Working with multiple plates

Mike Blazanin

Contents

Where are we so far?

1. Introduction: `vignette("gc01_gcplyr")`
2. Importing and reshaping data: `vignette("gc02_import_reshape")`
3. Incorporating experimental designs: `vignette("gc03_incorporate_designs")`
4. Pre-processing and plotting your data: `vignette("gc04_preprocess_plot")`
5. Processing your data: `vignette("gc05_process")`
6. Analyzing your data: `vignette("gc06_analyze")`
7. Dealing with noise: `vignette("gc07_noise")`
8. Best practices and other tips: `vignette("gc08_conclusion")`
9. **Working with multiple plates:** `vignette("gc09_multiple_plates")`
10. Using make_design to generate experimental designs: `vignette("gc10_using_make_design")`

So far, we've covered everything you can do with `gcplyr` with a single plate. In this vignette, I'll be briefly covering how `gcplyr` can also easily handle multiple plates of data at the same time. I'll be assuming that you've already read the analogous sections on working with a single plate at a time, so if you haven't, now is a good time to go back and do that!

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

```r
library(gcplyr)

library(dplyr)
library(ggplot2)
```

A brief primer on lists in R

`lists` are a powerful data format in `R`, because they can contain any set of `R` objects. However, this can also make them a little more difficult to navigate than simpler formats like `vectors` or `data.frames`.

Let's make a simple example to explore how lists can be navigated.

```r
mylist <- list("A", c(5, 6, 7), matrix(1:9, nrow = 3))
mylist
#> [[1]]
#> [1] "A"
#>
#> [[2]]
#> [1] 5 6 7
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]    1    4    7
#> [2,]    2    5    8
#> [3,]    3    6    9
```

We've created a list with 3 elements:

- the first element is a vector with only one entry, "A"
- the second element is a vector with three entries
- the third element is a matrix with the numbers 1 through 9

If we want to extract the contents of any of these elements, we simply use double brackets `[[`

```r
mylist[[1]]
#> [1] "A"
mylist[[2]]
#> [1] 5 6 7
mylist[[3]]
#>      [,1] [,2] [,3]
#> [1,]    1    4    7
#> [2,]    2    5    8
#> [3,]    3    6    9
```

If we want to modify the contents of any of these elements, we can also use double brackets

```r
mylist[[3]] <- mylist[[3]] + 7
mylist[[3]]
#>      [,1] [,2] [,3]
#> [1,]    8   11   14
#> [2,]    9   12   15
#> [3,]   10   13   16
```

When working with multiple plates, you'll often be dealing with lists of `data.frames`, rather than the single `data.frames` we've been working with before. With multiple plates, if you want to carry out an operation on only one of the plate `data.frames`, simply use double brackets to select it specifically. If you want to carry out an operation on all of the `data.frames`, `gcplyr` functions like `trans_wide_to_tidy` and `merge_dfs` will be able to handle the list automatically.

Importing data for multiple plates

If your data is block-shaped: use `import_blockmeasures` and start in the next section: **Reading files for multiple block-shaped plates**

If your data is wide-shaped: use `read_wides` and skip down to the **Reading files for multiple wide-shaped plates** section

Reading files for multiple block-shaped plates

If you're reading files for multiple block-shaped plates, your approach depends on how the files themselves are arranged:

- Each plate's worth of block-shaped files is easily separable (e.g. saved in different folders)
- The files from separate plates are interleaved such that they are alternating (as is common when multiple plates are read by a plate-reader robot in a single run)

Separated block-shaped plate files

In the first case, you will have to read each plate of files as an individual plate following the typical process with `import_blockmeasures`.

Let's generate some example files to see how this works. When working with real growth curve data, these files would be output by the plate reader.

```
filenames_sep <- make_example(vignette = 9, example = 1, dir = "./example_data")
#> Files have been written
head(filenames_sep)
#> [1] "./example_data/Plate1-0_00_00.csv" "./example_data/Plate1-0_15_00.csv"
#> [3] "./example_data/Plate1-0_30_00.csv" "./example_data/Plate1-0_45_00.csv"
#> [5] "./example_data/Plate1-1_00_00.csv" "./example_data/Plate1-1_15_00.csv"
```

As we can see, the names for the files have either "Plate1" or "Plate2" in them. We can use that to simply import them separately. If you would like to later work with them as a unit, you can save them into a list.

```
plates <- list(
  plate1 = import_blockmeasures(
    list.files(path = "./example_data/", pattern = "Plate1", full.names = TRUE),
    startrow = 4,
    metadata = list(Time = c(2, "C"))),
  plate2 = import_blockmeasures(
    list.files(path = "./example_data/", pattern = "Plate2", full.names = TRUE),
    startrow = 4,
    metadata = list(Time = c(2, "C"))))
```

Interleaved block-shaped plate files

In the second case, files from different plates are interleaved together. You can read and separate them with `import_blockmeasures`.

Let's generate some example files to see how this works. When working with real growth curve data, these files would be output by the plate reader.

```
filenames_mixed <- make_example(vignette = 9, example = 2, dir = "./example_data")
#> Files have been written
head(filenames_mixed)
#> [1] "./example_data/00_00_00.csv" "./example_data/00_00_01.csv"
#> [3] "./example_data/00_15_00.csv" "./example_data/00_15_01.csv"
#> [5] "./example_data/00_30_00.csv" "./example_data/00_30_01.csv"
```

In this scenario, there are two plates: the first plate was read every 15 minutes, and the second plate was read 1 second after each time the first plate was read. To read them and separate them out, we set the `num_plates` argument to the number of plates.

```
plates <- import_blockmeasures(
    filenames_mixed,
    startrow = 4,
    metadata = list(Time = c(2, "C")),
    num_plates = 2)
```

The output is a list, where the first element in the list is a wide-shaped `data.frame` for plate #1, and the second element is a wide-shaped `data.frame` for plate #2.

Reading files for multiple wide-shaped plates

You can also read multiple wide-shaped datasets. Whether they're from a single file, or from multiple files, simply input the corresponding information and `read_wides` will return a list containing each wide-shaped `data.frame`

```
make_example(vignette = 9, example = 3)
#> Files have been written
#> [1] "./widedata.csv"  "./widedata2.csv"

plates <- read_wides(files = c("widedata.csv", "widedata2.csv"),
           startrow = 5,
           metadata = list(Experiment = c(1, "B"),
                           Start_date = c(2, "B")))
```

Transforming multiple plates to tidy-shaped

Once you have your data in a list of wide-shaped `data.frames`, it's easy to transform it to tidy-shaped. Assuming all the plates have the same columns, you can simply put the list into `trans_wide_to_tidy`.

```
tidy_plates <-
  trans_wide_to_tidy(plates,
                     id_cols = c("file", "Experiment", "Start_date", "Time"))
```

The output is a list, where each element of the list is a tidy-shaped `data.frame` corresponding to one of the plates.

If you don't have any designs to merge, you can collapse this list into a single `data.frame` with `merge_dfs`:

```
tidy_plates_collapsed <- merge_dfs(tidy_plates, collapse = TRUE)
#> Joining with `by = join_by(file, Experiment, Start_date, Time, Well, Measurements)`

print_df(head(tidy_plates_collapsed), col.names = TRUE)
#> file Experiment Start_date Time Well Measurements
#> widedata Experiment_1 2024-07-09 0 A1 0.002
#> widedata Experiment_1 2024-07-09 0 B1 0.002
#> widedata Experiment_1 2024-07-09 0 C1 0.002
#> widedata Experiment_1 2024-07-09 0 D1 0.002
#> widedata Experiment_1 2024-07-09 0 E1 0.002
#> widedata Experiment_1 2024-07-09 0 F1 0.002
```

Merging designs with multiple plates

Plates with different designs

If you have different designs for each of your plates, you'll have to merge them with each plate separately using brackets:

```
example_design1 <- 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))
#> Inferred 'into' column names as: Bacteria_strain
example_design2 <- make_design(
  pattern_split = ",", nrows = 8, ncols = 12,
  "Bacteria_strain" = make_designpattern(
    values = paste("Strain", 49:96),
    rows = 1:8, cols = 1:6, pattern = 1:48, byrow = TRUE),
  "Bacteria_strain" = make_designpattern(
    values = paste("Strain", 49:96),
    rows = 1:8, cols = 7:12, pattern = 1:48, byrow = TRUE))
#> Inferred 'into' column names as: Bacteria_strain

tidy_plates[[1]] <- merge_dfs(tidy_plates[[1]], example_design1)
#> Joining with `by = join_by(Well)`
tidy_plates[[2]] <- merge_dfs(tidy_plates[[2]], example_design2)
#> Joining with `by = join_by(Well)`
```

Then afterwards, you can collapse your data together:

```
data_and_designs <- merge_dfs(tidy_plates, collapse = TRUE)
#> Joining with `by = join_by(file, Experiment, Start_date, Time, Well, Measurements,
#> Bacteria_strain)`
print_df(head(data_and_designs))
#> widedata Experiment_1 2024-07-09 0 A1 0.002  Strain 1
#> widedata Experiment_1 2024-07-09 0 B1 0.002  Strain 7
#> widedata Experiment_1 2024-07-09 0 C1 0.002 Strain 13
#> widedata Experiment_1 2024-07-09 0 D1 0.002 Strain 19
#> widedata Experiment_1 2024-07-09 0 E1 0.002 Strain 25
#> widedata Experiment_1 2024-07-09 0 F1 0.002 Strain 31
```

Plates with a shared design

If, instead, all your plates have the same design, you can merge the design with all of them at once by collapsing your list of data in the merge step. Don't worry about losing track of your plates, the `file` column that is automatically generated should differentiate the separate plates you read in.

```
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))
#> Inferred 'into' column names as: Bacteria_strain

data_and_designs <- merge_dfs(tidy_plates, example_design, collapse = TRUE)
#> Joining with `by = join_by(file, Experiment, Start_date, Time, Well, Measurements,
#> Bacteria_strain)`
#> Joining with `by = join_by(Well, Bacteria_strain)`
print_df(head(data_and_designs))
#> widedata Experiment_1 2024-07-09 0 A1 0.002  Strain 1
#> widedata Experiment_1 2024-07-09 0 B1 0.002  Strain 7
#> widedata Experiment_1 2024-07-09 0 C1 0.002 Strain 13
#> widedata Experiment_1 2024-07-09 0 D1 0.002 Strain 19
#> widedata Experiment_1 2024-07-09 0 E1 0.002 Strain 25
#> widedata Experiment_1 2024-07-09 0 F1 0.002 Strain 31
```