Analyzing growth curve data with gcplyr

A brief R + tidyverse tutorial

Albert Vill

2024-07-15

Background

R is a programming language for statistical computing. Hadley Wickham's `tidyverse` is a suite of tools designed for easy data processing and visualization in an R environment. Mike Blazanin's `gcplyr` is a package built with "tidy" grammar specifically to import, clean, and analyze microbial growth curve data. This short tutorial will take you through the process of installing `gcplyr` and processing real growth curve data.

Installing Software and Libraries

Download and install the most recent versions of R and RStudio Desktop here:

https://posit.co/download/rstudio-desktop/

Check out the RStudio IDE cheatsheet for an overview of the RStudio user interface.

From the RStudio Console (bottom left panel, by default), install the needed packages. In addition to `tidyverse` and `gcplyr`, the `lubridate` package will be used to process time data.

```r
install.packages("tidyverse")
install.packages("gcplyr")
install.packages("lubridate")
```

Once the packages are installed, load them.

```r
library("tidyverse")
library("gcplyr")
library("lubridate")
```

Loading and cleaning data

`gcplyr` includes functions to read the most common "shapes" of growth curve data. Please click this link to download a "wide" dataset. Then, specify the download directory to read your data into R as an object named `plate_data`.

```r
plate_data <-
  read_wides(files = "/path/to/raw/data_2024.03.12.csv")
```

Note that the `read_x` class of functions allows direct reading from URLs.

```r
url <-
  "https://github.com/acvill/KITP_QBio_2024_Turner/raw/main/data/data_2024.03.12.csv"
plate_data <-
  read_wides(files = url)
```

Inspect the data by running `View(plate_data)` or by clicking on the `plate_data` object in the Environment pane (top right panel, by default). Each row includes the file name, the time, the temperature, and the OD_{600} reading in every well of the plate. For this dataset, all rows have the same file name and temperature, so those columns (indices 1 and 3) can be removed with the `dplyr::select()` function.

```r
plate_data <-
  select(.data = plate_data, -c(1,3))
```

Inspect the last ~30 rows of the file. All of the time and OD_{600} observations are `NA`. These empty fields can be removed by filtering to include only rows where time is *not* `NA`. Similar to how `dplyr::select()` selects fields / columns, `dplyr::filter()` filters observations / rows.

```r
plate_data <-
  filter(.data = plate_data, !is.na(Time))
```

The time column is in `hour:minute:second` format. This is easy to read, but difficult to plot. To convert to hours, first specify the time format with `lubridate::hms()`. Then, use the `dplyr::mutate()` function to create a new column called `Hours`, where the minute and second data are encoded as decimal fractions of an hour.

```r
plate_data <-
  mutate(.data = plate_data, Time = hms(Time))
plate_data <-
  mutate(.data = plate_data,
         Hours = as.numeric(hour(Time) + minute(Time)/60 + second(Time)/3600))
plate_data <-
  select(.data = plate_data, -Time)
```

An important principle of "tidy" datasets is that each row should represent a single observation. Right now, each row includes 96 observations – one for each well at the specified time. `tidyr::pivot_longer()` allows us to lengthen the dataset so that one row represents one OD_{600} reading from one well at one time.

```r
plate_data <-
  pivot_longer(data = plate_data,
               cols = -Hours,
               names_to = "Well",
               values_to = "OD600")
```

In the process of selecting, filter, mutating, and pivoting data, things can get messy. When reading in a dataset, R guesses the class types of your data (logical, numeric, character, etc.). Sometimes, it will guess wrong, and this can lead to headaches during downstream analyses. Therefore, it's a good idea to check that your columns are the classes you expect with the `str()` command, which comes from the `utils` package included with base R.

```r
str(plate_data)
```

```
## tibble [10,464 x 3] (S3: tbl_df/tbl/data.frame)
##  $ Hours: num [1:10464] 0.154 0.154 0.154 0.154 0.154 ...
##  $ Well : chr [1:10464] "A1" "A2" "A3" "A4" ...
##  $ OD600: chr [1:10464] "0.091" "0.092" "0.09" "0.078" ...
```

Alas! Our `OD600` field is encoded as characters (`chr`), but we expect it to be numeric (`num`). This is easily fixed.

```
plate_data <-
  mutate(.data = plate_data,
         OD600 = as.numeric(OD600))
```

Instead of rewriting the `plate_data` object with each new manipulation, functions can be chained together with the R pipe (`|>`). When using the pipe with tidyverse functions, the `data` parameter is implicit.

```
url <-
  "https://github.com/acvill/KITP_QBio_2024_Turner/raw/main/data/data_2024.03.12.csv"
plate_data <-
  read_wides(files = url) |>
  select(-c(1,3)) |>
  filter(!is.na(Time)) |>
  mutate(Time = hms(Time)) |>
  mutate(Hours = as.numeric(hour(Time) + minute(Time)/60 + second(Time)/3600)) |>
  select(-Time) |>
  pivot_longer(cols = -Hours,
               names_to = "Well",
               values_to = "OD600") |>
  mutate(OD600 = as.numeric(OD600))
```

Incorporating experimental details

Now that the measurement data have been wrangled, we can add information about the experimental design using `gcplyr::import_blockdesigns()`, including bacterial strains and treatments.

```
strains <-
  "https://github.com/acvill/KITP_QBio_2024_Turner/raw/main/data/strains_2024.03.12.csv"
treatments <-
  "https://github.com/acvill/KITP_QBio_2024_Turner/raw/main/data/treatments_2024.03.12.csv"
plate_design <-
  import_blockdesigns(files = c(strains, treatments),
                      block_names = c("Strain", "Treatment"))
```

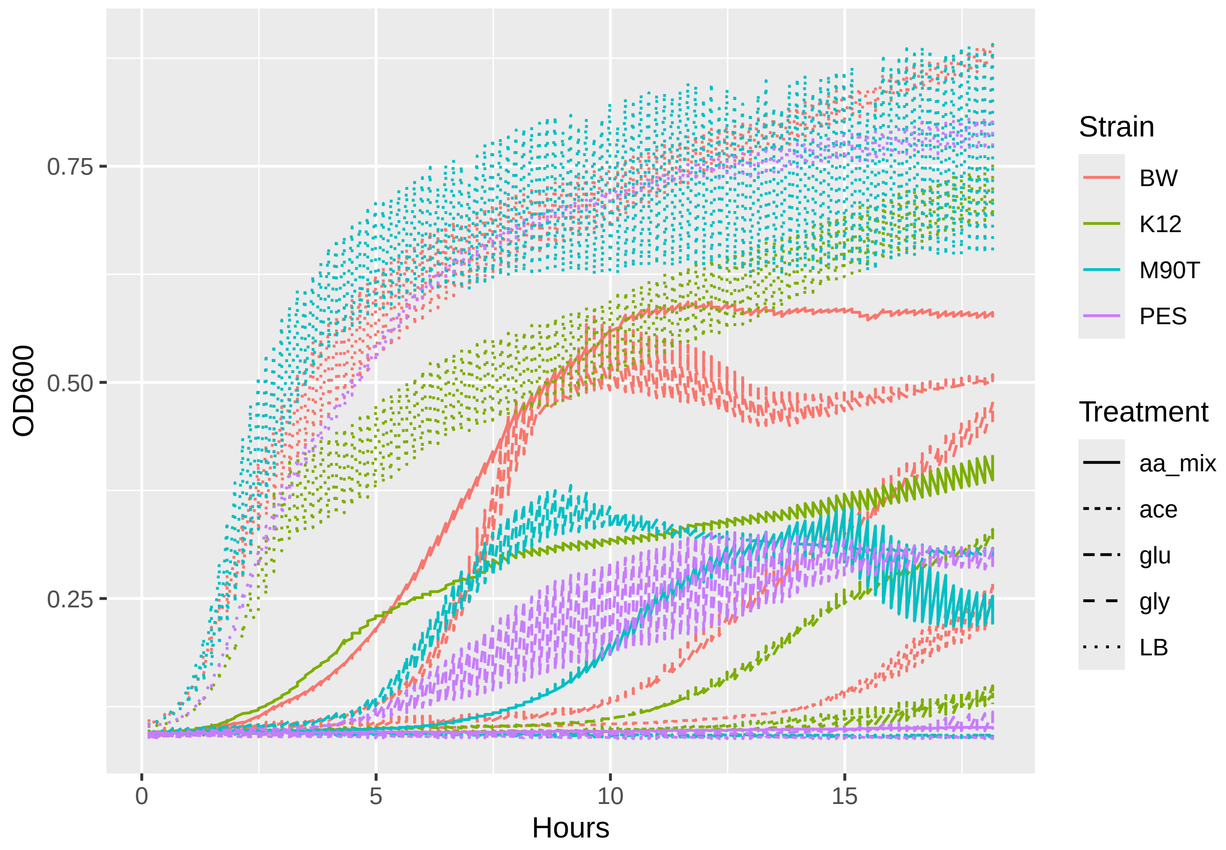
Using the shared `Well` column to associate OD_{600} measurements and the design data, the two objects can be merged with `gcplyr::merge_dfs()`. Empty wells are specified by “blank” in the `Strain` field, and should be removed.

```
data_design <-
  merge_dfs(plate_data, plate_design, by = "Well") |>
  filter(Strain != "blank")
```

Visualizing growth curves

A growth curve shows the change in the optical density of a bacterial culture through time. The simplest visualization is to plot each well as a line, with strains represented by color and treatments represented by linetype. To do this, we will use the `geom_line()` function from `ggplot2`. Check out the `ggplot2` cheatsheet to familiarize yourself with all the different `geoms`.

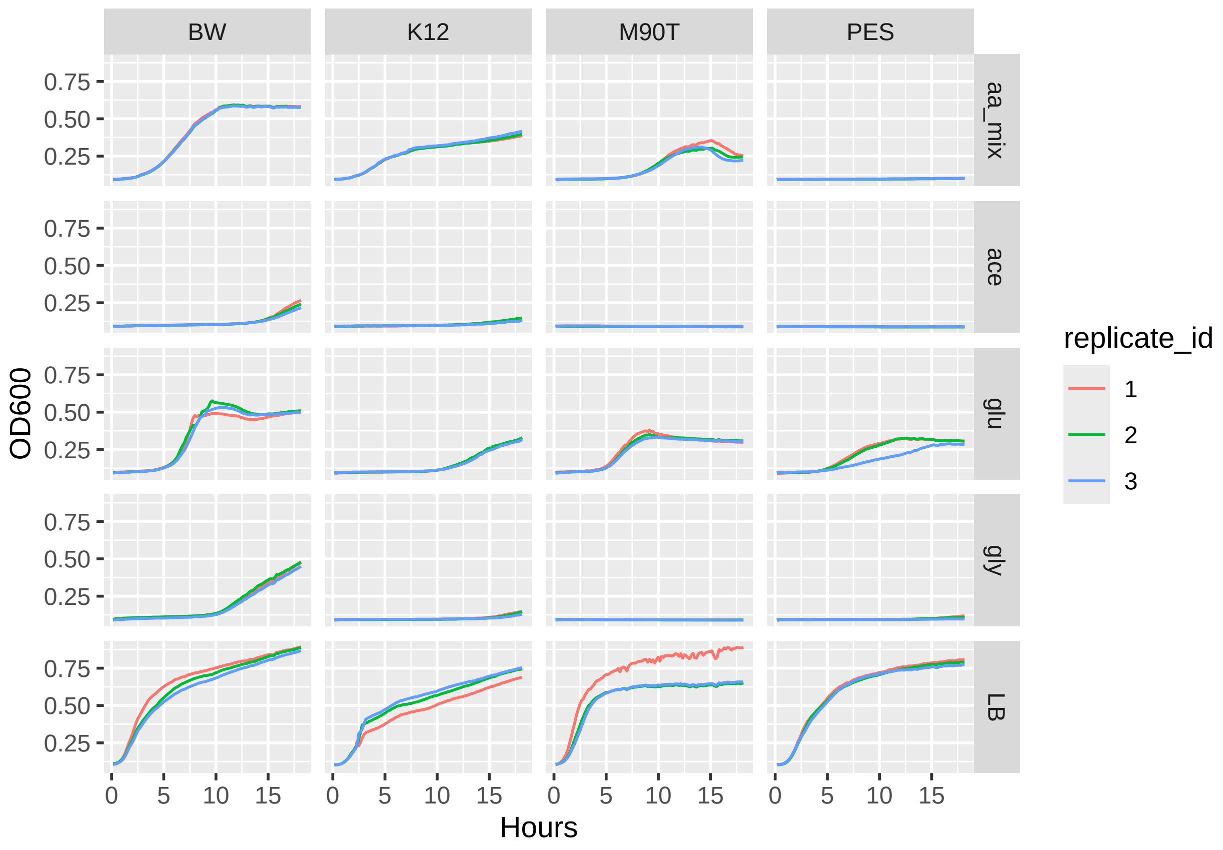
```
plot1 <-
  data_design |>
  ggplot(mapping = aes(x = Hours,
                       y = OD600,
                       color = Strain,
                       linetype = Treatment)) +
  geom_line()

plot1
```



This plot is hard to interpret because linetypes are hard to differentiate. And see that sawtooth pattern? This isn't due to some strange fluctuation in the growth of our strains. Rather, we've neglected to account for replicates: observations with the same `Strain`, `Treatment`, and `Hours` values.

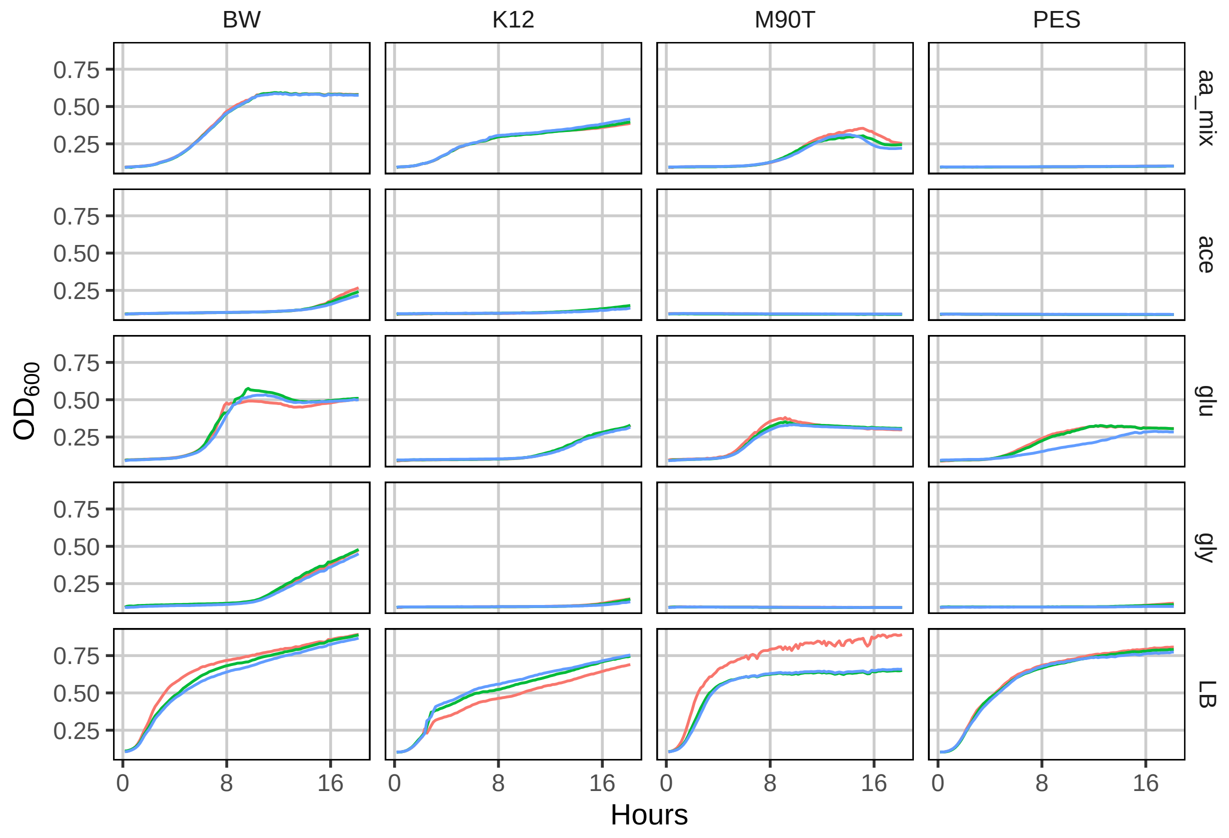
`ggplot2::facet_grid()` allows us to separate a single plot into many subplots. Let's redo the plot above, differentiating replicates by color and faceting by both `Strain` and `Treatment`.

```
plot2 <-
  data_design |>
  group_by(Strain, Treatment, Hours) |>
  mutate(replicate_id = as.factor(row_number())) |>
  ungroup() |>
  ggplot(mapping = aes(x = Hours,
                       y = OD600,
                       color = replicate_id)) +
  geom_line() +
  facet_grid(Treatment ~ Strain)

plot2
```



`ggplot2` allows for easy customization of plots with the `theme` and `scale` families of functions.

```
plot3 <-
  plot2 +
  labs(y = expression(OD[600])) +
  scale_x_continuous(labels = c(0,8,16),
                     breaks = c(0,8,16)) +
  theme(panel.background = element_rect(fill = "white"),
        panel.grid.major = element_line(color = "gray80"),
        panel.border = element_rect(color = "black", fill = NA),
        strip.background = element_blank(),
        legend.position = "none")

plot3
```



Plots can be saved using `ggplot2::ggsave()`, where you can specify the file type, resolution, and dimensions of your figure.

```
setwd("/path/to/plots")
ggsave(filename = "growth_curves_v1.pdf",
       plot = plot3,
       width = 8,
       height = 6,
       units = "in")
```