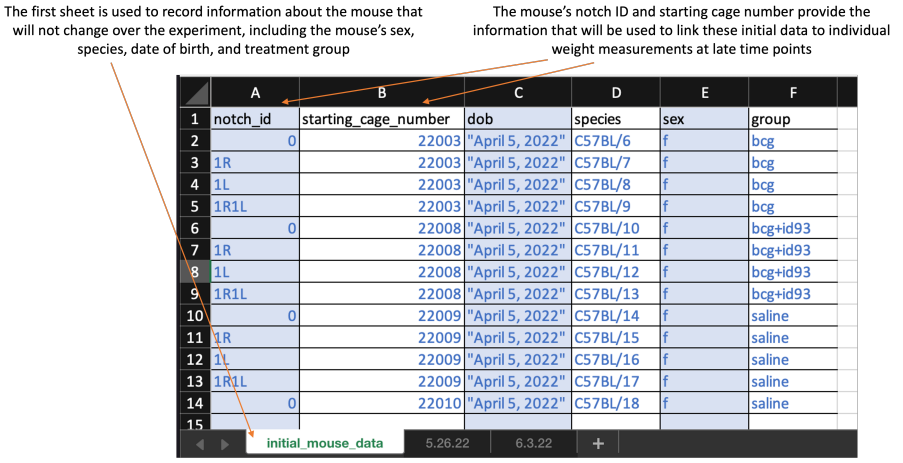
Other studies that use a mouse model of tuberculosis have collected mouse weights, as well (Smith et al., 2022; Segueni et al., 2016). We plan to investigate these data to visualize the trajectory of weight gain / loss in each mouse both before and after they are challenged with tuberculosis. We also plan to test whether each mouse's weight change after challenge is correlated with other metrics of the severity of disease and immune response. We will do this by testing the correlation between the percent change in weight between challenge and sacrifice with CFUs at sacrifice as well as expression of cytokines and other

biological markers (Smith et al., 2022).

3.0.3 Template description

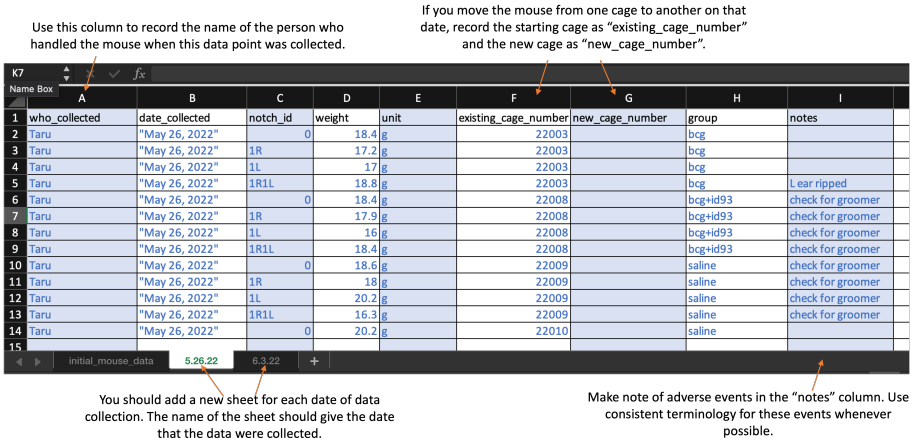
Both the animals' initial conditions and their weekly measures (adverse events, cage changes, and weights) should be recorded in an excel worksheet. You can download a copy of the template here.

The worksheet is divided into sheets. The first sheet is recorded at the first time point when the mice are measured and is used to record information about the mice that will remain unchanged over the course of the study, like species and sex. Here is what the first sheet of the template looks like:



	A	B	C	D	E	F
1	notch_id	starting_cage_number	dob	species	sex	group
2	0	22003	"April 5, 2022"	C57BL/6	f	bcg
3	1R	22003	"April 5, 2022"	C57BL/7	f	bcg
4	1L	22003	"April 5, 2022"	C57BL/8	f	bcg
5	1R1L	22003	"April 5, 2022"	C57BL/9	f	bcg
6	0	22008	"April 5, 2022"	C57BL/10	f	bcg+id93
7	1R	22008	"April 5, 2022"	C57BL/11	f	bcg+id93
8	1L	22008	"April 5, 2022"	C57BL/12	f	bcg+id93
9	1R1L	22008	"April 5, 2022"	C57BL/13	f	bcg+id93
10	0	22009	"April 5, 2022"	C57BL/14	f	saline
11	1R	22009	"April 5, 2022"	C57BL/15	f	saline
12	1L	22009	"April 5, 2022"	C57BL/16	f	saline
13	1R1L	22009	"April 5, 2022"	C57BL/17	f	saline
14	0	22010	"April 5, 2022"	C57BL/18	f	saline
15						

The second and later sheets are used to record the weight at each measured timepoint. The second sheet will record the weights on the first date they are measured, so it should be recorded at the same time as the first sheet—with initial mouse information—is completed. Here is what the first sheet of the template looks like:



	A	B	C	D	E	F	G	H	I
1	who_collected	date_collected	notch_id	weight	unit	existing_cage_number	new_cage_number	group	notes
2	Taru	"May 26, 2022"	0	18.4	g	22003		bcg	
3	Taru	"May 26, 2022"	1R	17.2	g	22003		bcg	
4	Taru	"May 26, 2022"	1L	17	g	22003		bcg	
5	Taru	"May 26, 2022"	1R1L	18.8	g	22003		bcg	L ear ripped
6	Taru	"May 26, 2022"	0	18.4	g	22008		bcg+id93	check for groomer
7	Taru	"May 26, 2022"	1R	17.9	g	22008		bcg+id93	check for groomer
8	Taru	"May 26, 2022"	1L	16	g	22008		bcg+id93	check for groomer
9	Taru	"May 26, 2022"	1R1L	18.4	g	22008		bcg+id93	check for groomer
10	Taru	"May 26, 2022"	0	18.6	g	22009		saline	check for groomer
11	Taru	"May 26, 2022"	1R	18	g	22009		saline	check for groomer
12	Taru	"May 26, 2022"	1L	20.2	g	22009		saline	check for groomer
13	Taru	"May 26, 2022"	1R1L	16.3	g	22009		saline	check for groomer
14	Taru	"May 26, 2022"	0	20.2	g	22010		saline	
15									

As you continue to measure at new timepoints, you should add a sheet at each timepoint, with each new sheet following the format of the second sheet in the template. The second and later sheets should be labeled with the date when those weights were measured (e.g., "5.26.22" for weights measured on May 26, 2022).

When you download the template, it will have example values filled out in blue. Use these to get an idea for how to record your own data. When you are ready to record your own data, delete these example values and replace them with data collected from your own experiment.

Column titles are as follows. First, in the first sheet, you will record:

- `notch_id`: Record the ear notch pattern in the mouse. Make sure that you record consistently across all timepoints, so that each mouse can be tracked across dates. If you are doing single notches, for example, this might be "0" for no notches, "1R" for one notch in the right ear, "1L" for one notch in the left ear, and "1R1L" for one notch in each ear.
- `starting_cage_number`: Record the number of the cage that the mouse is put into at the start of the experiment. In combination with the mouse's `notch_id`, this will provide a unique identifier for each mouse at the start of the experiment.
- `dob`: Record the date the mouse was born.
- `species`: Record the species of the mouse (e.g., "C57BL/6" for C57 black 6 mice or "CBA" for CBA mice).
- `sex`: Record as "m" for male or "f" for female
- `group`: Provide the experimental group of the mouse. Be sure that you use the same abbreviation or notation across each timepoint. Examples of group designations might be: bcg, saline, bcg+id93, saline+id93, saline+noMtb

For the second and later sheets, you will record:

- `who_collected`: Record the first name of the person who actually handled the mouse from the scale.
- `date_collected`: Record the date using quotation marks, with the month, then day, then year. For example, "May 31, 2022".
- `weight`: Record as a number, without a unit in this column. The next column will be used for the units.

- `unit`: Provide the units that were used to take the weight (e.g., "g" for grams). Be consistent across all animals and timepoints in the abbreviation that you use (e.g., always use "g" for grams, not "g" sometimes and "grams" sometimes)
- `existing_cage_number`: Provide the cage number that the mouse is in when you start weighing at that time point. If the mouse is moved to another cage on this day, you will specify that in the next column. If the animal was moved from one cage to another between the last weighing and the date of the timepoint you are measuring, put in this column the cage number that the animal was in the last time it was weighed.
- `new_cage_number`: If the animal is moved to a new cage on the date of the timepoint you are measuring, then use this column to record the number of the cage you move it too. Similarly, if the animal moved cages between the last measured timepoint and this one, use this column to record the cage it was moved to. Otherwise, if the animal stays in the same cage that it was at the last measured time point, leave this column empty.
- `group`: Provide the experimental group of the mouse. Be sure that you use the same abbreviation or notation across each timepoint. Examples of group designations might be: bcg, saline, bcg+id93, saline+id93, saline+noMtb
- `notes`: Record information regarding clinical observations (e.g., "back is balding", "barbering", "excessive grooming", "euthanized").

3.0.4 Processing collected data

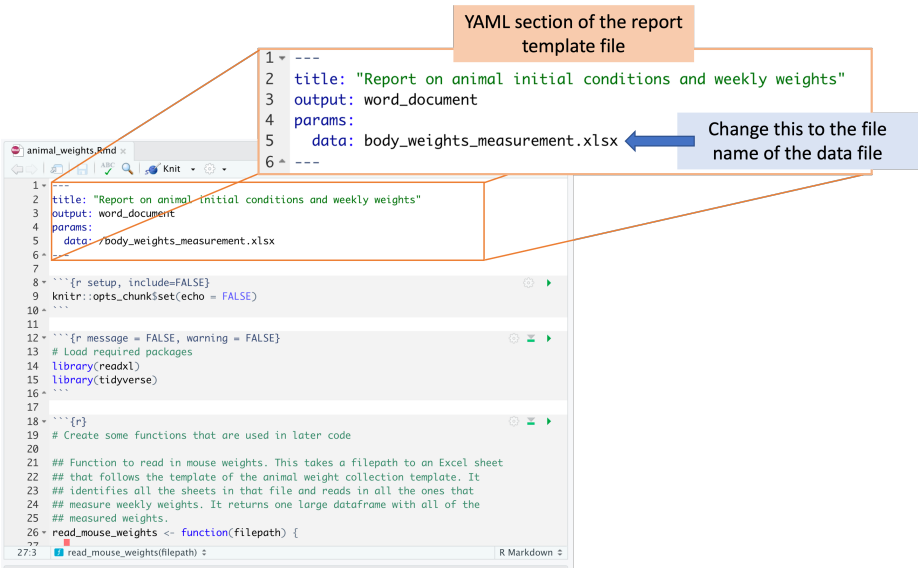
Once data are collected, the file can be run through an R workflow. This workflow will convert the data into a format that is easier to work with for data analysis and visualization. It will also produce a report on the data in the spreadsheet, and ultimately it will also write relevant results in a format that can be used to populate a global database for all experiments in the project.

The next section provides the details of the pipeline. It aims to explain the code that processes the data and generates visualizations. You do not need to run this code step-by-step, but instead can access a script with the full code here.

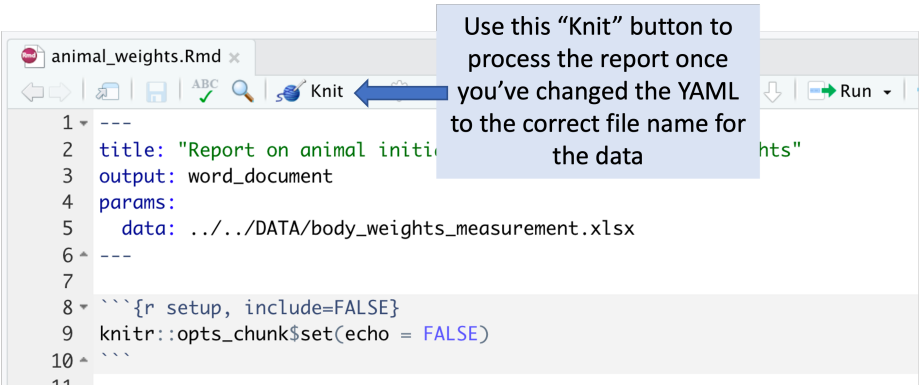
To use this reporting template, you need to download it to your computer and save it in the file directory where you saved the data you collected with the data collection template. You can then open RStudio and navigate so that you are working within this directory. You should also make sure that you have

installed a few required packages on R on the computer you are using to run the report. These packages are: `tidyverse`, `purrr`, `lubridate`, `readxl`, `knitr`, and `ggbeeswarm`.

Within RStudio, open the report template file. There is one spot where you will need to change the code in the template file, so it will read in the data from the version of the template that you saved, which you may have renamed. In the YAML of the report template file, change the file path beside "data:" so that it is the file name of your data file.



Once you've made this change, you can use the "Knit" button in RStudio to create a report from the data file and the report template file.



The report includes the following elements:

- Summary table of animals at the start of the experiment

- Time series plots of animal weights over the experiment, grouped by experimental group
- Boxplots of the distribution of animal weights within each experimental group at the last available time point
- Plot of measured weight, identified by the person who was handling the animal, to help determine if there are consistent differences by handler
- Table of all the animals in the experiment at the last measured time point, ordered by their weight change since the previous measurement. This table is meant to help in identifying animals that may need to be euthanized for animal welfare reasons.

You can download an example of a report created using this template by clicking here.

When you knit to create the report, it will create a Word file in the same file directory where you put your data file and report template. It will also create and output a version of the data that has been processed (in the case of the weights data, this mainly involves tracking mice as they change cages, to link all weights that are from a single animal). This output fill will be named "mouse_weights_output.csv" and, like the report file, will be saved in the same file directory as the data file and the report template.

3.0.5 Details of processing script

This section goes through the code within the report template. It explains each part of the code in detail. You do not need to understand these details to use the report template. However, if you have questions about how the data are being processed, or how the outputs are created, all those details are available in this section.

As a note, there are two places in the following code where there's a small change compared to the report template. In the report, you incorporate the path to the data file using the `data:` section in the YAML at the top of the document. In the following code, we've instead used the path of some example data within this book's file directory, so the code will run for this chapter as well.

First, the workflow loads some additional R libraries. You may need to install these on your local R session if you do not already have them installed.

```
library(readxl)
library(tidyverse)
library(ggbeeswarm)
```

These packages bring in some useful functions that are not available in the base installation of R. They are all open source. To cite any of them, you can use

the `citation` function. For example, to get the information you would need to cite the `readxl` package, in R you can run:

```
citation("readxl")
```

```
## To cite package 'readxl' in publications use:
##
##   Wickham H, Bryan J (2023). _readxl: Read Excel Files_. R package
##   version 1.4.3, <https://CRAN.R-project.org/package=readxl>.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {readxl: Read Excel Files},
##     author = {Hadley Wickham and Jennifer Bryan},
##     year = {2023},
##     note = {R package version 1.4.3},
##     url = {https://CRAN.R-project.org/package=readxl},
##   }
```

Next, the code in the report template creates a few custom functions to help process the data from the data collection template. The first of these functions checks the data collection template to identify all the timepoints that were collected and then reads each in, ultimately joining data from all time points into one large dataset.

The data collection template requires you to use a new sheet in the spreadsheet for each weight collection time point, with a first sheet that records initial information about the animals. If you only take weights at three time points, there would only be three time point sheets in the final file. Conversely, if you collect weight data at twenty time points, there would be twenty sheets in the final file. The first function, called “, reads the data file, checks to find all the weight recording sheets, whether it’s three or twenty, and then reads the data in from all the sheets and binds them together into a single dataframe.

```
## Function to read in mouse weights. This takes a filepath to an Excel sheet
## that follows the template of the animal weight collection template. It
## identifies all the sheets in that file and reads in all the ones that
## measure weekly weights. It returns one large dataframe with all of the
## measured weights.
read_mouse_weights <- function(filepath) {

  # getting info about all excel sheets
  mouse_weights_sheets <- readxl::excel_sheets(filepath)[-1] # First sheet is initial
```

```
  mouse_weights <- purrr::map(mouse_weights_sheets,
                              ~ readxl::read_excel(filepath, sheet = .x,
                                                   col_types = c("text",    # who_collected
                                                                 "text",    # date_collected
                                                                 "text",    # notch_id
                                                                 "numeric", # weight
                                                                 "text",    # unit
                                                                 "text",    # existing_cage_number
                                                                 "text",    # new_cage_number
                                                                 "text",    # group
                                                                 "text"     # notes
                                                                 ))) %>%
    dplyr::bind_rows() %>%
    mutate(date_collected = lubridate::mdy(date_collected))

  return(mouse_weights)
}
```

The remaining functions are all functions to help track a mouse over the experiment even if it changes cages. In processing this data, the key challenge is to track a single mouse over the experiment. The mice are identified by a pattern of notches in their ears. However, there are a limited number of notches that can be distinguished, so the notch information does not distinctly identify every mouse in the study, just every mouse within a certain cage. By knowing both an ear notch ID and a cage number, you can distinctly identify each mouse in the study.

However, mice are moved from one cage to another in some cases during a study. If mice within a cage are fighting, or if they are showing signs of excessive grooming, these can be reasons to move a mouse to a new cage once the experiment has started. The cage moves need to be resolved when processing the data so that each mouse can be tracked even as they move.

In the data collection template, we have created a design that aims to include information about cage moves, but to do so in a way that is as simple as possible for the person who is recording the data. The weights are recorded for each time point in a separate sheet of the data collection template. On the sheet for a time point, there are also columns to give the mouse's cage at the start of that data collection time point, as well as the cage the mouse was moved to, if it was moved. The report template code then uses this information to create a unique ID for each mouse (one that is constant across the experiment), and then attach it to the mouse's measurements even as the mouse is moved from one cage to another. The following two functions both help with this process:

```
# Function to get the next cage number based on the
# existing cage number and notch ID. If the mouse does not
```

```
# switch cages again, the output is a vector of length 0.
# This takes the dataframe and existing identifiers (notch id and
# existing cage number) as inputs. It returns the next cage
# that the mouse was moved to. If the mouse has not moved
# from the existing case, the output has length 0.
get_next_cage <- function(existing_cage_number, notch_id,
                          df = our_mouse_weights){
  next_cage <- df %>%
    filter(.data$existing_cage_number == {{existing_cage_number}} &
             .data$notch_id == {{notch_id}} &
             !is.na(.data$new_cage_number)) %>%
    pull(new_cage_number)

  return(next_cage)
}

# Function to get the full list of cages for each individual
# mouse, over the course of all data collected to date. This
# inputs the starting identifiers of the mouse (starting cage ID
# and notch ID). It then works through any cage changes to create
# a list for that mouse of all cages it was put in over the
# course of the experiment.
get_mouse_cages <- function(mouse_starting_cage, mouse_notch_id,
                            df = our_mouse_weights){
  mouse_cage_list <- mouse_starting_cage
  i <- 1

  while(TRUE){
    next_cage <- get_next_cage(existing_cage_number =
                               mouse_cage_list[i],
                               notch_id = mouse_notch_id,
                               df = df)
    if(length(next_cage) == 0) {
      break
      }
    i <- i + 1
    mouse_cage_list[i] <- next_cage
    }

  return(mouse_cage_list)
}
```

Next, the report template code gets to the workflow itself, where it uses both these custom functions and other R code to process the data and then to provide summaries and visualizations of the data.

The first step in the workflow is to read in the data from the spreadsheet. As long as the data are collected following the template that was described earlier, this code should be able to read it in correctly and create a master dataset with the data from all sheets of the spreadsheet. This step of the pipeline uses one of the custom functions that was defined at the start of the report template code:

```
# Read in the mouse weights from the Excel template. This creates one large
# dataframe with the weights from all the timepoints.
our_mouse_weights <- read_mouse_weights(filepath =
                                          "DATA/body_weights_measurement.xlsx")
```

Next, the code runs through a number of steps to create a unique ID for each mouse and then apply that ID to each time point, even if a mouse changes cages.

```
# Add a unique mouse ID for the first time point. This will become each mouse's
# unique ID across all measured timepoints.
our_mouse_weights <- our_mouse_weights %>%
  mutate(mouse_id = 1:n(),
         mouse_id = ifelse(date_collected ==
                                  first(date_collected),
                                mouse_id,
                                NA))

# Create a dataframe that lists all mice at the first time point,
# as well as a list of all the cages they have been in over the
# experiment
mice_cage_lists <- our_mouse_weights %>%
  filter(date_collected == first(date_collected)) %>%
  select(notch_id, existing_cage_number, mouse_id) %>%
  mutate(cage_list = map2(.x = existing_cage_number,
                          .y = notch_id,
                          .f = ~ get_mouse_cages(.x, .y, df = our_mouse_weights)))

# Add a column with the latest cage to the weight dataframe
our_mouse_weights$latest_cage <- NA

# Loop through all the individual mice, based on mice with a
# measurement at the first time point. Add the unique ID for
# each mouse, which will apply throughout the experiment. Also
# add the most recent cage ID, so the mouse can be identified
# by lab members based on it's current location
for(i in 1:nrow(mice_cage_lists)){
  this_notch_id <- mice_cage_lists[i, ]$notch_id
  this_cage_list <- mice_cage_lists[i, ]$cage_list[[1]]
  this_unique_id <- mice_cage_lists[i, ]$mouse_id
```

```
  latest_cage <- this_cage_list[length(this_cage_list)]

  our_mouse_weights$mouse_id[our_mouse_weights$notch_id == this_notch_id &
                       our_mouse_weights$existing_cage_number %in%
                       this_cage_list] <- this_unique_id

  our_mouse_weights$latest_cage[our_mouse_weights$notch_id == this_notch_id &
                       our_mouse_weights$existing_cage_number %in%
                       this_cage_list] <- latest_cage
}

# Add a label for each mouse based on its notch_id and latest cage
our_mouse_weights <- our_mouse_weights %>%
  mutate(mouse_label = paste("Cage:", latest_cage,
                             "Notch:", notch_id))
```

Ultimately, this creates both a unique ID for each mouse (in a column of the dataframe called `mouse_id`), as well as creates a unique label that can be used in plots and tables (given in the `mouse_label` column). The unique ID is set at the beginning of the study for each mouse and remains the same throughout the study. The label, on the other hand, is based on the mouse's ear notch pattern and the most recent cage it was recorded to be in. We made this choice for a labeling identifier, because it will help the researchers to quickly identify a mouse in the study based on it's current, rather than starting, cage.

The next part of the code reads in the initial data that were recorded for each animal in the experiment. The code then pulls in information from the processed weights dataset to match these initial data with each animals unique ID. Ultimately, these starting data are incorporated into the large dataset of mouse weights, creating a single large dataset to work with (`our_mouse_weights`) that includes all the information that was recorded in the data collection template.

```
# Read in the data from the original file with the initial animal
# characteristics
mouse_initial <- readxl::read_excel("DATA/body_weights_measurement.xlsx",
                                    sheet = 1,
                                    col_types = c("text", # notch_id
                                                  "text", # starting_cage_number
                                                  "text", # dob
                                                  "text", # species
                                                  "text", # sex
                                                  "text" # group
                                                  )) %>%
  mutate(dob = lubridate::mdy(dob),
         sex = forcats::as_factor(sex))
```

```
# Figure out the starting cage for each mouse, so they can be incorporated
# with the initial data so we can get the mouse ID that was added for the
# starting time point
mouse_ids <- our_mouse_weights %>%
  filter(date_collected == first(date_collected)) %>%
  select(notch_id, existing_cage_number, mouse_id) %>%
  rename(starting_cage_number = existing_cage_number)

# Merge in the mouse IDs with the dataframe of initial mouse characteristics
mouse_initial <- mouse_initial %>%
  left_join(mouse_ids, by = c("notch_id", "starting_cage_number"))

# Join the initial data with the weekly weights data into one large dataset
our_mouse_weights <- our_mouse_weights %>%
  left_join(mouse_initial, by = c("mouse_id", "notch_id", "group"))
```

At this point, the first few rows of this large dataset look like this:

```
our_mouse_weights %>%
  slice(1:5)
```

```
## # A tibble: 5 x 16
##   who_collected date_collected notch_id weight unit  existing_cage_number
##   <chr>         <date>         <chr>     <dbl> <chr> <chr>
## 1 Taru          2022-05-26     0          18.4 g     22003
## 2 Taru          2022-05-26     1R         17.2 g     22003
## 3 Taru          2022-05-26     1L         17   g     22003
## 4 Taru          2022-05-26     1R1L       18.8 g     22003
## 5 Taru          2022-05-26     0          18.4 g     22004
## # i 10 more variables: new_cage_number <chr>, group <chr>, notes <chr>,
## #   mouse_id <int>, latest_cage <chr>, mouse_label <chr>,
## #   starting_cage_number <chr>, dob <date>, species <chr>, sex <fct>
```

The rest of the code in the report template will create summaries and graphs of the data. First, there is some code that provides summaries of the research animals at the start of the experiment. It uses the `mouse_initial` dataset (which pulled in data from the first sheet of the data collection template). It uses a `summarize` call to summarize details from this sheet of data, including the species of the animal, the total number of animals, how many were males versus females, and which experimental groups were included. It uses some additional code to format the data so the resulting table will be clearer, and then uses the `kable` function to output the results as a nicely formatted table.

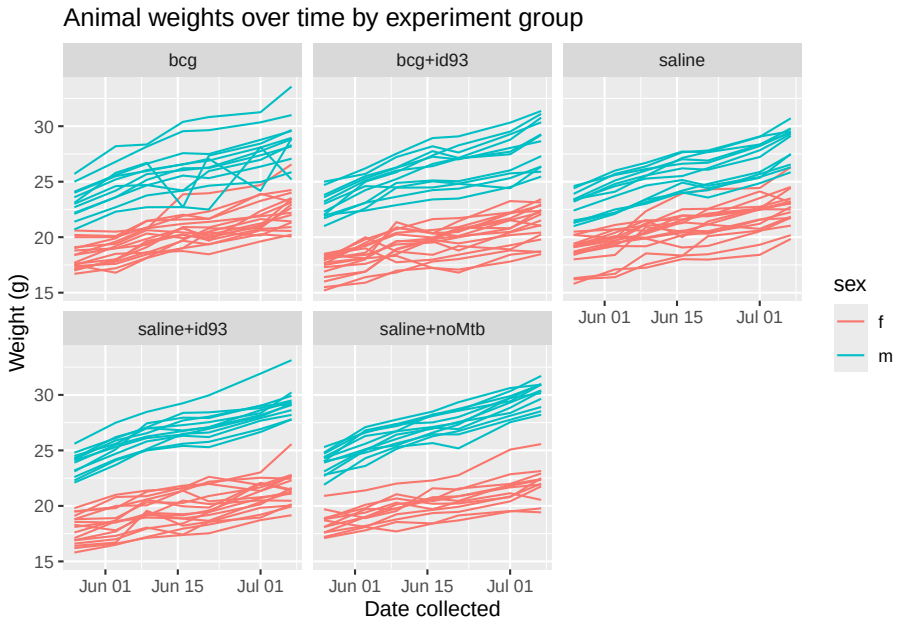
Table 3.1: Summary of experimental animals at the start of the experiment

Species:	C57BL/6
Total animals:	140
Sex distribution:	male: 60, female: 80
Experimental groups:	bcg, bcg+id93, saline, saline+id93, saline+noMtb
N. of starting cages:	34

```
# Create a table that summarizes the animals at the start of the experiment
mouse_initial %>%
  summarize(Species = paste(unique(species), collapse = ", "),
            `Total animals` = n(),
            `Sex distribution` = paste0("male: ", sum(sex == "m"),
                                        ", female: ", sum(sex == "f")),
            `Experimental groups` = paste(unique(group), collapse = ", "),
            `N. of starting cages` =
              length(unique(starting_cage_number))) %>%
  mutate_all(as.character) %>%
  pivot_longer(everything()) %>%
  mutate(name = paste0(name, ":")) %>%
  knitr::kable(col.names = c("", ""),
               caption = "Summary of experimental animals at the start of the experimer
               align = c("r", "l"))
```

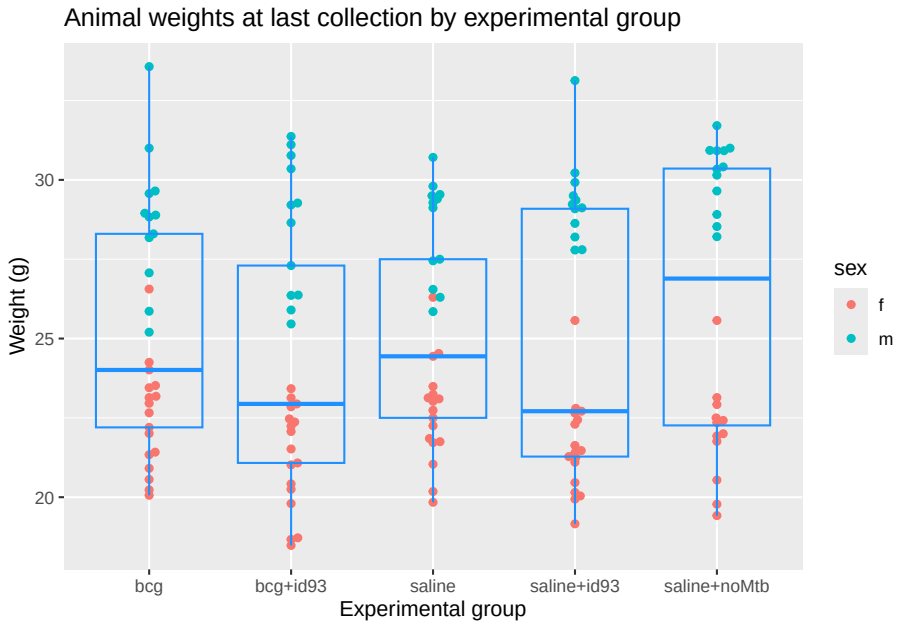
The next piece of code creates a time series of mouse weights over time. The points for each mouse are connected to create a line, so it's easy to see both variation across mice at a single time point and variation in a single mouse over the study. The lines are colored to distinguish male from female mouse (and there is a clear difference in average weights in the two groups). The plot is faceted so that the time series for mice in each experimental group are shown in different small "facets" of the plot, but with the same axis ranges used on each small plot to help comparisons across plots.

```
# Create a plot of mouse weights over time
our_mouse_weights %>%
  ggplot(aes(x = date_collected, y = weight,
             group = mouse_id, color = sex)) +
  geom_line() +
  facet_wrap(~ group) +
  ggtitle("Animal weights over time by experiment group") +
  labs(x = "Date collected",
       y = "Weight (g)")
```



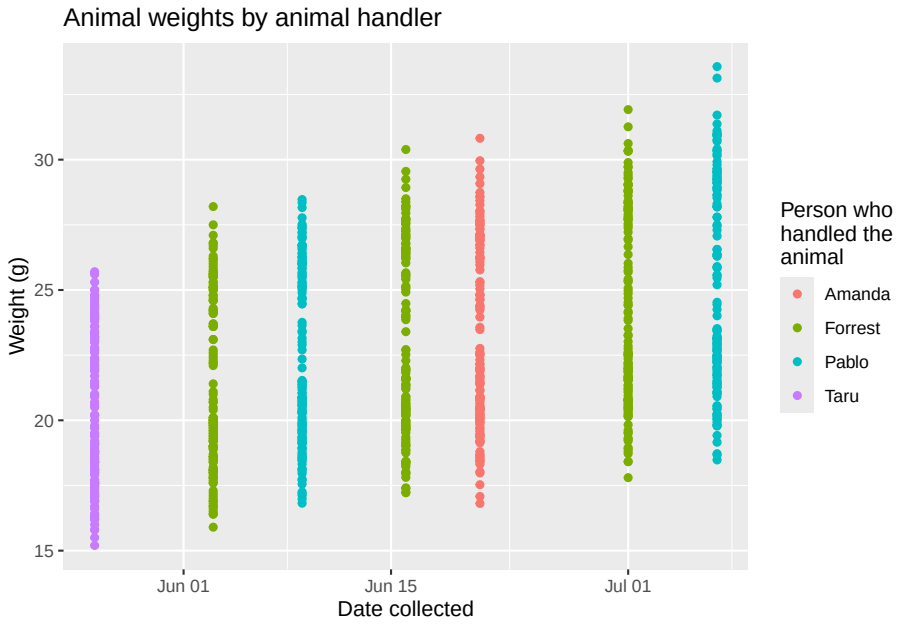
Next, the code creates boxplots that focus on differences in weights at the latest available timepoint. One boxplot is created for each experimental group, and the points for individual mice are shown behind the boxplot, to provide a better idea of the pattern of variation in individual mice. These points are colored based on sex, to help explore patterns by sex.

```
# Plot animal weight boxplots for the latest time point
our_mouse_weights %>%
  filter(date_collected == last(date_collected)) %>%
  ggplot(aes(x = group, y = weight)) +
  geom_beeswarm(aes(color = sex)) +
  geom_boxplot(fill = NA, color = "dodgerblue") +
  ggtitle("Animal weights at last collection by experimental group") +
  labs(x = "Experimental group",
       y = "Weight (g)")
```



The next piece of code shows how mouse weights vary by the person who was handling the animals at a certain time point. Different handlers may have small differences in how they handle and weight the mice. If there are noticable differences in the measured weights, this is something that could be corrected through statisical modeling in later analysis, so we included it as a potential check.

```
# Plot animal weights by animal handler
our_mouse_weights %>%
  ggplot(aes(x = date_collected, y = weight, color = who_collected)) +
  geom_point() +
  ggtitle("Animal weights by animal handler") +
  labs(x = "Date collected",
       y = "Weight (g)",
       color = "Person who\nhandled the\nanimal")
```



The next piece of code creates a table with each of the animals that was still tracked at the last time point (if animals were sacrificed prior to the last recorded time point, they would not be included here). This table focuses on the weight change since the previous measured time point. It is ordered by the change in weight, from the largest decrease to the largest increase. It is meant as an aide in identifying mice that are showing signs of suffering and may need to be considered for being euthanized. The animals are labeled in this table by their most recent cage location, so it will be easier to find them if necessary. For this example code, we've shown only a sample of 15 animals, but the report will show data for all animals.

```
# Create table of animal weight changes since previous time point
our_mouse_weights %>%
  select(date_collected, weight, group, mouse_label, sex) %>%
  group_by(mouse_label) %>%
  mutate(weight_change = (weight - lag(weight)) / lag(weight)) %>%
  ungroup() %>%
  filter(date_collected == last(date_collected)) %>%
  mutate(formatted_weight_change = paste0(formatC(weight_change * 100,
                                                  digits = 1, format = "f"), "%")) %>%
  arrange(weight_change) %>%
  select(mouse_label, group, sex, weight, formatted_weight_change) %>%
  slice(1:15) %>%  # Only for the chapter--show a sample, not all
  knitr::kable(col.names = c("Mouse", "Experimental group", "Sex",
                             "Weight (g)", "Weight change since last measure"),
```

Table 3.2: Individual data on weight changes in mice between current measurement and previous measurement.

Mouse	Experimental group	Sex	Weight (g)	Weight change since last measu
Cage: 22021 Notch: 1R	bcg	m	25.20	-10.4%
Cage: 22017 Notch: 1R	saline	f	23.13	-4.1%
Cage: 22015 Notch: 1L	bcg	f	23.18	-3.1%
Cage: 22476 Notch: 1R1L	saline+noMtb	f	20.54	-2.8%
Cage: 22014 Notch: 1L	saline+id93	f	21.47	-2.8%
Cage: 22014 Notch: 0	saline+id93	f	21.40	-2.7%
Cage: 22006 Notch: 1R1L	bcg+id93	f	21.02	-2.4%
Cage: 22015 Notch: 0	bcg	f	20.06	-1.0%
Cage: 22004 Notch: 1R	bcg	f	21.42	-1.0%
Cage: 22006 Notch: 0	bcg+id93	f	18.67	-0.7%
Cage: 22476 Notch: 0	saline+noMtb	f	19.42	-0.6%
Cage: 22016 Notch: 1R1L	bcg+id93	f	23.13	-0.5%
Cage: 22012 Notch: 1R	saline+id93	f	22.44	-0.4%
Cage: 22015 Notch: 2L	bcg	f	20.56	-0.3%
Cage: 22013B Notch: 1L	saline+id93	f	20.46	-0.2%

```
                caption = "Individual data on weight changes in mice between current mea
```

As a last step, the code in the template writes a CSV file with the processed data. This file will be an input into a script that will format the data to add to a database where we are collecting and integrating data from all the CSU experiments, and ultimately from there into project-wide storage.

```
# Write out processed data into a CSV file
write_csv(our_mouse_weights, "mouse_weights_output.csv")
```

Chapter 4

Colony forming units to determine bacterial counts

4.0.1 Downloads

The downloads for this chapter are:

- Data collection template for recording colony forming units counted on each plate or section of plate in the laboratory
- Report template to process data collected with the data template (when you go to this link, go to the "File" bar in your browser's menu bar, chose "Save As", then save the file as "animal_weights.Rmd")
- Example output from the report template

4.0.2 Overview TEST

In the experiments, we will need to estimate the bacterial load of *Mycobacterium tuberculosis* in organs—including lungs and spleens—of animals from experiments. These measurements help us assess how well a vaccine has worked in comparison to controls.

We will be estimating bacterial load in an animal organ using the plate count method with serial dilutions. Serial dilutions allow you to create a highly diluted sample without needing a massive amount of diluent: as you increase the dilution one step at a time, you can steadily bring the samples down to lower bacterial loads per volume. This method is common across laboratories that study tuberculosis drug efficacy as a method for estimating bacterial load in animal organs (Franzblau et al., 2012) and is a well-established method across microbiology in general, dating back to Koch in the late 1800s (Wilson, 1922; Ben-David and Davidson, 2014).

With this method, we homogenize part of the organ, and then create several increasingly dilute samples. Each dilution is then spread on a plate with a medium in which *Mycobacterium tuberculosis* can grow and left to grow for several weeks at a temperature conducive to *Mycobacterium tuberculosis* growth. The idea is that individual bacteria from the original sample end up randomly spread across the surface of the plate, and any bacteria that are viable (able to reproduce) will form a new colony that, after a while, you'll be able to see (Wilson, 1922; Goldman and Green, 2015). At the end of this incubation period, you can count the number of these colony-forming units (CFUs) on each plate.

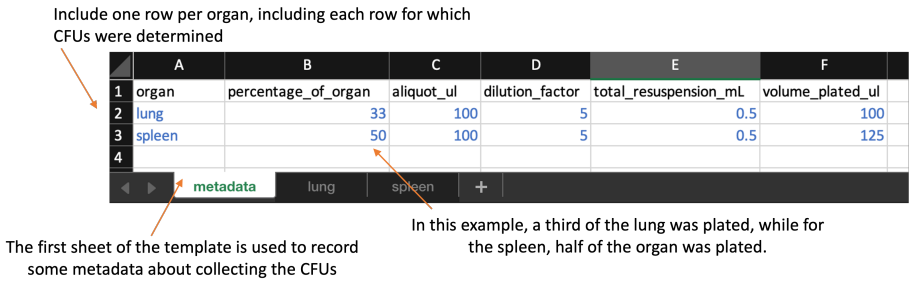
To count the number of CFUs, you need a "just right" dilution (and we often won't know what this is until after plating) to have a countable plate. If you have too high of a dilution (i.e., one with very few viable bacteria), randomness will play a big role in the CFU count, and you'll estimate the original with more variability, which isn't ideal. If you have too low of a dilution (i.e., one with lots of viable bacteria), it will be difficult to identify separate colonies, and they may compete for resources. (The pattern you see when the dilution is too low (i.e., too concentrated with bacteria) is called a lawn—colonies merge together).

Once you identify a good dilution for each sample, the CFU count from this dilution can be used to estimate the bacterial load in the animal's organ. To translate from diluted concentration to original concentration, you do a back-calculation, incorporating both the number of colonies counted at that dilution and how dilute the sample was (Ben-David and Davidson, 2014; Goldman and Green, 2015).

4.0.3 Template description

The data are collected in a spreadsheet with multiple sheets. The first sheet (named "metadata") is used to record some metadata for the experiment, while the following sheets are used to record CFUs counts from the plates used for samples from each organ, with one sheet per organ. For example, if you plated data from both the lung and spleen, there would be three sheets in the file: one with the metadata, one with the plate counts for the lung, and one with the plate counts for the spleen.

The first sheet, which is the metadata sheet, is shown below:

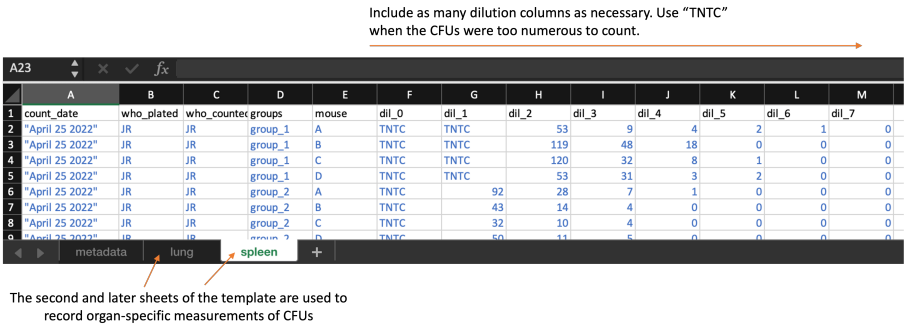


This metadata sheet is used to record information about the overall process of plating the data. Values from this sheet will be used in calculating the bacterial load in the original sample based on the CFU counts. This spreadsheet includes the following columns:

- `organ`: Include one row for each organ that was plated in the experiment. You should name the organ all in lowercase (e.g., "lung", "spleen"). You should use the same name to also name the sheet that records data for that organ for example, if you have rows in the metadata sheet for "lung" and "spleen", then you should have two other sheets in the file, one sheet named "lung" and one named "spleen", which you'll use to store the plate counts for each of those organs.
- `percentage_of_organ`: In this column, give the proportion of that organ that was plated. For example, if you plated half the lung, then in the "lung" row of this spread sheet, you should put 0.5 in the `prop_resuspended` column.
- `aliquot_ul`: 100 uL of the total_resuspended slurry would be considered an original aliquot and is used to peform serial dilutions.
- `dilution_factor`: Amount of the original stock solution that is present in the total solution, after dilution(s)
- `total_resuspended_mL`: This column contains an original volume of tissue homogenate. For example, raw lung tissue is homogenized in 0.5 mL of PBS in a tube containing metal beads.
- `volume_plated_ul`: Amount of suspension + diluent plated on section of solid agar

Following this first sheet in the file, you should have one sheet for each organ. The organs that you record in these sheets should match up with the rows on the first, metadata sheet of the template.

Each of these organ-specific sheets should look like this:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	count_date	who_plated	who_counte	groups	mouse	dil_0	dil_1	dil_2	dil_3	dil_4	dil_5	dil_6	dil_7
2	"April 25 2022"	JR	JR	group_1	A	TNTC	TNTC	53	9	4	2	1	0
3	"April 25 2022"	JR	JR	group_1	B	TNTC	TNTC	119	48	18	0	0	0
4	"April 25 2022"	JR	JR	group_1	C	TNTC	TNTC	120	32	8	1	0	0
5	"April 25 2022"	JR	JR	group_1	D	TNTC	TNTC	53	31	3	2	0	0
6	"April 25 2022"	JR	JR	group_2	A	TNTC	92	28	7	1	0	0	0
7	"April 25 2022"	JR	JR	group_2	B	TNTC	43	14	4	0	0	0	0
8	"April 25 2022"	JR	JR	group_2	C	TNTC	32	10	4	0	0	0	0

Each of these organ-specific sheets of the template include the following columns:

- `count_date`: The date that the CFUs were counted. In some cases, the same plates may be counted at multiple dates.

- `who_plated`: An identifier for the researcher who plated the sample
- `who_counted`: An identifier for the researcher who counted the plate on this specific date
- `groups`: The experimental group to which the mouse belonged to
- `mouse`: An identifier for the unique mouse within the group (*note: as we collect data from the new experiment, this can be a unique ID by mouse, based on notch ID and cage number*)
- `dil_0`, `dil_1`, `dil_2`, ...: The count at each dilution. You can add additional columns if there were more dilutions that are in the template or take away dilution columns if there were fewer. However, all dilution columns should be named consistently, with "dil_" followed by the dilution number (e.g., "0", "1", "2"). If the CFUs were too numerous to count for a sample at a particular dilution, put "TNTC" in that cell of the spreadsheet.

You can download the template here. When you download the template, it will have example values filled out in blue. Use these to get an idea for how to record your own data. When you are ready to record your own data, delete these example values and replace them with data collected from your own experiment.

4.0.4 Processing collected data

Once data are collected, the file can be run through an R workflow. This workflow will convert the data into a format that is easier to work with for data analysis and visualization. It will also produce a report on the data in the spreadsheet, and ultimately it will also write relevant results in a format that can be used to populate a global database for all experiments in the project.

The next section provides the details of the pipeline. It aims to explain the code that processes the data and generates visualizations. You do not need to run this code step-by-step, but instead can access a script with the full code here.

To use this reporting template, you need to download it to your computer and save it in the file directory where you saved the data you collected with the data collection template. You can then open RStudio and navigate so that you are working within this directory. You should also make sure that you have installed a few required packages on R on the computer you are using to run the report. These packages are: `tidyverse`, `readxl`, `ggbeeswarm`, `ggpubr`, `purrr`, `knitr`, and `broom`.

Within RStudio, open the report template file. There is one spot where you will need to change the code in the template file, so it will read in the data from the version of the template that you saved, which you may have renamed. In the YAML of the report template file, change the file path beside "data:" so that it is the file name of your data file.

Once you've made this change, you can use the "Knit" button in RStudio to create a report from the data file and the report template file.

The report includes the following elements:

- Organ-specific summaries of the experiment, including the number of mice, experimental groups, date counted, and dilutions used for each experiment
- Metadata on the CFU collection process (e.g., percent of organ plated for each organ, dilution factor)
- Plot showing the distribution of CFUs by group in each organ
- Table giving the results of an ANOVA analysis comparing log CFUs across groups within each organ

You can download an example of a report created using this template by clicking here.

When you knit to create the report, it will create a Word file in the same file directory where you put your data file and report template. It will also create and output a version of the data that has been processed (in the case of the weights data, this mainly involves tracking mice as they change cages, to link all weights that are from a single animal). This output fill will be named "cfu_output.csv" and, like the report file, will be saved in the same file directory as the data file and the report template.

4.0.5 Details of processing script

This section goes through the code within the report template. It explains each part of the code in detail. You do not need to understand these details to use the report template. However, if you have questions about how the data are being processed, or how the outputs are created, all those details are available in this section.

As a note, there are two places in the following code where there's a small change compared to the report template. In the report, you incorporate the path to the data file using the `data:` section in the YAML at the top of the document. In the following code, we've instead used the path of some example data within this book's file directory, so the code will run for this chapter as well.

First, the workflow loads some additional R libraries. You may need to install these on your local R session if you do not already have them installed.

Next, the pipeline reads in the organ-specific data. To do this, it creates a list of all of the sheets that are in the spreadsheet other than the metadata sheet. It then loops through each of these organ-specific sheets. It uses pivoting to convert all the dilution levels and values into two columns (a longer rather than wider format), so that the data from all the organs can be joined into a single large dataframe, even if a different number of dilutions were used for the different organs.

```
## [[1]]
## # A tibble: 880 x 10
##    count_date      who_plated who_counted group mouse day   sex   organ dilution
##    <chr>           <chr>      <chr>       <chr> <chr> <chr> <chr> <chr>    <dbl>
##  1 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         0
##  2 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         1
##  3 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         2
##  4 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         3
##  5 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         4
##  6 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         5
##  7 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         6
##  8 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         7
##  9 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         8
## 10 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ lung         9
## # i 870 more rows
## # i 1 more variable: CFUs <chr>
##
## [[2]]
## # A tibble: 870 x 10
##    count_date      who_plated who_counted group mouse day   sex   organ dilution
##    <chr>           <chr>      <chr>       <chr> <chr> <chr> <chr> <chr>    <dbl>
##  1 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        0
##  2 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        1
##  3 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        2
##  4 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        3
##  5 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        4
##  6 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        5
##  7 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        6
##  8 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        7
##  9 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        8
## 10 "\"Dec 28 2023~ FL         FL          BCG   1     14    fema~ sple~        9
## # i 860 more rows
## # i 1 more variable: CFUs <chr>


## # A tibble: 6 x 10
##   count_date who_plated who_counted group mouse day   sex   organ dilution CFUs
##   <chr>      <chr>      <chr>       <chr> <chr> <chr> <chr> <chr>    <dbl> <chr>
## 1 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         0 TNTC
## 2 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         1 TNTC
## 3 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         2 52
## 4 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         3 10
## 5 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         4 0
## 6 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         5 0
```

At this stage...

```
## # A tibble: 6 x 4
##   organ  who_plated who_counted count_date
##   <chr>  <chr>      <chr>       <chr>
## 1 lung   FL         FL          "\"Dec 28 2023\""
## 2 lung   FL         FL          "\"Feb 28 2023\""
## 3 lung   FL         FL          "\"Mar 22 24\""
## 4 spleen FL         FL          "\"Dec 28 2023\""
## 5 spleen FL         FL          "\"Feb 28 2023\""
## 6 spleen FL         FL          "\"Mar 22 24\""
```

```
## # A tibble: 6 x 10
##   count_date who_plated who_counted group mouse day   sex   organ dilution CFUs
##   <chr>      <chr>      <chr>       <fct> <chr> <chr> <chr> <chr>    <dbl> <chr>
## 1 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         2 52
## 2 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         3 10
## 3 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         4 0
## 4 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         5 0
## 5 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         6 0
## 6 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         7 0
```

You can see that, rather than having separate columns for each dilution level on a single row for a sample, there are now multiple rows per sample, with the CFUs at different dilutions given in a `CFUs` column, with the `dilution` column identifying which dilution level for each.

The next steps work through the data, identifying which dilution is an appropriate one to use to count CFUs for each sample.

```
## # A tibble: 171 x 10
##    count_date      who_plated who_counted group mouse day   sex   organ dilution
##    <chr>           <chr>      <chr>       <fct> <chr> <chr> <chr> <chr>    <dbl>
##  1 "\"Dec 28 2023~ FL         FL          Sali~ 1     14    fema~ lung         2
##  2 "\"Feb 28 2023~ FL         FL          Sali~ 1     56    fema~ lung         2
##  3 "\"Mar 22 24\"" FL         FL          Sali~ 1     90    fema~ lung         3
##  4 "\"Dec 28 2023~ FL         FL          Sali~ 1     14    male  lung         3
##  5 "\"Feb 28 2023~ FL         FL          Sali~ 1     56    male  lung         2
##  6 "\"Mar 22 24\"" FL         FL          Sali~ 1     90    male  lung         3
##  7 "\"Dec 28 2023~ FL         FL          Sali~ 1     14    fema~ sple~        0
##  8 "\"Feb 28 2023~ FL         FL          Sali~ 1     56    fema~ sple~        2
##  9 "\"Mar 22 24\"" FL         FL          Sali~ 1     90    fema~ sple~        2
## 10 "\"Dec 28 2023~ FL         FL          Sali~ 1     14    male  sple~        1
## # i 161 more rows
## # i 1 more variable: CFUs <dbl>
```

In the example data, this step has reduced the number observations to consider from over 1406 to 171.

```
## [1] 1406
```

```
## [1] 171
```

If you look at the first few rows of the data before and after cleaning, you can see that in particular it has removed a lot of "TNTC" values (as well as a lot of 0 values, although that's harder to see in this sample of the data):

```
## # A tibble: 5 x 10
##   count_date who_plated who_counted group mouse day   sex   organ dilution CFUs
##   <chr>      <chr>      <chr>       <fct> <chr> <chr> <chr> <chr>    <dbl> <chr>
## 1 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         2 52
## 2 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         3 10
## 3 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         4 0
## 4 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         5 0
## 5 "\"Dec 28~ FL         FL          BCG   1     14    fema~ lung         6 0
```

```
## # A tibble: 5 x 10
##   count_date who_plated who_counted group mouse day   sex   organ dilution  CFUs
##   <chr>      <chr>      <chr>       <fct> <chr> <chr> <chr> <chr>    <dbl> <dbl>
## 1 "\"Dec 28~ FL         FL          Sali~ 1     14    fema~ lung         2    61
## 2 "\"Feb 28~ FL         FL          Sali~ 1     56    fema~ lung         2    55
## 3 "\"Mar 22~ FL         FL          Sali~ 1     90    fema~ lung         3     4
## 4 "\"Dec 28~ FL         FL          Sali~ 1     14    male  lung         3    41
## 5 "\"Feb 28~ FL         FL          Sali~ 1     56    male  lung         2    85
```

Next, the code brings in the information from the metadata sheet, including data on what percent of each organ was resuspended, the dilution factor, and so on. It uses this information to take the CFU value at a given dilution and convert it to an estimate of CFUs per mL.

```
## # A tibble: 171 x 11
##    organ count_date      day   who_plated who_counted group sex   mouse dilution
##    <chr> <chr>           <chr> <chr>      <chr>       <fct> <chr> <chr>    <dbl>
##  1 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ fema~ 1            2
##  2 lung  "\"Feb 28 2023~ 56    FL         FL          Sali~ fema~ 1            2
##  3 lung  "\"Mar 22 24\"" 90    FL         FL          Sali~ fema~ 1            3
##  4 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ male  1            3
##  5 lung  "\"Feb 28 2023~ 56    FL         FL          Sali~ male  1            2
##  6 lung  "\"Mar 22 24\"" 90    FL         FL          Sali~ male  1            3
##  7 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ fema~ 2            3
##  8 lung  "\"Feb 28 2023~ 56    FL         FL          Sali~ fema~ 2            2
##  9 lung  "\"Mar 22 24\"" 90    FL         FL          Sali~ fema~ 2            3
## 10 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ male  2            3
## # i 161 more rows
## # i 2 more variables: CFUs <dbl>, CFUs_whole <dbl>
```

Table 4.1: Organ-specific summary of the experiment

organ	name	value
lung	Experimental groups:	Saline, BCG, ID93-GLA-SE, BCG+ID93-GLA-SE
lung	Dates counted:	”Dec 28 2023”, ”Feb 28 2023”, ”Mar 22 24”
lung	Total mice:	17
lung	Dilutions considered:	2, 3, 4, 5
spleen	Experimental groups:	Saline, BCG, ID93-GLA-SE, BCG+ID93-GLA-SE
spleen	Dates counted:	”Dec 28 2023”, ”Feb 28 2023”, ”Mar 22 24”
spleen	Total mice:	17
spleen	Dilutions considered:	0, 1, 2, 3

```
## # A tibble: 171 x 11
##    organ count_date      day   who_plated who_counted group sex   mouse dilution
##    <chr> <chr>           <chr> <chr>      <chr>       <fct> <chr> <chr>    <dbl>
##  1 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ fema~ 1            2
##  2 lung  "\"Feb 28 2023~ 56    FL         FL          Sali~ fema~ 1            2
##  3 lung  "\"Mar 22 24\"" 90    FL         FL          Sali~ fema~ 1            3
##  4 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ male  1            3
##  5 lung  "\"Feb 28 2023~ 56    FL         FL          Sali~ male  1            2
##  6 lung  "\"Mar 22 24\"" 90    FL         FL          Sali~ male  1            3
##  7 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ fema~ 2            3
##  8 lung  "\"Feb 28 2023~ 56    FL         FL          Sali~ fema~ 2            2
##  9 lung  "\"Mar 22 24\"" 90    FL         FL          Sali~ fema~ 2            3
## 10 lung  "\"Dec 28 2023~ 14    FL         FL          Sali~ male  2            3
## # i 161 more rows
## # i 2 more variables: CFUs <dbl>, CFUs_whole <dbl>
```

```
## # A tibble: 6 x 11
##   organ count_date day   who_plated who_counted group sex   mouse dilution  CFUs
##   <chr> <chr>      <chr> <chr>      <chr>       <fct> <chr> <chr>    <dbl> <dbl>
## 1 lung  "\"Dec 28~ 14    FL         FL          Sali~ fema~ 1            2    61
## 2 lung  "\"Feb 28~ 56    FL         FL          Sali~ fema~ 1            2    55
## 3 lung  "\"Mar 22~ 90    FL         FL          Sali~ fema~ 1            3     4
## 4 lung  "\"Dec 28~ 14    FL         FL          Sali~ male  1            3    41
## 5 lung  "\"Feb 28~ 56    FL         FL          Sali~ male  1            2    85
## 6 lung  "\"Mar 22~ 90    FL         FL          Sali~ male  1            3     9
## # i 1 more variable: CFUs_whole <dbl>
```

The rest of the report code is used to provide summaries, visualizations, and analysis of these data. First, there is code to provide a summary of the number of mice, experimental groups, and some other details for each of the organs:

Next, the pipeline provides a table with the conditions of the CFU collection, based on the collected metadata from the template:

Table 4.2: Conditions of the CFU collection

Organ	Percent of Organ Plated	Dilution Factor	Total resuspension mL	Volume Plated mL
lung	33	10	1.5	0.1
spleen	50	10	1.5	0.1

Next, the pipeline creates a plot showing the distribution of CFUs by experimental group in each of the organs:

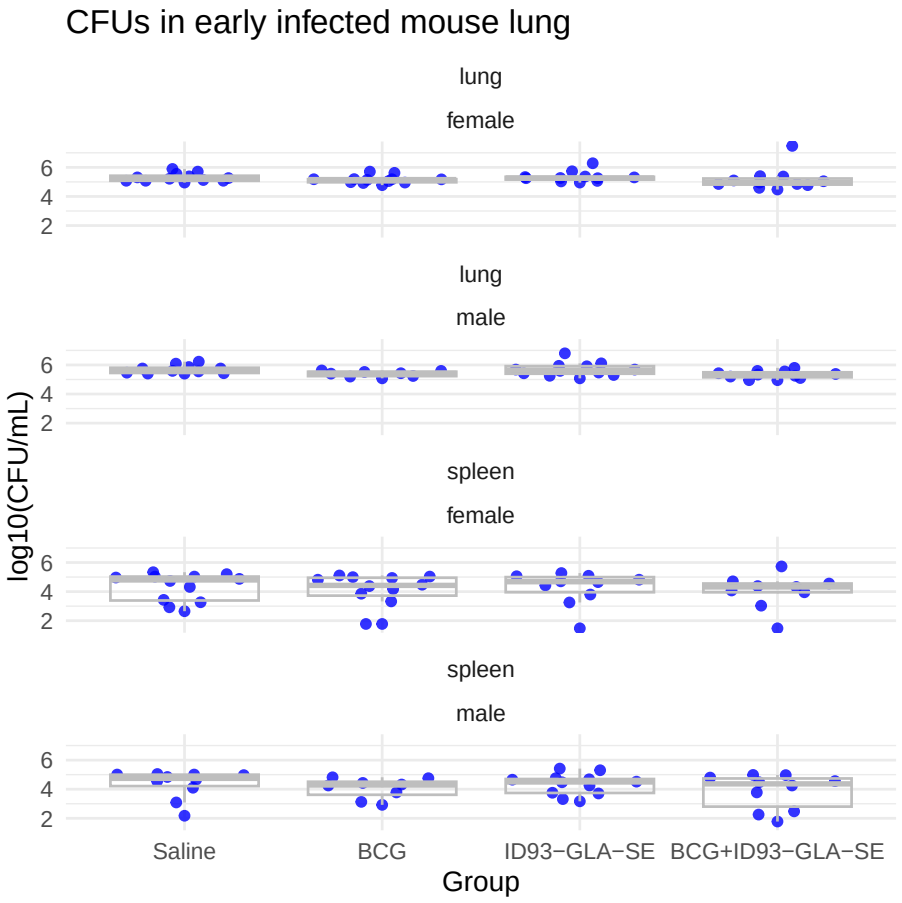
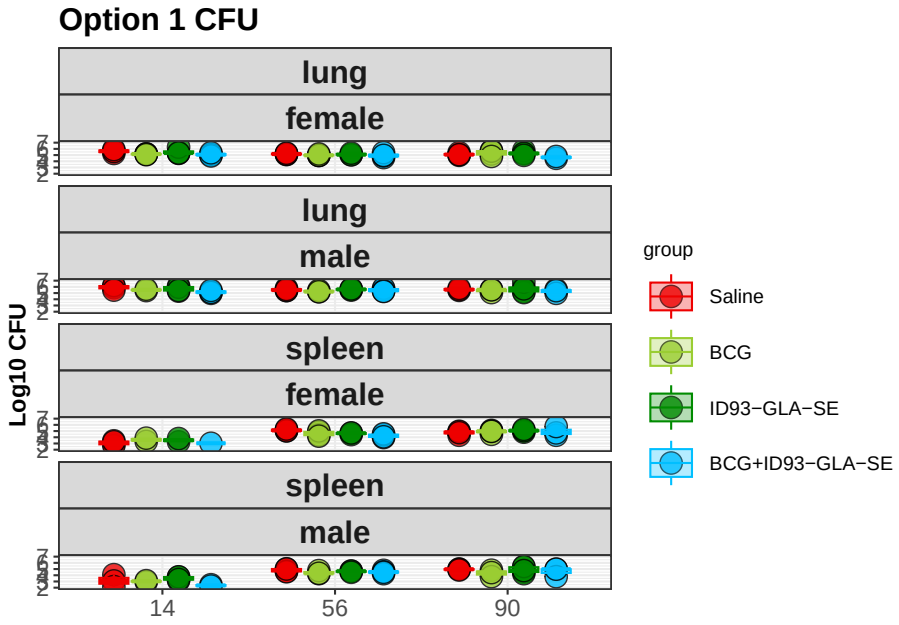


Table 4.3: ANOVA results comparing CFUs in each organ across the experimental groups

organ	contrast1	contrast2	estimate	conf.low	conf.high	adj.p.value
lung	BCG	Saline	-0.2368058	-0.5832132	0.1096016	0.2844430
lung	ID93 GLA SE	Saline	0.0407358	-0.2933692	0.3748408	0.9886291
lung	BCG+ID93 GLA SE	Saline	-0.2381384	-0.5760187	0.0997419	0.2587702
lung	ID93 GLA SE	BCG	0.2775416	-0.0688658	0.6239490	0.1614924
lung	BCG+ID93 GLA SE	BCG	-0.0013326	-0.3513826	0.3487175	0.9999996
lung	BCG+ID93 GLA SE	ID93 GLA SE	-0.2788742	-0.6167545	0.0590061	0.1420190
spleen	BCG	Saline	-0.2738479	-1.0919602	0.5442645	0.8159594
spleen	ID93 GLA SE	Saline	-0.0292823	-0.8276782	0.7691136	0.9996778
spleen	BCG+ID93 GLA SE	Saline	-0.4036624	-1.2329754	0.4256505	0.5798709
spleen	ID93 GLA SE	BCG	0.2445656	-0.5735467	1.0626780	0.8612381
spleen	BCG+ID93 GLA SE	BCG	-0.1298145	-0.9781256	0.7184965	0.9779295
spleen	BCG+ID93 GLA SE	ID93 GLA SE	-0.3743802	-1.2036931	0.4549328	0.6381738



Next, the pipeline runs an ANOVA analysis on the data. This is conducted after transforming the CFUs with a log-10 transform.

```
## [1] 0.6506753
```

```
## [1] 0.8720107
```

```
## [1] 0.3555729
```

```
## [1] 0.01680341
```

```
## [1] 0.2140449
```

```
## [1] 0.0267285
```

As a last step, the code in the template writes a CSV file with the processed data. This file will be an input into a script that will format the data to add to a database where we are collecting and integrating data from all the CSU experiments, and ultimately from there into project-wide storage.

Chapter 5

Enzyme-linked immunosorbest assay (ELISA)

ELISA is a standard molecular biology assay for detecting and quantifying a variety of compounds, including peptides, proteins, and antibodies in a sample. The sample could be serum, plasma, or bronchoalveolar lavage fluid (BALF).

5.1 Importance of ELISA

An antigen-specific reaction in the host results in the production of antibodies, which are proteins found in the blood. In the event of an infectious disease, it aids in the detection of antibodies in the body. ELISA is distinguishable from other antibody-assays in that it produces quantifiable findings and separates non-specific from specific interactions by serial binding to solid surfaces, which is often a polystyrene multi-well plate.

In IMPAc-TB project, it is crucial to evaluate the if the vaccine is eliciting humoral immunity and generating antibodies against vaccine antigen. ELISA will be used to determine the presence of Immunoglobulin (Ig) IgG, IgA, and IgM in the serum different time points post-vaccination.

5.1.1 Principle of ELISA

ELISA is based on the principle of antigen-antibody interaction. An antigen must be immobilized on a solid surface and then complexed with an enzyme-linked antibody in an ELISA. The conjugated enzyme's activity is evaluated

by incubating it with a substrate to yield a quantifiable result, which enables detection. There are four basic steps of ELISA:

1. Coating multiwell plate with antigen/antibody: This step depends on what we want to detect the sample. If we need to evaluate the the presence of antibody, the plate will be coated with the antigen, and vice versa. To coat the plate, a fixed concentration of antigen (protein) is added to a 96 well high-binding plate (charged plate). Plate is incubated over night with the antigen at 4 degree celsius (as proteins are temperature sensitive) so that antigens are completely bound to the well.

2. Blocking: It is possible that not each and every site of the well is coated with the targeted antigen, and there could be uncovered areas. It is important to block those empty spaces so that primary antibody (which we will add to the next step) binds to these spaces and give us false positive results. For this, microplate well surface-binding sites are blocked with an unrelated protein or other substance.Most common blocking agents are bovine serum albumin, skim milk, and casein. One of the best blocking agents is to use the serum from the organism in which your secondary (detection antibody) is raised. For example, if the secondary antibody is raised in goat, then we can use goat serum as a blocking agent.

3. Probing: Probing is the step where we add sample containing antibodies that we want to detect. This will be the primary antibody. If the antibodies against the antigen (which we have coated) are present in the sample, it will bind to the antigen with high affinity.

4. Washing: After the incubation of sample containing primary antibody, the wells are washed so that any unbound antibody is washed away. Washing solution contains phosphate buffer saline + 0.05% tween-20 (a mild detergent). 0.05% tween-20 washes away all the non-specific interactions as those are not strong, but keeps all the specific interaction as those are strong and cannot be detached with mild detergent.

5. Detection: To detect the presence of antibody-antigen complex, a secondary antibody labelled with an enzyme (usually horseradish peroxidase) is added to the wells, incubated and washed.

6. Signal Measurement: Finally to detect "if" and "how much" of the antibody is present, a chromogenic substrate (like 3,3',5,5'-Tetramethylbenzidine) is added to the wells, which can be cleaved the the enzyme that is tagged to the secondary antibody. The color compund is formed after the addition of the substrate, which is directly proportional to the amount of antibody present in the sample. The plate is read on a plate reader, where color is converted to numbers.

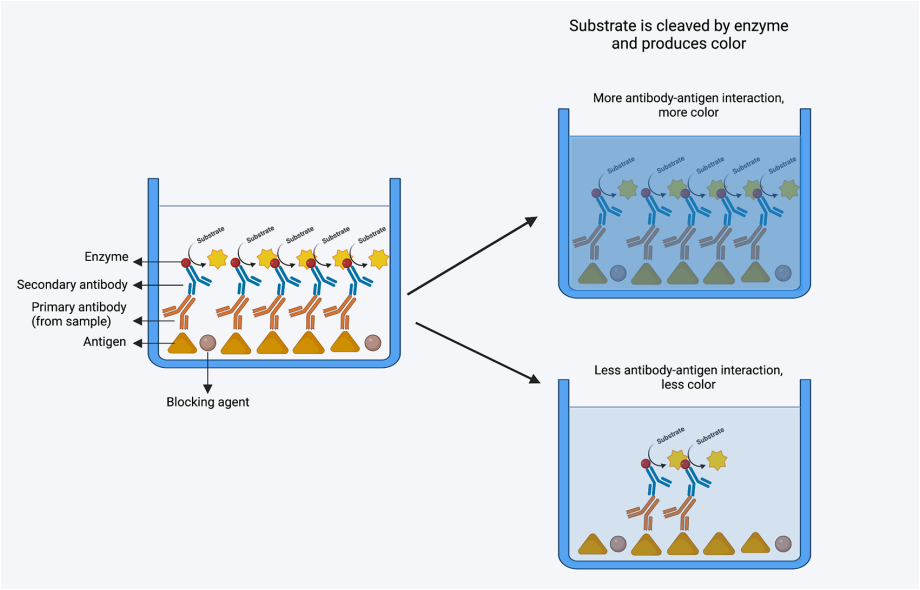


Figure 5.1: A caption

5.1.2 Loading libraries

5.2 ELISA data analysis

Analysis of ELISA data is the most important part of the ELISA experiment. ELISA data can be analyzed in different ways based on how the data is acquired. There are a a few examples of the type of ELISA data :

1. With standard curve: ELISA can be used to determine the concentrations of the antigen and antibody. This type of ELISA data usually have a standard curve with different concentrations of the known analyte and the concentration in the sample is determined by extrapolating the unknown values in the curve. This type of assay is straightforward, easy to interpret and are more robust.

2. Without standard curve: Usually vaccine studies involve investigating the presence of high-affinity (and novel) antibodies against the vaccine antigens. Therefore, plotting a standard curve is not feasible as there is no previous information available for antibody concentration or type of antibody. Also, because antibody response to a vaccine will differ depending on the individual, it is not practical to generate a calibration curve from which absolute concentrations can be extrapolated. For this type of ELISA, quantification of the antibody titers is performed using serial dilutions of the test samples, and analysis can be performed using the following three methods (Hartman et al., 2018):

1. Fitting sigmoid model
2. Endpoint titer method 3: Absorbance summation method

Let's have a look at these methods, how we can apply these methods in our data, and R-based packages that we can utilize to perform this analysis.

5.3 1. Curve fitting model:

The curve in ELISA data represents a plot of known concentrations versus their corresponding signal responses. The typical range of these calibration curves is one to two orders of magnitude on the response axis and two or more orders of magnitude on the concentration axis. The real curve of each assay could be easily identified if an infinite number of concentration dilutions with an infinite number of repetitions could be tested. The correct curve must be approximated from a relatively small number of noisy points, though, because there are a finite number of dilutions that may be performed. To estimate the dose-response relationship between standard dilutions, a method of interpolating between standards is required because there cannot be a standard at every concentration. This process is typically performed using a mathematical function or regression to approximate the true shape of the curve. A curve model is the name given to this approximating function, which commonly uses two or more parameters to describe a family of curves, and are then adjusted in order to find the curve from the family of curves that best fits the assay data.

Three qualities should be included in a good curve fitting model. 1. The true curve's shape must be accurately approximated by the curve model. If the curve model does not accomplish this, there is no way to adjust for this component of the total error that results from a lack of fit. 2. In order to get concentration estimates with minimal inaccuracy, a decent curve model must be able to average away as much of the random variation as is practical. 3. A successful curve model must be capable of accurately predicting concentration values for points between the anchor points of the standard dilutions.

5.3.1 How do we perform curve fitting model

There are two major steps in performing curve fitting model for non-linear data like ELISA: 1. Finding the initial starting estimates of the parameters 2. locating the optimal solution in a region of the initial estimates

We have presented an example below where we have performed a 8-10 point serial dilution of our sample and fitted a 4 parameter curve model.

5.3.2 An example of the curve fitting model

5.3.2.1 Read in the data

This information comes from the 2018 study conducted by Hartman et al. Hartman et al. analyzed the ELISA data in their study utilizing fitted sigmoid analysis, end point titer, and absorbance summation. We utilized this information to determine whether our formulas and calculations provide the same outcomes and values as theirs.

5.3.2.2 Tidying the data

We next performed tidying the data and make it in a format so that we can plot a sigmoid curve with that.

```
## # A tibble: 6 x 5
##   numerator denominator absorbance dilution log_dilution
##       <dbl>       <dbl>      <dbl>    <dbl>        <dbl>
## 1         1          30       4    0.0333          -3.10
## 2         1          90       3.73 0.0111          -4.10
## 3         1         270       2.34 0.00370         -5.10
## 4         1         810       1.1  0.00123         -6.10
## 5         1        2430       0.51 0.000412        -7.10
## 6         1        7290       0.22 0.000137        -8.10
```

5.3.2.3 Create function for curve fitting model

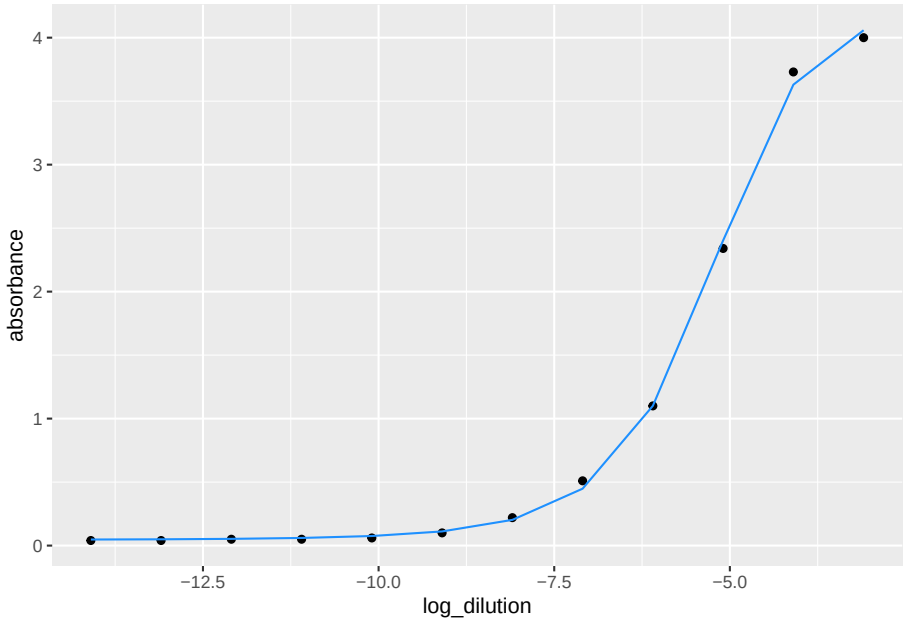
We next created the curve fitting model function by using nlsLM function from "minpack.lm" package. The purpose of nlslm is to minimize the sum square of the vector returned by the function fn, by a modification of the Levenberg-Marquardt algorithm. In the early 1960s, the Levenberg-Marquardt algorithm was developed to address nonlinear least squares problems. Through a series of well-chosen updates to model parameter values, Levenberg-Marquardt algorithm lower the sum of the squares of the errors between the model function and the data points.

```
## Nonlinear regression model
##   model: absorbance ~ ((a - d)/(1 + (log_dilution/c)^b)) + d
##    data: elisa_example_data
##        a        d        c        b
##  4.12406  0.04532 -5.31056  7.62972
##  residual sum-of-squares: 0.02221
##
## Number of iterations to convergence: 9
## Achieved convergence tolerance: 1.49e-08
```

```
##
## Formula: absorbance ~ ((a - d)/(1 + (log_dilution/c)^b)) + d
##
## Parameters:
##   Estimate Std. Error  t value Pr(>|t|)
## a  4.12406    0.05820   70.860 1.75e-12 ***
## d  0.04532    0.02268    1.998   0.0808 .
## c -5.31056    0.03933 -135.037 1.01e-14 ***
## b  7.62972    0.35854   21.280 2.50e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05269 on 8 degrees of freedom
##
## Number of iterations to convergence: 9
## Achieved convergence tolerance: 1.49e-08
```

5.3.2.4 Apply the function to the data

5.3.2.5 Plot the sigmoid curve with fitted sigmoid model



5.4 2. Endpoint titer method

The endpoint titer approach chooses an absorbance value just above the background noise (or the lower asymptotic level). **The highest dilution with an absorbance greater than this predetermined value is the endpoint titer.** This method is based on the assumption that a sample with a higher protein concentration will require a higher dilution factor to achieve an absorbance just above the level of background noise.

5.4.1 Create an endpoint titer function and apply it to the output of the fitted sigmoid model values.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -8.113  -8.113  -8.113  -8.113  -8.113  -8.113
```

```
## [1] -8.113285
```

5.4.2 Other methods to analyze ELISA data

5.4.2.1 Absorption summation

5.4.2.2 Area under the curve

In this model of data analysis, we sum all the absorbance values from each sample to obtain one value. This value is termed as absorption summation (AS). Using the above data, the AS will be calculated as below:

```
## [1] 12.24
```

5.5 Apply the fitting sigmoid model and endpoint titer function in our dataset

The presented data is from a mouse study. In this data, presence of IgG antibody has been evaluated against receptor binding domain (RBD) of SARS-CoV-2 virus in two different groups of mice. We need to elucidate which group has higher concentration of the antibodies.

5.5.0.1 Read in the data

5.5.0.2 Tidy the data

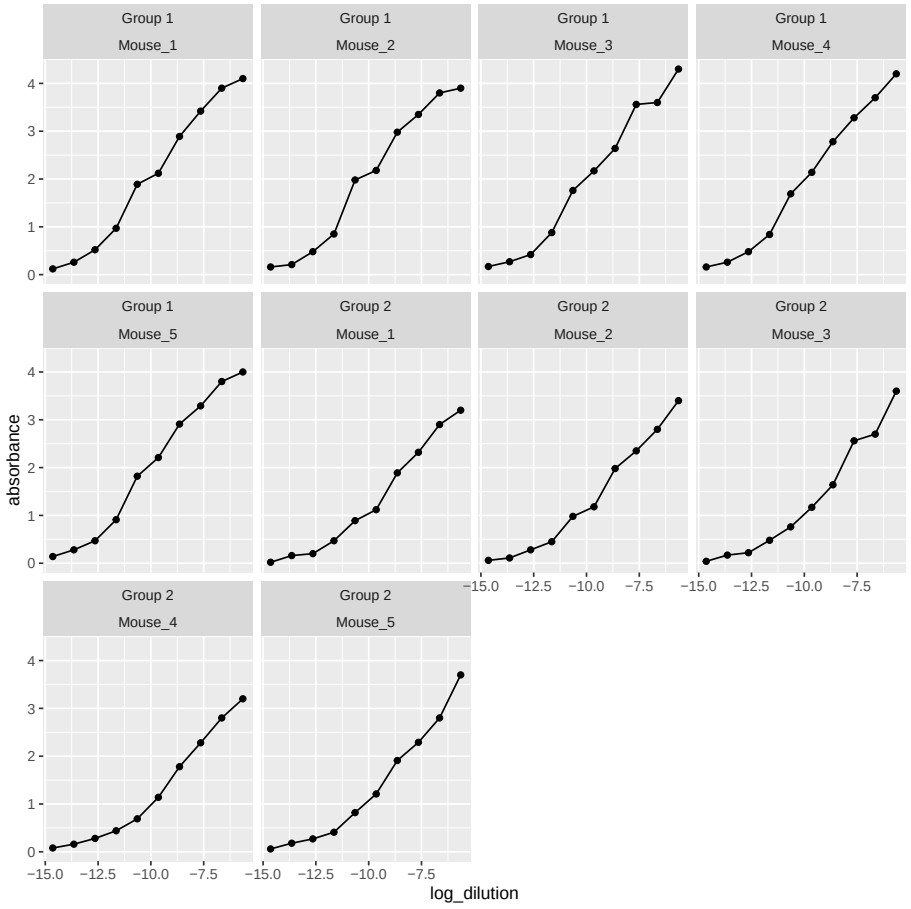
```
## # A tibble: 6 x 4
##   Groups  Dilution mouse_id absorbance
##   <chr>   <chr>    <chr>         <dbl>
## 1 Group 1 1/50     Mouse_1         4.1
## 2 Group 1 1/50     Mouse_2         3.9
## 3 Group 1 1/50     Mouse_3         4.3
## 4 Group 1 1/50     Mouse_4         4.2
## 5 Group 1 1/50     Mouse_5         4
## 6 Group 1 1/100    Mouse_1         3.9
```

```
##    Groups numerator denomenator mouse_id absorbance dilution log_dilution
## 1 Group 1         1          50  Mouse_1        4.1     0.02    -5.643856
## 2 Group 1         1          50  Mouse_2        3.9     0.02    -5.643856
## 3 Group 1         1          50  Mouse_3        4.3     0.02    -5.643856
## 4 Group 1         1          50  Mouse_4        4.2     0.02    -5.643856
## 5 Group 1         1          50  Mouse_5        4.0     0.02    -5.643856
## 6 Group 1         1         100  Mouse_1        3.9     0.01    -6.643856
```

5.5.0.2.1 converting data into dataframe

```
## # A tibble: 6 x 3
## # Groups:   Groups, mouse_id [6]
##   Groups  mouse_id data
##   <chr>   <chr>    <list>
## 1 Group 1 Mouse_1  <tibble [10 x 5]>
## 2 Group 1 Mouse_2  <tibble [10 x 5]>
## 3 Group 1 Mouse_3  <tibble [10 x 5]>
## 4 Group 1 Mouse_4  <tibble [10 x 5]>
## 5 Group 1 Mouse_5  <tibble [10 x 5]>
## 6 Group 2 Mouse_1  <tibble [10 x 5]>
```

5.5.0.2.2 plot the curves to evaluate the a, d, c, and b



Based on the curve, the values are:

a = 4, d = 0 c = 2 b = 1

5.5.1 Creating a function for fitting model

5.5.1.1 Fitting the model into the dataset

```
## Nonlinear regression model
##   model: absorbance ~ ((a - d)/(1 + (log_dilution/c)^b)) + d
##    data: df_elisa
##        a        d        c        b
##   4.3070  -0.6009 -10.2577   5.2893
##  residual sum-of-squares: 0.1199
##
```

```
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 1.49e-08
```

5.5.1.2 Apply the fitted model function to the whole dataframe

```
## # A tibble: 6 x 4
## # Groups:   Groups, mouse_id [6]
##   Groups  mouse_id data              fitted_data
##   <chr>   <chr>    <list>            <list>
## 1 Group 1 Mouse_1  <tibble [10 x 5]> <nls>
## 2 Group 1 Mouse_2  <tibble [10 x 5]> <nls>
## 3 Group 1 Mouse_3  <tibble [10 x 5]> <nls>
## 4 Group 1 Mouse_4  <tibble [10 x 5]> <nls>
## 5 Group 1 Mouse_5  <tibble [10 x 5]> <nls>
## 6 Group 2 Mouse_1  <tibble [10 x 5]> <nls>
```

5.5.1.3 Take out the summary of the data

```
## Nonlinear regression model
##   model: absorbance ~ ((a - d)/(1 + (log_dilution/c)^b)) + d
##    data: df_elisa
##        a        d        c        b
##   4.3070  -0.6009 -10.2577   5.2893
##  residual sum-of-squares: 0.1199
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 1.49e-08
```

5.6 Create function of Fitted model and endpoint titer, where the output of the fitted model data will be the input of the endpoint titer

5.6.0.1 Apply the fitted model fuction into the nested data and use the output of the fitted data as the input for endpoint titer value evaluation

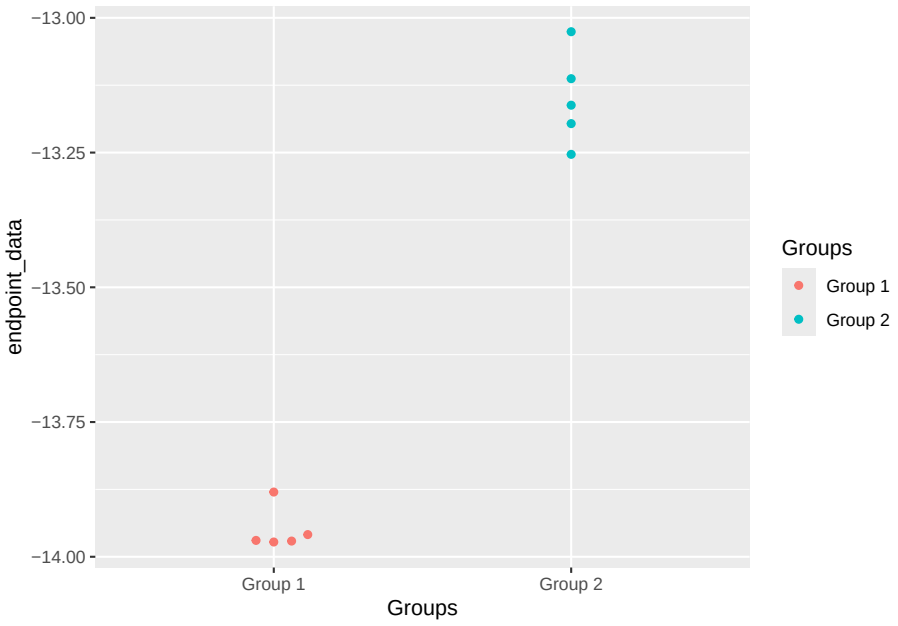
5.6.0.1.1 Run fitted model on the data

```
## # A tibble: 6 x 4
## # Groups:   Groups, mouse_id [6]
```

```
##   Groups  mouse_id data              fitted_data
##   <chr>   <chr>    <list>            <list>
## 1 Group 1 Mouse_1  <tibble [10 x 5]> <nls>
## 2 Group 1 Mouse_2  <tibble [10 x 5]> <nls>
## 3 Group 1 Mouse_3  <tibble [10 x 5]> <nls>
## 4 Group 1 Mouse_4  <tibble [10 x 5]> <nls>
## 5 Group 1 Mouse_5  <tibble [10 x 5]> <nls>
## 6 Group 2 Mouse_1  <tibble [10 x 5]> <nls>
```

5.6.0.1.2 Taking output of the fitted model function and into endpoint titer function

5.6.0.2 Plot the endpoint titer data for the two groups



5.6.0.3 Perform statistical analysis on the data

```
## # A tibble: 1 x 10
##   estimate estimate1 estimate2 statistic    p.value parameter conf.low conf.high
##      <dbl>     <dbl>     <dbl>     <dbl>      <dbl>     <dbl>    <dbl>     <dbl>
## 1   -0.800     -14.0     -13.2     -18.8 0.00000268      5.63   -0.906    -0.695
## # i 2 more variables: method <chr>, alternative <chr>
```

5.6.0.4 Statistical data analysis for more than two groups

5.7 ELISA data processing

We read ELISA plate in a 96 well plate using a plate reader. The plate reader generates the data in form of number in an excel sheet. We have created this pipeline/worksheet to bring out the information from the excl sheet to a tidy format in which the above created fitted model and endpoint titer functions can be applied.

5.7.0.1 Read in the first dataset

Below is the example ELISA data that has came straight out of the plate reader. This data is arranged in a 96-well plate format and contains Optical Density (OD) values.

```
## # A tibble: 6 x 12
##   ...1          ...2  ...3 ...4   ...5  ...6 ...7   ...8  ...9 ...10 ...11 ...12
##   <chr>        <dbl> <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl> <chr> <dbl> <dbl>
## 1 5.199999999~ 0.05  0.069 6.3E~ 0.061 0.122 0.16~ 0.145 0.135 6.80~ 0.053 0.05
## 2 7.900000000~ 0.098 0.069 6.80~ 0.115 0.202 5.89~ 0.134 0.069 0.106 0.05  0.075
## 3 8.899999999~ 0.133 0.119 OVRF~ 3.87  2.32  OVRF~ 3.85  2.12  OVRF~ 3.21  1.02
## 4 OVRFLW       3.46  1.16  OVRF~ 3.80  2.36  OVRF~ 3.70  1.49  OVRF~ 3.68  1.63
## 5 3.815999999~ 1.82  0.446 3.89~ 3.42  1.13  OVRF~ 2.33  0.608 OVRF~ 3.41  1.10
## 6 OVRFLW       3.69  1.43  OVRF~ 3.66  1.27  3.839 1.74  0.444 2.49~ 0.637 0.704
```

5.7.0.2 Tidy dataset 1

It is important to clean the data and arrange it in a format on which we can apply formulas and functions.

```
## # A tibble: 6 x 2
##   well_id od_450nm
##   <chr>      <dbl>
## 1 A1         0.052
## 2 A2         0.05
## 3 A3         0.069
## 4 A4         0.063
## 5 A5         0.061
## 6 A6         0.122
```

5.7.0.3 Read in the second data set

The second dataset contains the information such as groups, mouse id, and dilutions for the respective wells of the 96 well plate for the dataset-1.

```
## # A tibble: 6 x 12
##   ...1        ...2   ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11 ...12
##   <chr>       <chr>  <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 blank       secon~ naïv~ 1A-1~ 1A-1~ 1A-1~ 1A-2~ 1A-2~ 1A-2~ 1A-3~ 1A-3~ 1A-3~
## 2 1A-4 (1/250 1A-4 ~ 1A-4~ 1B-1~ 1B-1~ 1B-1~ 1B-2~ 1B-2~ 1B-2~ 1B-3~ 1B-3~ 1B-3~
## 3 1B-4 (1/250 1B-4 ~ 1B-4~ 2A-1~ 2A-1~ 2A-1~ 2A-2~ 2A-2~ 2A-2~ 2A-3~ 2A-3~ 2A-3~
## 4 2B-1 (1/250 2B-1 ~ 2B-1~ 2B-2~ 2B-2~ 2B-2~ 2B-3~ 2B-3~ 2B-3~ 2B-4~ 2B-4~ 2B-4~
## 5 3A-1 (1/250 3A-1 ~ 3A-1~ 3A-2~ 3A-2~ 3A-2~ 3A-3~ 3A-3~ 3A-3~ 3A-4~ 3A-4~ 3A-4~
## 6 3B-1 (1/250 3B-1 ~ 3B-1~ 3B-2~ 3B-2~ 3B-2~ 3B-3~ 3B-3~ 3B-3~ 3B-4~ 3B-4~ 3B-4~
```

5.7.0.4 Tidy dataset-2

```
## # A tibble: 6 x 2
##   well_id information
##   <chr>   <chr>
## 1 A1      blank
## 2 A2      secondary
## 3 A3      naïve (1/250)
## 4 A4      1A-1 (1/250
## 5 A5      1A-1 (1/1250
## 6 A6      1A-1 (1/6250
```

5.7.0.5 Merge dataset-1 (with OD information) with dataset-2 (with respective data information)

To create a complete full dataset with Groups, mouse-id, dilutions, and OD, we merged the dataset-1 and dataset-2 together. We also cleaned the data set so that mouse-ID and dilution columns are separate and have their own columns.

```
## # A tibble: 6 x 3
##   well_id od_450nm information
##   <chr>      <dbl> <chr>
## 1 A1         0.052 blank
## 2 A2         0.05  secondary
## 3 A3         0.069 naïve (1/250)
## 4 A4         0.063 1A-1 (1/250
## 5 A5         0.061 1A-1 (1/1250
## 6 A6         0.122 1A-1 (1/6250
```

```
## # A tibble: 6 x 4
##   well_id od_450nm sample_id   dilution
##   <chr>      <dbl> <chr>       <chr>
## 1 A1         0.052 "blank"     <NA>
## 2 A2         0.05  "secondary" <NA>
## 3 A3         0.069 "naïve "    1/250)
## 4 A4         0.063 "1A-1 "     1/250
## 5 A5         0.061 "1A-1 "     1/1250
## 6 A6         0.122 "1A-1 "     1/6250
```

```
## # A tibble: 6 x 4
##   well_id sample_id   dilution od_450nm
##   <chr>   <chr>          <dbl>    <dbl>
## 1 A1      "blank"           NA    0.052
## 2 A2      "secondary"       NA    0.05
## 3 A3      "naïve "         250    0.069
## 4 A4      "1A-1 "          250    0.063
## 5 A5      "1A-1 "         1250    0.061
## 6 A6      "1A-1 "         6250    0.122
```

Chapter 6

Flow cytometry

Flow cytometry data can be quantified in many different ways and with different techniques. For the purpose of these data analyses, manual gating has been achieved in FlowJo and cell frequencies and populations exported as a `.csv` file. This `.csv` file is the primary input for this R pipeline which aims to output box plots for each gated cell population.

This example data set is from an innate response study whcih investigated the immune response in the lungs during the first 28 days of infection.

Immune cells are very diverse, and the make-up of immune cells within a sample can provide important insights on immune processes based on measures of this composition. Immune cells can be categorized into large groups (e.g., T cells, B cells, macrophages, dendritic cells). They can also be characterized into different populations within these large groups, based on things like activation and differentiation [?] within the group [?] (Maecker et al., 2012) (e.g., T cells can be divided into naive T cells versus memory T cells, helper T cells versus cytotoxic T cells, [others] [?]). This process of characterizing the immune cells in a sample is called immunophenotyping.

To make these classifications, flow cytometry uses a pretty clever mix of physics and biology. First, it starts by leveraging the biological knowledge that antibodies can have a very specific affinity for a certain protein. This means that you can find a set of antibodies that will target and stick specifically to certain proteins.

Flow cytometry starts by creating a panel of up to [x] protein markers, focusing on proteins that can help in identifying a specific cell type. Typically, the panel will include several proteins that are "CD" proteins (the "CD" stands for cluster

of differentiation or classification determinant). These are proteins that show up on the surface of immune cells, with specific CD proteins common to only certain types of cells, making their presence or absence helpful in classifying cells.

Two of the most common CDs to include on a panel are CD3, CD4, and CD8. T cells have CD3 on their surface, so a marker for CD3 can be used to distinguish T cells from other types of white blood cells, including granulocytes, monocytes, and B cells. Among T cells, the helper T cells have the CD4 protein on their surface, while the cytotoxic T cells have the CD8 protein on their surface, so the CD4 and CD8 markers can help in refining a T cell into a more specific type.

In flow cytometry, you can characterize immune cells into populations based on proteins on the cell surface and inside the cell, as well as cell size and granularity (Maecker et al., 2012, Barnett et al. (2008)). For example, macrophages can be distinguished from T cells and B cells based on … [size? granularity?], which T cells and B cells can be distinguished from each other based on whether the cell has the [surface protein? CD3?], and helper T cells versus cytotoxic T cells can be distinguished from each other based on whether the cell has the [surface protein CD8? CD4?].

6.1 Loading packages

MULTI CSV SHEETS MALE FEMALE

6.2 Loading data

```
## # A tibble: 64 x 30
##    Sample          sex    day   `CD3+ ` `CD3+ CD4+ ` `CD3+ CD4+  CD62L-  CD44+ `
##    <chr>           <chr>  <chr>   <dbl>        <dbl>                         <dbl>
##  1 " Saline_1.fcs" female 14       30          14.1                           5.19
##  2 " Saline_2.fcs" female 14       29.5        12.4                           2.47
##  3 " Saline_3.fcs" female 14       26.7        12.6                           4.86
##  4 " Saline_4.fcs" female 14       17.2         7.2                           3
##  5 " BCG_1.fcs"    female 14       43.2        24.9                          14.8
##  6 " BCG_2.fcs"    female 14       11.7         6.42                          1.45
##  7 " BCG_3.fcs"    female 14       24.1        13.5                           4.59
##  8 " BCG_4.fcs"    female 14       41.2        24                            10.2
##  9 " ID93_1.fcs"   female 14       38.9        24.2                          11.9
## 10 " ID93_2.fcs"   female 14       34.2        18.9                          10.1
## # i 54 more rows
## # i 24 more variables: `CD3+ CD4+  CD62L-  CD44+  PD1-  KLRG1+ ` <dbl>,
## #   `CD3+ CD4+  CD62L-  CD44+  PD1+  KLRG1+ ` <dbl>,
## #   `CD3+ CD4+  CD62L-  CD44+  PD1+  KLRG1- ` <dbl>,
```

```
## #   `CD3+ CD4+  CD62L-  CD44+  PD1-  KLRG1- ` <dbl>,
## #   `CD3+ CD4+  CD62L+  CD44+ ` <dbl>, `CD3+ CD4+  CD62L+  CD44- ` <dbl>,
## #   `CD3+ CD4+  CD62L+  CD44-  PD1-  KLRG1+ ` <dbl>, ...
```

```
##                                 Sample                                    sex
##                                      0                                      0
##                                    day                                   CD3+
##                                      0                                      0
##                               CD3+ CD4+                CD3+ CD4+  CD62L-  CD44+
##                                      0                                      0
## CD3+ CD4+  CD62L-  CD44+  PD1-  KLRG1+  CD3+ CD4+  CD62L-  CD44+  PD1+  KLRG1+
##                                      0                                      0
## CD3+ CD4+  CD62L-  CD44+  PD1+  KLRG1-  CD3+ CD4+  CD62L-  CD44+  PD1-  KLRG1-
##                                      0                                      0
##               CD3+ CD4+  CD62L+  CD44+                CD3+ CD4+  CD62L+  CD44-
##                                      0                                      0
## CD3+ CD4+  CD62L+  CD44-  PD1-  KLRG1+  CD3+ CD4+  CD62L+  CD44-  PD1+  KLRG1+
##                                      0                                      0
## CD3+ CD4+  CD62L+  CD44-  PD1+  KLRG1-  CD3+ CD4+  CD62L+  CD44-  PD1-  KLRG1-
##                                      0                                      0
##               CD3+ CD4+  CD62L-  CD44-                              CD3+ CD8+
##                                      0                                      0
##               CD3+ CD8+  CD62L-  CD44+  CD3+ CD8+  CD62L-  CD44+  PD1-  KLRG1+
##                                      0                                      0
## CD3+ CD8+  CD62L-  CD44+  PD1+  KLRG1+  CD3+ CD8+  CD62L-  CD44+  PD1+  KLRG1-
##                                      0                                      0
## CD3+ CD8+  CD62L-  CD44+  PD1-  KLRG1-                CD3+ CD8+  CD62L+  CD44+
##                                      0                                      0
##               CD3+ CD8+  CD62L+  CD44-  CD3+ CD8+  CD62L+  CD44-  PD1-  KLRG1+
##                                      0                                      0
## CD3+ CD8+  CD62L+  CD44-  PD1+  KLRG1+  CD3+ CD8+  CD62L+  CD44-  PD1+  KLRG1-
##                                      0                                      0
## CD3+ CD8+  CD62L+  CD44-  PD1-  KLRG1-                CD3+ CD8+  CD62L-  CD44-
##                                      0                                      0
```

MULTIDAY SHEETS FOR EXCEL

SINGLE SHEETS Loading data

MAKING DATA TIDY FOR PLOTTING

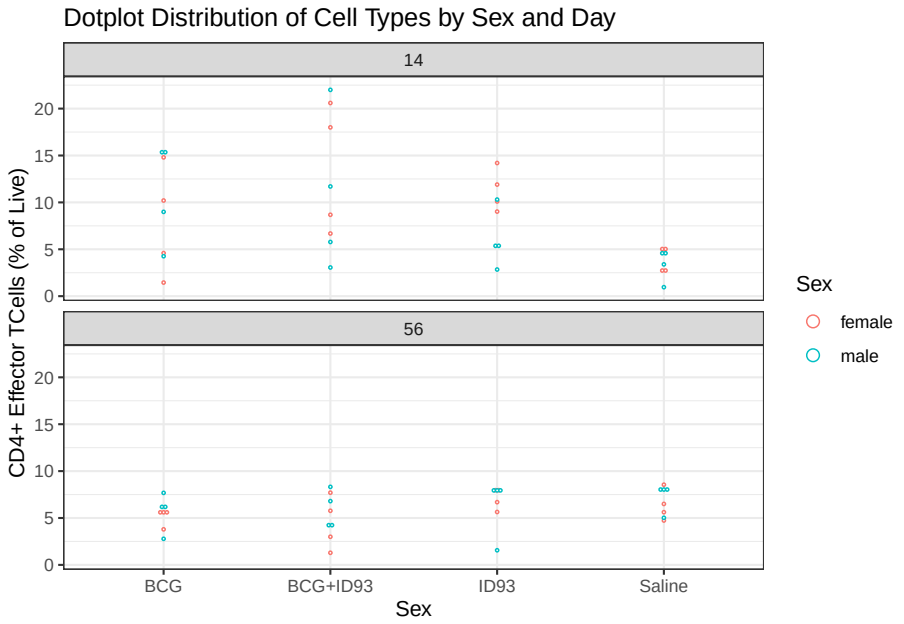
```
## # A tibble: 1,728 x 6
##    group  mouse_ID sex    day   cell_types                    percentage_of_LIVE
##    <chr>  <chr>    <chr>  <chr> <chr>                                      <dbl>
##  1 Saline 1        female 14    CD3+                                        30
##  2 Saline 1        female 14    CD3+ CD4+                                   14.1
```

```
##  3 Saline 1        female 14    CD3+ CD4+ CD62L- CD44+                   5.19
##  4 Saline 1        female 14    CD3+ CD4+ CD62L- CD44+ PD1- ~            0.42
##  5 Saline 1        female 14    CD3+ CD4+ CD62L- CD44+ PD1+ ~            0.24
##  6 Saline 1        female 14    CD3+ CD4+ CD62L- CD44+ PD1+ ~            1.87
##  7 Saline 1        female 14    CD3+ CD4+ CD62L- CD44+ PD1- ~            2.66
##  8 Saline 1        female 14    CD3+ CD4+ CD62L+ CD44+                   0.72
##  9 Saline 1        female 14    CD3+ CD4+ CD62L+ CD44-                   3.9
## 10 Saline 1        female 14    CD3+ CD4+ CD62L+ CD44- PD1- ~            0.00482
## # i 1,718 more rows
```

```
## tibble [1,728 x 6] (S3: tbl_df/tbl/data.frame)
##  $ group             : chr [1:1728] "Saline" "Saline" "Saline" "Saline" ...
##  $ mouse_ID          : chr [1:1728] "1" "1" "1" "1" ...
##  $ sex               : chr [1:1728] "female" "female" "female" "female" ...
##  $ day               : chr [1:1728] "14" "14" "14" "14" ...
##  $ cell_types        : chr [1:1728] "CD3+" "CD3+ CD4+" "CD3+ CD4+ CD62L- CD44+" "CD3
##  $ percentage_of_LIVE: num [1:1728] 30 14.1 5.19 0.42 0.24 1.87 2.66 0.72 3.9 0.0048
```

CHECK FIRST ASSUMPTION IN STATS (Independent obervations)

```
##  [1] "CD3+"                                "CD3+ CD4+"
##  [3] "CD3+ CD4+ CD62L- CD44+"              "CD3+ CD4+ CD62L- CD44+ PD1- KLRG1+"
##  [5] "CD3+ CD4+ CD62L- CD44+ PD1+ KLRG1+" "CD3+ CD4+ CD62L- CD44+ PD1+ KLRG1-"
##  [7] "CD3+ CD4+ CD62L- CD44+ PD1- KLRG1-" "CD3+ CD4+ CD62L+ CD44+"
##  [9] "CD3+ CD4+ CD62L+ CD44-"              "CD3+ CD4+ CD62L+ CD44- PD1- KLRG1+"
## [11] "CD3+ CD4+ CD62L+ CD44- PD1+ KLRG1+" "CD3+ CD4+ CD62L+ CD44- PD1+ KLRG1-"
## [13] "CD3+ CD4+ CD62L+ CD44- PD1- KLRG1-" "CD3+ CD4+ CD62L- CD44-"
## [15] "CD3+ CD8+"                          "CD3+ CD8+ CD62L- CD44+"
## [17] "CD3+ CD8+ CD62L- CD44+ PD1- KLRG1+" "CD3+ CD8+ CD62L- CD44+ PD1+ KLRG1+"
## [19] "CD3+ CD8+ CD62L- CD44+ PD1+ KLRG1-" "CD3+ CD8+ CD62L- CD44+ PD1- KLRG1-"
## [21] "CD3+ CD8+ CD62L+ CD44+"             "CD3+ CD8+ CD62L+ CD44-"
## [23] "CD3+ CD8+ CD62L+ CD44- PD1- KLRG1+" "CD3+ CD8+ CD62L+ CD44- PD1+ KLRG1+"
## [25] "CD3+ CD8+ CD62L+ CD44- PD1+ KLRG1-" "CD3+ CD8+ CD62L+ CD44- PD1- KLRG1-"
## [27] "CD3+ CD8+ CD62L- CD44-"
```



CHECKING EQUAL VARIANCES OF POPULATION OF INTEREST

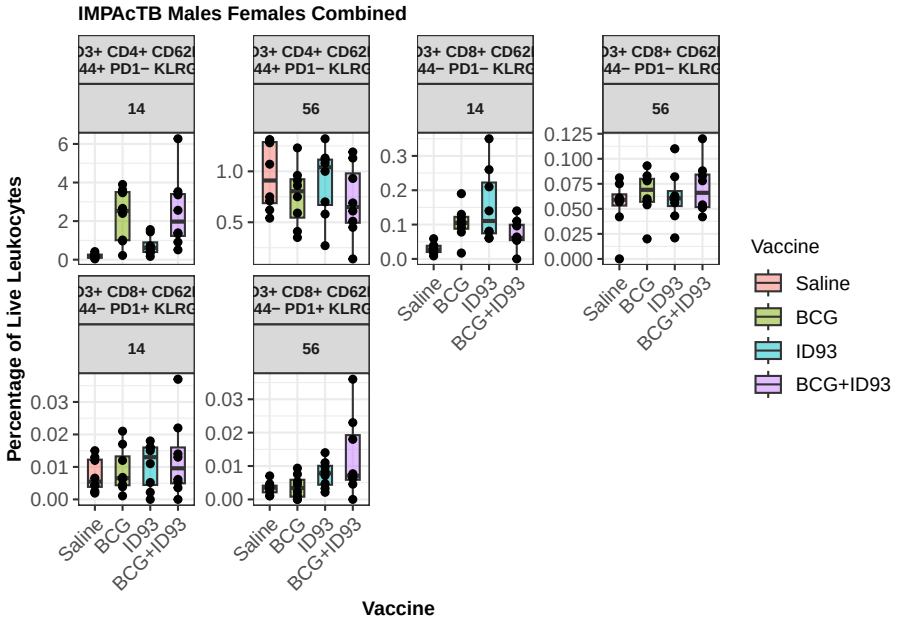
```
## Levene's Test for Homogeneity of Variance (center = mean)
##       Df F value    Pr(>F)
## group  7  8.3438 6.034e-07 ***
##       56
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  percentage_of_LIVE by interaction(group, day)
## Bartlett's K-squared = 36.214, df = 7, p-value = 6.605e-06
```

WORKING ON STATS June 1, 2023

EXPLORATORY ANALYSIS

```
## [1] "Saline"    "BCG"       "ID93"      "BCG+ID93"
```



PLOT POPULATIONS

WORKING ON THIS

Saving plots to PDF

Looking at specific populations

Chapter 7

Pathology

Chapter 8

Proteomics

8.0.1 Downloads

The downloads for this chapter are:

- Data collection format (pre-defined by the assay equipment and software run prior on [software] for most of the pre-processing)
- Report template to process data collected with the data template (when you go to this link, go to the "File" bar in your browser's menu bar, chose "Save As", then save the file as "animal_weights.Rmd")
- Example output from the report template

8.0.2 Overview

Proteomics allows us to measure which proteins are in each sample, as well as how much of specific proteins are in each sample. This information is very important, as proteins play such a critical role in immunological processes. For example, … [cytokines? receptors? enzymes? others?]. Proteins perform critical processes in the body, including catalzying reactions (enzymes), providing structure (…), and sending signals across cells (…). They have such a critical role in health and disease that most drug targets are proteins [ref].

> "Protein molecules, rather than DNA or RNA, carry out most cellular functions. The direct measurement of protein levels and activity within the cell is therefore the best determinant of overall cell function." (Lakhani and Ashworth, 2001)

Other assays can also help in determining protein activity. For example, transcriptomics measure messenger RNA in a sample. Since these messenger RNA

are key to building proteins, based on the expression of genes in the cell, these transcriptomics can be helpful in understanding which proteins are being created in certain cells or samples. However, transcriptomics does not provide a direct measure of protein content, and its results are not perfectly correlated with protein content. This is because … Proteomics directly measures the proteins in the sample, and so provides a more direct picture of these "machines" in the sample.

Proteomics is a bulk, rather than single-cell assay [always true?]. In other words, it will provide estimates of the protein composition of a sample, but for the sample as a whole. The final aim of proteomics will often be to compare the protein composition of different samples. For example, you might identify which proteins are present in samples from treated animals versus control animals, or you may compare whether the amount of a certain protein is higher in diseased versus healthy animals (in fact, this kind of analysis might be used to identify proteins that can serve as a biomarker of that disease).

The analysis process for proteomics is quite complex, but it's helpful to understand when trying to interpret the data. In the assays that we use, the proteins are measured using a mass spectrometer, coupled with a liquid chromatography column (Steen and Mann, 2004).

Before the sample proteins are sent through this equipment, they are broken down into peptides, which are shorter chains of amino acids. A peptide can be measured in terms of its number of residues (another name for amino acids, in the context of their role as monomers in protein sequences), and typically you aim for peptides with about 20 residues for sending through the equipment (Steen and Mann, 2004). [Why not send the whole protein through?] There are some proteins that could not be sent through the mass spectrometer, or if they were would not produce meaningful results, because of characteristics like stability or solubility of that particular protein and how it interacts with the equipment's set-up (Steen and Mann, 2004).

> "After protein purification, the first step is to convert proteins to a set of peptides using a sequence-specific protease. Even though mass spectrometers can measure the mass of intact proteins, there are a number of reasons why peptides, and not proteins, are analysed in proteomics. Proteins can be difficult to handle and might not all be soluble under the same conditions… In addition, the sensitivity of the mass spectrometer for proteins is much lower than for peptides, and the protein may be processed and modified such that the combinatorial effect makes determining the masses of the numerous resulting isoforms impossible. … Most importantly, if the purpose is to identify the protein, sequence information is needed and the mass spectrometer is most efficient at obtaining sequence information from peptides that are up to ~20 residues long, rather than whole proteins. Nevertheless, with very specialized equipment, it is

> becoming possible to derive partial sequence information from intact proteins, which can then be used for identification purposes or the analysis of protein modifications in an approach called 'top-down' protein sequencing." (Steen and Mann, 2004)
>
> "Digesting the protein into a set of peptides also means that the physico-chemical properties of the protein, such as solubility and 'stickiness', become irrelevant. As long as the protein generates a sequence of peptides, at least some of them can be sequenced by the mass spectrometer, even if the protein itself would have been unstable or insoluble under the conditions used." (Steen and Mann, 2004)

To break the proteins down into peptides, we use proteases, which are enzymes that break down proteins at certain spots. [How does this work?]

These peptides are then sent through the equipment. The first part of the process sends them through a liquid chromatography column. The purpose of this column is to separate the peptides, and in a meaningful way, before they are sent through the mass spectrometer. By separating them, you can get some added information to help determine their identity based on how long it takes them to get through the equipment (retention time). Often, the column will be designed to separate them by hydrophobicity, so that a peptide is retained longer or moves through the column more quickly based on its affinity for water. The information about how quickly the peptide moved through this column will be recorded as the "retention time" in the LC/MS read-out data, and will provide a clue to help in identifying which peptide is represented by a certain part of the read-out data.

Once the peptide has passed through the liquid chromatography column, it goes through the mass spectrometer. As it enters, it is ionized [definition]. The chamber [?] of the mass spectrometer employs differences in electrical charge to help identify the ion fragments of the peptide. [More on this] This equipment gives an output in terms of the intensity [?] recorded at specific mass-to-charge ratios.

This process involves breaking down the original sample into small components (proteins are broken into peptides, and then those are broken into ion fragments). The data preprocessing for the resulting data, therefore, involves a lot of work in putting things back together—in other words, identifying the ion fragments and then trying to determine the original peptides and then proteins that they came from.

In proteomics assays that leverage mass spectrometry, the proteins in the sample are broken down into smaller components (peptides), and it is these peptides

rather than the original proteins that are measured in the mass spectrometer (Steen and Mann, 2004). The proteins are broken down into peptides of about 20 residues (amino acids) each using protease, and this is done because the peptides can be measured more easily and efficiently than full proteins (Steen and Mann, 2004). However, this also means that the information will need to be put back together at the end. Once the different peptides are identified and quantified, it's necessary to figure out which protein each originally came from in order to give a picture of the protein composition of the original sample.

This step links the data output from the equipment with specific proteins, to use the read-out to characterize the protein make-up of the original sample. This protein make-up is the scientifically interesting result, as it can help the scientist answer questions about the biological processes at play under the conditions which the sample represents. However, translating the equipment read-out in this way requires several steps that incorporate scientific principles. For example, one piece of the read-out that can help in linking data to a protein is the retention time of each measure. The sample is sent through a liquid chromatography column before it is vaporized in the mass spectrometer. This LC column is designed to separate components of a sample based on their physical characteristics, often using hydrophobicity as the characteristics to separate on. The LC column is designed so that, for example, components that are extremely hydrophobic will pass through very quickly, while those that are extremely hydrophillic will be retained on the column longer, and so will have a longer retention time. Knowing the set-up of the column, as well as knowing the hydrophillic / hydrophobic characteristics of different peptides, means that the retention time can be used as a clue in helping to identify which peptide a given measure from the mass spectrometer represent. To leverage this at the scale of many peptides in several or many samples, it's necessary to have tools that can incorporate this in a way that's more efficient than using the clue by hand to try to identify peptides. Often, preprocessing software will incorporate many of the algorithms like this that are necessary to translate read-outs from a piece of laboratory equipment into something that is scientifically meaningful.

> "The peptides that are generated by protein digestion are not introduced to the mass spectrometer all at once. Instead, they are injected onto a microscale capillary high-performance liquid chromatography (HPLC) column that is directly coupled to, or is 'online' with, the mass spectrometer. The peptides are eluted from these columns using a solvent gradient of increasing organic content, so that the peptide species elute in order of their hydrophobicity." (Steen and Mann, 2004)

> "At the beginning of the 1990s, researchers realized that the peptide-sequencing problem could be converted to a database-matching problem, which would be much simpler to solve. The reason database searching is easier than de novo sequencing is that

> only an infinitesimal fraction of the possible peptide amino-acid sequences actually occur in nature. A peptide-fragmentation spectrum might therefore not contain sufficient information to unambiguously derive the complete amino-acid sequence, but it might still have sufficient information to match it uniquely to a peptide sequence in the database on the basis of the observed and expected fragment ions. There are several different algorithms that are used to search sequence databases with tandem MS-spectra data, and they have names such as PeptideSearch, Sequest, Mascot, Sonar ms/ms, and ProteinProspector." (Steen and Mann, 2004)

8.0.3 Data format description

For proteomics data, we will be getting data that have already been collected and pre-processed by another part of the team.

[More about how these data were pre-processed. Software: Skyline]

The following shows an example of the type of data we will get as an input:

```
## # A tibble: 3,393 x 18
##    Peptide     Protein Replicate  `Precursor Mz` `Precursor Charge` `Product Mz`
##    <chr>       <chr>   <chr>               <dbl>              <dbl>        <dbl>
##  1 QELDEISTNIR Cfp10   091322_LT1           659.                  2        1061.
##  2 QELDEISTNIR Cfp10   091322_LT2           659.                  2        1061.
##  3 QELDEISTNIR Cfp10   091322_LT3           659.                  2        1061.
##  4 QELDEISTNIR Cfp10   091322_LT4           659.                  2        1061.
##  5 QELDEISTNIR Cfp10   091322_LT5           659.                  2        1061.
##  6 QELDEISTNIR Cfp10   091322_LT6           659.                  2        1061.
##  7 QELDEISTNIR Cfp10   091322_LT7           659.                  2        1061.
##  8 QELDEISTNIR Cfp10   091322_LT8           659.                  2        1061.
##  9 QELDEISTNIR Cfp10   091322_LT~           659.                  2        1061.
## 10 QELDEISTNIR Cfp10   091322_LT~           659.                  2        1061.
## # i 3,383 more rows
## # i 12 more variables: `Product Charge` <dbl>, `Fragment Ion` <chr>,
## #   `Retention Time` <dbl>, Area <dbl>, Background <dbl>, `Peak Rank` <dbl>,
## #   `Ratio Dot Product` <chr>, `Total Area Normalized` <chr>,
## #   `Total Area Ratio` <chr>, `Library Dot Product` <dbl>,
## #   RatioLightToHeavy <dbl>, DotProductLightToHeavy <dbl>
```

These data include the following columns:

- `Peptide`: A short string of peptides that are being measured
- `Protein`: The protein that those peptides come from
- `Replicate`: An identifier for the sample that the measurement was taken on

- `Precursor Mz`, `Precursor Charge`, `Product Mz`, `Product Charge`, `Fragment Ion`, `Retention Time`: Measurements that help in identifying the peptide that is being measured (?)
- `Area`:
- `Background`:
- `Peak Rank`:
- `Ratio Dot Product`:
- `Total Area Normalized`:
- `Total Area Ratio`
- `Library Dot Product`:
- `RatioLightToHeavy`:
- `DotProductLightToHeavy`:

Here are all the unique replicates in this file:

```
##  [1] "091322_LT1"  "091322_LT2"  "091322_LT3"  "091322_LT4"  "091322_LT5"
##  [6] "091322_LT6"  "091322_LT7"  "091322_LT8"  "091322_LT10" "091322_LT11"
## [11] "091322_LT12" "091322_LT13" "091322_LT14" "091322_H1"   "091322_H2"
## [16] "091322_H3"   "091322_H4"   "091322_H5"   "091322_H6"   "091322_H7"
## [21] "091322_H8"   "091322_H9"   "091322_H10"  "091322_H11"  "091322_H12"
## [26] "091322_H13"  "091322_H14"  "091322_TB1"  "091322_TB2"  "091322_TB3"
## [31] "091322_TB4"  "091322_TB5"  "091322_TB6"  "091322_TB7"  "091322_TB8"
## [36] "091322_TB9"  "091322_TB10" "091322_TB11" "091322_TB12"
```

The three groups in this data are labeled with "LT", "H", and "TB" somewhere in the identifier. We can create a new column in the dataset that pulls out this treatment group information:

```
## # A tibble: 3 x 2
## # Groups:   treatment_group [3]
##   treatment_group     n
##   <chr>           <int>
## 1 H                 140
## 2 LT                130
## 3 TB                120
```

```
## # A tibble: 10 x 19
##    Peptide     Protein Replicate  `Precursor Mz` `Precursor Charge` `Product Mz`
##    <chr>       <chr>   <chr>               <dbl>              <dbl>        <dbl>
##  1 QELDEISTNIR Cfp10   091322_LT1           659.                  2        1061.
##  2 QELDEISTNIR Cfp10   091322_LT1           659.                  2         832.
##  3 QELDEISTNIR Cfp10   091322_LT1           659.                  2         703.
##  4 QELDEISTNIR Cfp10   091322_LT1           659.                  2         590.
##  5 QELDEISTNIR Cfp10   091322_LT1           659.                  2         503.
```

```
##  6 QELDEISTNIR Cfp10   091322_LT1          664.                  2        1071.
##  7 QELDEISTNIR Cfp10   091322_LT1          664.                  2         842.
##  8 QELDEISTNIR Cfp10   091322_LT1          664.                  2         713.
##  9 QELDEISTNIR Cfp10   091322_LT1          664.                  2         600.
## 10 QELDEISTNIR Cfp10   091322_LT1          664.                  2         513.
## # i 13 more variables: `Product Charge` <dbl>, `Fragment Ion` <chr>,
## #   `Retention Time` <dbl>, Area <dbl>, Background <dbl>, `Peak Rank` <dbl>,
## #   `Ratio Dot Product` <chr>, `Total Area Normalized` <chr>,
## #   `Total Area Ratio` <chr>, `Library Dot Product` <dbl>,
## #   RatioLightToHeavy <dbl>, DotProductLightToHeavy <dbl>,
## #   treatment_group <chr>
```

```
## [1] "Cfp10"                  "acpM"                   "Ag85A"
## [4] "MtbH37Rv|Rv3841|BfrB"   "MtbH37Rv|Rv1837c|GlcB"  "MtbH37Rv|Rv3418c|GroES"
## [7] "MtbH37Rv|Rv3248c|SahH"  "MtbH37Rv|Rv2031c|hspX"
```

- Cfp10
- acpM
- Ag85A
- MtbH37Rv|Rv3841|BfrB
- MtbH37Rv|Rv1837c|GlcB
- MtbH37Rv|Rv3418c|GroES
- MtbH37Rv|Rv3248c|SahH
- MtbH37Rv|Rv2031c|hspX

8.0.4 Processing collected data

Once data are collected, the file can be run through an R workflow. This workflow will convert the data into a format that is easier to work with for data analysis and visualization. It will also produce a report on the data in the spreadsheet, and ultimately it will also write relevant results in a format that can be used to populate a global database for all experiments in the project.

The next section provides the details of the pipeline. It aims to explain the code that processes the data and generates visualizations. You do not need to run this code step-by-step, but instead can access a script with the full code [here].

To use this reporting template, you need to download it to your computer and save it in the file directory where you saved the data you collected with the data collection template. You can then open RStudio and navigate so that you are working within this directory. You should also make sure that you have installed a few required packages on R on the computer you are using to run the report. These packages are: `tidyverse`, … .

Within RStudio, open the report template file. There is one spot where you will need to change the code in the template file, so it will read in the data from the

version of the template that you saved, which you may have renamed. In the YAML of the report template file, change the file path beside "data:" so that it is the file name of your data file.

Once you've made this change, you can use the "Knit" button in RStudio to create a report from the data file and the report template file.

The report includes the following elements:

- [Element 1]
- [Element 2]
- …

You can download an example of a report created using this template by clicking [here].

When you knit to create the report, it will create a Word file in the same file directory where you put your data file and report template. It will also create and output a version of the data that has been processed (in the case of the weights data, this mainly involves tracking mice as they change cages, to link all weights that are from a single animal). This output fill will be named "…" and, like the report file, will be saved in the same file directory as the data file and the report template.

8.0.5 Details of processing script

This section goes through the code within the report template. It explains each part of the code in detail. You do not need to understand these details to use the report template. However, if you have questions about how the data are being processed, or how the outputs are created, all those details are available in this section.

…

Chapter 9

Single-cell RNA-seq

9.0.1 Downloads

The downloads for this chapter are:

- Data collection format (pre-defined by the assay equipment)
- Report template to process data collected with the data template (when you go to this link, go to the "File" bar in your browser's menu bar, chose "Save As", then save the file as "animal_weights.Rmd")
- Example output from the report template

9.0.2 Overview

Single-cell RNA-seq belongs to an area of assays called transcriptomics, which aim to measure the presence and level of messenger RNA from different genes in a sample. These types of assays help in understanding biological processes because they capture a snapshot of gene expression. Genes within a cell, when expressed, will create new protein products through a process of transcription and translation, with messenger RNA serving to transfer the genetic instructions for the protein product from the DNA to where it can be made into the associated protein. The level of mRNA from a gene isn't a perfect measure of how much of its associated protein is available in the cell, as proteins can degrade and be secreted at different rates, all of which contribute to the amount of protein present within the cell. However, it can help it identifying the proteins that the cell is in the process of making at the moment of sample collection, which is both correlated to some degree with protein composition in the cell and also is indicative of actions the cell might be taking in response to stimuli and environmental conditions.

> "Single-cell transcriptomics relies on the reverse transcription of RNA to complementary DNA and subsequent amplification by PCR or in vitro transcription before deep sequencing—procedures prone to losses or biases. The biases are exaggerated by the need for very high amplification from the small amounts of RNA found in an individual cell. Although technical noise confounds precise measurements of low-abundance transcripts, modern protocols have progressed to the point that single-cell measurements are rich in biological information." (Sandberg, 2014)

Until recently, transcriptomics assays primarily were conducted at a bulk resolution, capturing the gene expression levels on average across the cells in the sample. Recently, methods have been developed to capture gene expression for each cell in the sample, an assay called single-cell RNA-sequencing (scRNA-seq). This assay provides an estimate of the expression of each of a set of genes (often a very large set) for each cell in the original sample. Thus, like flow cytometry, it provides insights at the level of each cell, and as a result, the data collected will include many of [thousands?] of observations, rather than a single observation per sample.

> "Our notion of transcriptomes has been forged mainly by population-level observations that have been the mainstream in biology over the last two decades. We are used to thinking about differences in expression in terms of graded or subtle fold changes when comparing data across entire tissues or conditions. But the actual differences between cells may be far larger. Subsets of cells may experience dramatic changes that are averaged out or diluted by the presence of a large number of nonresponsive cells. In fact, it was shown over 60 years ago that inductive cues often result in all-or-none responses in single cells but these responses are observed as a gradual increase when quantified across the population1. It is clear that assessing gene expression in single cells is critical to better understand cellular behaviors and compositions in developing, adult and pathological tissues. To this end, a long-standing goal has been to enable genome-wide RNA profiling, or transcriptomics, in single cells2,3. Only recently has the technology matured so that biologically meaningful differences can be robustly detected with single-cell RNA-seq. … Widespread adoption of these techniques will have a major impact on our understanding and appreciation of cellular states, the nature of transcription and gene regulation, and our ability to characterize pathological states in disease." (Sandberg, 2014)

This information is helpful in identifying molecular processes at work in the cells in the sample. Further, it can be used to classify each cell in the sample by cell type. To understand how, it's helpful to think about how the different

types of cells in a multicellular organism differ. Every cell in an organism has the same genetic code, outside the potential for a few random variations in DNA from mutations to that cell or its ancestors (e.g., mutation to a cell from exposure to radiation). What makes cells different types—some nerve cells, some muscle cells, some lung tissue cells, some immune cells, and so on—is not their genetics, but how the genes in that genetic code are expressed. Some genes will be regularly expressed in cells of one type but turned off in cells of another type, while other genes might express across multiple cell types, but at different levels. A few genes are expressed in common across all types of cells (for example, housekeeping genes, which help …), but there is enough variation in gene expression across cell types that a profile of this gene expression can be used to group cells in a sample into different types of cells.

> "For example, a recurrent theme in single-cell transcriptome studies is that cells reliably group by their cell type or state when subjected to unsupervised clustering7,8,9,10. Gene expression associated with cell identity or developmental stages thus has a stronger signal than technical noise or biological variability related to dynamic processes such as phase of the cell cycle. Moreover, the power to detect meaningful biological differences from single-cell data is demonstrated by the identification of hundreds to thousands of genes with differences in abundances between cell types7,9." (Sandberg, 2014)

> "The measurement of gene expression in single cells will revolutionize our understanding of gene regulation and resolve many longstanding debates in biology. Cells cluster by cell type or developmental state when grouped according to their expression profiles7,8,9,10. Thus, expression-based clustering allows for the unbiased reconstruction or 'reverse engineering' of cell types in any population or tissue after sequencing enough individual cells (Fig. 1). If the sampling of cells is extensive and sufficiently free from biases, such clustering can reveal all cell types present, including new ones. All cells in a cluster can also be used to derive robust cell-type expression profiles, again in a data-driven manner and without previous knowledge of which marker genes define a tissue or cell type. Single-cell profiling of RNAs is therefore the first method that could lay a foundation for a quantitative, data-driven classification of cell types." (Sandberg, 2014)

This is one key thing that can be done using scRNA-seq. There are other methods that can identify and count cells of different types, providing a profile of the composition of cells in a sample. For example, flow cytometry is often used to characterize immune cell composition in a sample. However, to do so, flow cytometry tags and measures certain proteins on each cell's surface or interior, and rather than measuring every protein, this is limited to a set of

proteins specified in a predefined panel of proteins. This panel is limited in the number of proteins it can include, because only so many different fluourescent tags can be independently distinguished. With scRNA-seq, you can measure the expression levels of orders of magnitude [maybe one order?] more proteins. This means that you may be able to identify clusters of cells of the same type that are discovered *de novo* in that experiment, in addition to cell populations that were anticipated in the sample.

The gene expression within single cells is now studied through a process of single cell RNA-seq, which leverages biological process, like use of the enzyme? reverse transcriptase and PCR?, as well as complex computer algorithms to [put things back together?].

Single cell RNA-seq aims to characterize the messenger RNA levels within each cell of a sample, as this provides a measure of the expression of specific genes within the cell—when a gene is expressed, its information is copied into messenger RNA, which carries the information to [other parts of the cell] where it can be used to create the protein associated with that gene. Different types of cells will express different genes, and cells will express different genes under different conditions, so information about gene expression can be used both to classify cells by cell type and to characterize the activity of cells under different conditions. [Information on transcripts rather than genes?]

To measure levels of messenger RNA within each cell of a sample, sc-RNA-seq leverages a fairly involved set of steps that leverage advanced biological and computational ideas. First, the cells in the sample are lysed (essentially, broken open) to access their contents. The content of messenger RNA in any specific cell is small (Brennecke et al., 2013), and so a process is conducted to amplify that content so it can be measured. This involves first transcribing the information in each messenger RNA to cDNA [complementary DNA?], which can then be amplified using [PCR] (Haque et al., 2017). The transcription uses the enzyme reverse transcriptase (Haque et al., 2017), which [clever leveraging of an existing biological process…]. In the course of this transcriptions, unique molecular identifiers (UMIs) can be added to the transcription primers to identify each initial mRNA molecule and the cell from which the mRNA came [?] (Haque et al., 2017). PCR leverages … [also using biological processes]. Once the cDNA are amplified, the information encoded in them can be extracted using next-generation sequencing (Haque et al., 2017) and analysis of the resulting raw sequencing data, which leverages [advanced algorithms?].

9.0.3 Data format description

9.0.4 Processing collected data

9.0.5 Details of processing script

1. Process the raw sequencing data and "demultiplex" the data
2. Quality control—identify and remove low quality cells and some transcripts

> "Before analysing the single-cell gene expression data, we must ensure that all cellular barcode data correspond to viable cells. Cell QC is commonly performed based on three QC covariates: the number of counts per barcode (count depth), the number of genes per barcode, and the fraction of counts from mitochondrial genes per barcode (Ilicic et al, 2016; Griffiths et al, 2018). The distributions of these QC covariates are examined for outlier peaks that are filtered out by thresholding (Fig 2). These outlier barcodes can correspond to dying cells, cells whose membranes are broken, or doublets. For example, barcodes with a low count depth, few detected genes, and a high fraction of mitochondrial counts are indicative of cells whose cytoplasmic mRNA has leaked out through a broken membrane, and thus, only mRNA located in the mitochondria is still conserved (Fig 2). In contrast, cells with unexpectedly high counts and a large number of detected genes may represent doublets. Thus, high-count depth thresholds are commonly used to filter out potential doublets. Three recent doublet detection tools offer more elegant and potentially better solutions (DoubletDecon: preprint: DePasquale et al, 2018; Scrublet: Wolock et al, 2019; Doublet Finder: McGinnis et al, 2018)." (Luecken and Theis, 2019)

> "In addition to checking the integrity of cells, QC steps must also be performed at the level of transcripts. Raw count matrices often include over 20,000 genes. This number can be drastically reduced by filtering out genes that are not expressed in more than a few cells and are thus not informative of the cellular heterogeneity. A guideline to setting this threshold is to use the minimum cell cluster size that is of interest and leaving some leeway for dropout effects. For example, filtering out genes expressed in fewer than 20 cells may make it diffi- cult to detect cell clusters with fewer than 20 cells. For datasets with high dropout rates, this threshold may also complicate the detection of larger clusters. The choice of threshold should scale with the number of cells in the dataset and the intended downstream analysis." (Luecken and Theis, 2019)

3. Normalization

   "Each count in a count matrix represents the successful capture, reverse transcription and sequencing of a molecule of cellular mRNA (Box 1). Count depths for identical cells can differ due to the variability inherent in each of these steps. Thus, when gene expres- sion is compared between cells based on count data, any difference may have arisen solely due to sampling effects. Normalization addresses this issue by e.g. scaling count data to obtain correct rela- tive gene expression abundances between cells." (Luecken and Theis, 2019)

   "In the same way that cellular count data can be normalized to make them comparable between cells, gene counts can be scaled to improve comparisons between genes. Gene normalization constitutes scaling gene counts to have zero mean and unit variance (z scores). … There is currently no consensus on whether or not to perform normalization over genes. While the popular Seurat tutorials (Butler et al, 2018) generally apply gene scaling, the authors of the Slingshot method opt against scaling over genes in their tutorial (Street et al, 2018). The preference between the two choices revolves around whether all genes should be weighted equally for downstream analysis, or whether the magnitude of expression of a gene is an informative proxy for the importance of the gene. In order to retain as much biological information as possible from the data, we opt to refrain from scaling over genes in this tutorial." (Luecken and Theis, 2019)

4. Transformation

   "After normalization, data matrices are typically log(x+1)-transformed. This transformation has three important effects. Firstly, distances between log-transformed expression values represent log fold changes, which are the canonical way to measure changes in expression. Secondly, log transformation mitigates (but does not remove) the mean–variance relationship in single-cell data (Bren- necke et al, 2013). Finally, log transformation reduces the skew- ness of the data to approximate the assumption of many downstream analysis tools that the data are normally distributed. While scRNA-seq data are not in fact log-normally distributed (Vieth et al, 2017), these three effects make the log transformation a crude, but useful tool. This usefulness is highlighted by downstream applications for differential expression testing (Finak et al, 2015; Ritchie et al, 2015) or batch correction (Johnson et al, 2006; Buttner et al, 2019) that use log transformation for these purposes. It should however be noted that log transformation of normalized data can introduce spurious differential expression effects into the

data (preprint: Lun, 2018). This effect is particularly pronounced when normalization size factor distributions differ strongly between tested groups." (Luecken and Theis, 2019)

Bibliography

Baazim, H., Antonio-Herrera, L., and Bergthaler, A. (2022). The interplay of immunology and cachexia in infection and cancer. *Nature Reviews Immunology*, 22(5):309–321.

Baazim, H., Schweiger, M., Moschinger, M., Xu, H., Scherer, T., Popa, A., Gallage, S., Ali, A., Khamina, K., Kosack, L., et al. (2019). Cd8+ t cells induce cachexia during chronic viral infection. *Nature immunology*, 20(6):701–710.

Barnett, D., Walker, B., Landay, A., and Denny, T. N. (2008). Cd4 immunophenotyping in hiv infection. *Nature Reviews Microbiology*, 6(11):S7–S15.

Ben-David, A. and Davidson, C. E. (2014). Estimation method for serial dilution experiments. *Journal of microbiological methods*, 107:214–221.

Brennecke, P., Anders, S., Kim, J. K., Kołodziejczyk, A. A., Zhang, X., Proserpio, V., Baying, B., Benes, V., Teichmann, S. A., Marioni, J. C., et al. (2013). Accounting for technical noise in single-cell rna-seq experiments. *Nature methods*, 10(11):1093–1095.

Franzblau, S. G., DeGroote, M. A., Cho, S. H., Andries, K., Nuermberger, E., Orme, I. M., Mdluli, K., Angulo-Barturen, I., Dick, T., Dartois, V., et al. (2012). Comprehensive analysis of methods used for the evaluation of compounds against mycobacterium tuberculosis. *Tuberculosis*, 92(6):453–488.

Goldman, E. and Green, L. H. (2015). *Practical Handbook of Microbiology*. CRC Press.

Haque, A., Engel, J., Teichmann, S. A., and Lönnberg, T. (2017). A practical guide to single-cell rna-sequencing for biomedical research and clinical applications. *Genome medicine*, 9(1):1–12.

Hartman, H., Wang, Y., Schroeder Jr, H. W., and Cui, X. (2018). Absorbance summation: a novel approach for analyzing high-throughput elisa data in the absence of a standard. *PloS one*, 13(6):e0198528.

Lakhani, S. R. and Ashworth, A. (2001). Microarray and histopathological analysis of tumours: the future and the past? *Nature Reviews Cancer*, 1(2):151–157.

Luecken, M. D. and Theis, F. J. (2019). Current best practices in single-cell rna-seq analysis: a tutorial. *Molecular systems biology*, 15(6):e8746.

Maecker, H. T., McCoy, J. P., and Nussenblatt, R. (2012). Standardizing immunophenotyping for the human immunology project. *Nature Reviews Immunology*, 12(3):191–200.

Sandberg, R. (2014). Entering the era of single-cell transcriptomics in biology and medicine. *Nature methods*, 11(1):22–24.

Segueni, N., Tritto, E., Bourigault, M.-L., Rose, S., Erard, F., Le Bert, M., Jacobs, M., Di Padova, F., Stiehl, D. P., Moulin, P., et al. (2016). Controlled mycobacterium tuberculosis infection in mice under treatment with anti-il-17a or il-17f antibodies, in contrast to tnf-alpha neutralization. *Scientific Reports*, 6(1):1–17.

Smith, C. M., Baker, R. E., Proulx, M. K., Mishra, B. B., Long, J. E., Park, S. W., Lee, H.-N., Kiritsy, M. C., Bellerose, M. M., Olive, A. J., et al. (2022). Host-pathogen genetic interactions underlie tuberculosis susceptibility in genetically diverse mice. *Elife*, 11:e74419.

Steen, H. and Mann, M. (2004). The abc's (and xyz's) of peptide sequencing. *Nature reviews Molecular cell biology*, 5(9):699–711.

Wilson, G. (1922). The proportion of viable bacteria in young cultures with especial reference to the technique employed in counting. *Journal of bacteriology*, 7(4):405.