

# Data wrangling

dplyr



Roland Krause | rworkshop | 2021-09-09

# Introduction

## ✂ Data munging

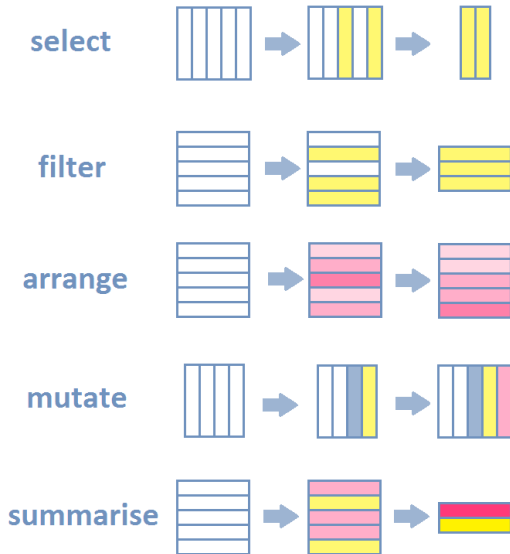
- Preparing data is the most time consuming part of data analysis.
- Individual steps might look *easy*.
- Essential part of *understanding* the data you're working with.
- Additional data preparation before modeling is impossible to avoid.

## 🧰 At a glance

**dplyr** is a tool box for working with data in *tibbles*, offering a unified language for operations scattered through base R.



## ☰ Key operations



source: [Lise Vaudor](#)

# This lecture

## Example data

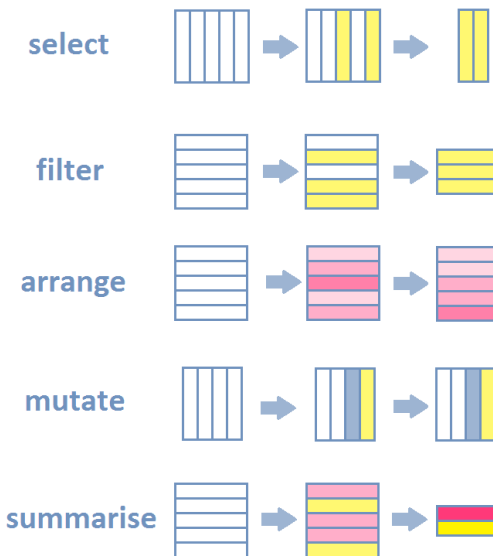
Van 't Veer, Anna; Sleegers, Willem, **2019**, "Psychology data from an exploration of the effect of anticipatory stress on disgust vs. non-disgust related moral judgments". *Journal of Open Psychology Data*.

- Data is (largely) *tidy*.
- Typical data you might see in the wild.

## Learning objectives

- Learn the **grammar** to operate on rows and columns of a table
- Selection and manipulation of
  - observations,
  - variables and
  - values.
- Grouping and summarizing
- Joining and intersecting tibbles
- Pivoting column headers and variables

## Key operations



source: [Lise Vaudor](#)

# dplyr Introduction: Cheat sheets

## Data transformation with dplyr : CHEAT SHEET



dplyr functions work with pipes and expect tidy data. In tidy data:



Each **variable** is in its own **column**



Each **observation**, or **case**, is in its own **row**



$x \%>\% f(y)$  becomes  $f(x, y)$

pipes

### Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function



**summarise**(data, ...) Compute table of summaries. summarise(mtcars, avg = mean(mpg))



**count**(data, ..., wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in ... Also **tally**(). count(mtcars, cyl)

### Group Cases

Use **group\_by**(data, ..., add = FALSE, drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.



mtcars %>%  
group\_by(cyl) %>%  
summarise(avg = mean(mpg))

Use **rowwise**(data, ...) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidy cheat sheet for list-column workflow.



starwars %>%  
rowwise() %>%  
mutate(film\_count = length(films))

**ungroup**(x, ...) Returns ungrouped copy of table. ungroup(g\_mtcars)

### Manipulate Cases

#### EXTRACT CASES

Row functions return a subset of rows as a new table.



**filter**(data, ..., preserve = FALSE) Extract rows that meet logical criteria. filter(mtcars, mpg > 20)



**distinct**(data, ..., keep\_all = FALSE) Remove rows with duplicate values. distinct(mtcars, gear)



**slice**(data, ..., preserve = FALSE) Select rows by position. slice(mtcars, 10:15)



**slice\_sample**(data, ..., n, prop, weight\_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows. slice\_sample(mtcars, n = 5, replace = TRUE)



**slice\_min**(data, order\_by, ..., n, prop, with\_ties = TRUE) and **slice\_max**() Select rows with the lowest and highest values. slice\_min(mtcars, mpg, prop = 0.25)



**slice\_head**(data, ..., n, prop) and **slice\_tail**() Select the first or last rows. slice\_head(mtcars, n = 5)

#### Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%		xor()
!=	>	>=	!is.na()	!	&	

See ?base::Logic and ?Comparison for help.

#### ARRANGE CASES



**arrange**(data, ..., by\_group = FALSE) Order rows by values of a column or columns (low to high), use with **desc**() to order from high to low. arrange(mtcars, mpg)

#### ADD CASES



**add\_row**(data, ..., before = NULL, after = NULL) Add one or more rows to a table. add\_row(cars, speed = 1, dist = 1)

### Manipulate Variables

#### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



**pull**(data, var = -1, name = NULL, ...) Extract column values as a vector, by name or index. pull(mtcars, wt)



**select**(data, ...) Extract columns as a table. select(mtcars, mpg, wt)



**relocate**(data, ..., before = NULL, after = NULL) Move columns to new position. relocate(mtcars, mpg, cyl, after = last\_col())

#### Use these helpers with select() and across()

e.g. select(mtcars, mpg:cyl)

<b>contains</b> (match)	<b>num_range</b> (prefix, range)	; e.g. mpg:cyl
<b>ends_with</b> (match)	<b>all_of</b> (x)/ <b>any_of</b> (x, ..., vars)	; e.g. gear
<b>starts_with</b> (match)	<b>matches</b> (match)	<b>everything</b> ()

#### MANIPULATE MULTIPLE VARIABLES AT ONCE



**across**(cols, funs, ..., names = NULL) Summarise or mutate multiple columns in the same way. summarise(mtcars, across(everything(), mean))



**c\_across**(cols) Compute across columns in row-wise data. transmute(rowwise(UKgas), total = sum(c\_across(1:2)))

#### MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function



**mutate**(data, ..., keep = "all", before = NULL, after = NULL) Compute new column(s). Also **add\_column**(), **add\_count**(), and **add\_tally**(). mutate(mtcars, gpm = 1 / mpg)



**transmute**(data, ...) Compute new column(s), drop others. transmute(mtcars, gpm = 1 / mpg)



**rename**(data, ...) Rename columns. Use **rename\_with**() to rename with a function. rename(cars, distance = dist)



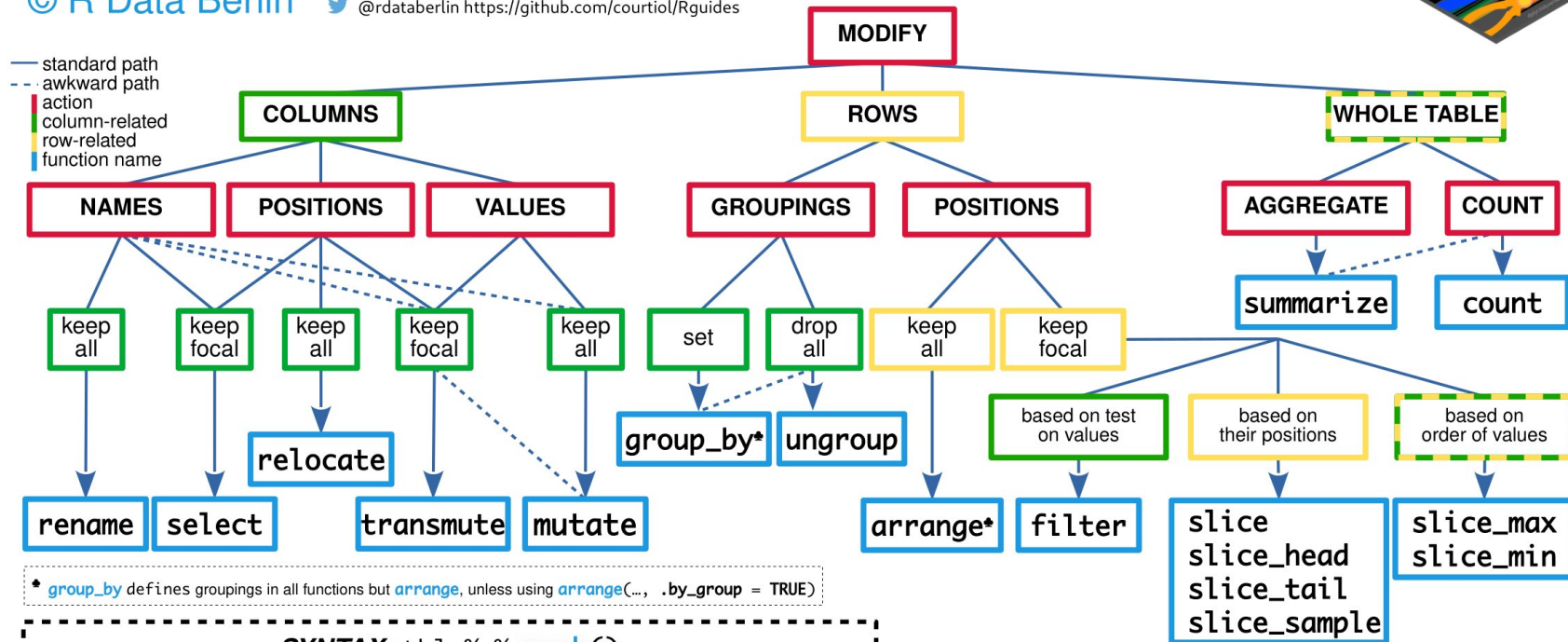
RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at [dplyr.tidyverse.org](https://dplyr.tidyverse.org) • dplyr 1.0.7 • Updated: 2021-07

# Data Transformation with **dplyr** 1.0 (part 1)

A guide to 17 modifications applied to one tibble (tbl) or data.frame

© R Data Berlin

[@rdatBerlin https://github.com/courtiol/Rguides](https://github.com/courtiol/Rguides)



## SYNTAX: tbl %>% verb()

[font style varies to tease apart placeholders from true R commands]

```
tbl %>% rename(new_name_col_X = old_name_col_X)
tbl %>% select(name_col_X, name_col_Y [+ std.op.*], selection_helper*)
tbl %>% relocate(same_as_for_select, .before[or .after] = name_col_Z/selection_helper*)
tbl %>% transmute(name_col_Z = fn*(name_col_X))
tbl %>% mutate(name_col_Z = fn*(name_col_X))
tbl %>% group_by(name_col_X, name_col_Y) %>% verb() %>% ungroup()
tbl %>% arrange(name_col_X, desc(name_col_Y))
tbl %>% filter(fn_test_vectorized(name_col_X), fn_test_vectorized(name_col_Y))
tbl %>% slice(row_indices); tbl %>% slice_head/tail/sample(number_rows_to_keep);
tbl %>% slice_min/max(name_col_X, n = nb_rows_to_keep [or prop = proportion_rows_to_keep])
tbl %>% summarize(name_col_Z = fn*(name_col_X))
tbl %>% count(); tbl %>% count(name_col_X)
```

\* standard operators may be used to combine (c(), &, |) or negate elements (!)

\* selection helpers from pkg tidyselect may be used to select columns based on:

- column values → where(fn), e.g. fn = is.numeric
- column names → starts\_with("text"), ends\_with("text"), contains("text"), matches("regex"), num\_range("text", min:max), all\_of(vector\_of\_text), any\_of(vector\_of\_text)
- column positions → everything(), last\_col() → See guide part 2 for more details

▼ if the function fn does not return a scalar or a row-long output, use list(fn()) to create a list column (i.e. for nesting the content); for creating multiple columns at once fn should return a data.frame or tibble and no name should be defined when calling the dplyr verb (i.e. name\_col\_Z =; if a name is defined, the output will be nested); to unnest to content of a list column, use one of the functions provided by the pkg tidyr (e.g. unnest\_wider)

# Using the dplyr package

## Do not use these packages!

**dplyr** supersedes previous packages from Hadley Wickham.

- **reshape**
- **reshape2**
- **plyr**

All functionality can be found in **dplyr** or **tidyr**.

## dplyr current version is 1.0.7



- **dplyr** has seen many changes.
- Watch out for deprecated examples on Stack overflow!
- Code will break sooner or later (you might get lucky)
- But handled with clear [lifecycle stages](#) now
- Changes are generally introduced to simplify operations.

## Loading **dplyr**

```
library(dplyr) # AND  
library(tidyr)  
  
# OR (recommended  
# for the workshop)  
  
library(tidyverse)
```

# Preparing the data

## Description

Van 't Veer, Anna; Sleegers, Willem. (2019) "[Psychology data from an exploration of the effect of anticipatory stress on disgust vs. non-disgust related moral judgments](#)". *Journal of Open Psychology Data*.  

- Moral dilemmata (trolley problem, survival after plane crash, etc. [moral](#))
- Standard questionnaires
  - Private Body Consciousness ([PBC](#), range 0 - 4)
  - Rational-Experiential Inventory ([REI](#), range 1 - 5)
  - Multidimensional Assessment of Interoceptive Awareness ([MAIA](#), range 0 - 5)
  - State Trait Anxiety Inventory ([STAI](#))

```
judge_url <- "https://biostat2.uni.lu/practicals/data/judgments.tsv"
judgments <- readr::read_tsv(judge_url)
```

Rows: 188 Columns: 158

— Column specification —

Delimiter: "\t"

chr (5): start\_date, end\_date, condition, gender, logbook

dbl (153): finished, subject, age, mood\_pre, mood\_post, STAI\_pre\_1\_1, S

- Use ``spec()`` to retrieve the full column specification for this data.
- Specify the column types or set ``show_col_types = FALSE`` to quiet thi

```
# https://dataverse.nl/api/access/datafile/11863
#"https://biostat2.uni.lu/practicals/data/judgments.tsv")
```

# Your turn

Load the data into your RStudio session.

```
judgments <- readr::read_tsv("https://biostat2.uni.lu/practicals/data/judgments.tsv")
```

Rows: 188 Columns: 158

— Column specification —

Delimiter: "\t"

chr (5): start\_date, end\_date, condition, gender, logbook

dbl (153): finished, subject, age, mood\_pre, mood\_post, STAI\_pre\_1\_1, STAI\_p...

- Use ``spec()`` to retrieve the full column specification for this data.
- Specify the column types or set ``show_col_types = FALSE`` to quiet this message.

02:00

# Pipes to simplify multiple function calls

## Speaking code

```
verb(subject, complement)
```

```
subject %>% verb(complement)
```

Example adapted from [Romain François](#)

**tidyverse** functions are consistently designed to work with **%>%**

**data** (*subject*) is the **first** argument.

## Current developments

- Many languages are adding pipes.
- Several packages exist in R.
- R 4.1 introduced a base pipe `|>`.
  - Works mostly like the magrittr/tidyverse **%>%** but is not identical.
- Will use **%>%** only in the course.

[Luke Tierney useR!2020](#)

# Best practices for using the pipe

## Reminder

First arguments for all **tidyverse** functions is the **.data**: A tibble / data.frame.

But for single operations -- function calls -- it is recommended **NOT** to use **%>%**.

## Warning

```
judgments %>%  
  mutate(judgments, # Wrong!  
         new_col = mean(age))
```

Don't specify **data** twice!

## Better

```
judgments %>%  
  mutate(new_col =  
         mean(age))
```

Harder to read for simple operations.

## Recommended

```
mutate(judgments,  
       new_col = mean(age))
```

For single operations only.

**Inspecting tibbles**

# Inspect tibbles with `glimpse()`

## Column-wise description

Shows some values and the type of **each** column.

The *Environment* tab in RStudio tab does it too.

Clicking object `judgments` triggers `View()`.

Similar to the `base::str()` function

```
glimpse(judgments)
```

```
Rows: 188
Columns: 158
$ start_date      <chr> "11/3/2014", "11/3/2014", "11/3/2014", "11/...
$ end_date        <chr> "11/3/2014", "11/3/2014", "11/3/2014", "11/...
$ finished        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ condition       <chr> "control", "stress", "stress", "stress", "c...
$ subject         <dbl> 2, 1, 3, 4, 7, 6, 5, 9, 16, 13, 18, 14, 12,...
$ gender          <chr> "female", "female", "female", "female", "fe...
$ age             <dbl> 24, 19, 19, 22, 22, 22, 18, 20, 21, 19, 19,...
$ mood_pre        <dbl> 81, 59, 22, 53, 48, 73, NA, 100, 67, 30, 55...
$ mood_post       <dbl> NA, 42, 60, 68, NA, 73, NA, NA, 74, 68, 57,...
$ STAI_pre_1_1    <dbl> 2, 3, 4, 2, 1, 2, 2, 1, 2, 4, 2, 1, 2, 1, 1...
$ STAI_pre_1_2    <dbl> 1, 2, 3, 2, 1, 2, 2, 1, 2, 2, 3, 2, 2, 1, 1...
$ STAI_pre_1_3    <dbl> 2, 3, 3, 2, 1, 1, 1, 1, 1, 3, 1, 2, 2, 2, 2...
$ STAI_pre_1_4    <dbl> 2, 1, 3, 2, 1, 1, 1, 1, 1, 3, 1, 2, 1, 1, 1...
$ STAI_pre_1_5    <dbl> 2, 3, 4, 3, 2, 2, 2, 1, 2, 3, 2, 2, 2, 2, 2...
$ STAI_pre_1_6    <dbl> 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1...
$ STAI_pre_1_7    <dbl> 2, 3, 3, 1, 1, 2, 1, 1, 1, 3, 1, 1, 2, 1, 3...
$ STAI_pre_2_1    <dbl> 2, 3, 4, 3, 3, 2, 2, 2, 2, 4, 3, 3, 2, 4, 3...
$ STAI_pre_2_2    <dbl> 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1...
$ STAI_pre_2_3    <dbl> 1, 2, 3, 3, 3, 2, 2, 1, 2, 3, 2, 3, 3, 3, 2...
$ STAI_pre_2_4    <dbl> 1, 2, 4, 3, 3, 2, 2, 1, 2, 4, 3, 3, 3, 3, 2...
$ STAI_pre_2_5    <dbl> 1, 2, 4, 1, 1, 2, 1, 1, 1, 3, 1, 2, 1, 2, 1...
$ STAI_pre_2_6    <dbl> 1, 3, 4, 1, 1, 2, 1, 1, 1, 3, 1, 1, 1, 2, 2...
$ STAI_pre_2_7    <dbl> 1, 1, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2, 3, 1...
$ STAI_pre_3_1    <dbl> 2, 3, 4, 3, 1, 2, 2, 1, 2, 4, 2, 2, 3, 2, 3...
$ STAI_pre_3_2    <dbl> 2, 3, 3, 3, 2, 2, 2, 1, 2, 3, 2, 2, 2, 3, 2...
$ STAI_pre_3_3    <dbl> 2, 3, 2, 2, 2, 2, 1, 1, 1, 3, 1, 1, 2, 1, 2...
$ STAI_pre_3_4    <dbl> 1, 2, 3, 1, 1, 1, 2, 1, 2, 3, 1, 1, 1, 1, 1...
$ STAI_pre_3_5    <dbl> 2, 3, 4, 3, 3, 2, 2, 1, 2, 4, 2, 2, 3, 2, 3...
```

# Selecting columns

## `select()`

- Select the columns you want: `select(tibble, your_column1, ...)`

```
select(judgments, gender, age, condition)
```

```
# A tibble: 188 × 3
  gender age condition
  <chr> <dbl> <chr>
1 female 24 control
2 female 19 stress
3 female 19 stress
4 female 22 stress
5 female 22 control
6 female 22 stress
7 female 18 control
8 male 20 control
9 female 21 stress
10 female 19 stress
# ... with 178 more rows
```

## Warning

The `biomaRt` package of Bioconductor (amongst others) provides a `select()` function. If loaded, we need to address the `dplyr`-package using `::`!

```
judgments %>%
  dplyr::select(age, gender)
```

```
# A tibble: 188 × 2
  age gender
  <dbl> <chr>
1 24 female
2 19 female
3 19 female
4 22 female
5 22 female
6 22 female
7 18 female
8 20 male
9 21 female
10 19 female
# ... with 178 more rows
```

# Reminder tidyselect

## Helper function

To select columns with names that:

- `contains()` - a string
- `starts_with()` - a string
- `ends_with()` - a string
- `one_of()` - names in a character **vector**
- `matches()` - using regular expressions
- `everything()` - all **remaining** columns
- `last_col()` - last column

```
select(judgments, starts_with("moral"))
```

```
# A tibble: 188 × 10
  moral_dilemma_dog moral_dilemma_wallet moral_dilemma_plane moral_dilemma_kit
  <dbl>          <dbl>          <dbl>          <dbl>
1           9           9           8
2           9           9           9
3           8           7           8
4           8           4           8
5           3           9           9
6           9           9           9
7           9           5           7
8           9           4           1
9           6           9           3
10          6           8           9
# ... with 178 more rows, and 6 more variables: moral_dilemma_kit
# moral_dilemma_trolley <dbl>, moral_dilemma_control <dbl>,
# moral_judgment <dbl>, moral_judgment_disgust <dbl>,
# moral_judgment_non_disgust <dbl>
```

# Combining helpers

## Remark

Helpers are found in several functions, e.g. `across()`.

```
select(judgments, ends_with("date"), contains("dilemma"))
```

```
# A tibble: 188 × 9
  start_date end_date moral_dilemma_dog moral_dilemma_wallet moral_dilemma_pl...
  <chr>      <chr>          <dbl>                <dbl>                <dbl>
1 11/3/2014 11/3/2014             9                    9                    8
2 11/3/2014 11/3/2014             9                    9                    9
3 11/3/2014 11/3/2014             8                    7                    8
4 11/3/2014 11/3/2014             8                    4                    8
5 11/3/2014 11/3/2014             3                    9                    9
6 11/3/2014 11/3/2014             9                    9                    9
7 11/3/2014 11/3/2014             9                    5                    7
8 11/3/2014 11/3/2014             9                    4                    1
9 11/3/2014 11/3/2014             6                    9                    3
10 11/3/2014 11/3/2014             6                    8                    9
# ... with 178 more rows, and 4 more variables: moral_dilemma_resume <dbl>,
# moral_dilemma_kitten <dbl>, moral_dilemma_trolley <dbl>,
# moral_dilemma_control <dbl>
```

# Selecting columns using `select()`

## ❏ Negative selection

Drop columns by **negating** their names `-`.

Works with the **tidyselect** helper functions.

```
select(judgments,  
  -gender,  
  -starts_with(c("STAI", "REI")),  
  -ends_with("id"))
```

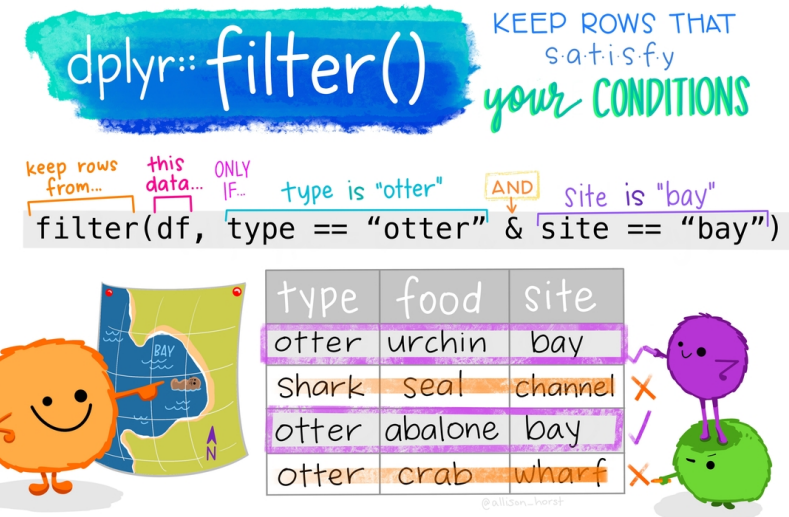
```
# A tibble: 188 × 71  
  start_date end_date finished condition subject age mood_pre mood_post  
  <chr>      <chr>      <dbl> <chr>      <dbl> <dbl> <dbl> <dbl>  
1 11/3/2014 11/3/2014      1 control      2 24 81 NA  
2 11/3/2014 11/3/2014      1 stress      1 19 59 42  
3 11/3/2014 11/3/2014      1 stress      3 19 22 60  
4 11/3/2014 11/3/2014      1 stress      4 22 53 68  
5 11/3/2014 11/3/2014      1 control      7 22 48 NA  
6 11/3/2014 11/3/2014      1 stress      6 22 73 73  
7 11/3/2014 11/3/2014      1 control      5 18 NA NA  
8 11/3/2014 11/3/2014      1 control      9 20 100 NA  
9 11/3/2014 11/3/2014      1 stress     16 21 67 74  
10 11/3/2014 11/3/2014      1 stress     13 19 30 68  
# ... with 178 more rows, and 63 more variables: moral_dilemma_dog <dbl>,  
# moral_dilemma_wallet <dbl>, moral_dilemma_plane <dbl>,  
# moral_dilemma_resume <dbl>, moral_dilemma_kitten <dbl>,  
# moral_dilemma_trolley <dbl>, moral_dilemma_control <dbl>,  
# presentation_experience <dbl>, presentation_unpleasant <dbl>,  
# presentation_fun <dbl>, presentation_challenge <dbl>, PBC_1 <dbl>,  
# PBC_2 <dbl>, PBC_3 <dbl>, PBC_4 <dbl>, PBC_5 <dbl>, MAIA_1_1 <dbl>, ...
```

# Filtering for rows: `filter()`

Let's take a look at all the data that was excluded. `exclude` is coded as `0 = include` and `1 = exclude`.

```
filter(judgments, exclude == 1)
```

```
# A tibble: 3 × 158
  start_date end_date finished condition subject gender age
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr> <dbl>
1 11/3/2014  11/3/2014      1 stress      28 male    22
2 11/3/2014  11/3/2014      1 stress      32 female  19
3 11/7/2014  11/7/2014      1 stress     181 male    22
# ... with 149 more variables: STAI_pre_1_1 <dbl>, STAI_pre_1_2 <dbl>,
# STAI_pre_1_3 <dbl>, STAI_pre_1_4 <dbl>, STAI_pre_1_5 <dbl>,
# STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>, STAI_pre_2_1 <dbl>,
# STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>,
# STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>,
# STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>,
# STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>, STAI_pre_3_6 <dbl>,
```



Artwork by [@allison\\_horst](#)

# Filtering rows

## Multiple conditions: AND

- **comma** separated conditions are equivalent to **&** (AND)
- Cases that are included and have a high mood before the experiment.

```
filter(judgments,
  exclude == 0,
  mood_pre > 85)
```

```
# A tibble: 8 × 158
  start_date end_date finished condition subject gender age
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr>  <dbl>
1 11/3/2014 11/3/2014         1 control         9 male    20
2 11/3/2014 11/3/2014         1 control        19 male    23
3 11/3/2014 11/3/2014         1 control        39 male    20
4 11/4/2014 11/4/2014         1 control        74 female  23
5 11/5/2014 11/5/2014         1 control       105 female  21
6 11/6/2014 11/6/2014         1 stress       142 female  22
7 11/7/2014 11/7/2014         1 control       167 male   18
8 11/7/2014 11/7/2014         1 control       168 female  21
# ... with 149 more variables: STAI_pre_1_1 <dbl>, STAI_pre_1_2 <dbl>,
# STAI_pre_1_3 <dbl>, STAI_pre_1_4 <dbl>, STAI_pre_1_5 <dbl>,
# STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>, STAI_pre_2_1 <dbl>,
# STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>,
# STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>,
# STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>,
# STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>, STAI_pre_3_6 <dbl>,
```

## Multiple conditions: OR

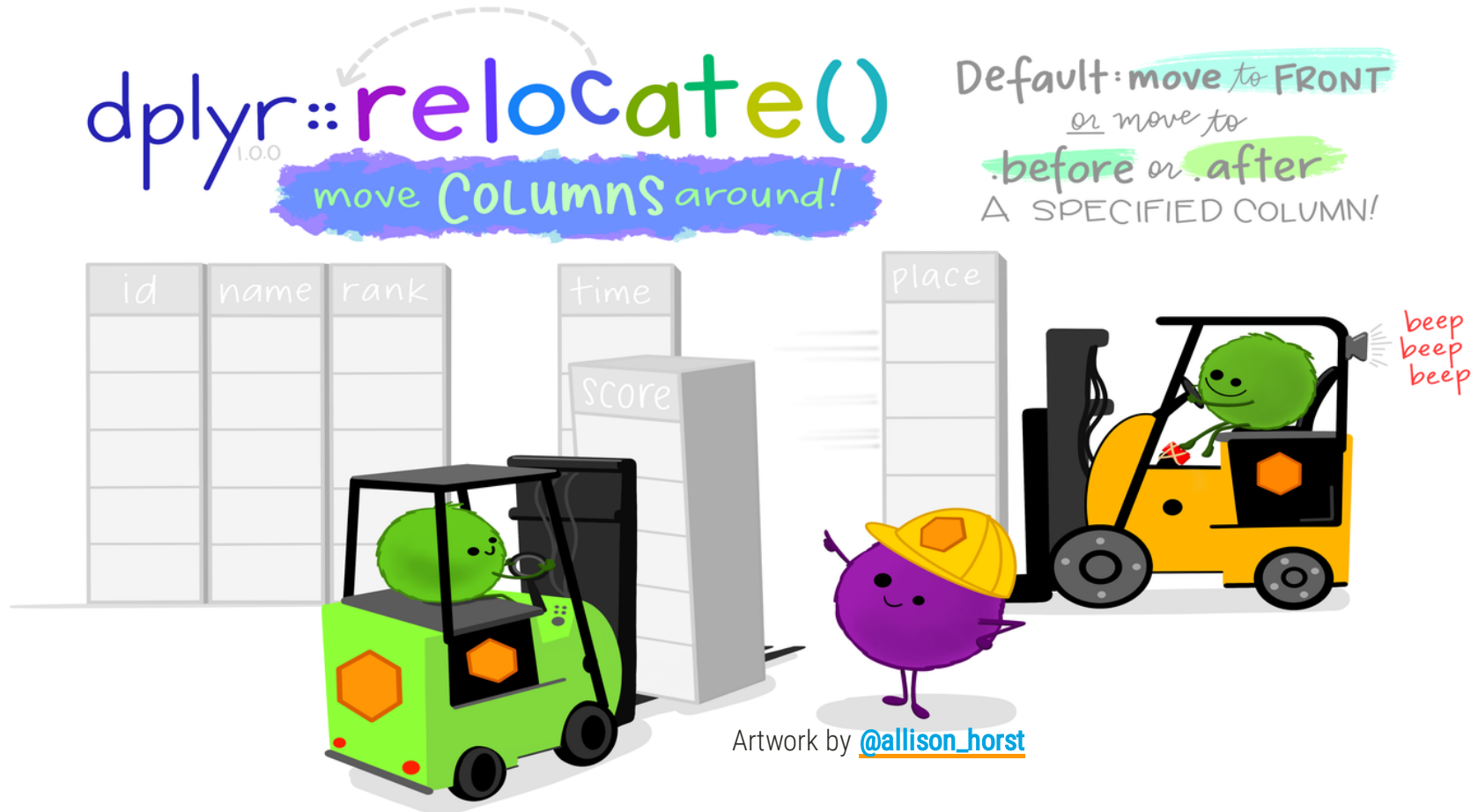
- **pipe (|)** separated conditions are combined with OR
- Filter participants that have a low mood before or after the experiment.

```
filter(judgments,
  mood_pre < 20 |
  mood_post < 20 )
```

```
# A tibble: 8 × 158
  start_date end_date finished condition subject gender age
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr>  <dbl>
1 11/3/2014 11/3/2014         1 stress         24 female  20
2 11/3/2014 11/3/2014         1 stress         22 female  18
3 11/3/2014 11/3/2014         1 control        40 male   22
4 11/5/2014 11/5/2014         1 stress         81 female  17
5 11/6/2014 11/6/2014         1 stress       127 female  18
6 11/6/2014 11/6/2014         1 stress       138 male   21
7 11/6/2014 11/6/2014         1 stress       159 female  18
8 11/7/2014 11/7/2014         1 stress       161 female  19
# ... with 149 more variables: STAI_pre_1_1 <dbl>, STAI_pre_1_2 <dbl>,
# STAI_pre_1_3 <dbl>, STAI_pre_1_4 <dbl>, STAI_pre_1_5 <dbl>,
# STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>, STAI_pre_2_1 <dbl>,
# STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>,
# STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>,
# STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>,
# STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>, STAI_pre_3_6 <dbl>,
```

# relocate()

- Only present since `dplyr` version **1.0**
- `.before` and `.after` for fine placement.



# Filtering out rows

## Row vs column selection

- `filter()` acts on rows
- `select()` acts on columns

- Remove excluded participants
- Combine with `relocate()` to place `mood` column first

```
filter(judgments, exclude == 0) %>%  
  relocate(contains("mood"))
```

```
# A tibble: 185 × 158  
  mood_pre mood_post start_date end_date finished condition subject gender  
    <dbl>    <dbl> <chr>    <chr>      <dbl> <chr>      <dbl> <chr>  
1      81      NA 11/3/2014 11/3/2014      1 control      2 female  
2      59      42 11/3/2014 11/3/2014      1 stress      1 female  
3      22      60 11/3/2014 11/3/2014      1 stress      3 female  
4      53      68 11/3/2014 11/3/2014      1 stress      4 female  
5      48      NA 11/3/2014 11/3/2014      1 control      7 female  
6      73      73 11/3/2014 11/3/2014      1 stress      6 female  
7      NA      NA 11/3/2014 11/3/2014      1 control      5 female  
8     100      NA 11/3/2014 11/3/2014      1 control      9 male  
9      67      74 11/3/2014 11/3/2014      1 stress     16 female  
10     30      68 11/3/2014 11/3/2014      1 stress     13 female  
# ... with 175 more rows, and 150 more variables: age <dbl>, STAI_pre_1_1 <dbl>,  
# STAI_pre_1_2 <dbl>, STAI_pre_1_3 <dbl>, STAI_pre_1_4 <dbl>,  
# STAI_pre_1_5 <dbl>, STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>,  
# STAI_pre_2_1 <dbl>, STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>,  
# STAI_pre_2_4 <dbl>, STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>,  
# STAI_pre_2_7 <dbl>, STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>,  
# STAI_pre_3_3 <dbl>, STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>, ...
```

# Filter rows, helper between()

```
judgments %>%  
  filter(between(mood_pre, 40, 60)) %>%  
  select(age, gender, condition, mood_pre, mood_post)
```

```
# A tibble: 64 × 5  
   age gender condition mood_pre mood_post  
  <dbl> <chr>   <chr>      <dbl>   <dbl>  
1    19 female stress        59       42  
2    22 female stress        53       68  
3    22 female control       48      NA  
4    19 female stress        55       57  
5    18 female stress        53       38  
6    19 female control       59      NA  
7    19 female stress        60       53  
8    22 male   stress        53       68  
9    18 male   control       50      NA  
10   19 female stress        58       45  
# ... with 54 more rows
```

# Set operations with `filter()`

- The two below are equivalent
- For larger operations use `inner_join()`
- Below, `tidyselect` helper `:` for a range of columns

```
judgments %>%  
  filter(is.element(start_date, c("11/3/2014", "11/5/2014")))
```

```
# A tibble: 79 × 158  
  start_date end_date finished condition subject gender age  
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr> <dbl>  
1 11/3/2014 11/3/2014         1 control         2 female 24  
2 11/3/2014 11/3/2014         1 stress         1 female 19  
3 11/3/2014 11/3/2014         1 stress         3 female 19  
4 11/3/2014 11/3/2014         1 stress         4 female 22  
5 11/3/2014 11/3/2014         1 control         7 female 22  
6 11/3/2014 11/3/2014         1 stress         6 female 22  
7 11/3/2014 11/3/2014         1 control         5 female 18  
8 11/3/2014 11/3/2014         1 control         9 male 20  
9 11/3/2014 11/3/2014         1 stress        16 female 21  
10 11/3/2014 11/3/2014         1 stress        13 female 19  
# ... with 69 more rows, and 150 more variables: mood_post <dbl>,  
# STAI_pre_1_1 <dbl>, STAI_pre_1_2 <dbl>, STAI_pre_1_3 <dbl>,  
# STAI_pre_1_4 <dbl>, STAI_pre_1_5 <dbl>, STAI_pre_1_6 <dbl>,  
# STAI_pre_1_7 <dbl>, STAI_pre_2_1 <dbl>, STAI_pre_2_2 <dbl>,  
# STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>, STAI_pre_2_5 <dbl>,  
# STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>, STAI_pre_3_1 <dbl>,  
# STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>, STAI_pre_3_4 <dbl>,
```

```
judgments %>%  
  filter(start_date %in% c("11/3/2014", "11/5/2014")) %>%  
  select(start_date:age)
```

```
# A tibble: 79 × 7  
  start_date end_date finished condition subject gender age  
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr> <dbl>  
1 11/3/2014 11/3/2014         1 control         2 female 24  
2 11/3/2014 11/3/2014         1 stress         1 female 19  
3 11/3/2014 11/3/2014         1 stress         3 female 19  
4 11/3/2014 11/3/2014         1 stress         4 female 22  
5 11/3/2014 11/3/2014         1 control         7 female 22  
6 11/3/2014 11/3/2014         1 stress         6 female 22  
7 11/3/2014 11/3/2014         1 control         5 female 18  
8 11/3/2014 11/3/2014         1 control         9 male 20  
9 11/3/2014 11/3/2014         1 stress        16 female 21  
10 11/3/2014 11/3/2014         1 stress        13 female 19  
# ... with 69 more rows
```

# Filter out rows that are unique: `distinct()`

Suppose we select the dates and control status (`condition`) for a particular start date. Do we have different end dates?

```
judgments %>%  
  select(start_date, end_date, condition)
```

```
# A tibble: 188 × 3  
  start_date end_date condition  
  <chr>      <chr>      <chr>  
1 11/3/2014 11/3/2014 control  
2 11/3/2014 11/3/2014 stress  
3 11/3/2014 11/3/2014 stress  
4 11/3/2014 11/3/2014 stress  
5 11/3/2014 11/3/2014 control  
6 11/3/2014 11/3/2014 stress  
7 11/3/2014 11/3/2014 control  
8 11/3/2014 11/3/2014 control  
9 11/3/2014 11/3/2014 stress  
10 11/3/2014 11/3/2014 stress  
# ... with 178 more rows
```

- Multiple identical rows.
- Use `distinct()` to remove duplicated rows:

```
judgments %>%  
  filter(exclude == 0) %>%  
  select(start_date, end_date, condition) %>%  
  distinct()
```

```
# A tibble: 10 × 3  
  start_date end_date condition  
  <chr>      <chr>      <chr>  
1 11/3/2014 11/3/2014 control  
2 11/3/2014 11/3/2014 stress  
3 11/4/2014 11/4/2014 stress  
4 11/4/2014 11/4/2014 control  
5 11/5/2014 11/5/2014 stress  
6 11/5/2014 11/5/2014 control  
7 11/6/2014 11/6/2014 control  
8 11/6/2014 11/6/2014 stress  
9 11/7/2014 11/7/2014 stress  
10 11/7/2014 11/7/2014 control
```

- Also possible (except columns order):

```
judgments %>%  
  filter(exclude == 0) %>%  
  distinct(start_date, end_date, condition)
```

# Sort columns: `arrange()`

## A nested sorting example

1. sort by `mood_pre`
2. within each group of `mood_pre`, sort by `mood_post`

```
judgments %>%  
  arrange(mood_pre, mood_post) %>%  
  select(subject, mood_pre, mood_post)
```

```
# A tibble: 188 × 3  
  subject mood_pre mood_post  
    <dbl>   <dbl>   <dbl>  
1     159         9         0  
2     161        11        31  
3      22        13        37  
4     127        15        20  
5     138        18        44  
6      40        19        NA  
7      99        20        28  
8     150        20        62  
9      84        21        48  
10       3        22        60  
# ... with 178 more rows
```

## Reverse sort columns

- Use `arrange()` with the helper function `desc()`
- For example, *oldest* participant first

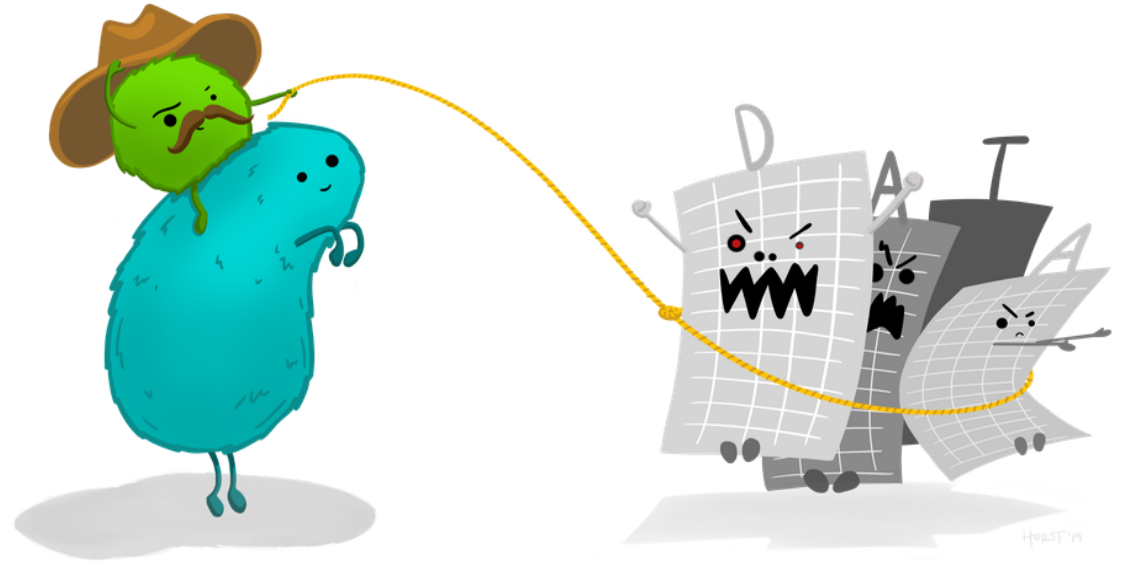
```
arrange(judgments, desc(age)) %>%  
  select(subject, age, condition)
```

```
# A tibble: 188 × 3  
  subject    age condition  
    <dbl> <dbl> <chr>  
1     107     31 stress  
2      61     27 stress  
3      41     26 control  
4     183     25 stress  
5       2     24 control  
6      93     24 stress  
7     115     24 stress  
8     137     24 control  
9      23     23 control  
10     19     23 control  
# ... with 178 more rows
```

# Verbs to inspect data

## Summary

- `glimpse()` to get an overview of each column's content
- `select()` to pick and/or omit columns
  - helper functions
- `relocate()` re-arrange columns order
- `filter()` to subset
  - AND/OR conditions (`, , |`)
- `arrange()` to sort
  - combine with `desc()` to reverse the sorting



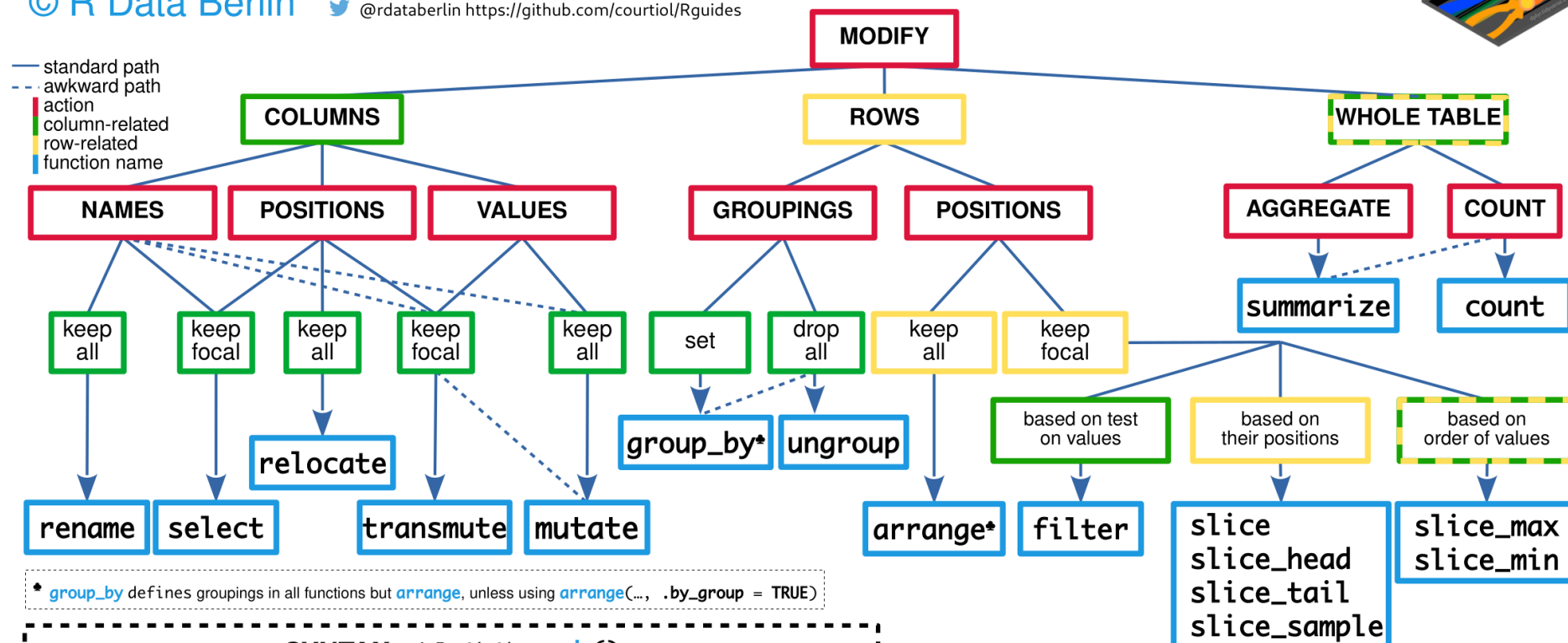
Artwork by [@allison\\_horst](https://twitter.com/allison_horst)

# Data Transformation with dplyr 1.0 (part 1)

A guide to 17 modifications applied to one tibble (tbl) or data.frame

© R Data Berlin

[@rdatabelin https://github.com/courtiol/Rguides](https://github.com/courtiol/Rguides)



## SYNTAX: `tbl %>% verb()`

font style varies to tease apart placeholders from true R commands

```

tbl %>% rename(new_name_col_X = old_name_col_X)
tbl %>% select(name_col_X, name_col_Y [+ std_op.*], selection_helper*)
tbl %>% relocate( , .before[or .after] = name_col_Z/selection_helper*)
tbl %>% transmute(name_col_Z = fn(name_col_X))
tbl %>% mutate(name_col_Z = fn(name_col_X), [ , .keep = "all"/"used"/"unused"/"none"])
tbl %>% group_by(name_col_X, name_col_Y) %>% verb() %>% ungroup()
tbl %>% arrange(name_col_X, desc(name_col_Y))
tbl %>% filter(fn_test_vectorized(name_col_X), fn_test_vectorized(name_col_Y))
tbl %>% slice(row_indices); tbl %>% slice_head/tail/sample(number_rows_to_keep);
tbl %>% slice_min/max(name_col_X, n = nb_rows_to_keep [or prop = proportion_rows_to_keep])
tbl %>% summarize(name_col_Z = fn(name_col_X))
tbl %>% count(); tbl %>% count(name_col_X)
  
```

\* standard operators may be used to combine (`c()`), `&`, `!` or negate elements (`!`)

\* selection helpers from pkg `tidyselect` may be used to select columns based on:

- column values → `where(fn)`, e.g. `fn = is.numeric`
- column names → `starts_with("text")`, `ends_with("text")`, `contains("text")`, `matches("regex")`, `num_range("text", min:max)`, `all_of(vector_of_text)`, `any_of(vector_of_text)`
- column positions → `everything()`, `last_col()` → See guide part 2 for more details

▼ if the function `fn` does not return a scalar or a row-long output, use `list(fn())` to create a list column (i.e. for nesting the content); for creating multiple columns at once `fn` should return a data.frame or tibble and no name should be defined when calling the **dplyr verb** (i.e. `name_col_Z =`; if a name is defined, the output will be nested); to unnest to content of a list column, use one of the functions provided by the pkg `tidyr` (e.g. `unnest_wider`)

# Transforming columns

# Changing column names

`rename()`

`rename(tib, new_name = old_name).`

*to remember the order of appearance, consider = as "was".*

```
rename(judgments,
       sex = gender,
       remove = exclude)
```

```
# A tibble: 188 × 158
  start_date end_date finished condition subject sex age
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr> <dbl>
1 11/3/2014 11/3/2014      1 control      2 female 24
2 11/3/2014 11/3/2014      1 stress      1 female 19
3 11/3/2014 11/3/2014      1 stress      3 female 19
4 11/3/2014 11/3/2014      1 stress      4 female 22
5 11/3/2014 11/3/2014      1 control      7 female 22
6 11/3/2014 11/3/2014      1 stress      6 female 22
7 11/3/2014 11/3/2014      1 control      5 female 18
8 11/3/2014 11/3/2014      1 control      9 male 20
9 11/3/2014 11/3/2014      1 stress     16 female 21
10 11/3/2014 11/3/2014      1 stress     13 female 19
# ... with 178 more rows, and 149 more variables: STAI_pre_1_1 <dbl>,
# STAI_pre_1_2 <dbl>, STAI_pre_1_3 <dbl>, STAI_pre_1_4 <dbl>,
# STAI_pre_1_5 <dbl>, STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>,
# STAI_pre_2_1 <dbl>, STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>,
# STAI_pre_2_4 <dbl>, STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>,
# STAI_pre_2_7 <dbl>, STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>,
# STAI_pre_3_3 <dbl>, STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>,
```

With a function: `rename_with()`

Uppercase certain columns using `stringr` and `tidyselect`

```
rename_with(judgments,
            stringr::str_to_lower,
            starts_with("STAI")) %>%
  select(contains("pre"))
```

```
# A tibble: 188 × 27
  mood_pre stai_pre_1_1 stai_pre_1_2 stai_pre_1_3 stai_pre_1_4
  <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
1      81          2          1          2          2
2      59          3          2          3          1
3      22          4          3          3          3
4      53          2          2          2          2
5      48          1          1          1          1
6      73          2          2          1          1
7      NA          2          2          1          1
8     100          1          1          1          1
9      67          2          2          1          1
10     30          4          2          3          3
# ... with 178 more rows, and 21 more variables: stai_pre_1_6 <dbl>,
# stai_pre_1_7 <dbl>, stai_pre_2_1 <dbl>, stai_pre_2_2 <dbl>,
# stai_pre_2_3 <dbl>, stai_pre_2_4 <dbl>, stai_pre_2_5 <dbl>,
# stai_pre_2_6 <dbl>, stai_pre_2_7 <dbl>, stai_pre_3_1 <dbl>,
# stai_pre_3_2 <dbl>, stai_pre_3_3 <dbl>, stai_pre_3_4 <dbl>,
# stai_pre_3_5 <dbl>, stai_pre_3_6 <dbl>, presentation_experi
# presentation_unpleasant <dbl>, presentation_fun <dbl>, ...
```

# Adding columns: mutate()

Let's create a new column `mood_change` that describes the change of the mood of the participant across the experiment.

- New column name: `mood_change`
- Computation: subtract `mood_pre` from `mood_post`

```
judgments %>%  
  mutate(mood_change = mood_post - mood_pre) %>%  
  relocate(starts_with("mood"))
```

```
# A tibble: 188 × 159  
  mood_pre mood_post mood_change start_date end_date finished  
    <dbl>    <dbl>    <dbl> <chr>    <chr>    <dbl>  
1      81      NA      NA 11/3/2014 11/3/20...      1  
2      59      42     -17 11/3/2014 11/3/20...      1  
3      22      60      38 11/3/2014 11/3/20...      1  
4      53      68      15 11/3/2014 11/3/20...      1  
5      48      NA      NA 11/3/2014 11/3/20...      1  
6      73      73       0 11/3/2014 11/3/20...      1  
7      NA      NA      NA 11/3/2014 11/3/20...      1  
8     100      NA      NA 11/3/2014 11/3/20...      1  
9      67      74       7 11/3/2014 11/3/20...      1  
10     30      68      38 11/3/2014 11/3/20...      1  
# ... with 178 more rows, and 151 more variables: gender <chr>, a  
# STAI_pre_1_1 <dbl>, STAI_pre_1_2 <dbl>, STAI_pre_1_3 <dbl>,  
# STAI_pre_1_4 <dbl>, STAI_pre_1_5 <dbl>, STAI_pre_1_6 <dbl>,  
# STAI_pre_1_7 <dbl>, STAI_pre_2_1 <dbl>, STAI_pre_2_2 <dbl>,  
# STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>, STAI_pre_2_5 <dbl>,  
# STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>, STAI_pre_3_1 <dbl>,  
# STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>, STAI_pre_3_4 <dbl>,
```



Artwork by [@allison\\_horst](https://twitter.com/allison_horst)

# Within one mutate statement

## Instant availability

Use new variables in the same function call right away!

```
judgments %>%  
  mutate(  
    mood_change = mood_post - mood_pre,  
    mood_change_norm =  
      abs(mood_change / mean(mood_change, na.rm = TRUE))) %>%  
    relocate(starts_with("mood")) %>%  
    arrange(desc(mood_change_norm))
```

```
# A tibble: 188 × 160  
  mood_pre mood_post mood_change mood_change_norm start_date end_date finished  
    <dbl>    <dbl>    <dbl>         <dbl> <chr>      <chr>      <dbl>  
1      66         0      -66          9.12 11/3/2014 11/3/2014      1  
2      77        22     -55          7.60 11/4/2014 11/4/2014      1  
3      47       100      53          7.32 11/5/2014 11/5/2014      1  
4      25        72      47          6.49 11/4/2014 11/4/2014      1  
5      22        69      47          6.49 11/5/2014 11/5/2014      1  
6      37        83      46          6.36 11/6/2014 11/6/2014      1  
7      20        62      42          5.80 11/6/2014 11/6/2014      1  
8      60       100      40          5.53 11/4/2014 11/4/2014      1  
9      22        60      38          5.25 11/3/2014 11/3/2014      1  
10     30        68      38          5.25 11/3/2014 11/3/2014      1  
# ... with 178 more rows, and 153 more variables: condition <chr>, subject <dbl>,  
# gender <chr>, age <dbl>, STAI_pre_1_1 <dbl>, STAI_pre_1_2 <dbl>,  
# STAI_pre_1_3 <dbl>, STAI_pre_1_4 <dbl>, STAI_pre_1_5 <dbl>,  
# STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>, STAI_pre_2_1 <dbl>,  
# STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>,  
# STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>,  
# STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>, ...
```

# Replacing columns

## Update

Using existing columns updates their content.

## Warn

If not using names the actions is used as name (avoid)

`mutate()` existing columns

Rescaling `mood` columns

```
judgments %>%  
  mutate(mood_pre = mood_pre/mean(mood_pre, na.rm = TRUE),  
         mood_post = mood_post/mean(mood_post, na.rm = TRUE),  
         mood_pre / mean(mood_post, na.rm = TRUE)) %>%  
  select( starts_with("mood"))
```

```
# A tibble: 188 × 3  
  mood_pre mood_post `mood_pre/mean(mood_post, na.rm = TRUE)`  
    <dbl>    <dbl>          <dbl>  
1     1.36      NA             1.36  
2     0.994    0.680            0.994  
3     0.371    0.971            0.371  
4     0.893    1.10            0.893  
5     0.809    NA             0.809  
6     1.23    1.18            1.23  
7      NA     NA             NA  
8     1.68    NA             1.68  
9     1.13    1.20            1.13  
10    0.505    1.10            0.505  
# ... with 178 more rows
```

# case\_when()

Categorize the `mood_pre` column

```
judgments %>%  
  mutate(mood_pre_cat = case_when(  
    mood_pre < 25 ~ "poor",  
    mood_pre > 75 ~ "great",  
    TRUE ~ "normal")) %>%  
  select(starts_with("mood")) %>%  
  arrange(desc(mood_pre))
```

```
# A tibble: 188 × 3  
  mood_pre mood_post mood_pre_cat  
    <dbl>    <dbl>    <chr>  
1     100      NA great  
2     100      NA great  
3      96     98 great  
4      95      NA great  
5      94      NA great  
6      93      NA great  
7      87      NA great  
8      87      NA great  
9      85     91 great  
10     84      NA great  
# ... with 178 more rows
```

Artwork by [@allison\\_horst](#)

**dplyr::case\_when()** IF ELSE... (but you love it?)

df %>% *ADD COLUMN 'danger'*  
 mutate(danger = case\_when(  
 *IF type is kraken* type == "kraken" *THEN* ~ "extreme!",  
 *TRUE ~ "high"* TRUE ~ "high"))  
*OTHERWISE, danger is high.*



# Act on multiple columns at once using across()

## Usage

Can be plugged into `mutate()`, `filter()`, `summarise()`...

`across(ON WHO, DO WHAT)`

- **Columns** selection:
  - Argument `.cols =`
  - `tidyselect` helpers
  - `everything()` to select all columns.
  - **Conditions** (boolean) needs `where()`, `across(where(is.numeric))`
- **Actions** using functions:
  - Argument `.fns =`
  - `function, arg1, arg2`
  - `~ function(.x)`, with placeholder `.x`
  - Multiple functions need to be wrapped in a `list()`
  - New column names can be controlled

## Recent changes ⚠


The functions `mutate_all()`, `mutate_if()` and `mutate_at()` have been superseded by using `mutate()` with `across()` and `where()`.

`dplyr::across()`

use within `mutate()` or `summarize()` to apply function(s) to a selection of columns!

### EXAMPLE:

```
df %>%  
  group_by(species) %>%  
  summarize(  
    across(where(is.numeric), mean)  
  )
```



species	mass_g	age_yr	range_sqmi
pika	163	2.4	0.46
marmot	1509	3.0	0.87
marmot	2417	5.6	0.62

Artwork by [@allison\\_horst](#)

# Examples of across() usage

Find rows where both mood data columns are missing

```
judgments %>%  
  filter(across(starts_with("mood_p"), is.na)) %>%  
  select(subject, starts_with("mood"))
```

```
# A tibble: 1 × 3  
  subject mood_pre mood_post  
    <dbl>   <dbl>   <dbl>  
1         5      NA      NA
```

Of note: interesting **new** function: `if_any()`:  
`filter(if_any(everything(), is.na))`

Add 1 to the PBC data

To convert Likert scales

```
judgments %>%  
  mutate(across(contains("STAI"),  
    `+`, 1)) %>%  
  select(starts_with("STAI"))
```

```
# A tibble: 188 × 42  
  STAI_pre_1_1 STAI_pre_1_2 STAI_pre_1_3 STAI_pre_1_4 STAI_pre_1_5  
    <dbl>      <dbl>      <dbl>      <dbl>      <dbl>  
1         3         2         3         3         3  
2         4         3         4         2         2  
3         5         4         4         4         4  
4         3         3         3         3         3  
5         2         2         2         2         2  
6         3         3         2         2         2  
7         3         3         2         2         2  
8         2         2         2         2         2  
9         3         3         2         2         2  
10        5         3         4         4         4  
# ... with 178 more rows, and 36 more variables: STAI_pre_1_7 <dbl>  
# STAI_pre_2_1 <dbl>, STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>,  
# STAI_pre_2_4 <dbl>, STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>,  
# STAI_pre_2_7 <dbl>, STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>,  
# STAI_pre_3_3 <dbl>, STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>,  
# STAI_pre_3_6 <dbl>, STAI_post_1_1 <dbl>, STAI_post_1_2 <dbl>,  
# STAI_post_1_3 <dbl>, STAI_post_1_4 <dbl>, STAI_post_1_5 <dbl>
```

``+`` to access the sum function name, a bit tricky

# Simplifying actions with the placeholder .x

## Add 1 to the MAIA questionnaire data

Activate the placeholder (`.x`) with the `~`

```
judgments %>%
  mutate(across(contains("MAIA"),
    ~ .x + 1)) %>%
  select(starts_with("MAIA"))
```

```
# A tibble: 188 × 40
  MAIA_1_1 MAIA_1_2 MAIA_1_3 MAIA_1_4 MAIA_1_5 MAIA_1_6 MAIA_1_7
    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1         3         5         5         5         3         3         3
2        NA        NA        NA        NA        NA        NA        NA
3        NA        NA        NA        NA        NA        NA        NA
4        NA        NA        NA        NA        NA        NA        NA
5         4         4         4         4         4         4         4
6        NA        NA        NA        NA        NA        NA        NA
7         5         5         5         6         3         3         3
8         4         5         5         5         6         3         3
9        NA        NA        NA        NA        NA        NA        NA
10       NA        NA        NA        NA        NA        NA        NA
# ... with 178 more rows, and 32 more variables: MAIA_1_9 <dbl>,
# MAIA_1_11 <dbl>, MAIA_1_12 <dbl>, MAIA_1_13 <dbl>, MAIA_1_14
# MAIA_1_15 <dbl>, MAIA_1_16 <dbl>, MAIA_2_1 <dbl>, MAIA_2_2
# MAIA_2_3 <dbl>, MAIA_2_4 <dbl>, MAIA_2_5 <dbl>, MAIA_2_6 <dbl>,
# MAIA_2_7 <dbl>, MAIA_2_8 <dbl>, MAIA_2_9 <dbl>, MAIA_2_10 <dbl>,
# MAIA_2_11 <dbl>, MAIA_2_12 <dbl>, MAIA_2_13 <dbl>, MAIA_2_14
# MAIA_2_15 <dbl>, MAIA_2_16 <dbl>, MAIA_noticing <dbl>, ...
```

## Placeholder allows to use function calls

```
judgments %>%
  mutate(across(contains("mood"),
    ~ .x / 100)) %>%
  select(starts_with("mood"))
```

```
# A tibble: 188 × 2
  mood_pre mood_post
    <dbl>    <dbl>
1     0.81         NA
2     0.59     0.42
3     0.22     0.6
4     0.53     0.68
5     0.48         NA
6     0.73     0.73
7         NA         NA
8         1         NA
9     0.67     0.74
10    0.3     0.68
# ... with 178 more rows
```

This is advanced usage, ok to find it hard.



# Selecting columns with a predicate where()

## Add 1 to numeric columns for all questionnaire data

- predicate means return **TRUE** or **FALSE**

```
judgments %>%  
  mutate(across(where(is.numeric),  
    ~ .x + 1))
```

```
# A tibble: 188 × 158  
  start_date end_date finished condition subject gender age  
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr> <dbl>  
1 11/3/2014 11/3/2014      2 control      3 female 25  
2 11/3/2014 11/3/2014      2 stress      2 female 20  
3 11/3/2014 11/3/2014      2 stress      4 female 20  
4 11/3/2014 11/3/2014      2 stress      5 female 23  
5 11/3/2014 11/3/2014      2 control      8 female 23  
6 11/3/2014 11/3/2014      2 stress      7 female 23  
7 11/3/2014 11/3/2014      2 control      6 female 19  
8 11/3/2014 11/3/2014      2 control     10 male 21  
9 11/3/2014 11/3/2014      2 stress     17 female 22  
10 11/3/2014 11/3/2014      2 stress     14 female 20  
# ... with 178 more rows, and 150 more variables: mood_post <dbl>  
# STAI_pre_1_1 <dbl>, STAI_pre_1_2 <dbl>, STAI_pre_1_3 <dbl>,  
# STAI_pre_1_4 <dbl>, STAI_pre_1_5 <dbl>, STAI_pre_1_6 <dbl>,  
# STAI_pre_1_7 <dbl>, STAI_pre_2_1 <dbl>, STAI_pre_2_2 <dbl>,  
# STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>, STAI_pre_2_5 <dbl>,  
# STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>, STAI_pre_3_1 <dbl>,  
# STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>, STAI_pre_3_4 <dbl>,
```



Now we also get **subject** changed!

# More advanced across: multiple functions

## Summarise by the mean of mood

```
judgments %>%  
  summarise(across(starts_with("mood"),  
                    list(mean, sd)))
```

```
# A tibble: 1 × 4  
  mood_pre_1 mood_pre_2 mood_post_1 mood_post_2  
    <dbl>    <dbl>    <dbl>    <dbl>  
1      NA      NA      NA      NA
```

NA is obviously not playing nice here. Need to supply additional arguments (`na.rm = TRUE`)

## Better with naming and function arguments

```
judgments %>%  
  summarise(across(starts_with("moral_dil"),  
                    list(aveg = ~ mean(.x, na.rm = TRUE),  
                          sdev = ~ sd(.x, na.rm = TRUE))))
```

```
# A tibble: 1 × 14  
  moral_dilemma_dog_aveg moral_dilemma_dog... moral_dilemma_wal...  
    <dbl>          <dbl>          <dbl>  
1      7.35          2.17          7.14  
# ... with 10 more variables: moral_dilemma_plane_aveg <dbl>,  
#   moral_dilemma_plane_sdev <dbl>, moral_dilemma_resume_aveg <dbl>,  
#   moral_dilemma_resume_sdev <dbl>, moral_dilemma_kitten_aveg <dbl>,  
#   moral_dilemma_kitten_sdev <dbl>, moral_dilemma_trolley_aveg <dbl>,  
#   moral_dilemma_trolley_sdev <dbl>, moral_dilemma_control_aveg <dbl>,  
#   moral_dilemma_control_sdev <dbl>
```

# Manipulation by row

## Summing all scores for the STAI questionnaire

```
judgments %>%  
  mutate(total_stai =  
    sum(across(contains("STAI")),  
      na.rm = TRUE)) %>%  
  select(subject, total_stai, contains("STAI"))
```

```
# A tibble: 188 × 44  
  subject total_stai STAI_pre_1_1 STAI_pre_1_2 STAI_pre_1_3  
  <dbl>    <dbl>    <dbl>    <dbl>    <dbl>  
1       2     22866         2         1         1  
2       1     22866         3         2         1  
3       3     22866         4         3         1  
4       4     22866         2         2         1  
5       7     22866         1         1         1  
6       6     22866         2         2         1  
7       5     22866         2         2         1  
8       9     22866         1         1         1  
9      16     22866         2         2         1  
10     13     22866         4         2         1  
# ... with 178 more rows, and 38 more variables: STAI_pre_1_4 <dbl>,  
# STAI_pre_1_5 <dbl>, STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>, STAI_pre_2_1  
# STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>,  
# STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>,  
# STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>,  
# STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>, STAI_pre_3_6 <dbl>,  
# STAI_post_1_1 <dbl>, STAI_post_1_2 <dbl>, STAI_post_1_3 <dbl>,  
# STAI_post_1_4 <dbl>, STAI_post_1_5 <dbl>, STAI_post_1_6 <dbl>,  
# STAI_post_1_7 <dbl>, STAI_post_2_1 <dbl>, STAI_post_2_2 <dbl>,  
# STAI_post_2_3 <dbl>, STAI_post_2_4 <dbl>, STAI_post_2_5 <dbl>,  
# STAI_post_2_6 <dbl>, STAI_post_2_7 <dbl>, STAI_post_3_1 <dbl>,  
# STAI_post_3_2 <dbl>, STAI_post_3_3 <dbl>, STAI_post_3_4 <dbl>,  
# STAI_post_3_5 <dbl>, STAI_post_3_6 <dbl>
```

- `rowwise()` - computation by row
  - `c_across()` - selects columns with tidyselect helpers

```
judgments %>%  
  rowwise() %>%  
  mutate(total_stai = sum(c_across(  
    starts_with("STAI")))) %>%  
  select(subject, total_stai, contains("STAI"))
```

```
# A tibble: 188 × 44  
# Rowwise:  
  subject total_stai STAI_pre_1_1 STAI_pre_1_2 STAI_pre_1_3  
  <dbl>    <dbl>    <dbl>    <dbl>    <dbl>  
1       2         NA         2         1         1  
2       1       200         3         2         1  
3       3       212         4         3         1  
4       4       148         2         2         1  
5       7         NA         1         1         1  
6       6       134         2         2         1  
7       5         NA         2         2         1  
8       9         NA         1         1         1  
9      16       122         2         2         1  
10     13       196         4         2         1  
# ... with 178 more rows, and 38 more variables: STAI_pre_1_4 <dbl>,  
# STAI_pre_1_5 <dbl>, STAI_pre_1_6 <dbl>, STAI_pre_1_7 <dbl>, STAI_pre_2_1  
# STAI_pre_2_2 <dbl>, STAI_pre_2_3 <dbl>, STAI_pre_2_4 <dbl>,  
# STAI_pre_2_5 <dbl>, STAI_pre_2_6 <dbl>, STAI_pre_2_7 <dbl>,  
# STAI_pre_3_1 <dbl>, STAI_pre_3_2 <dbl>, STAI_pre_3_3 <dbl>,  
# STAI_pre_3_4 <dbl>, STAI_pre_3_5 <dbl>, STAI_pre_3_6 <dbl>,  
# STAI_post_1_1 <dbl>, STAI_post_1_2 <dbl>, STAI_post_1_3 <dbl>,  
# STAI_post_1_4 <dbl>, STAI_post_1_5 <dbl>, STAI_post_1_6 <dbl>,  
# STAI_post_1_7 <dbl>, STAI_post_2_1 <dbl>, STAI_post_2_2 <dbl>,  
# STAI_post_2_3 <dbl>, STAI_post_2_4 <dbl>, STAI_post_2_5 <dbl>,  
# STAI_post_2_6 <dbl>, STAI_post_2_7 <dbl>, STAI_post_3_1 <dbl>,  
# STAI_post_3_2 <dbl>, STAI_post_3_3 <dbl>, STAI_post_3_4 <dbl>,  
# STAI_post_3_5 <dbl>, STAI_post_3_6 <dbl>
```

# Data Transformation with dplyr 1.0 (part 2)

A guide to using (c\_)across() to apply the same functions repeatedly

© R Data Berlin [@rdatabelin https://github.com/courtio/Rguides](https://github.com/courtio/Rguides)



Use **across()** to apply the same function(s) on multiple columns

tbl %>%  
[group\_by(name\_grouping\_col\_X, name\_grouping\_col\_Y...) %>% ] ## grouping is optional  
verb\*(across(selected\_columns, fn\_with\_args, [.names = prototype\_for\_new\_col\_names\*])) ## prototype for naming is optional

**mutate** ☺  
**transmute** 🎵  
**group\_by** 🍏  
**arrange**  
**filter** ⚙️  
**summarize** 🎵  
**count** 📊

**selection from column-:**  
names, values, positions

**name\_col\_X:name\_col\_Y** ☺  
**c(name\_col\_X, name\_col\_Y)** ☺  
**all\_of/any\_of(c("name\_col\_X", "name\_col\_Y"))**  
**starts\_with/ends\_with/contains("partial\_name\_col")** 🎵  
**matches("regular\_expression")** 🍏  
**num\_range("name\_col\_without\_number", vector\_numbers)**  
**where(fn\_returning\_TRUE/FALSE\_for\_each\_col)** ⚙️  
**c(position\_col\_X, position\_col\_Y)**  
**everything()** 🎵  
**last\_col()**

⚡ when fn runs, .x will be internally replaced by the content of the selected columns

1 function  
fn [, args\*] ☺  
~ fn(.x\* [, args\*]) ⚙️  
\*args = possible arguments for fn

≥ 1 functions  
**list(suffix1\_new\_col\_names\* = fn1, suffix2\_new\_col\_names\* = fn2), [, args\*]** 🍏  
**list(suffix1\_new\_col\_names\* = ~ fn1(.x\* [, args\*]), suffix2\_new\_col\_names\* = ~ fn2(.x\* [, args\*]))** 🎵

A quoted expression with any text and the placeholders { .fn } referring to function position and/or { .col } referring to former column names  
Only useful when no suffix defined in fn\_with\_args

**EXAMPLES for across()**  
[!!! do not forget to load dplyr before running examples: library(dplyr)]  
## turning specific columns into z-scores:  
iris %>%  
mutate(across(c("Sepal.Length", "Sepal.Width"), scale, .names = "{.col}\_z"))  
## keeping only rows where numeric values are > 2 in all numeric columns:  
iris %>% filter(across(where(is.numeric), ~ .x > 2))  
## defining 3 groups for all columns with name made of 2 words around a point:  
iris %>% group\_by(across(matches("\\w\\.\\w"), list(discrete = cut, breaks = 3)))  
## counting NA in each column:  
iris %>% summarise(across(everything(), list("NA" = ~ sum(is.na(.x)))))

⚡ if no name is defined, some will automatically be generated by dplyr

\* across() and c\_across() should not be used within dplyr verbs shown in part 1 and not here, as it would not make sense. Note however that there is a special function to rename multiple columns: rename\_with(fn\_with\_args, selected\_columns)

Use **c\_across()** to apply the same function across multiple columns within each row

tbl %>%  
rowwise() %>%  
verb\*(fn\*(c\_across(selected\_columns), [args\*])) %>% ungroup()

**mutate** ☺  
**transmute** 🎵  
**filter** ⚙️  
**count** 📊

!!! don't forget rowwise(); if you do, c\_across() will concatenate values across rows, which is wrong

as for across(), see above

**EXAMPLES for c\_across()**  
[!!! don't forget ungroup() unless you want future dplyr operations to be executed rowwise]  
## computing the area of petal (approximated as rectangles) and only keep that:  
iris %>%  
rowwise() %>%  
transmute(Petal\_Area = prod(c\_across(contains("Petal")))) %>% ungroup()  
## counting rows where at least one numeric values is > 6 for a range of columns:  
iris %>%  
rowwise() %>%  
count(any(c\_across(Sepal.Length:Petal.Width) > 6)) %>% ungroup()

⚡ unlike fn\_with\_args in across() calls, the function fn is not here provided as a definition but used directly, so you can only use one function that has already been defined (you cannot define it on the fly using ~ & .x)

# Grouping and summarizing

# Counting elements

How many participants per condition?

## Replacing calls to `table()`

```
head(as.data.frame(table(judgments$condition)))
```

	Var1	Freq
1	control	91
2	stress	97

Ugly and convoluted. 🤖

- `count()` groups by specified columns
- Result is a tibble that can be processed further
- `sort = TRUE` avoid piping to `arrange(desc(n))`

```
count(judgments, condition,  
      sort = TRUE)
```

```
# A tibble: 2 × 2  
  condition     n  
  <chr>    <int>  
1 stress      97  
2 control     91
```

`count()` is a shortcut for:

```
group_by(judgments, condition) %>%  
  summarise(n = n(), .groups = "drop")
```

# Summarise data

## When counting is not enough

```
judgments %>%  
  summarise(  
    min = min(mood_pre, na.rm = TRUE),  
    max = max(mood_pre, na.rm = TRUE))
```

```
# A tibble: 1 × 2  
  min    max  
<dbl> <dbl>  
1     9   100
```

- **summarise** returns as many rows as groups, one grouping being peeled off from the right for functions with a single return value.
- **mutate** returns as many rows as given

```
judgments %>%  
  mutate( min = min(mood_pre, na.rm = TRUE),  
          max = max(mood_pre, na.rm = TRUE))
```

```
# A tibble: 188 × 160  
  start_date end_date finished condition subject gender age  
  <chr>      <chr>      <dbl> <chr>      <dbl> <chr> <dbl>  
1 11/3/2014 11/3/2014      1 control      2 female 24  
2 11/3/2014 11/3/2014      1 stress      1 female 19  
3 11/3/2014 11/3/2014      1 stress      3 female 19  
4 11/3/2014 11/3/2014      1 stress      4 female 22  
5 11/3/2014 11/3/2014      1 control      7 female 22  
6 11/3/2014 11/3/2014      1 stress      6 female 22
```

## If we want the min/max per biotype

And keep the grouping at the same time

```
judgments %>%  
  group_by(condition) %>%  
  summarise(min = min(mood_pre, na.rm = TRUE),  
            max = max(mood_pre, na.rm = TRUE),  
            .groups = "keep")
```

```
# A tibble: 2 × 3  
# Groups:   condition [2]  
  condition min    max  
  <chr>    <dbl> <dbl>  
1 control     19   100  
2 stress      9    96
```

# Grouping by more than one variable

## Peeling effect

```
judgments %>%  
  group_by(condition, gender) %>%  
  summarise(n = n()) -> n_part
```

`summarise()` has grouped output by 'condition'. You can overri

```
slice_head(n_part, n = 1)
```

```
# A tibble: 2 × 3  
# Groups:   condition [2]  
  condition gender      n  
  <chr>      <chr> <int>  
1 control   female     65  
2 stress    female     82
```

⚠ Most functions in **dplyr** are group-aware!

## No grouping after additional calls to `summarise()`

```
summarise(n_part, n = n()) %>%  
  slice_head(n = 3)
```

```
# A tibble: 2 × 2  
  condition      n  
  <chr>      <int>  
1 control      2  
2 stress       2
```

# Grouping (cont.)

## Remark

Ask explicitly to **ungroup** data or use the `ungroup()` function.

How many different answers for a variable by condition and gender?

```
judgments %>%  
  group_by(condition, gender ) %>%  
  summarise(n_ans = n_distinct( STAI_post_1_1),  
            .groups = "drop")
```

```
# A tibble: 4 × 3  
  condition gender n_ans  
  <chr>      <chr> <int>  
1 control   female     1  
2 control   male       1  
3 stress    female     4  
4 stress    male       3
```

# Summarise can deal with multiple values per group

Since v1.0

- `range()` returns min **and** max
- `summarise()` duplicates the key names

```
judgments %>%
  group_by(condition, gender) %>%
  summarise(range = range(mood_pre - mood_post),
            n = n(),
            .groups = "keep")
```

```
# A tibble: 8 × 4
# Groups:   condition, gender [4]
  condition gender range      n
  <chr>      <chr> <dbl> <int>
1 control   female    NA     65
2 control   female    NA     65
3 control   male      NA     26
4 control   male      NA     26
5 stress    female   -53     82
6 stress    female    66     82
7 stress    male    -32     15
8 stress    male      0     15
```

More advanced but useful: 3 quantiles

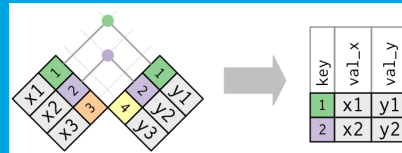
```
judgments %>%
  filter(!is.na(mood_pre)) %>%
  group_by(condition) %>%
  summarise(
    quan = quantile(mood_pre,
                    c(0.25, 0.5, 0.75)),
    q = c(0.25, 0.5, 0.75),
    n = n(),
    .groups = "keep") %>%
  rmarkdown::paged_table()
```

condition	quan	q	n
<chr>	<dbl>	<dbl>	<int>
control	53	0.25	90
control	65	0.50	90
control	76	0.75	90
stress	44	0.25	97
stress	58	0.50	97
stress	67	0.75	97

6 rows

# Joining data frames

## Relational operations



# Matching subjects in 2 tibbles

## Additional data for some participants for coffee consumption

Manual input using `tribble()`

```
coffee_drinkers <-  
  tribble(~student, ~coffee_shots,  
          21,      1,  
          23,      4,  
          28,      2)  
  
coffee_drinkers
```

```
# A tibble: 3 × 2  
  student coffee_shots  
  <dbl>     <dbl>  
1     21           1  
2     23           4  
3     28           2
```

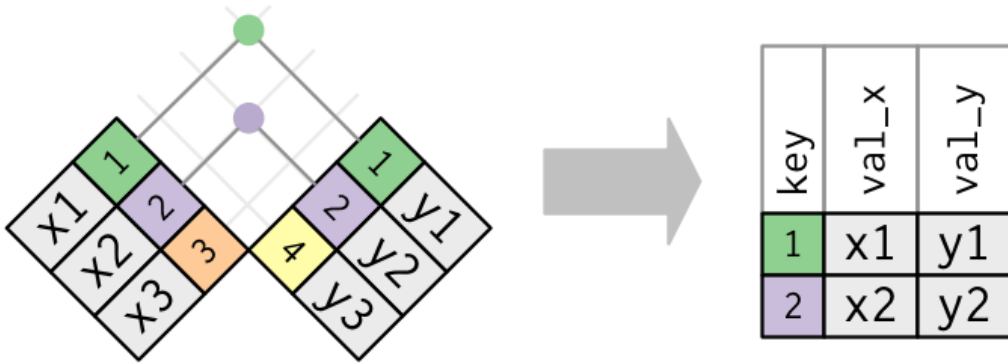
## Smaller sample set

```
subject_mood <- judgments %>%  
  select(subject, condition, gender, starts_with("mood")) %>%  
  distinct()
```

This is made up data for demonstration purposes only.

# Combining the two tables

## Inner join



**Error!** no common key.

```
inner_join(coffee_drinkers,  
            subject_mood)
```

Error: `by` must be supplied when `x` and `y` have no common variables  
use `by = character()` to perform a cross-join.

Provide the corresponding columns names

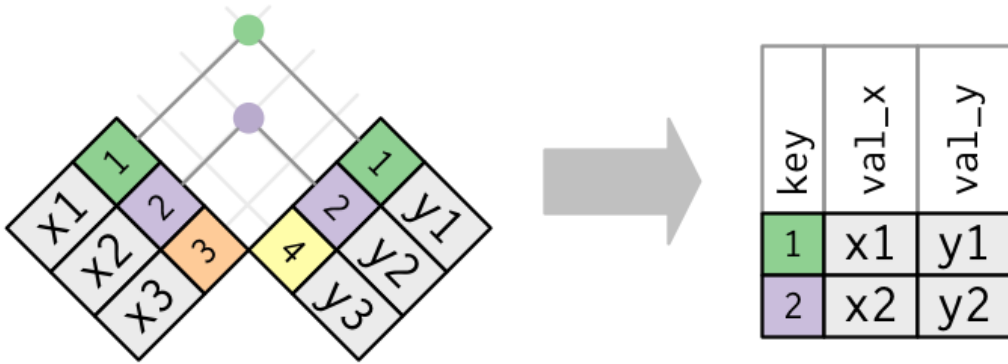
```
inner_join(subject_mood,  
            coffee_drinkers,  
            by = c(subject = "student"))
```

```
# A tibble: 3 × 6  
  subject condition gender mood_pre mood_post coffee_shots  
  <dbl> <chr>      <chr>    <dbl>    <dbl>      <dbl>  
1     23 control   female      78      NA          4  
2     21 control   female      68      NA          1  
3     28 stress    male       53      68          2
```

bare name for `subject` as evaluated in `subject_mood`

# Mutating joins

## Inner join

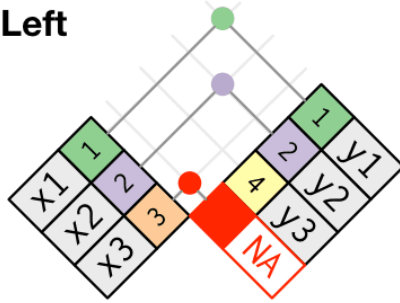


credit: [Hadley Wickham, R for data science](#)

## Creating new tables through *joins*

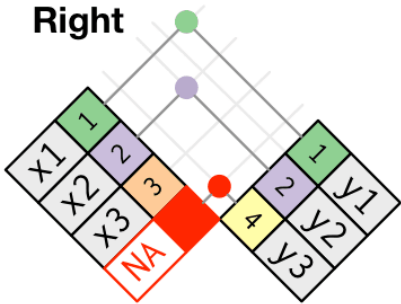
- Key operations in data processing
- Role of *observations* as row changes
- `inner_join()` is the most strict *join* operations
- `merge` is a similar operation in base R

Left



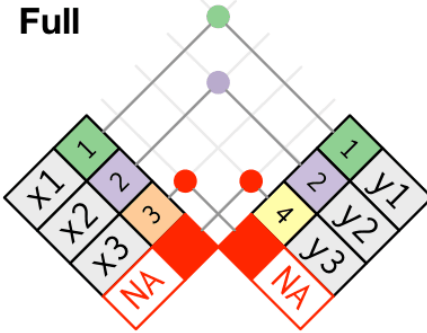
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Full



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

## Example with full\_join()

```
subject_mood %>%
  full_join(coffee_drinkers,
            by = c(subject = "student"))
```

```
# A tibble: 187 × 6
  subject condition gender mood_pre mood_post coffee_shots
  <dbl> <chr> <chr> <dbl> <dbl> <dbl>
1     2 control female      81      NA      NA
2     1 stress female      59      42      NA
3     3 stress female      22      60      NA
4     4 stress female      53      68      NA
5     7 control female      48      NA      NA
6     6 stress female      73      73      NA
7     5 control female      NA      NA      NA
8     9 control male      100      NA      NA
9    16 stress female      67      74      NA
10    13 stress female      30      68      NA
# ... with 177 more rows
```

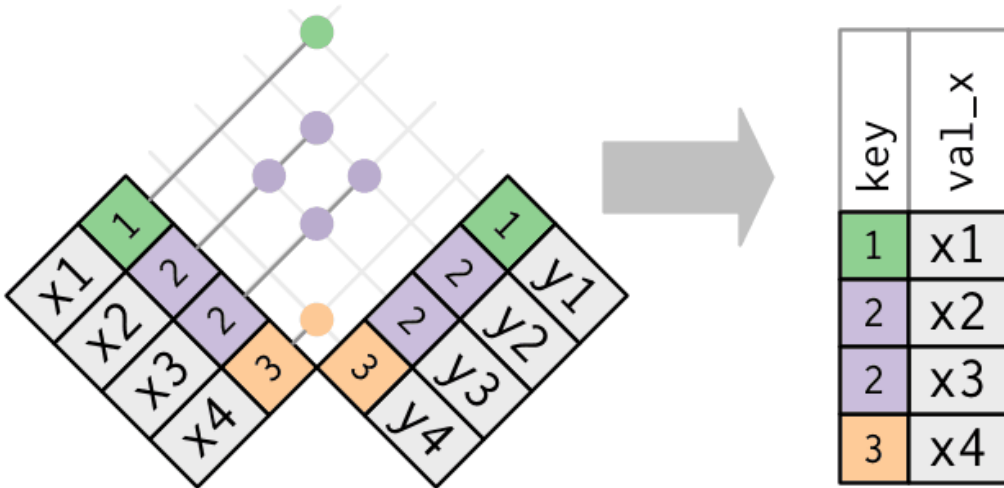
# Filtering joins -

Only the **existence** of a match is important; it doesn't matter which observation is matched. This means that **filtering** joins **never duplicate rows** like **mutating** joins do

– Hadley Wickam - [R for data science](#)

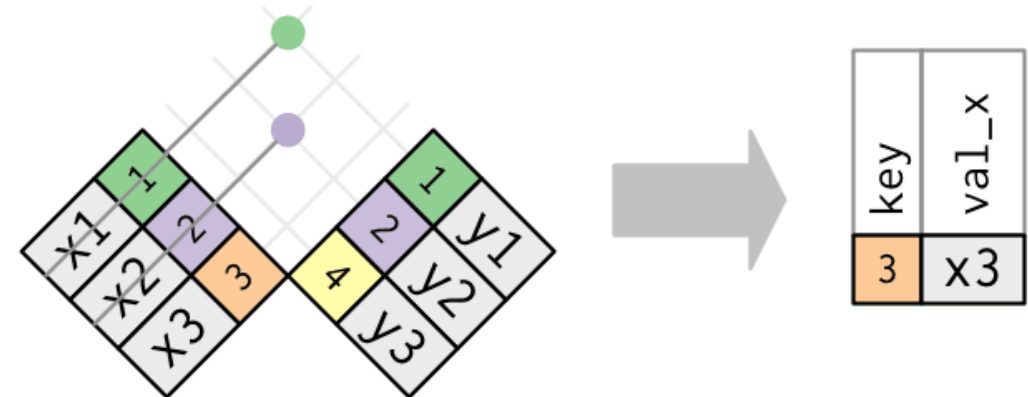
## semi\_join()

Filter matches in x, no duplicates.



## anti\_join()

Extract what does **not** match.



# semi\_join() does not alter original

```
(tx <- tribble(~ x, ~key,  
  "x1", 1,  
  "x2", 2,  
  "x3", 2,  
  "x4", 3))
```

```
# A tibble: 4 × 2  
  x      key  
  <chr> <dbl>  
1 x1      1  
2 x2      2  
3 x3      2  
4 x4      3
```

```
(ty <- tribble(~ y, ~key,  
  "y1", 1,  
  "y2", 2,  
  "y3", 2,  
  "y4", 3))
```

```
# A tibble: 4 × 2  
  y      key  
  <chr> <dbl>  
1 y1      1  
2 y2      2  
3 y3      2  
4 y4      3
```

## Filtering

```
semi_join(tx, ty)
```

Joining, by = "key"

```
# A tibble: 4 × 2  
  x      key  
  <chr> <dbl>  
1 x1      1  
2 x2      2  
3 x3      2  
4 x4      3
```

## Mutating

```
inner_join(tx, ty)
```

Joining, by = "key"

```
# A tibble: 6 × 3  
  x      key y  
  <chr> <dbl> <chr>  
1 x1      1 y1  
2 x2      2 y2  
3 x2      2 y3  
4 x3      2 y2  
5 x3      2 y3  
6 x4      3 y4
```

**Exchange of variables and values with pivot functions**

**tidyr functionality**

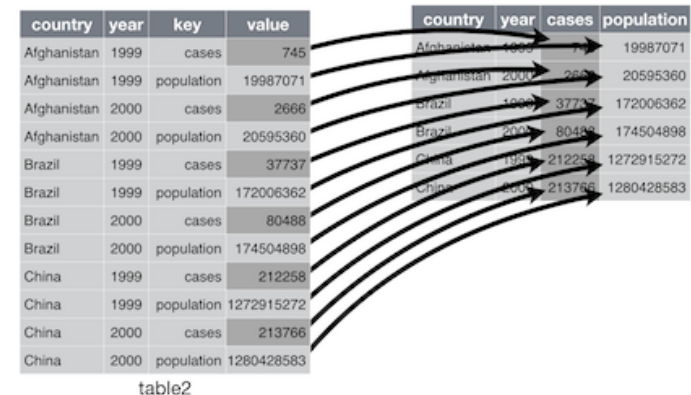
# Converting into long or wide formats - pivot functions

- The wide format is generally **untidy** but found in the majority of datasets
- The **wide** format makes computation on columns sometimes easier
- The new functions `pivot_longer()` and `pivot_wider()` supersede older functions called `gather()` and `spread()`.
- **pivot** functions are easier to use. No really!



country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

table4



country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

# Making a wide data set longer

## Calculations involving column and values A longer, tidier table

### A toy data set

Variants by sample and positions.

Why is this not **tidy**?

```
variants_wide <- tribble(
  ~sample_id, ~`3`, ~`5`, ~`8`,
  "L1002",     "A",  "C",  NA,
  "L1034",     "A",  NA,   "T",
  "L1234",     NA,   "C",  "T"
)
variants_wide
```

```
# A tibble: 3 × 4
  sample_id `3` `5` `8`
  <chr>     <chr> <chr> <chr>
1 L1002     A     C    <NA>
2 L1034     A    <NA>  T
3 L1234    <NA>  C     T
```

```
pivot_longer(variants_wide,
  -contains("sample"), # columns argument, required
  names_to = "pos",
  values_to = "variant")
```

```
# A tibble: 9 × 3
  sample_id pos variant
  <chr>     <chr> <chr>
1 L1002     3     A
2 L1002     5     C
3 L1002     8    <NA>
4 L1034     3     A
5 L1034     5    <NA>
6 L1034     8     T
7 L1234     3    <NA>
8 L1234     5     C
9 L1234     8     T
```

# Values across dilemmas

```
readr::read_tsv("https://biostat2.uni.lu/practicals/data/j  
select(subject, condition, age, starts_with("moral_dilem  
pivot_longer(starts_with("moral_dilemma"),  
  names_to = "dilemma",  
  values_to = "dilemma_val",  
  names_pattern = "moral_dilemma_(.*)") %>%  
count(condition, dilemma_val) %>%  
pivot_wider(names_from = dilemma_val,  
  values_from = n,  
  names_prefix = "n")
```

Rows: 188 Columns: 158

— Column specification —

Delimiter: "\t"

chr (5): start\_date, end\_date, condition, gender, logbook

dbl (153): finished, subject, age, mood\_pre, mood\_post, STAI\_pre\_1\_1,

- Use `spec()` to retrieve the full column specification for this data
- Specify the column types or set `show\_col\_types = FALSE` to quiet th

# A tibble: 2 × 10

	condition	n1	n2	n3	n4	n5	n6	n7	n8	n9
	<chr>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
1	control	25	31	48	44	49	65	103	110	162
2	stress	13	35	32	51	56	54	103	133	202

# Helpful tools

# Removing the data frame context through `pull()`

## Remark

By default tidyverse operations that receive a **tibble** as input return a **tibble**.

```
pull(judgments, age)[1:5]
```

```
[1] 24 19 19 22 22
```

```
pull(judgments, -20)[1:10]
```

```
[1] NA 51 41 32 NA 33 NA NA 30 38
```

```
# TODO: make this work  
judgments %>%  
  distinct(mood_pre) %>%  
  pull(mood_pre)
```

# Comparing to data in other rows

## Leading and lagging rows

`lead()` for data in following row and  
`lag()` for data in the row above

## Calculate differences between subjects

Let's assume the subject IDs are in order of the tests being conducted. What is the difference to the previous subject for the "initial mood"?

```
judgments %>%  
  dplyr::select(subject, mood_pre) %>%  
  arrange(subject) %>%  
  mutate(prev_mood_pre = lag(mood_pre),  
         mood_diff = mood_pre - lag(mood_pre))
```

```
# A tibble: 188 × 4  
  subject mood_pre prev_mood_pre mood_diff  
  <dbl>   <dbl>      <dbl>    <dbl>  
1       1       59          NA         NA  
2       2       81          59         22  
3       3       22          81        -59  
4       4       53          22         31  
5       5       NA          53         NA  
6       6       73          NA         NA  
7       7       48          73        -25  
8       8       59          48         11  
9       9      100          59         41  
10      10       72         100        -28  
# ... with 178 more rows
```

# Summary

## Most commonly used - 80%

- `select()` - columns
- `filter()` - rows meeting condition
- `arrange()` - sort
- `glimpse()` - inspect
- `rename()` - change column name
- `relocate()` - move columns
- `mutate()` - create columns
- `across()`, `c_across()` - work on >1 column
- `group_by()`, `ungroup()`, `rowwise()`
- `summarise()` - group-wise summaries
- `lead()` and `lag()` - Values in other rows
- `inner_join` and friends - Merging tables
- `case_when()` simplifies if/else/if/else

source: Lise Vaudor [blog](#)

## Comments

- Represent the verbs you will use 80% of your time. Go to the website to see additional functions.
- `tidyr` and `dplyr` are replacing the `reshape` and `reshape2` packages
  - tidy data
  - <http://tidyr.tidyverse.org/>
  - `vignette("tidy-data")`
- Further reading
  - `tidyjson` - Retrieve json data as nested tibbles



# The other 20%

## Occasionally handy

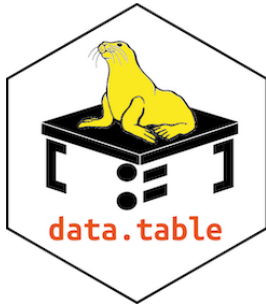
- Assembly: [bind\\_rows](#), [bind\\_cols](#)
- Windows function, [min\\_rank](#), [dense\\_rank](#), [cumsum](#).  
See [vignette](#)
- working with [list-columns](#)
- [multidplyr](#) for parallelized code

## SQL mapping allows database access

- [dplyr](#) code can be translated into SQL and query databases online (using [dbplyr](#))
- different types of tabular data ([dplyr SQL backend](#), databases,

# Bigger data

## Go for `data.table`



- See this interesting [thread](#) about comparing `data.table` versus `dplyr`
  - [data.table](#), see [introduction](#) is very efficient but the syntax is not so easy.
  - Main advantage: inline replacement (tidyverse is frequently copying)
  - As a summary: *tl;dr data.table for speed, dplyr for readability and convenience* [Prashanth Sriram](#)
- Hadley recommends that for data > 1-2 Gb, if speed is your main matter, go for `data.table`
- [dtplyr](#) is for learning the `data.table` from `dplyr` API input
- [tidytable](#) is actually using `data.table` but with `dplyr` syntax
- `data.table` might be not useful for specialized applications with high volumes such as genomics.

# Before we stop

## You learned to:

- Selection and manipulation of
  - observations,
  - variables and
  - values.
- Grouping and summarizing
- Joining and intersecting tibbles
- Reshaping column headers and variables

## Acknowledgments

- Aurelien Ginolhac, Roland Krause (development)
- Hadley Wickham
- Lionel Henry
- Romain François
- [Gina Reynolds](#) for fantastic flipbooks
- [Lise Vaudor](#) nice blog
- [Allison Horst](#) for the great ArtWork
- [Alexandre Courtiol](#) for cheatsheets
- Jenny Bryan
- [poorman](#) by Nathan Eastwood, a re-implementation of [dplyr](#) in base only

**Thank you for your attention!**