Incorporating design information

Mike Blazanin

Contents

Where are we so far?

1. Introduction: `vignette("gcplyr")`
2. Importing and transforming data: `vignette("import_transform")`
3. **Incorporating design information:** `vignette("incorporate_designs")`
4. Pre-processing and plotting your data: `vignette("preprocess_plot")`
5. Processing your data: `vignette("process")`
6. Analyzing your data: `vignette("analyze")`

7. Dealing with noise: `vignette("noise")`
8. Statistics, merging other data, and other resources: `vignette("conclusion")`

So far, we've imported and transformed our measures data into `R`. Now we're going to address how to incorporate our experimental design.

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

```r
library(gcplyr)
```

Including design elements

We often want to combine information about the experimental design with our data. `gcplyr` enables incorporation of design elements in two ways:

1. Designs can be imported from files
2. Designs can be generated in `R` using `make_design`

Reading design elements from files

Users can read block-shaped or tidy-shaped design files:

- If design files are block-shaped, they can be read with `import_blockdesigns`
- If design files are tidy-shaped, they can simply be read with `read_tidys`

Importing block-shaped design files

To import block-shaped design files, use `import_blockdesigns`, which will return a tidy-shaped designs data frame (or list of data frames).

`import_blockdesigns` only requires a list of filenames (or relative file paths) and will return a data.frame (or list of data frames) in a **tidy format** that you can save in R.

A basic example

Let's look at an example. First, we need to create an example file for the sake of this tutorial. **Don't worry how the below code works**, just imagine that you've created this file in Excel.

```r
write.csv(
  file = "mydesign.csv",
  x = matrix(rep(c("Tr1", "Tr2"), each = 48),
             nrow = 8, ncol = 12, dimnames = list(LETTERS[1:8], 1:12)))
```

Now let's take a look at what the file looks like:

```
print_df(read.csv("mydesign.csv", header = FALSE, colClasses = "character"))
#>     1   2   3   4   5   6   7   8   9  10  11  12
#> A Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
#> B Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
#> C Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
#> D Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
#> E Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
#> F Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
#> G Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
#> H Tr1 Tr1 Tr1 Tr1 Tr1 Tr1 Tr2 Tr2 Tr2 Tr2 Tr2 Tr2
```

Here we can see that our design has Treatment 1 on the left-hand side of the plate (wells in columns 1 through 6), and Treatment 2 on the right-hand side of the plate (wells in columns 7 through 12). Let's import this design using `import_blockdesigns` saving it with the column name `Treatment_numbers`.

```
my_design <- import_blockdesigns(files = "mydesign.csv",
                                 block_names = "Treatment_numbers")
head(my_design, 20)
#>    Well Treatment_numbers
#> 1    A1               Tr1
#> 2    A2               Tr1
#> 3    A3               Tr1
#> 4    A4               Tr1
#> 5    A5               Tr1
#> 6    A6               Tr1
#> 7    A7               Tr2
#> 8    A8               Tr2
#> 9    A9               Tr2
#> 10  A10               Tr2
#> 11  A11               Tr2
#> 12  A12               Tr2
#> 13   B1               Tr1
#> 14   B2               Tr1
#> 15   B3               Tr1
#> 16   B4               Tr1
#> 17   B5               Tr1
#> 18   B6               Tr1
#> 19   B7               Tr2
#> 20   B8               Tr2
```

Importing multiple block-shaped design elements

What do you do if you have multiple designs? For instance, what if you have several strains each in several treatments? In that case, simply save each design component as a separate file, and import them all in one go with `import_blockdesigns`.

First, let's create another example designs file. Again, **don't worry how the below code works**, just imagine that you've created this file in Excel.

```
write.csv(
  file = "mydesign2.csv",
  x = matrix(rep(c("StrA", "StrB", "StrC", "StrD"), each = 24),
             nrow = 8, ncol = 12, dimnames = list(LETTERS[1:8], 1:12),
             byrow = TRUE))
```

Now let's take a look at what the file looks like:

```
print_df(read.csv("mydesign2.csv", header = FALSE, colClasses = "character"))
#>      1    2    3    4    5    6    7    8    9   10   11   12
#> A StrA StrA StrA StrA StrA StrA StrA StrA StrA StrA StrA StrA
#> B StrA StrA StrA StrA StrA StrA StrA StrA StrA StrA StrA StrA
#> C StrB StrB StrB StrB StrB StrB StrB StrB StrB StrB StrB StrB
#> D StrB StrB StrB StrB StrB StrB StrB StrB StrB StrB StrB StrB
#> E StrC StrC StrC StrC StrC StrC StrC StrC StrC StrC StrC StrC
#> F StrC StrC StrC StrC StrC StrC StrC StrC StrC StrC StrC StrC
#> G StrD StrD StrD StrD StrD StrD StrD StrD StrD StrD StrD StrD
#> H StrD StrD StrD StrD StrD StrD StrD StrD StrD StrD StrD StrD
```

Here we can see that our design has Strain A in the first two rows, Strain B in the next two rows, and so on.

Let's now import both designs using `import_blockdesigns`, saving them to columns named `Treatment_numbers` and `Strain_letters`.

```
my_design <-
  import_blockdesigns(files = c("mydesign.csv", "mydesign2.csv"),
                      block_names = c("Treatment_numbers", "Strain_letters"))
head(my_design, 20)
#>    Well Treatment_numbers Strain_letters
#> 1    A1               Tr1           StrA
#> 2    A2               Tr1           StrA
#> 3    A3               Tr1           StrA
#> 4    A4               Tr1           StrA
#> 5    A5               Tr1           StrA
#> 6    A6               Tr1           StrA
#> 7    A7               Tr2           StrA
#> 8    A8               Tr2           StrA
#> 9    A9               Tr2           StrA
#> 10  A10               Tr2           StrA
#> 11  A11               Tr2           StrA
#> 12  A12               Tr2           StrA
#> 13   B1               Tr1           StrA
#> 14   B2               Tr1           StrA
#> 15   B3               Tr1           StrA
#> 16   B4               Tr1           StrA
#> 17   B5               Tr1           StrA
#> 18   B6               Tr1           StrA
#> 19   B7               Tr2           StrA
#> 20   B8               Tr2           StrA
```

Importing tidy-shaped design files

You can import tidy-shaped designs with `read_tidys`.

`read_tidys` only requires a filename (or vector of filenames, or relative file paths) and will return a `data.frame` (or list of data.frames) that you can save in R.

Once these design elements have been read into the `R` environment, you won't need to transform them. So you can skip down to learning how to merge them with your data in the **Merging spectrophotometric and design data** section.

Generating designs in R

If you'd rather make your design data.frames in R, `make_design` can create:

- block-shaped data.frames with your design information (for saving to files)
- tidy-shaped data.frames with your design information (for saving to files and merging with tidy-shaped data)

An example with a single design

Let's start with a simple design.

Imagine you have a 96 well plate (12 columns and 8 rows) with a different bacterial strain in each row, leaving the first and last rows and columns empty.

Row names	Column 1	Column 2	Column 3	...	Column 11	Column 12
Row A	Blank	Blank	Blank	...	Blank	Blank
Row B	Blank	Strain #1	Strain #1	...	Strain #1	Blank
Row B	Blank	Strain #2	Strain #2	...	Strain #2	Blank
...	...	...	...	...	...	...
Row G	Blank	Strain #5	Strain #5	...	Strain #5	Blank
Row G	Blank	Strain #6	Strain #6	...	Strain #6	Blank
Row H	Blank	Blank	Blank	...	Blank	Blank

Typing a design like this manually into a spreadsheet can be tedious. But generating it with `make_design` is easier.

`make_design` first needs some general information, like the `nrows` and `ncols` in the plate, and the `output_format` you'd like (typically `blocks` or `tidy`).

Then, for each different design component, `make_design` needs five different pieces of information:

- a vector containing the possible values
- a vector specifying which rows these values should be applied to
- a vector specifying which columns these values should be applied to
- a string or vector of the pattern of these values
- a Boolean for whether this pattern should be filled byrow (defaults to TRUE)

```r
my_design_blk <- make_design(
  output_format = "blocks",
  nrows = 8, ncols = 12,
  Bacteria = list(c("Str1", "Str2", "Str3", "Str4", "Str5", "Str6"),
                  2:7,
                  2:11,
                  "123456",
                  FALSE)
)
```

So for our example above, we can see:

- the possible values are `c("Strain 1", "Strain 2", "Strain 3", "Strain 4", "Strain 5", "Strain 6")`
- the rows these values should be applied to are `2:7`
- the columns these values should be applied to are `2:11`
- the pattern these values should be filled in by is `"123456"`
- and these values should *not* be filled by row (they should be filled by column)

```
my_design_blk
#> [[1]]
#> [[1]]$data
#>   1  2      3      4      5      6      7      8      9      10     11     12
#> A NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> B NA "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" NA
#> C NA "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" NA
#> D NA "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" NA
#> E NA "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" NA
#> F NA "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" NA
#> G NA "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" NA
#> H NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#>
#> [[1]]$metadata
#> block_name
#> "Bacteria"
```

This produces a `data.frame` with `Bacteria` as the `block_name` in the metadata. If we save this design to a file or transform it to tidy-shaped, this `block_name` metadata will come in handy.

A few notes on the pattern

The pattern in `make_design` is flexible to make it easy to input designs.

The "0" character is reserved for `NA` values, and can be put into your pattern anywhere you'd like to have the value be `NA`

```
my_design_blk <- make_design(
  output_format = "blocks",
  nrows = 8, ncols = 12,
  Bacteria = list(c("Str1", "Str2", "Str3",
                     "Str4", "Str5", "Str6"),
                  2:7,
                  2:11,
                  "123056",
                  FALSE)
)
my_design_blk
#> [[1]]
#> [[1]]$data
#>   1  2      3      4      5      6      7      8      9      10     11     12
#> A NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> B NA "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" NA
#> C NA "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" NA
#> D NA "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" NA
#> E NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> F NA "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" NA
#> G NA "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" NA
#> H NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#>
#> [[1]]$metadata
#> block_name
#> "Bacteria"
```

In the previous examples, I used the numbers 1 through 6 to correspond to our values. If you have more than 9 values, you can use letters too. By default, the order is numbers first, then uppercase letters, then lowercase letters (so “A” is the 10th index). However, if you’d like to only use letters, you can simply specify a different `lookup_tbl_start` so that `make_design` knows what letter you’re using as the `1` index.

```
my_design_blk <- make_design(
  output_format = "blocks",
  nrows = 8, ncols = 12, lookup_tbl_start = "A",
  Bacteria = list(
    c("Str1", "Str2", "Str3", "Str4", "Str5", "Str6"),
    2:7,
    2:11,
    "ABCDEF",
    FALSE)
)
```

You can also specify the pattern as a vector rather than a string.

```
my_design_blk <- make_design(
  output_format = "blocks",
  nrows = 8, ncols = 12,
  Bacteria = list(
    c("Str1", "Str2", "Str3", "Str4", "Str5", "Str6"),
    2:7,
    2:11,
    c(1,2,3,4,5,6),
    FALSE)
)
```

Continuing with the example: multiple designs

Now let’s return to our example growth curve experiment. *In addition* to having a different bacterial strain in each row, we now also have a different media in each column of the plate.

Row names	Column 1	Column 2	Column 3	...	Column 11	Column 12
Row A	Blank	Blank	Blank	...	Blank	Blank
Row B	Blank	Media #1	Media #2	...	Media #10	Blank
...	...	...	...	...	...	...
Row G	Blank	Media #1	Media #2	...	Media #10	Blank
Row H	Blank	Blank	Blank	...	Blank	Blank

We can generate both designs with `make_design`:

```
my_design_blk <- make_design(
  output_format = "blocks",
  nrows = 8, ncols = 12, lookup_tbl_start = "a",
  Bacteria = list(c("Str1", "Str2", "Str3",
                    "Str4", "Str5", "Str6"),
                  2:7,
                  2:11,
                  "abcdef",
                  FALSE),
  Media = list(c("Med1", "Med2", "Med3",
                 "Med4", "Med5", "Med6",
                 "Med7", "Med8", "Med9",
                 "Med10", "Med11", "Med12"),
               2:7,
               2:11,
               "abcdefghij")
  )

my_design_blk
#> [[1]]
#> [[1]]$data
#>   1  2      3      4      5      6      7      8      9      10     11     12
#> A NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> B NA "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" NA
#> C NA "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" NA
#> D NA "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" NA
#> E NA "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" "Str4" NA
#> F NA "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" NA
#> G NA "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" NA
#> H NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#>
#> [[1]]$metadata
#> block_name
#> "Bacteria"
#>
#>
#> [[2]]
#> [[2]]$data
#>   1  2      3      4      5      6      7      8      9      10     11      12
#> A NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA      NA
#> B NA "Med1" "Med2" "Med3" "Med4" "Med5" "Med6" "Med7" "Med8" "Med9" "Med10" NA
#> C NA "Med1" "Med2" "Med3" "Med4" "Med5" "Med6" "Med7" "Med8" "Med9" "Med10" NA
#> D NA "Med1" "Med2" "Med3" "Med4" "Med5" "Med6" "Med7" "Med8" "Med9" "Med10" NA
#> E NA "Med1" "Med2" "Med3" "Med4" "Med5" "Med6" "Med7" "Med8" "Med9" "Med10" NA
#> F NA "Med1" "Med2" "Med3" "Med4" "Med5" "Med6" "Med7" "Med8" "Med9" "Med10" NA
#> G NA "Med1" "Med2" "Med3" "Med4" "Med5" "Med6" "Med7" "Med8" "Med9" "Med10" NA
#> H NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA      NA
#>
#> [[2]]$metadata
#> block_name
#>    "Media"
```

However, the real strength of `make_design` is that it is not limited to simple alternating patterns. `make_design` can use irregular patterns too, replicating them as needed to fill all the wells.

```
my_design_blk <- make_design(
  output_format = "blocks",
  nrows = 8, ncols = 12, lookup_tbl_start = "a",
  Bacteria = list(c("Str1", "Str2"),
                  2:7,
                  2:11,
                  "abaaabbbab",
                  FALSE),
  Media = list(c("Med1", "Med2", "Med3"),
               2:7,
               2:11,
               "aabbbc000abc"))

my_design_blk
#> [[1]]
#> [[1]]$data
#>   1  2      3      4      5      6      7      8      9      10     11     12
#> A NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> B NA "Str1" "Str2" "Str1" "Str1" "Str1" "Str1" "Str2" "Str1" "Str1" "Str1" NA
#> C NA "Str2" "Str2" "Str1" "Str2" "Str2" "Str2" "Str2" "Str1" "Str2" "Str2" NA
#> D NA "Str1" "Str1" "Str1" "Str1" "Str2" "Str1" "Str1" "Str1" "Str1" "Str2" NA
#> E NA "Str1" "Str2" "Str2" "Str2" "Str2" "Str1" "Str2" "Str2" "Str2" "Str2" NA
#> F NA "Str1" "Str1" "Str2" "Str1" "Str1" "Str1" "Str1" "Str2" "Str1" "Str1" NA
#> G NA "Str2" "Str2" "Str2" "Str1" "Str2" "Str2" "Str2" "Str2" "Str1" "Str2" NA
#> H NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#>
#> [[1]]$metadata
#> block_name
#> "Bacteria"
#>
#>
#> [[2]]
#> [[2]]$data
#>   1  2      3      4      5      6      7      8      9      10     11     12
#> A NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> B NA "Med1" "Med1" "Med2" "Med2" "Med2" "Med3" NA     NA     NA     "Med1" NA
#> C NA "Med2" "Med3" "Med1" "Med1" "Med2" "Med2" "Med2" "Med3" NA     NA     NA
#> D NA NA     "Med1" "Med2" "Med3" "Med1" "Med1" "Med2" "Med2" "Med2" "Med3" NA
#> E NA NA     NA     NA     "Med1" "Med2" "Med3" "Med1" "Med1" "Med2" "Med2" NA
#> F NA "Med2" "Med3" NA     NA     NA     "Med1" "Med2" "Med3" "Med1" "Med1" NA
#> G NA "Med2" "Med2" "Med2" "Med3" NA     NA     NA     "Med1" "Med2" "Med3" NA
#> H NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#>
#> [[2]]$metadata
#> block_name
#>    "Media"
```

There is also an optional helper function called `make_designpattern`. `make_designpattern` just reminds us what arguments are necessary for each design. For example:

```
my_design_blk <- make_design(
  output_format = "blocks",
  nrows = 8, ncols = 12, lookup_tbl_start = "a",
  Bacteria = make_designpattern(
    values = c("Str1", "Str2", "Str3",
               "Str4", "Str5", "Str6"),
    rows = 2:7, cols = 2:11, pattern = "abc0ef",
    byrow = FALSE),
  Media = make_designpattern(
    values = c("Med1", "Med2", "Med3",
               "Med4", "Med5", "Med6",
               "Med7", "Med8", "Med9",
               "Med10", "Med11", "Med12"),
    rows = 2:7, cols = 2:11, pattern = "abcde0ghij"))

my_design_blk
#> [[1]]
#> [[1]]$data
#>   1  2      3      4      5      6      7      8      9      10     11     12
#> A NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> B NA "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" "Str1" NA
#> C NA "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" "Str2" NA
#> D NA "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" "Str3" NA
#> E NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#> F NA "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" "Str5" NA
#> G NA "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" "Str6" NA
#> H NA NA     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA
#>
#> [[1]]$metadata
#> block_name
#> "Bacteria"
#>
#>
#> [[2]]
#> [[2]]$data
#>   1  2      3      4      5      6      7  8      9      10     11      12
#> A NA NA     NA     NA     NA     NA     NA NA     NA     NA     NA      NA
#> B NA "Med1" "Med2" "Med3" "Med4" "Med5" NA "Med7" "Med8" "Med9" "Med10" NA
#> C NA "Med1" "Med2" "Med3" "Med4" "Med5" NA "Med7" "Med8" "Med9" "Med10" NA
#> D NA "Med1" "Med2" "Med3" "Med4" "Med5" NA "Med7" "Med8" "Med9" "Med10" NA
#> E NA "Med1" "Med2" "Med3" "Med4" "Med5" NA "Med7" "Med8" "Med9" "Med10" NA
#> F NA "Med1" "Med2" "Med3" "Med4" "Med5" NA "Med7" "Med8" "Med9" "Med10" NA
#> G NA "Med1" "Med2" "Med3" "Med4" "Med5" NA "Med7" "Med8" "Med9" "Med10" NA
#> H NA NA     NA     NA     NA     NA     NA NA     NA     NA     NA      NA
#>
#> [[2]]$metadata
#> block_name
#>    "Media"
```

For merging our designs with plate reader data, we need it tidy-shaped, so we just need to change the `output_format` to `tidy`.

```
my_design_tdy <- make_design(
  output_format = "tidy",
  nrows = 8, ncols = 12, lookup_tbl_start = "a",
  Bacteria = make_designpattern(
    values = c("Str1", "Str2", "Str3",
               "Str4", "Str5", "Str6"),
    rows = 2:7, cols = 2:11, pattern = "abc0ef",
    byrow = FALSE),
  Media = make_designpattern(
    values = c("Med1", "Med2", "Med3",
               "Med4", "Med5", "Med6",
               "Med7", "Med8", "Med9",
               "Med10", "Med11", "Med12"),
    rows = 2:7, cols = 2:11, pattern = "abcde0ghij"))

head(my_design_tdy, 20)
#>    Well Bacteria Media
#> 1    A1     <NA>  <NA>
#> 2    A2     <NA>  <NA>
#> 3    A3     <NA>  <NA>
#> 4    A4     <NA>  <NA>
#> 5    A5     <NA>  <NA>
#> 6    A6     <NA>  <NA>
#> 7    A7     <NA>  <NA>
#> 8    A8     <NA>  <NA>
#> 9    A9     <NA>  <NA>
#> 10  A10     <NA>  <NA>
#> 11  A11     <NA>  <NA>
#> 12  A12     <NA>  <NA>
#> 13   B1     <NA>  <NA>
#> 14   B2     Str1  Med1
#> 15   B3     Str1  Med2
#> 16   B4     Str1  Med3
#> 17   B5     Str1  Med4
#> 18   B6     Str1  Med5
#> 19   B7     Str1  <NA>
#> 20   B8     Str1  Med7
```

Saving designs to files

If you'd like to save the designs you've created with `make_design` to files, you just need to decide if you'd like them tidy-shaped or block-shaped. Both formats can easily be read back into `R` by `gcplyr`.

Saving tidy-shaped designs

These design files will be less human-readable, but easier to import and merge. Additionally, tidy-shaped files are often better for data repositories, like Dryad. To save tidy-shaped designs, simply use the built-in `write.csv` function.

```
#See the previous section where we created my_design_tdy
write.csv(x = my_design_tdy, file = "tidy_design.csv",
          row.names = FALSE)
```

Saving block-shaped designs

These design files will be more human-readable but slightly more computationally involved to import and merge. For these, use the `gcplyr` function `write_blocks`. Typically, you'll use `write_blocks` to save files in one of two formats:

- `multiple` - each block will be saved to its own `.csv` file
- `single` - all the blocks will be saved to a single `.csv` file, with an empty row in between them

Saving block-shaped designs to multiple files The default setting for `write_blocks` is `output_format = 'multiple'`. This creates one `csv` file for each block. If we set `file = NULL`, the default is to name the files according to the `block_names` in the metadata.

```
# See the previous section where we created my_design_blk
write_blocks(my_design_blk, file = NULL)

# Let's see what the files look like
print_df(read.csv("Bacteria.csv", header = FALSE, colClasses = "character"))
#>    1    2    3    4    5    6    7    8    9   10   11  12
#> A
#> B   Str1 Str1 Str1 Str1 Str1 Str1 Str1 Str1 Str1 Str1
#> C   Str2 Str2 Str2 Str2 Str2 Str2 Str2 Str2 Str2 Str2
#> D   Str3 Str3 Str3 Str3 Str3 Str3 Str3 Str3 Str3 Str3
#> E
#> F   Str5 Str5 Str5 Str5 Str5 Str5 Str5 Str5 Str5 Str5
#> G   Str6 Str6 Str6 Str6 Str6 Str6 Str6 Str6 Str6 Str6
#> H

print_df(read.csv("Media.csv", header = FALSE, colClasses = "character"))
#>    1    2    3    4    5    6 7    8    9   10    11  12
#> A
#> B   Med1 Med2 Med3 Med4 Med5   Med7 Med8 Med9 Med10
#> C   Med1 Med2 Med3 Med4 Med5   Med7 Med8 Med9 Med10
#> D   Med1 Med2 Med3 Med4 Med5   Med7 Med8 Med9 Med10
#> E   Med1 Med2 Med3 Med4 Med5   Med7 Med8 Med9 Med10
#> F   Med1 Med2 Med3 Med4 Med5   Med7 Med8 Med9 Med10
#> G   Med1 Med2 Med3 Med4 Med5   Med7 Med8 Med9 Med10
#> H
```

Saving block-shaped designs to a single file The other setting for `write_blocks` is `output_format = 'single'`. This creates a single `csv` file that contains all the blocks, putting metadata like `block_names` in rows that precede each block.

Let's take a look what the `single` output format looks like:

```
# See the previous section where we created my_design_blk
write_blocks(my_design_blk, file = "Design.csv", output_format = "single")

# Let's see what the file looks like
print_df(read.csv("Design.csv", header = FALSE, colClasses = "character"))
#> block_name Bacteria
#>                    1    2    3    4    5    6    7    8    9   10     11  12
#>           A
#>           B          Str1 Str1 Str1 Str1 Str1 Str1 Str1 Str1 Str1  Str1
#>           C          Str2 Str2 Str2 Str2 Str2 Str2 Str2 Str2 Str2  Str2
#>           D          Str3 Str3 Str3 Str3 Str3 Str3 Str3 Str3 Str3  Str3
#>           E
#>           F          Str5 Str5 Str5 Str5 Str5 Str5 Str5 Str5 Str5  Str5
#>           G          Str6 Str6 Str6 Str6 Str6 Str6 Str6 Str6 Str6  Str6
#>           H
#>
#> block_name    Media
#>                    1    2    3    4    5    6    7    8    9   10     11 12
#>           A
#>           B          Med1 Med2 Med3 Med4 Med5      Med7 Med8 Med9 Med10
#>           C          Med1 Med2 Med3 Med4 Med5      Med7 Med8 Med9 Med10
#>           D          Med1 Med2 Med3 Med4 Med5      Med7 Med8 Med9 Med10
#>           E          Med1 Med2 Med3 Med4 Med5      Med7 Med8 Med9 Med10
#>           F          Med1 Med2 Med3 Med4 Med5      Med7 Med8 Med9 Med10
#>           G          Med1 Med2 Med3 Med4 Med5      Med7 Med8 Med9 Med10
#>           H
```

Here we can see all our design information has been saved to a single file, and the metadata has been added in rows before each block.

Merging spectrophotometric and design data

Once we have both our design and data in theRenvironment and tidy-shaped, we can merge them using `merge_dfs`.

For this, we'll use the data in the `example_widedata_noiseless` dataset that is included with `gcplyr`, and which was the source for our previous examples with `import_blockmeasures` and `read_wides`.

In the `example_widedata_noiseless` dataset, we have 48 different bacterial strains. The left side of the plate has all 48 strains in a single well each, and the right side of the plate also has all 48 strains in a single well each:

Row names	Column 1	…	Column 6	Column 7	…	Column 12
Row A	Strain #1	…	Strain #6	Strain #1	…	Strain #6
Row B	Strain #7	…	Strain #12	Strain #7	…	Strain #12
…	…	…	…	…	…	…
Row G	Strain #37	…	Strain #42	Strain #37	…	Strain #42
Row H	Strain #43	…	Strain #48	Strain #43	…	Strain #48

Then, on the right hand side of the plate a phage was also inoculated (while the left hand side remained bacteria-only):

Row names	Column 1	…	Column 6	Column 7	…	Column 12
Row A	No Phage	…	No Phage	Phage Added	…	Phage Added
Row B	No Phage	…	No Phage	Phage Added	…	Phage Added
…	…	…	…	…	…	…
Row G	No Phage	…	No Phage	Phage Added	…	Phage Added
Row H	No Phage	…	No Phage	Phage Added	…	Phage Added

Let's generate our design:

```r
example_design <- make_design(
  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"))
```

Here's what the resulting data.frame looks like:

```r
head(example_design, 20)
#>    Well Bacteria_strain       Phage
#> 1    A1        Strain 1    No Phage
#> 2    A2        Strain 2    No Phage
#> 3    A3        Strain 3    No Phage
#> 4    A4        Strain 4    No Phage
#> 5    A5        Strain 5    No Phage
#> 6    A6        Strain 6    No Phage
#> 7    A7        Strain 1 Phage Added
#> 8    A8        Strain 2 Phage Added
#> 9    A9        Strain 3 Phage Added
#> 10  A10        Strain 4 Phage Added
#> 11  A11        Strain 5 Phage Added
#> 12  A12        Strain 6 Phage Added
#> 13   B1        Strain 7    No Phage
#> 14   B2        Strain 8    No Phage
#> 15   B3        Strain 9    No Phage
#> 16   B4       Strain 10    No Phage
#> 17   B5       Strain 11    No Phage
#> 18   B6       Strain 12    No Phage
#> 19   B7        Strain 7 Phage Added
#> 20   B8        Strain 8 Phage Added
```

Now let's transform the `example_widedata_noiseless` to tidy-shaped.

```r
example_tidydata <- trans_wide_to_tidy(example_widedata_noiseless,
                                       id_cols = "Time")
```

And finally, we merge the two using `merge_dfs`, saving the result to `ex_dat_mrg`, short for example_data_merged. `merge_dfs` merges using columns with the same name between the two data.frames.

```
ex_dat_mrg <- merge_dfs(example_tidydata, example_design)
#> Joining with `by = join_by(Well)`

head(ex_dat_mrg)
#>   Time Well Measurements Bacteria_strain    Phage
#> 1    0   A1        0.002        Strain 1 No Phage
#> 2    0   B1        0.002        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        0.002       Strain 31 No Phage
```

What's next?

Now that you've merged your data and designs, you can pre-process and plot your data

1. Introduction: `vignette("gcplyr")`
2. Importing and transforming data: `vignette("import_transform")`
3. Incorporating design information: `vignette("incorporate_designs")`
4. **Pre-processing and plotting your data: `vignette("preprocess_plot")`**
5. Processing your data: `vignette("process")`
6. Analyzing your data: `vignette("analyze")`
7. Dealing with noise: `vignette("noise")`
8. Statistics, merging other data, and other resources: `vignette("conclusion")`