ImpactTB/BAA: Standard Operating Procedures for Data Analysis

Colorado State University Coding Team

2022-12-06

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.

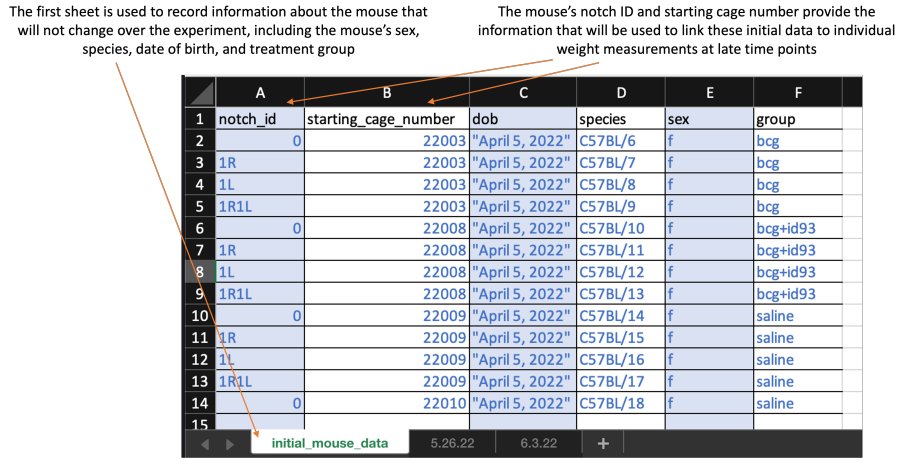
Given these relationships between weight loss, diseases, and immune processes, it is possible that mouse weight might provide a regularly measurable insight into the severity of disease in each animal. While many of data points are collected to measure the final disease state of each animal, fewer are 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

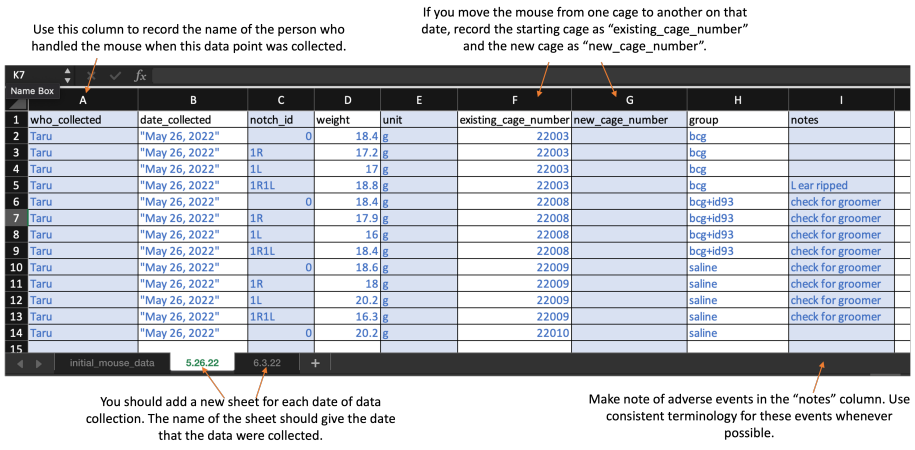
Both the animals' initial conditions and their weekly measures (adverse events, cage changes, and weights) should be recorded in an excel worksheet. You can download a copy of the template here.

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").

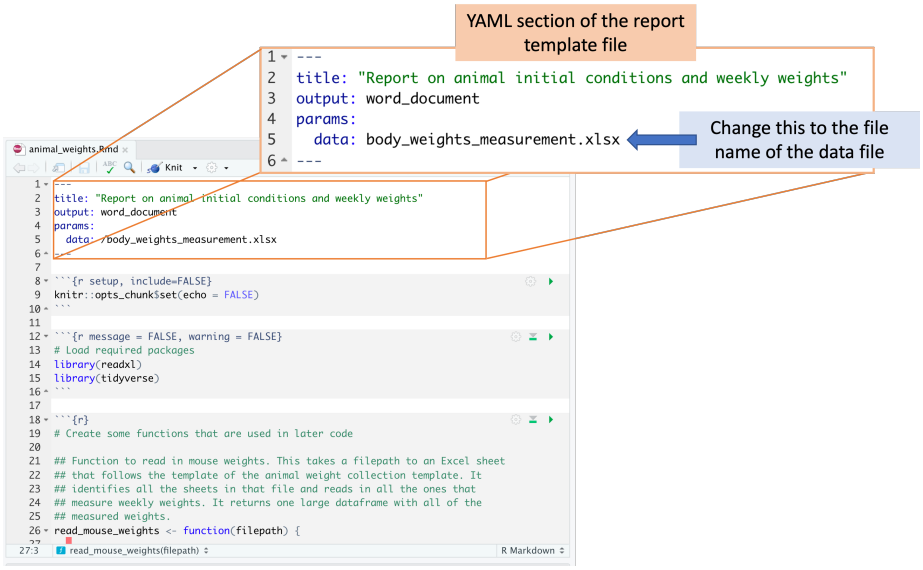
3.0.4 Processing collected data

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.

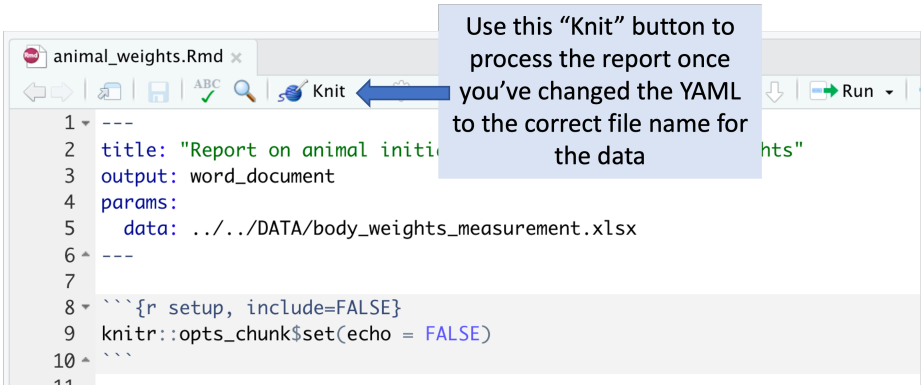
The next section provides the details of the pipeline. It aims to explain the code that processes the data and generates visualizations. You do not need to run this code step-by-step, but instead can access a script with the full code here.

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:
##
##   Hadley Wickham and Jennifer Bryan (2019). readxl: Read Excel Files. R
##   package version 1.3.1. 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 = {2019},
##     note = {R package version 1.3.1},
##     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_colle
                                                                 "text",    # date_coll
                                                                 "text",    # notch_id
                                                                 "numeric", # weight
                                                                 "text",    # unit
                                                                 "text",    # existing_
                                                                 "text",    # new_cage_
                                                                 "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
## # ... with 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.

```
# 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 experiment",
               align = c("r", "l"))
```

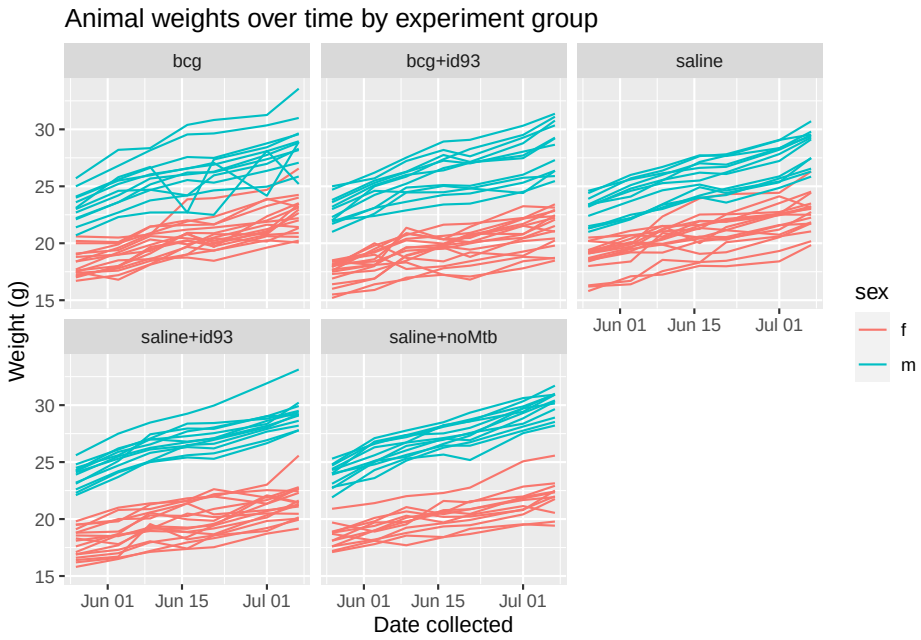
The next piece of code creates a time series of mouse weights over time. The points for each mouse are connected to create a line, so it's easy to see both variation across mice at a single time point and variation in a single mouse over the study. The lines are colored to distinguish male from female 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

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

different small "facets" of the plot, but with the same axis ranges used on each small plot to help comparisons across plots.

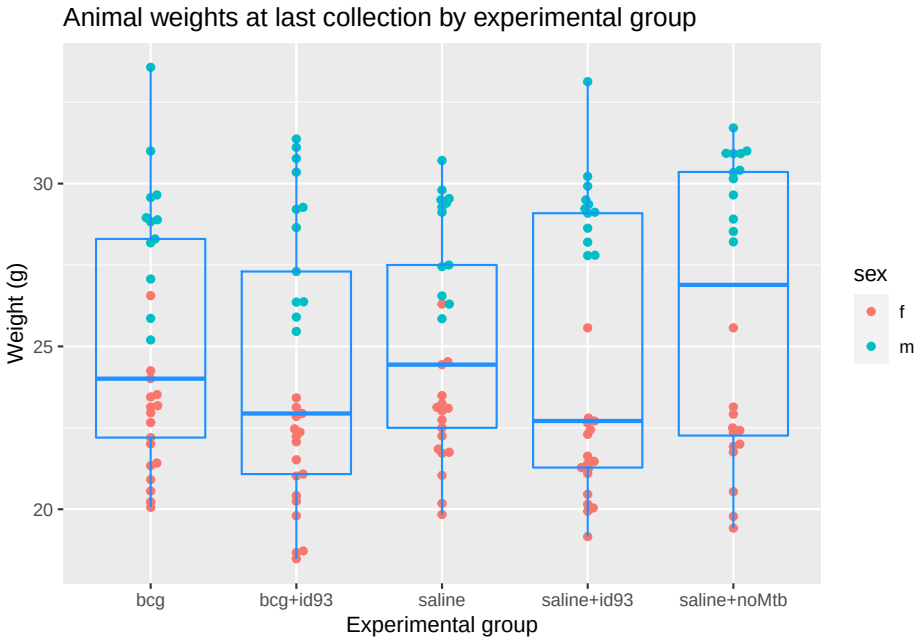
```
# Create a plot of mouse weights over time
our_mouse_weights %>%
  ggplot(aes(x = date_collected, y = weight,
             group = mouse_id, color = sex)) +
  geom_line() +
  facet_wrap(~ group) +
  ggtitle("Animal weights over time by experiment group") +
  labs(x = "Date collected",
       y =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

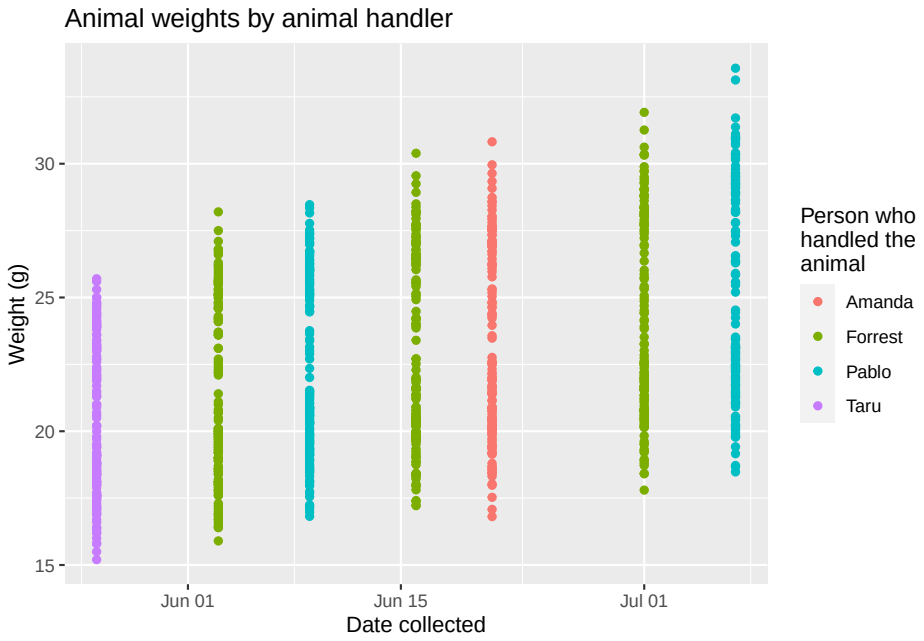
```
# Plot animal weight boxplots for the latest time point
our_mouse_weights %>%
  filter(date_collected == last(date_collected)) %>%
  ggplot(aes(x = group, y = weight)) +
  geom_beeswarm(aes(color = sex)) +
  geom_boxplot(fill = NA, color = "dodgerblue") +
  ggtitle("Animal weights at last collection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

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 measure
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%

```
  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"),
               caption = "Individual data on weight changes in mice between current measurement a
```

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

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.

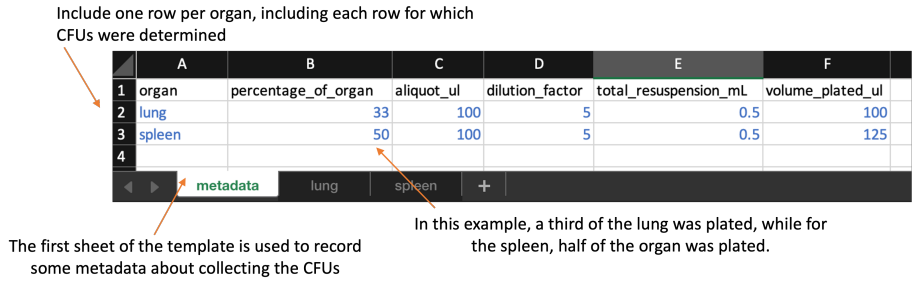
To count the number of CFUs, you need a "just right" dilution (and we often won't know what this is until after plating) to have a countable plate. If you have too high of a dilution (i.e., one with very few viable bacteria), randomness will play a big role in the CFU count, and you'll estimate the original 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:



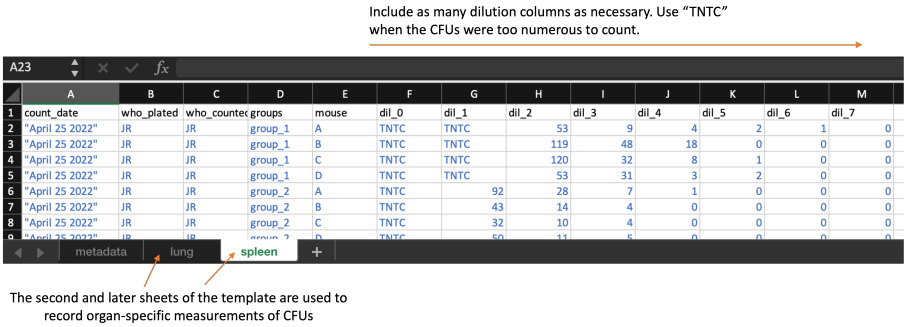
This metadata sheet is used to record information about the overall process of

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:



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: … .

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:

-

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.

```
library(readxl)
library(purrr)
library(tidyverse)
library(gridExtra)
library(ggpubr)
```

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.

```
# Create a list of all the sheets in the file. Exclude the metadata sheet
# to get a list of the organ-specific sheets
sheet_names <- excel_sheets("DATA/baa_cfu_sheet.xlsx")
organ_sheets <- sheet_names[sheet_names != "metadata"]

# Loop through all the organ-specific sheets and read them in as an
```

```
# element in a list. Use pivoting to collect all the dilutions in a
# way that will let you bind across all organs, even if a different
# set of dilutions were used for the different organs.
merged_data <- list()
for(i in 1:length(organ_sheets)){

  data <- read_excel("DATA/baa_cfu_sheet.xlsx",
                     sheet = organ_sheets[i]) %>%
    mutate(organ = paste0(organ_sheets[i]))

  data <- data %>%
    #mutate(missing_col = NA) %>%
    mutate_if(is.double, as.numeric) %>%
    mutate_if(is.numeric, as.character) %>%
    pivot_longer(starts_with("dil_"), names_to = "dilution",
                 values_to = "CFUs") %>%
    mutate(dilution = str_extract(dilution, "[0-9]+"),
           dilution = as.numeric(dilution))

  merged_data[[i]] <- data
}

# Bind all the organ-specific datasets into a single large dataset
all_data <- bind_rows(merged_data, .id = "column_label") %>%
    select(-column_label)
```

At this stage, here is how the beginning of this dataset looks:

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

```
## # A tibble: 5 x 8
##   count_date            who_plated who_counted groups mouse organ dilution CFUs
##   <chr>                 <chr>      <chr>       <chr>  <chr> <chr>    <dbl> <chr>
## 1 "\"February 21 2022\~ BK         BK          group~ A     lung         0 TNTC
## 2 "\"February 21 2022\~ BK         BK          group~ A     lung         1 TNTC
## 3 "\"February 21 2022\~ BK         BK          group~ A     lung         2 TNTC
## 4 "\"February 21 2022\~ BK         BK          group~ A     lung         3 53
## 5 "\"February 21 2022\~ BK         BK          group~ A     lung         4 9
```

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.

```
# Pull out numeric dilution value and filter for dilutions with CFUs between 10-75
tidy_cfu_data <- all_data %>%
  mutate(dilution = str_extract(dilution, "[0-9]+"),
         dilution = as.numeric(dilution)) %>%
  filter((CFUs >= 5 & CFUs <= 95) | groups == "control") %>%
  mutate(CFUs = as.numeric(CFUs))
```

In the example data, this step has reduced the number observations to consider from over 450 to 60.

```
nrow(all_data)
```

```
## [1] 454
```

```
nrow(tidy_cfu_data)
```

```
## [1] 60
```

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):

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

```
## # A tibble: 5 x 8
##   count_date            who_plated who_counted groups mouse organ dilution CFUs
##   <chr>                 <chr>      <chr>       <chr>  <chr> <chr>    <dbl> <chr>
## 1 "\"February 21 2022\~ BK         BK          group~ A     lung         0 TNTC
## 2 "\"February 21 2022\~ BK         BK          group~ A     lung         1 TNTC
## 3 "\"February 21 2022\~ BK         BK          group~ A     lung         2 TNTC
## 4 "\"February 21 2022\~ BK         BK          group~ A     lung         3 53
## 5 "\"February 21 2022\~ BK         BK          group~ A     lung         4 9
```

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

```
## # A tibble: 5 x 8
##   count_date            who_plated who_counted groups mouse organ dilution  CFUs
##   <chr>                 <chr>      <chr>       <chr>  <chr> <chr>    <dbl> <dbl>
## 1 "\"February 21 2022\~ BK         BK          group~ A     lung         3    53
## 2 "\"February 21 2022\~ BK         BK          group~ A     lung         4     9
## 3 "\"February 21 2022\~ BK         BK          group~ C     lung         5     8
## 4 "\"February 21 2022\~ BK         BK          group~ D     lung         3    53
## 5 "\"February 21 2022\~ BK         BK          group~ A     lung         2    92
```

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.

```
# Calculate CFU/ml for every qualifying replicate between 10-75 CFUs.
# Column binding by organ name to the metadata sheet via inner_join().
meta <- read_excel("DATA/baa_cfu_sheet.xlsx", sheet = "metadata")

tidy_cfu_meta_joined <- inner_join(meta, tidy_cfu_data) %>%
  mutate(CFUs_per_ml = (CFUs * (dilution_factor ^ dilution) *
                          (total_resuspension_mL / volume_plated_ul) * 1000)) %>%
  select(organ, count_date, who_plated, who_counted, groups,  mouse, dilution,
         CFUs, CFUs_per_ml)
```

```
## Joining, by = "organ"
```

```
head(tidy_cfu_meta_joined)
```

```
## # A tibble: 6 x 9
##   organ count_date            who_plated who_counted groups mouse dilution  CFUs
##   <chr> <chr>                 <chr>      <chr>       <chr>  <chr>    <dbl> <dbl>
## 1 lung  "\"February 21 2022\~ BK         BK          group~ A            3    53
## 2 lung  "\"February 21 2022\~ BK         BK          group~ A            4     9
## 3 lung  "\"February 21 2022\~ BK         BK          group~ C            5     8
## 4 lung  "\"February 21 2022\~ BK         BK          group~ D            3    53
## 5 lung  "\"February 21 2022\~ BK         BK          group~ A            2    92
## 6 lung  "\"February 21 2022\~ BK         BK          group~ A            4     7
## # ... with 1 more variable: CFUs_per_ml <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:

```
tidy_cfu_meta_joined %>%
  group_by(organ) %>%
  summarize(`Experimental groups` = paste(unique(groups), collapse = ", "),
            `Dates counted` = paste(unique(count_date), collapse = ", "),
            `Total mice` = length(unique(paste(groups, mouse))),
            `Dilutions considered` = paste(sort(unique(as.numeric(dilution))),
                                collapse = ", ")) %>%
  mutate_all(as.character) %>%
  pivot_longer(-organ) %>%
  mutate(name = paste0(name, ":")) %>%
  knitr::kable(align = c("c", "r", "l"),
               caption = "Organ-specific summary of the experiment")
```

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

```
meta %>%
  knitr::kable(col.names = c("Organ", "Percent of organ plated", "Aliquot",
                             "Dilution factor", "Total resuspension",
```

Table 4.1: Organ-specific summary of the experiment

organ	name	value
lung	Experimental groups:	group_1, group_2, group_3, group_4
lung	Dates counted:	"February 21 2022", "April 18 2022"
lung	Total mice:	19
lung	Dilutions considered:	2, 3, 4, 5, 6
spleen	Experimental groups:	group_1, group_2, group_3
spleen	Dates counted:	"April 25 2022"
spleen	Total mice:	10
spleen	Dilutions considered:	1, 2, 3, 4

Table 4.2: Conditions of the CFU collection

Organ	Percent of organ plated	Aliquot	Dilution factor	Total resuspension	Volume plated
lung	33	100	5	0.5	100
spleen	50	100	5	0.5	125

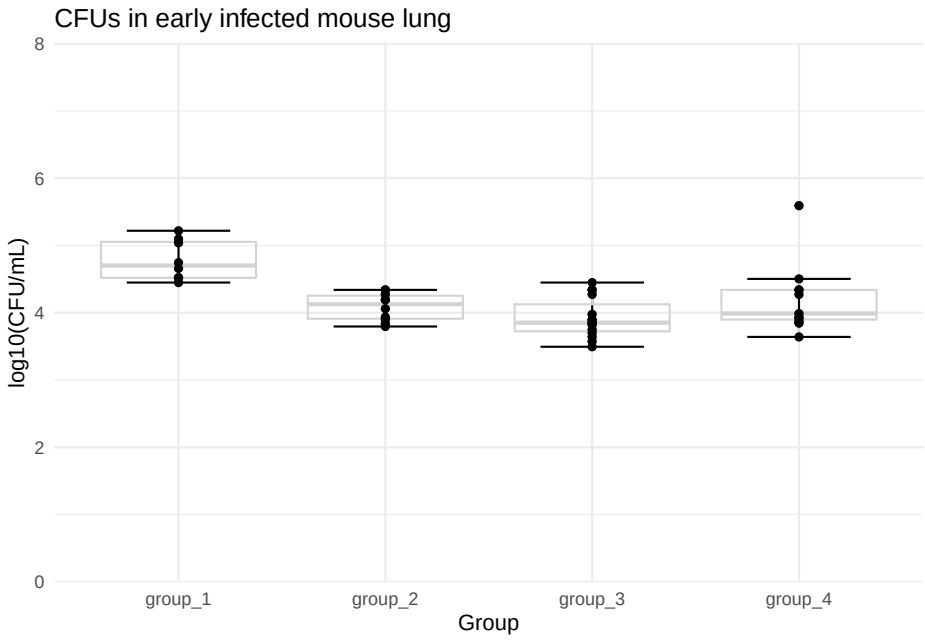
```
                                "Volume plated"),
                    caption = "Conditions of the CFU collection")
```

Distribution of CFUs at each dilution:

Here's a plot that shows how many plates were too numerous to count at each dilution level:

Here is a plot that shows how the CFU counts were distributed by dilution level in the data:

```
tidy_lung_cfu_plot <- tidy_cfu_meta_joined %>%
  filter(organ == "lung") %>%
  mutate(group = fct_relevel(groups, "group_1", "group_2", "group_3", "group_4")) %>%
  ggplot(aes(x = groups, y = log10(CFUs_per_ml), fill = groups))+
  stat_boxplot( aes(x = groups, y = log10(CFUs_per_ml)),
    geom='errorbar', linetype=1, width=0.5)+
  geom_boxplot(aes(group = groups), fill = NA, show.legend = FALSE, color = "lightgrey")+
  geom_point(show.legend = FALSE)+
  labs(title = paste0("CFUs in early infected mouse lung"), x = "Group", y = "log10(CFU/mL)",
       color = "Group")+
  guides(shape = "none")+
  theme_minimal()+
  stat_compare_means(label = "p.signif", method = "t.test", ref.group = "group_1") +
  scale_y_continuous(expand = c(0, 0), limits = c(0, 8))

tidy_lung_cfu_plot
```




```
cfu_stats <- tidy_cfu_meta_joined %>%
  group_by(organ) %>%
  nest() %>%
  mutate(aov_result = map(data, ~aov(CFUs_per_ml ~ groups, data = .x)),
         tukey_result = map(aov_result, TukeyHSD),
         tidy_tukey = map(tukey_result, broom::tidy)) %>%
  unnest(tidy_tukey, .drop = TRUE) %>%
  separate(contrast, into = c("contrast1", "contrast2"), sep = "-") %>%
  select(-data, -aov_result, -tukey_result, -term, -null.value)# %>%
```

```
## Warning: The `.drop` argument of `unnest()` is deprecated as of tidyr 1.0.0.
## All list-columns are now preserved.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated
```

```
  # filter(adj.p.value <= 0.05)

cfu_stats
```

```
## # A tibble: 9 x 7
## # Groups:   organ [2]
##   organ  contrast1 contrast2 estimate conf.low conf.high adj.p.value
##   <chr>  <chr>     <chr>        <dbl>    <dbl>     <dbl>       <dbl>
## 1 lung   group_2   group_1    -60953. -138742.    16836.      0.171
## 2 lung   group_3   group_1    -63903. -135699.     7893.      0.0963
## 3 lung   group_4   group_1    -26214. -102416.    49987.      0.793
```

```
## 4 lung   group_3   group_2     -2950.  -69900.    64000.      0.999
## 5 lung   group_4   group_2     34739.  -36915.   106393.      0.569
## 6 lung   group_4   group_3     37689.  -27410.   102787.      0.417
## 7 spleen group_2   group_1     -6565   -13529.      399.      0.0656
## 8 spleen group_3   group_1     -7310   -13341.    -1279.      0.0178
## 9 spleen group_3   group_2      -745.   -6776.     5286.      0.943
```

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(tidy_cfu_meta_joined, "cfu_output.csv")
```

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.

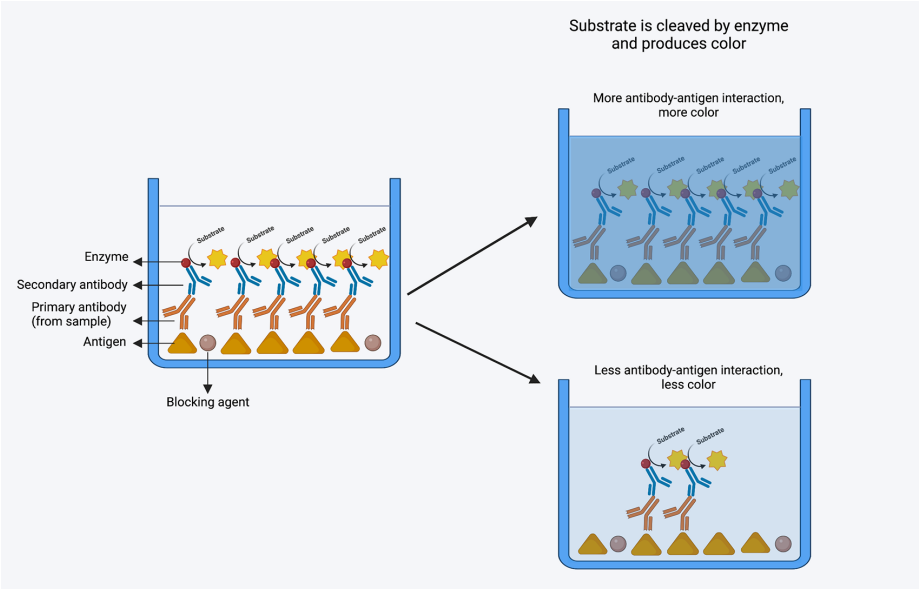


Figure 5.1: A caption

5.1.2 Loading libraries

```
library(readxl)
library(tidyverse)
library(minpack.lm)
library(broom)
library(purrr)
library(ggbeeswarm)
```

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 infor-

mation 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.

```
elisa_example_data <- read_excel("DATA/example_elisa_data.xlsx")
```

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.

```
# Divide dilution column into two seoparate columns

elisa_example_data <- separate(elisa_example_data,
                               col = "dilution",
                               into = c("numerator", "denominator"),
                               sep = "\\/")

# Convert the tabke from character to numeric
elisa_example_data <- elisa_example_data %>%
  mutate_if(is.character, as.numeric)


elisa_example_data$dilution <-
  elisa_example_data$numerator/elisa_example_data$denominator

elisa_example_data <- elisa_example_data %>%
  mutate(log_dilution = log(dilution, base = 3))

head(elisa_example_data)
```

```
## # 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.

```
mod_1 <- nlsLM(absorbance ~
                ((a-d)/(1+(log_dilution/c)^b)) + d,
data = elisa_example_data,
start = list (a = 4, d= 0, c = -5, b = 1))

# a = maximum absorbance
# d = minimum absobance
# c = point of maximum growth
# b = slope at c

mod_1
```

```
## 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
```

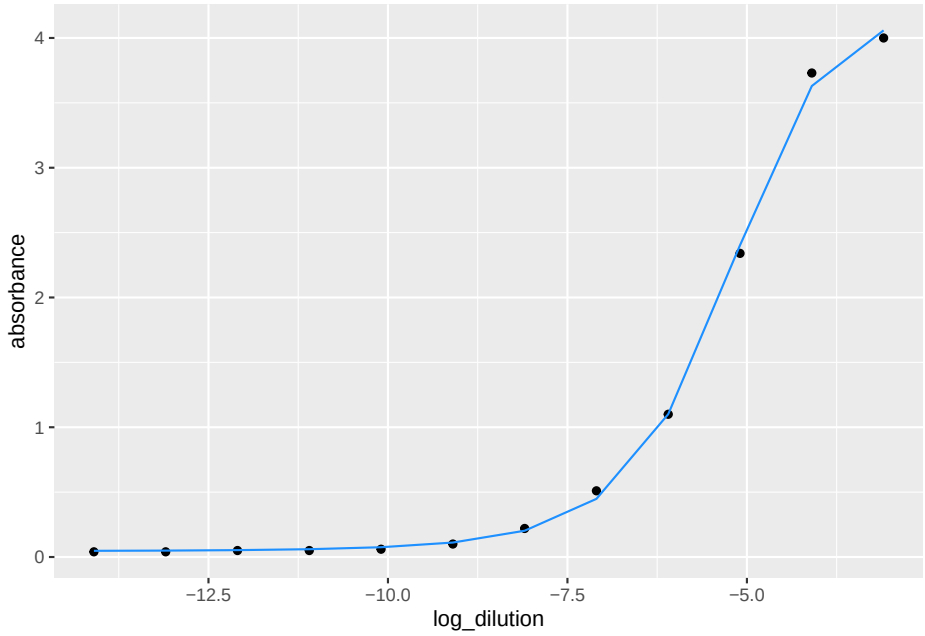
```
summary(mod_1)
```

```
##
## 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
```

```
tidy_params <- mod_1 %>% tidy()

a <- tidy_params$estimate[tidy_params$term == "a"]
b <- tidy_params$estimate[tidy_params$term == "b"]
c <- tidy_params$estimate[tidy_params$term == "c"]
d <- tidy_params$estimate[tidy_params$term == "d"]

elisa_example_data <- elisa_example_data %>%
  mutate(fitted = predict(mod_1))

elisa_example_data <- elisa_example_data %>%
  mutate(fitted = predict(mod_1))
```

```
elisa_example_data %>%
  ggplot(aes(x = log_dilution, y = absorbance)) +
  geom_point() +
  geom_line(aes(y=fitted), color = "dodgerblue")
```



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.

```
endpoint_titer <- c * (((a - d) / (0.2 - d)) - 1) ^ (1 / b)

summary(endpoint_titer)
```

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

```
endpoint_titer
```

```
## [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:

```
AS = 0.04 + 0.04 + 0.05 + 0.05 + 0.06 +
  0.1 + 0.22 + 0.51 + 1.1 + 2.34 + 3.73 + 4.0

AS
```

```
## [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

```
elisa_data <- read_excel("DATA/elisa_data_serial_dilution.xlsx")
```

5.5.0.2 Tidy the data

```
elisa_data <- pivot_longer(data = elisa_data,
                           cols = "Mouse_1":"Mouse_5",
                           names_to = "mouse_id",
                           values_to = "absorbance")

head(elisa_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
```

```
# separate dilution column and convert it to log2

elisa_data <- separate(elisa_data,
                       col = "Dilution",
                       into = c("numerator",
                                "denomenator"),
                       sep = "\\/")

elisa_data <- elisa_data %>%
  transform(numerator = as.numeric(numerator),
            denomenator = as.numeric(denomenator))

elisa_data <- elisa_data %>%
  mutate(dilution =
           elisa_data$numerator/elisa_data$denomenator)

elisa_data <- elisa_data %>%
  mutate(log_dilution = log2(dilution))

head(elisa_data)
```

```
##    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
```

```
elisa_data_df <- elisa_data %>%
  group_by(Groups, mouse_id) %>%
  summarize(log_dilution = log_dilution,
            absorbance = absorbance)
```

5.5.0.2.1 converting data into dataframe

```
## `summarise()` has grouped output by 'Groups', 'mouse_id'. You can override using
## the `.groups` argument.
```

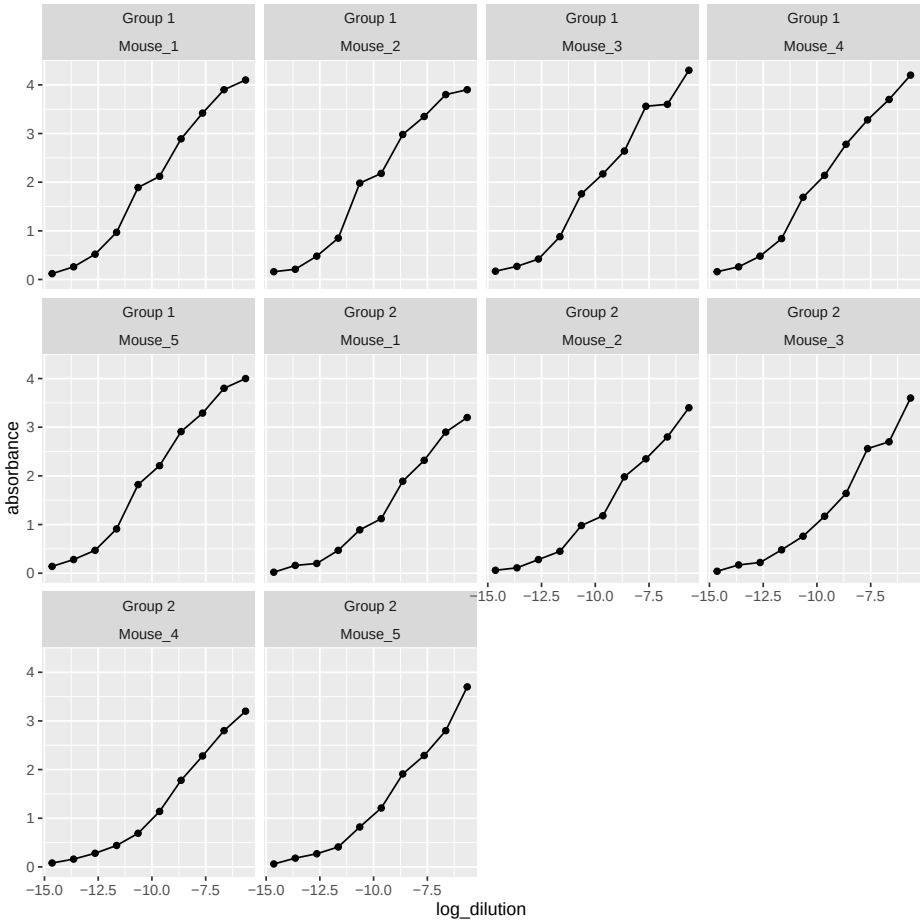
```
elisa_data_nested <- elisa_data %>%
  group_by(Groups, mouse_id) %>%
  nest()

head(elisa_data_nested)
```

```
## # 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]>
```

```
elisa_data %>%
  ggplot(aes(x = log_dilution, y = absorbance)) +
  geom_point() +
  geom_line() +
  facet_wrap(Groups ~ mouse_id)
```

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

```
fitted_model_elisa <- function(df_elisa,
                               start_a, start_d,
                               start_c, start_b) {
  mod_1 <- nlsLM(absorbance ~
                   ((a-d)/(1+(log_dilution/c)^b)) + d,
data = df_elisa,
start = list(a = start_a, d = start_d, c = start_c, b = start_b))
  return(mod_1)
}
```

5.5.1.1 Fitting the model into the dataset

```
fitted_model_elisa(elisa_data_nested$data[[1]],
                   start_a = 4,
                   start_d = 0,
                   start_c = -8,
                   start_b = 1)
```

```
## 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

```
elisa_fitted_data <- elisa_data_nested %>%
  mutate(fitted_data =
           purrr::map(data, ~
                        fitted_model_elisa(.x,start_a = 4,
                                           start_d = 0,
                                           start_c = -8,
                                           start_b = 1)))

head(elisa_fitted_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>
```

5.5.1.3 Take out the summary of the data

```
elisa_fitted_data_summary <- elisa_fitted_data %>%
  mutate(elisa_fitted_data_summary =
           purrr::map(fitted_data, broom::glance))
```

```
unnested <- elisa_fitted_data_summary %>%
unnest(elisa_fitted_data_summary) %>%
ungroup() %>%
dplyr::select(Groups, mouse_id, fitted_data)

unnested$fitted_data[[1]]
```

```
## 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

```
# Fitted model function
fitted_model_elisa <- function(df_elisa,
                               start_a,
                               start_d,
                               start_c,
                               start_b) {
  mod_1 <- nlsLM(absorbance ~
                   ((a-d)/(1+(log_dilution/c)^b)) + d,
data = df_elisa,
start = list(a = start_a, d = start_d, c = start_c, b = start_b))
  return(mod_1)
}

# Endpoint titer function

endpoint_titer_elisa <- function(fitted_data, back_value) {
  tidy_fitted <- broom::tidy(fitted_data)
  est_a <- tidy_fitted$estimate[tidy_fitted$term == "a"]
  est_b <- tidy_fitted$estimate[tidy_fitted$term == "b"]
  est_c <- tidy_fitted$estimate[tidy_fitted$term == "c"]
```

```
  est_d <- tidy_fitted$estimate[tidy_fitted$term == "d"]
  endpoint_titer <- est_c * (((est_a - est_d) / (back_value - est_d)) - 1) ^ (1 / est_b)
  return(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

```
elisa_data_with_fit_model <- elisa_data_nested %>%
  mutate(fitted_data = purrr::map(data,
                                  ~ fitted_model_elisa(.x, start_a = 4,
                                                       start_d = 0,
                                                       start_c = -8,
                                                       start_b = 1)))

head(elisa_data_with_fit_model)
```

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>
```

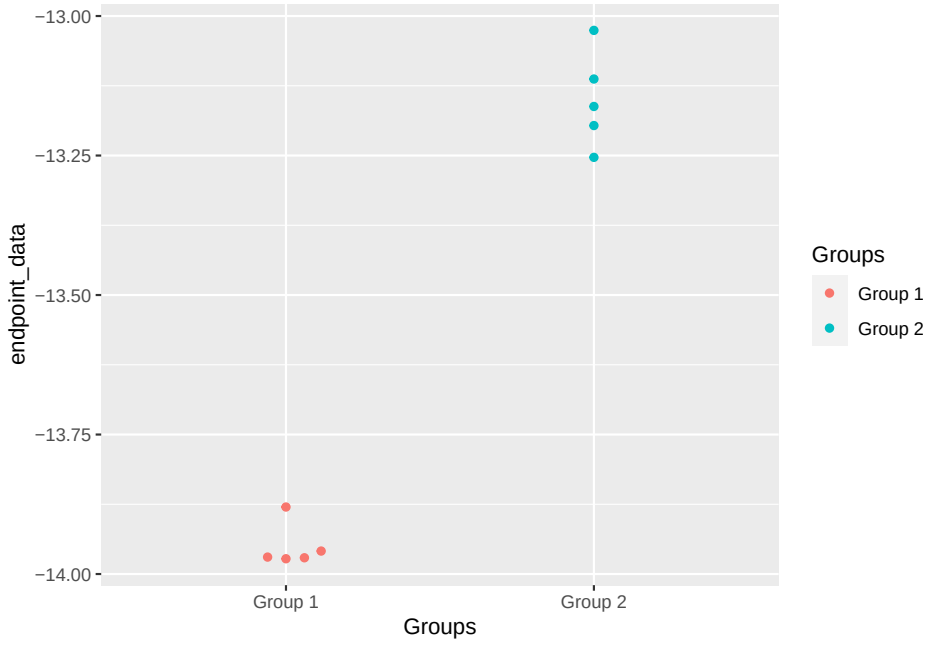
```
elisa_data_with_endpoint_titer <- elisa_data_with_fit_model %>%
  mutate(endpoint_data =
           purrr::map(fitted_data,
                      ~ endpoint_titer_elisa(.x, back_value = 0.2)))
```

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

```
elisa_data_with_endpoint_titer$endpoint_data=
  as.numeric(elisa_data_with_endpoint_titer$endpoint_data)
```

```
elisa_data_with_endpoint_titer %>%
  ggplot(aes(x = Groups, y = endpoint_data, color = Groups)) +
 geom_beeswarm(cex = 3)
```



5.6.0.3 Perform statistical analysis on the data

```
elisa_data_stats <- t.test(endpoint_data ~ Groups,
                           data = elisa_data_with_endpoint_titer)

elisa_data_stats %>%
  tidy()
```

```
## # 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
## # ... with 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.

```
elisa_raw_data <- read_excel("DATA/elisa_s1_07-25-20.xlsx",
                             sheet = "S1", col_names = FALSE,
                             range = "B2:M9")
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
## * ...
```

```
head(elisa_raw_data)
```

```
## # 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.

```
# Convert all columns to numeric

elisa_raw_data_numeric <- elisa_raw_data %>%
  mutate_if(is.character, as.numeric)
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion

## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion

## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```
# pivot longer the data

elisa_raw_data_tidy <- pivot_longer(data = elisa_raw_data_numeric, cols = "...1":"...1

# remove "..." from the first column

elisa_raw_data_tidy$well_id <- str_replace(elisa_raw_data_tidy$well_id, "...", "")

# Add new column to the data_frame

elisa_raw_data_tidy_new <- elisa_raw_data_tidy %>%
  mutate(name = rep(LETTERS[1:8], each = 12))

elisa_raw_data_tidy_new <- elisa_raw_data_tidy_new %>%
  mutate(well_id = paste0(name, well_id)) %>%
  select(-name)

head(elisa_raw_data_tidy_new)
```

```
## # 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.

```
elisa_label_data <- read_excel("DATA/elisa_s1_07-25-20.xlsx",
                               sheet = "S1", col_names = FALSE,
                               range = "Q2:AB9")
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
## * `` -> ...5
```

```
## * ...
```

```
head(elisa_label_data)
```

```
## # 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

```
# pivot longer the data

elisa_label_data_tidy <- pivot_longer(data = elisa_label_data,
                                      cols = "...1":"...12",
                                      names_to = "well_id",
                                      values_to = "information")

# remove "..." from the first column

elisa_label_data_tidy$well_id <- str_replace(elisa_label_data_tidy$well_id, "...", "")

# Add new column to the data_frame

elisa_label_data_tidy_new <- elisa_label_data_tidy %>%
  mutate(name = rep(LETTERS[1:8], each = 12))

elisa_label_data_tidy_new <- elisa_label_data_tidy_new %>%
  mutate(well_id = paste0(name, well_id)) %>%
  select(-name)

head(elisa_label_data_tidy_new)
```

```
## # 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.

```
#Merge the two datasets

elisa_data = elisa_raw_data_tidy_new %>% inner_join(elisa_label_data_tidy_new,
                                                    by="well_id")

head(elisa_data)
```

```
## # 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
```

```
### Separate the information table into sample ID and dilution columns

tidy_elisa_data <- separate(elisa_data, col = "information",
                            into = c("sample_id", "dilution"),
                            sep = "\\(")
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 2 rows [1, 2].
```

```
head(tidy_elisa_data)
```

```
## # 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
```

```
tidy_elisa_data <- tidy_elisa_data %>%
  mutate(dilution = str_extract(dilution, "(/)[0-9]+"),
         dilution = str_replace(dilution, "/", ""),
```

```
         dilution = as.numeric(dilution))

tidy_elisa_data <- tidy_elisa_data %>%
  select(well_id, sample_id, dilution, od_450nm)

head(tidy_elisa_data)
```

```
## # 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.

6.1 Loading packages

```
library(readxl)
library(ggplot2)
library(RColorBrewer)
library(dplyr)
library(tidyverse)
library(scales)
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(stringr)
library(tidyr)
library(knitr)
library(forcats)
library(broom)
library(ggfortify)
library(stats)
library(ggpubr)
library(grDevices)
library(rstatix)
```

```
##
## Attaching package: 'rstatix'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
library(writexl)
```

6.2 panel information

```
# antibody_panel <- read_excel
```

6.3 Loading data

```
Df <- read_excel("DATA/innate_normalizedto45.xlsx", sheet = "CD3CD11b No Day 14")

marker_legend <- read_excel("DATA/marker legend.xlsx")

# Remove Freq of Parent columns
Df1 <- Df %>%
  select(-matches("Parent"))
# Remove "Leukocytes/LIVE/Single Cells/" from col names
names(Df1) <- str_remove(names(Df1), "Leukocytes/LIVE/Single Cells/")

Df1 <- Df1 %>%
  rename_all(funs(str_replace(., "\\|.+", "")))# Remove "|Freq of..." from col names
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
```

```
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```
Df1 <- Df1 %>%
  rename_all(funs(str_replace_all(., "\\/Q[:digit:]+\\:", ""))) %>%
  rename_all(funs(str_replace(., "\\/", " "))) %>%
  rename_all(funs(str_replace(., "\\,", " "))) %>%
  rename_all(funs(str_replace(., "\\ \\,", " ")))

# str_extract_all(names(Df1), "[:alpha:]+[:digit:]+[\\+\\-]")
#
#
#
#
#
#
# marker_select <- function(col_title) {
#   marker_df <- str_detect(names(DATA1), "[\\+\\-]")
#   return(marker_df)
# }
```

6.4 Making the data tidy for plotting

```
tidy_Df1 <- pivot_longer(data = Df1, cols =  starts_with("CD45+"), names_to = "cell_types", value

tidy_Df1 <- tidy_Df1 %>%
  separate(col = "SAMPLE", into = c("day", "replicate"))


tidy_Df1 %>%
  select(cell_types) %>%
  unique()
```

```
## # A tibble: 128 x 1
##    cell_types
##    <chr>
##  1 "CD45+ "
##  2 "CD45+ CD3-   CD11b+ "
##  3 "CD45+ CD3-   CD11b+ CD25+ "
```

```
##  4 "CD45+ CD3-   CD11b+ CD103+ "
##  5 "CD45+ CD3-   CD11b+ gamma_delta "
##  6 "CD45+ CD3-   CD11b+ NKp46+ "
##  7 "CD45+ CD3-   CD11b+ CD11c+  CD64- "
##  8 "CD45+ CD3-   CD11b+ CD11c-  CD64- "
##  9 "CD45+ CD3-   CD11b+ CD86-  CD64+ "
## 10 "CD45+ CD3-   CD11b+ CD86+  CD64+ "
## # ... with 118 more rows
```

```
tidy_Df1 <- tidy_Df1 %>%
  filter(percentage_of_CD45 > 0.005)

head(tidy_Df1, n=10)
```

```
## # A tibble: 10 x 4
##    day   replicate cell_types                          percentage_of_CD45
##    <chr> <chr>     <chr>                                            <dbl>
##  1 CNT   1         "CD45+ "                                          82.9
##  2 CNT   1         "CD45+ CD3-   CD11b+ "                            29.3
##  3 CNT   1         "CD45+ CD3-   CD11b+ CD25+ "                       0.88
##  4 CNT   1         "CD45+ CD3-   CD11b+ CD103+ "                      0.75
##  5 CNT   1         "CD45+ CD3-   CD11b+ gamma_delta "                 4.77
##  6 CNT   1         "CD45+ CD3-   CD11b+ NKp46+ "                      7.3
##  7 CNT   1         "CD45+ CD3-   CD11b+ CD11c+  CD64- "               3.65
##  8 CNT   1         "CD45+ CD3-   CD11b+ CD11c-  CD64- "              24.3
##  9 CNT   1         "CD45+ CD3-   CD11b+ CD86-  CD64+ "               0.43
## 10 CNT   1         "CD45+ CD3-   CD11b+ CD86+  CD64+ "               0.85
```

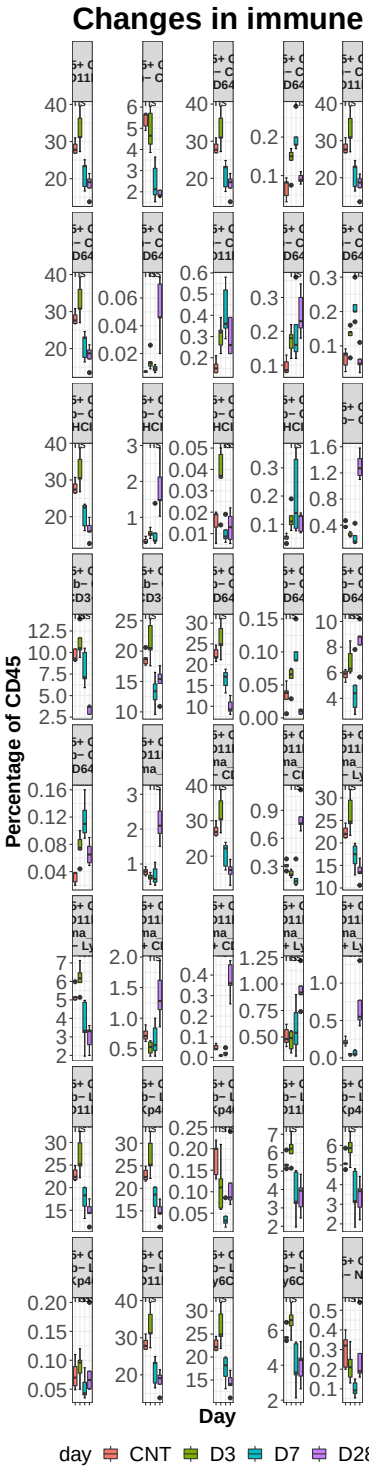
```
# Select CD3 & CD11b populations and create new data frames

CD3pos_CD11bneg <- tidy_Df1 %>%
  filter(str_detect(cell_types, "CD3\\+ + CD11b\\-"))

CD3neg_CD11bpos <- tidy_Df1 %>%
  filter(str_detect(cell_types, "CD3\\- + CD11b\\+"))

CD3neg_CD11bneg <- tidy_Df1 %>%
  filter(str_detect(cell_types, "CD3\\- + CD11b\\-"))
```

6.5 boxplot

```
CD3pos_CD11bneg_bar_plot <- CD3pos_CD11bneg %>%
mutate(day = fct_relevel(day,
            "CNT", "D3", "D7",
            "D28")) %>%
```

```
  ggplot(aes(x = day, y = percentage_of_CD45, fill= day)) +
  stat_boxplot( aes(day, percentage_of_CD45),
    geom='errorbar', linetype=1, width=0.5)+
  geom_boxplot(aes(day, percentage_of_CD45)) +
  facet_wrap(~cell_types, scale = "free_y", labeller = label_wrap_gen(width=15), ncol = 5, nrow =
  theme_bw() +
  theme(axis.text.x = element_blank(), axis.text.y = element_text(size = 20),
        axis.title.x = element_text(size = 20, face = "bold"),
        axis.title.y = element_text(size = 20, face = "bold"),
        legend.text = element_text(size = 20),
        legend.title = element_text(size = 20),
        plot.title = element_text(color="black", size=30, face="bold")) +
  labs (y="Percentage of CD45", x = "Day") +
  theme(strip.text = element_text(size=12, face = "bold")) + theme(legend.position="bottom") +
  ggtitle("Changes in immune cell populations (lung) CD3+ CD11b-") +
  stat_compare_means(label = "p.signif", method = "t.test",
                     ref.group = "CNT")



CD3neg_CD11bpos_bar_plot <- CD3neg_CD11bpos %>%
mutate(day = fct_relevel(day,
             "CNT", "D3", "D7",
             "D28")) %>%
  ggplot(aes(x = day, y = percentage_of_CD45, fill= day)) +
  stat_boxplot( aes(day, percentage_of_CD45),
    geom='errorbar', linetype=1, width=0.5)+
  geom_boxplot( aes(day, percentage_of_CD45)) +
  facet_wrap(~cell_types, scale = "free_y", labeller = label_wrap_gen(width=15), ncol = 5, nrow =
  theme_bw() +
  theme(axis.text.x = element_blank(), axis.text.y = element_text(size = 20),
        axis.title.x = element_text(size = 20, face = "bold"),
        axis.title.y = element_text(size = 20, face = "bold"),
        legend.text = element_text(size = 20),
        legend.title = element_text(size = 20),
        plot.title = element_text(color="black", size=30, face="bold")) +
  labs (y="Percentage of CD45", x = "Day") +
  theme(strip.text = element_text(size=12, face = "bold")) + theme(legend.position="bottom") +
  ggtitle("Changes in immune cell populations (lung) CD3- CD11b+") +
  stat_compare_means(label = "p.signif", method = "t.test",
                     ref.group = "CNT")



CD3neg_CD11bneg_bar_plot <- CD3neg_CD11bneg %>%
```

```
mutate(day = fct_relevel(day,
            "CNT", "D3", "D7",
            "D28")) %>%
  ggplot(aes(x = day, y = percentage_of_CD45, fill= day)) +
  stat_boxplot( aes(day, percentage_of_CD45),
    geom='errorbar', linetype=1, width=0.5)+
  geom_boxplot( aes(day, percentage_of_CD45)) +
  facet_wrap(~cell_types, scale = "free_y", labeller = label_wrap_gen(width=15), ncol =
  theme_bw() +
  theme(axis.text.x = element_blank(), axis.text.y = element_text(size = 20),
        axis.title.x = element_text(size = 20, face = "bold"),
        axis.title.y = element_text(size = 20, face = "bold"),
        legend.text = element_text(size = 20),
        legend.title = element_text(size = 20),
        plot.title = element_text(color="black", size=30, face="bold")) +
  labs (y="Percentage of CD45", x = "Day") +
  theme(strip.text = element_text(size=12, face = "bold")) + theme(legend.position="bot
  ggtitle("Changes in immune cell populations (lung) CD3- CD11b-") +
  stat_compare_means(label = "p.signif", method = "t.test",
                     ref.group = "CNT")

CD3pos_CD11bneg_bar_plot
```


Changes in immune
Percentage of CD45
Day
day CNT D3 D7 D28

```
# CD3neg_CD11bpos_bar_plot
# CD3neg_CD11bneg_bar_plot
```

Chapter 7

Pathology

Chapter 8

Proteomics

For proteomics data, we will be getting data that have already been collected and pre-processed by another part of the team. The following shows an example of the type of data we will get as an input:

```
library(tidyverse)

prot_a <- read_csv("DATA/Transition Results_CCTSI_A.csv")
```

```
## Rows: 3393 Columns: 18
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): Peptide, Protein, Replicate, Fragment Ion, Ratio Dot Product, Tota...
## dbl (11): Precursor Mz, Precursor Charge, Product Mz, Product Charge, Retent...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
prot_a
```

```
## # A tibble: 3,393 x 18
##    Peptide     Protein Replicate   `Precursor Mz` `Precursor Char~` `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_LT10           659.                 2        1061.
```

```
## 10 QELDEISTNIR Cfp10   091322_LT11           659.                 2        1061.
## # ... with 3,383 more rows, and 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`:

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

Here are all the unique replicates in this file:

```
prot_a %>%
  pull(Replicate) %>%
  unique()
```

```
##  [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:

```
prot_a <- prot_a %>%
  mutate(treatment_group = str_extract(Replicate, "[A-Z]+"))

prot_a %>%
  filter(Peptide == first(Peptide)) %>%
  group_by(treatment_group) %>%
  count()
```

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

```
prot_a %>%
  filter(Peptide == first(Peptide) &
           Replicate == first(Replicate))
```

```
## # 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.
## # ... with 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>
```

```
prot_a %>%
  pull(Protein) %>%
  unique()
```

```
## [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

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.

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

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.

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.

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.

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.