Pre-processing and plotting data

Mike Blazanin

2023-01-22

Table of Contents

# Workflow

1. Introduction
2. Importing & transforming data
3. Incorporating design information
4. **Pre-processing and plotting your data**
5. Processing your data
6. Analyzing your data
7. Statistics, merging other data, and other resources

If you haven’t already, load the necessary packages.

library(gcplyr)  
  
library(dplyr)  
library(ggplot2)  
library(lubridate)

#This code was explained in sections 2 and 3  
#Here we're re-running it so it's available for us to work with  
example\_tidydata <- trans\_wide\_to\_tidy(example\_widedata,  
 id\_cols = "Time")  
example\_design <- make\_design(  
 pattern\_split = ",", nrows = 8, ncols = 12,  
 "Bacteria\_strain" = make\_designpattern(  
 values = paste("Strain", 1:48),  
 rows = 1:8, cols = 1:6,  
 pattern = 1:48,  
 byrow = TRUE),  
 "Bacteria\_strain" = make\_designpattern(  
 values = paste("Strain", 1:48),  
 rows = 1:8, cols = 7:12,  
 pattern = 1:48,  
 byrow = TRUE),  
 "Phage" = make\_designpattern(  
 values = c("No Phage"),  
 rows = 1:8, cols = 1:6,  
 pattern = "1"),  
 "Phage" = make\_designpattern(  
 values = c("Phage Added"),  
 rows = 1:8, cols = 7:12,  
 pattern = "1"))  
ex\_dat\_mrg <- merge\_dfs(example\_tidydata, example\_design)  
#> Joining, by = "Well"

# Pre-processing

Now that we have our data and designs merged, we’re almost ready to start processing and analyzing them. However, first we need to carry out any necessary pre-processing steps, like excluding wells that were contaminated or empty, and converting time formats to numeric.

## Pre-processing: excluding data

In some cases, we want to remove some of the wells from our growth curves data before we carry on with downstream analyses. For instance, they may have been left empty, contained negative controls, or were contaminated. We can use dplyr’s filter function to remove those wells that meet criteria we want to exclude.

For instance, let’s imagine that we realized that we put the wrong media into Well B1, and so we should remove it from our analyses. In that case, we can simply:

example\_data\_and\_designs\_filtered <- filter(ex\_dat\_mrg, Well != "B1")  
head(example\_data\_and\_designs\_filtered)  
#> Time Well Measurements Bacteria\_strain Phage  
#> 1 0 A1 0.003 Strain 1 No Phage  
#> 2 0 C1 0.002 Strain 13 No Phage  
#> 3 0 D1 0.002 Strain 19 No Phage  
#> 4 0 E1 0.002 Strain 25 No Phage  
#> 5 0 F1 0.001 Strain 31 No Phage  
#> 6 0 G1 0.002 Strain 37 No Phage

Now we can see that all rows from Well B1 have been excluded. We could do something similar if we realized that a Bacterial strain was contaminated. For instance, if strain 13 was contaminated, we could exclude it (and Well B1) as follows:

example\_data\_and\_designs\_filtered <-   
 filter(ex\_dat\_mrg,   
 Well != "B1", Bacteria\_strain != "Strain 13")  
head(example\_data\_and\_designs\_filtered)  
#> Time Well Measurements Bacteria\_strain Phage  
#> 1 0 A1 0.003 Strain 1 No Phage  
#> 2 0 D1 0.002 Strain 19 No Phage  
#> 3 0 E1 0.002 Strain 25 No Phage  
#> 4 0 F1 0.001 Strain 31 No Phage  
#> 5 0 G1 0.002 Strain 37 No Phage  
#> 6 0 H1 0.002 Strain 43 No Phage

## Pre-processing: converting dates & times into numeric

Growth curve data produced by a plate reader often encodes the timestamp information as a string (e.g. “2:45:11” for 2 hours, 45 minutes, and 11 seconds), while downstream analyses need timestamp information as a numeric (e.g. number of seconds elapsed). Luckily, others have written great packages that make it easy to convert from common date-time text formats into plain numeric formats. Here, we’ll see how to use lubridate to do so:

First we have to create a data frame with time saved as it might be by a plate reader. As usual, **don’t worry how this block of code works**, since it’s just creating an example file in the same format as that output by a plate reader.

ex\_dat\_mrg$Time <-  
 paste(ex\_dat\_mrg$Time %/% 3600,  
 formatC((ex\_dat\_mrg$Time %% 3600) %/% 60,   
 width = 2, flag = 0),  
 formatC((ex\_dat\_mrg$Time %% 3600) %% 60,  
 width = 2, flag = 0),  
 sep = ":")

Let’s take a look at this data.frame. This shows the Time column as it might be written by a plate reader.

head(ex\_dat\_mrg)  
#> Time Well Measurements Bacteria\_strain Phage  
#> 1 0:00:00 A1 0.003 Strain 1 No Phage  
#> 2 0:00:00 B1 0.001 Strain 7 No Phage  
#> 3 0:00:00 C1 0.002 Strain 13 No Phage  
#> 4 0:00:00 D1 0.002 Strain 19 No Phage  
#> 5 0:00:00 E1 0.002 Strain 25 No Phage  
#> 6 0:00:00 F1 0.001 Strain 31 No Phage

We can see that our Time aren’t written in an easy numeric. Instead, they’re in a format that’s easy for a human to understand (but unfortunately not very usable for analysis).

Let’s use lubridate to convert this text into a usable format. lubridate has a whole family of functions that can parse text with hour, minute, and/or second components. You can use hms if your text contains hour, minute, and second information, hm if it only contains hour and minute information, and ms if it only contains minute and second information.

Since the example has all three, we’ll use hms. Once hms has parsed the text, we’ll use another function to convert the output of hms into a pure numeric value: time\_length. By default, time\_length returns in units of seconds, but you can change that by changing the unit argument to time\_length. See ?time\_length for details.

#We have previously loaded lubridate, but if you haven't already then  
#make sure to add the line:  
# library(lubridate)  
  
ex\_dat\_mrg$Time <- time\_length(hms(ex\_dat\_mrg$Time))  
  
head(ex\_dat\_mrg)  
#> Time Well Measurements Bacteria\_strain Phage  
#> 1 0 A1 0.003 Strain 1 No Phage  
#> 2 0 B1 0.001 Strain 7 No Phage  
#> 3 0 C1 0.002 Strain 13 No Phage  
#> 4 0 D1 0.002 Strain 19 No Phage  
#> 5 0 E1 0.002 Strain 25 No Phage  
#> 6 0 F1 0.001 Strain 31 No Phage

And now we can see that we’ve gotten nice numeric Time values! So we can proceed with the next steps of the analysis.

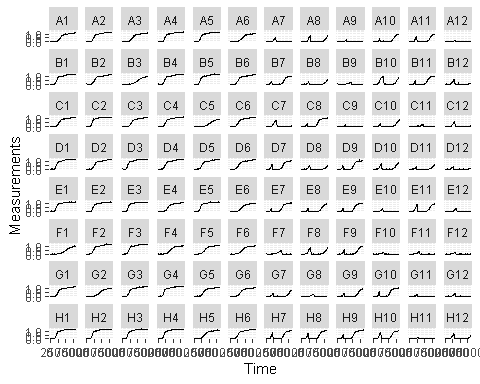
# Plotting your data

Once your data has been merged and times have been converted to numeric, we can easily plot our data using the ggplot2 package. That’s because ggplot2 was specifically built on the assumption that data would be tidy-shaped, which ours is! We won’t go into depth on how to use ggplot here, but there are three main commands to the plot below:

* ggplot - the ggplot function is where you specify the data.frame you would like to use and the *aes*thetics of the plot (the x and y axes you would like)
* geom\_line - tells ggplot how we would like to plot the data, in this case with a line (another common geom for time-series data is geom\_point)
* facet\_wrap - tells ggplot to plot each Well in a separate facet

**We’ll be using this format to plot our data throughout the remainder of this vignette**

#We have previously loaded ggplot2, but if you haven't already then  
#make sure to add the line:  
# library(ggplot2)  
  
#First, we'll reorder the Well levels so they plot in the correct order  
ex\_dat\_mrg$Well <-   
 factor(ex\_dat\_mrg$Well,  
 levels = paste(rep(LETTERS[1:8], each = 12), 1:12, sep = ""))  
  
ggplot(data = ex\_dat\_mrg, aes(x = Time, y = Measurements)) +  
 geom\_line() +  
 facet\_wrap(~Well, nrow = 8, ncol = 12)



Generally speaking, **from here on you should plot your data frequently**, and in every way you can think of! **After every processing and analysis step, visualize both the input data and output data** to understand what the processing and analysis steps are doing and whether they are the right choices for your particular data (this vignette will be doing that too!)

# What’s next?

Now that you’ve pre-processed and visualized your data, it’s time to process (in most cases) and analyze (pretty much always) it!

1. Introduction
2. Importing & transforming data
3. Incorporating design information
4. **Pre-processing and plotting your data**
5. Processing your data
6. Analyzing your data
7. Statistics, merging other data, and other resources