# Joining and pivoting tables

Raphael Eisenhofer

2022_06_02

# Outline for today:

- 1. Introduction to tidy data
- 2. How to pivot tables longer or wider
- 3. Joining tables

# 1. Introduction to tidy data

# Tidy data

- Hadley Wickham: see original paper here: https://vita.had.co.nz/papers/tidy-data.pdf

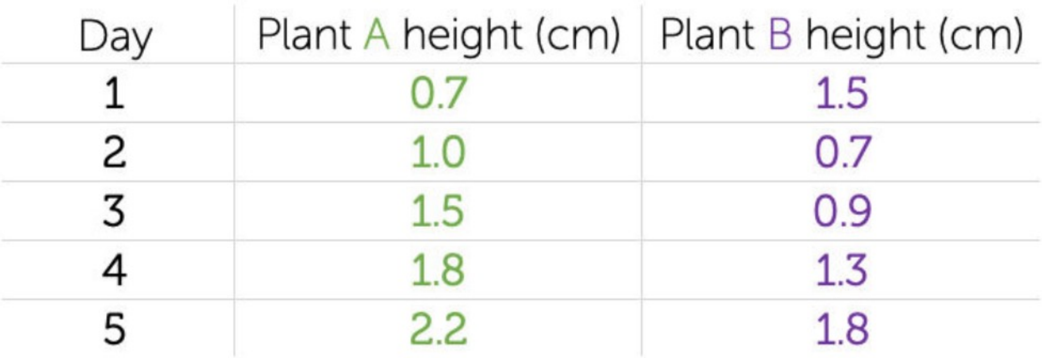
- Tidy data is a consistent way of representing data
  - Benefits include:
    - Easier to interpret
    - Easier to learn tools that manipulate it (e.g. dplyr)
    - Faster (takes advantage of R's vectorized nature)

- The three rules of tidy data:
  - **1: Each variable forms a column**
  - **2: Each observation forms a row**
  - **3: Each value must have its own cell**

# Tidy data example:

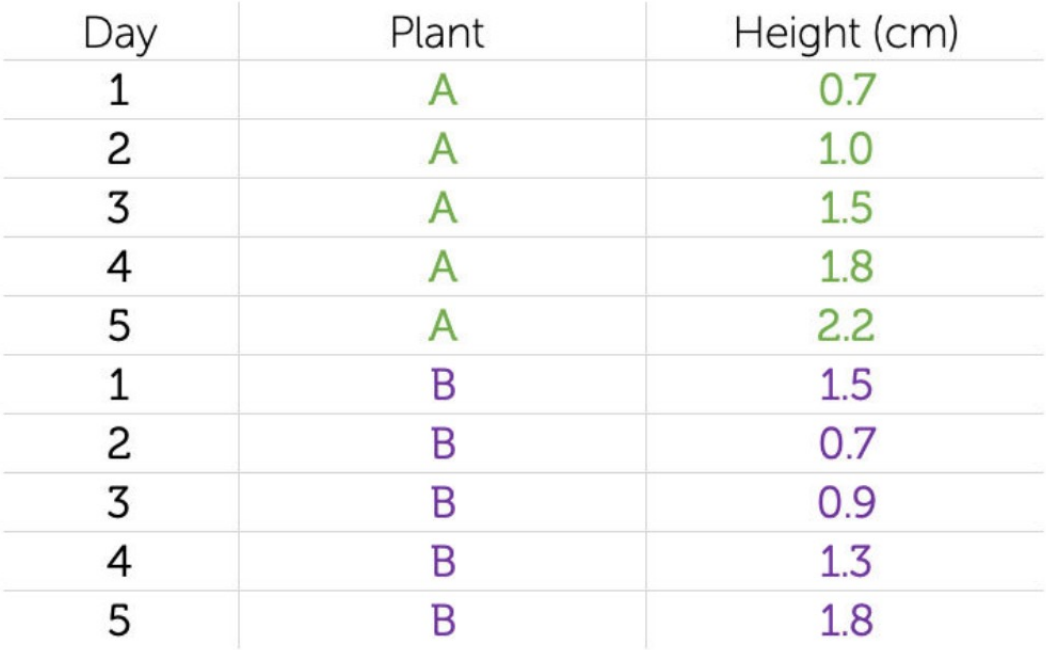
- https://about.dataclassroom.com/blog/keep-your-data-tidy

**Not tidy!**



| Day | Plant A height (cm) | Plant B height (cm) |
|---|---|---|
| 1 | 0.7 | 1.5 |
| 2 | 1.0 | 0.7 |
| 3 | 1.5 | 0.9 |
| 4 | 1.8 | 1.3 |
| 5 | 2.2 | 1.8 |

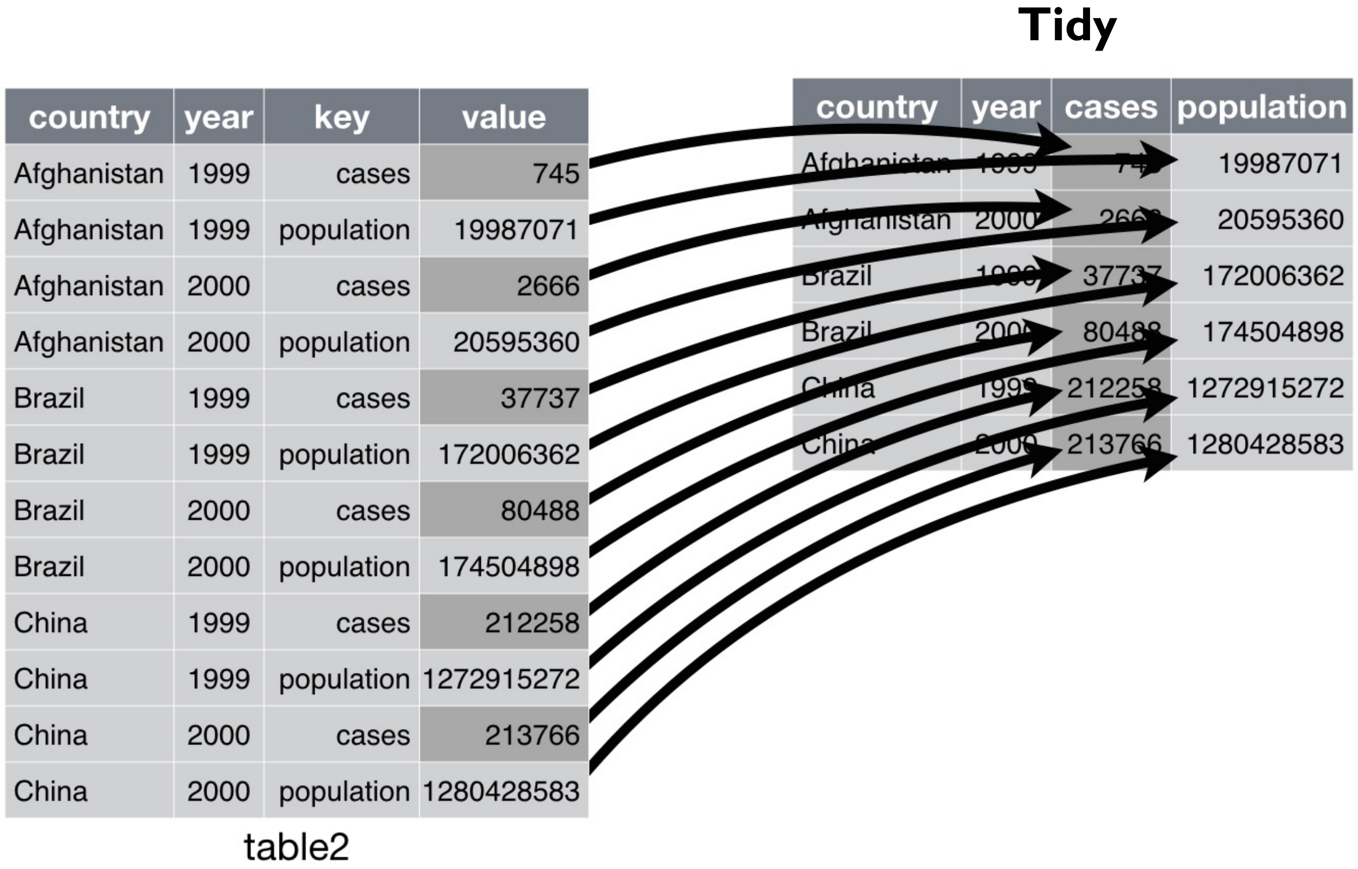
**Tidy**



| Day | Plant | Height (cm) |
|---|---|---|
| 1 | A | 0.7 |
| 2 | A | 1.0 |
| 3 | A | 1.5 |
| 4 | A | 1.8 |
| 5 | A | 2.2 |
| 1 | B | 1.5 |
| 2 | B | 0.7 |
| 3 | B | 0.9 |
| 4 | B | 1.3 |
| 5 | B | 1.8 |

# Tidy data example:

- https://r4ds.had.co.nz/tidy-data.html



| 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 |

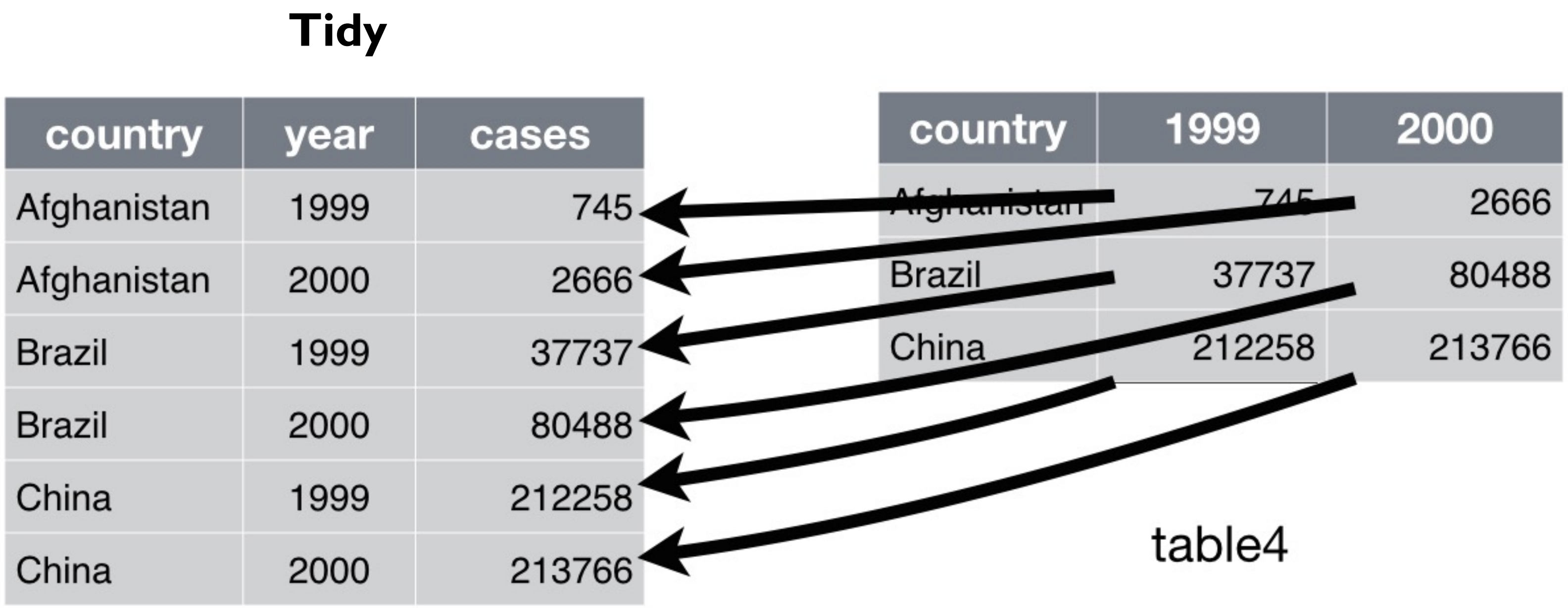
table2

Tidy

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

# Tidy data example:

- https://r4ds.had.co.nz/tidy-data.html



Tidy

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

| country | 1999 | 2000 |
|---|---|---|
| Afghanistan | 745 | 2666 |
| Brazil | 37737 | 80488 |
| China | 212258 | 213766 |

table4

# 2. How to get tidy

# Pivoting longer or wider

pivot_longer()

© @ketama.de.galgo/SPLITPICS UK

pivot_wider()

# pivot_longer()

- **Syntax:**
  - **x** = *dataframe*
  - **cols** = *columns to make longer* [e.g. c(1999, 2000) ]
    - can also use ! (for not), e.g. cols = !country
  - **names_to** = *name of new column,* e.g. "year"
    - Can choose multiple, but need to provide **names_sep** or **names_pattern**
  - **values_to** = *name of new column to store values,* e.g. "cases"

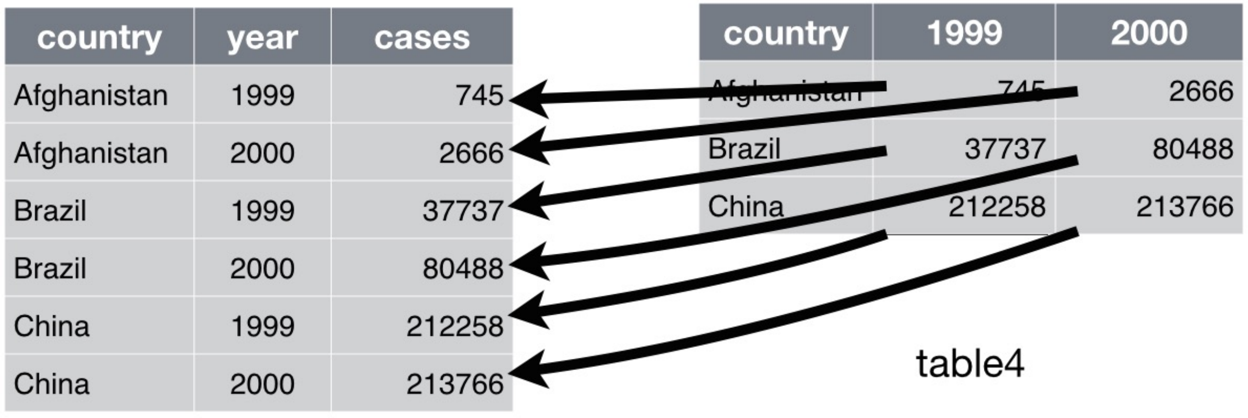
**Optional:**

**values_drop_na** = TRUE

- Will remove redundant rows ☺

**names_pattern** = "pattern"

- Determines how name will be broken, can use regular expressions e.g. (.*)_suffix



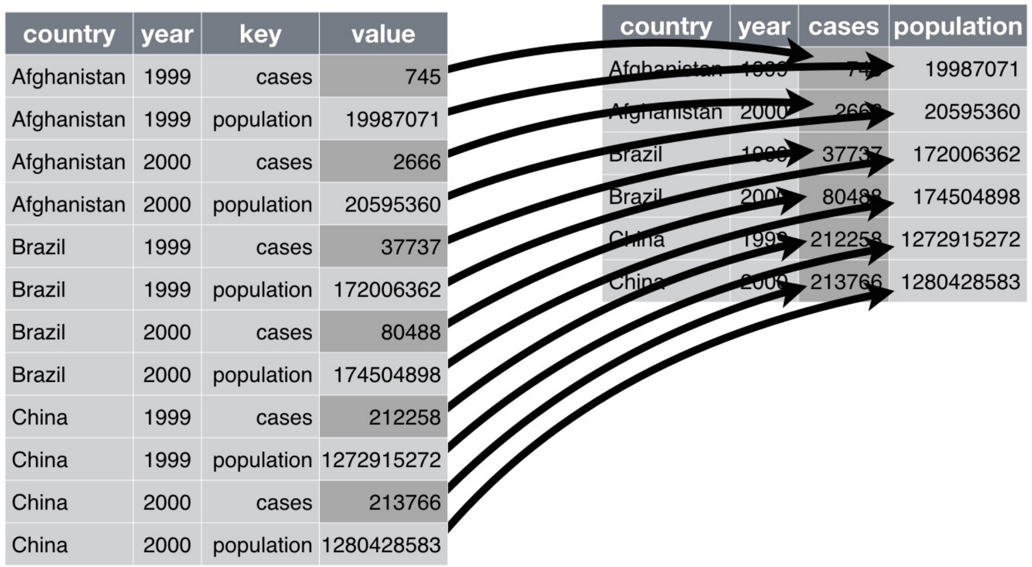
| country | year | cases |
|---|---|---|
| Afghanistan | 1999 | 745 |
| Afghanistan | 2000 | 2666 |
| Brazil | 1999 | 37737 |
| Brazil | 2000 | 80488 |
| China | 1999 | 212258 |
| China | 2000 | 213766 |

| country | 1999 | 2000 |
|---|---|---|
| Afghanistan | 745 | 2666 |
| Brazil | 37737 | 80488 |
| China | 212258 | 213766 |

table4

# pivot_wider()

- **Syntax:**
  - **x** = *dataframe*
  - **names_from** = *name of column to widen,* e.g. "key"
    - Can choose multiple, but need to provide **names_sep** or **names_pattern**
  - **values_from** = *name of column that has values* e.g. "value"
- **Optional:**
  - **values_fill** = value for missing values e.g. 0



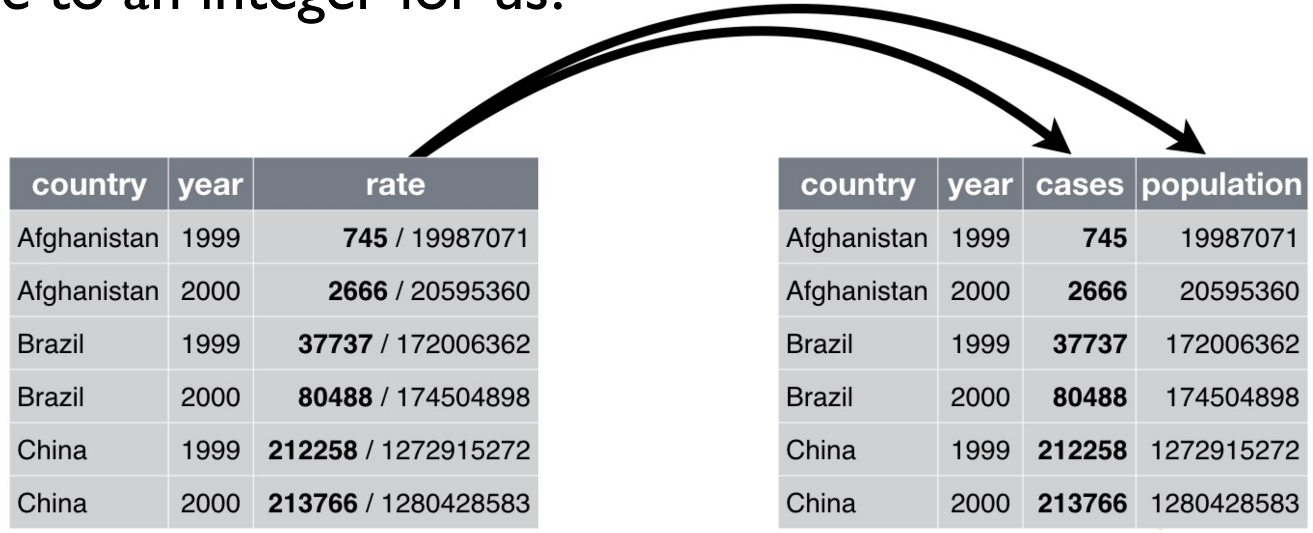
| country | year | key | value |
|---|---|---|---|
| Afghanistan | 1999 | cases | 745 |
| Afghanistan | 1999 | population | 19987071 |
| Afghanistan | 2000 | cases | 2666 |
| Afghanistan | 2000 | population | 20595360 |
| Brazil | 1999 | cases | 37737 |
| Brazil | 1999 | population | 172006362 |
| Brazil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

table2

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

# separate()

- **Syntax:**
  - **col** = *name of the column* (e.g. rate)
  - **into** = *name of new columns* -- e.g. c("cases", "population")
  - **sep** = *separator character* (e.g. "/") – by default it will guess non-alphanumeric chars
  - **convert** = TRUE or FALSE (default behaviour is to leave column types as is – '*chr*', so by using TRUE it will guess and change to an integer for us!



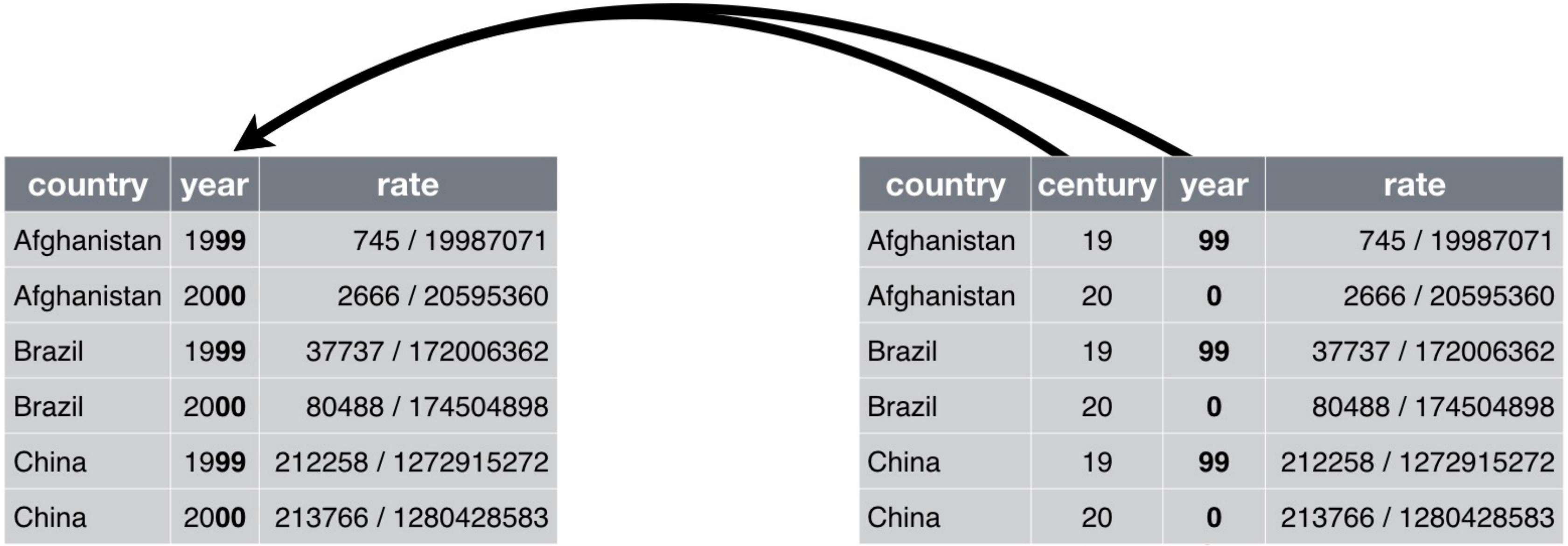
| country | year | rate |
|---|---|---|
| Afghanistan | 1999 | **745** / 19987071 |
| Afghanistan | 2000 | **2666** / 20595360 |
| Brazil | 1999 | **37737** / 172006362 |
| Brazil | 2000 | **80488** / 174504898 |
| China | 1999 | **212258** / 1272915272 |
| China | 2000 | **213766** / 1280428583 |

table3

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

# unite()

- **Syntax:**
  - **col** = *name of new column* e.g. year
  - **…** = *columns you want to join* e.g. century, year
  - **sep** = *delimiter to put between columns* (use "" for no separator)



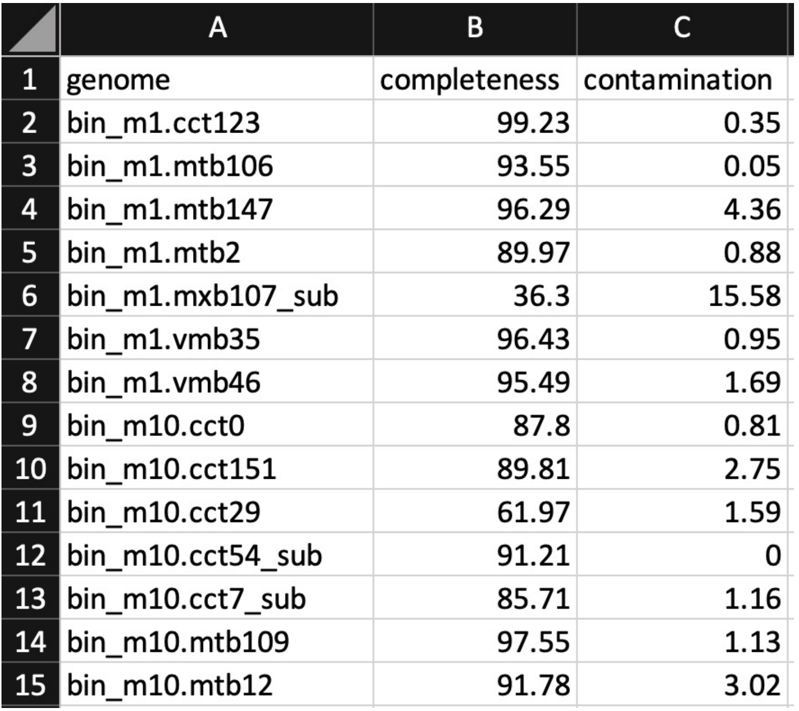
| country | year | rate |
|---|---|---|
| Afghanistan | 19**99** | 745 / 19987071 |
| Afghanistan | 20**00** | 2666 / 20595360 |
| Brazil | 19**99** | 37737 / 172006362 |
| Brazil | 20**00** | 80488 / 174504898 |
| China | 19**99** | 212258 / 1272915272 |
| China | 20**00** | 213766 / 1280428583 |

| country | century | year | rate |
|---|---|---|---|
| Afghanistan | 19 | **99** | 745 / 19987071 |
| Afghanistan | 20 | **0** | 2666 / 20595360 |
| Brazil | 19 | **99** | 37737 / 172006362 |
| Brazil | 20 | **0** | 80488 / 174504898 |
| China | 19 | **99** | 212258 / 1272915272 |
| China | 20 | **0** | 213766 / 1280428583 |

table6

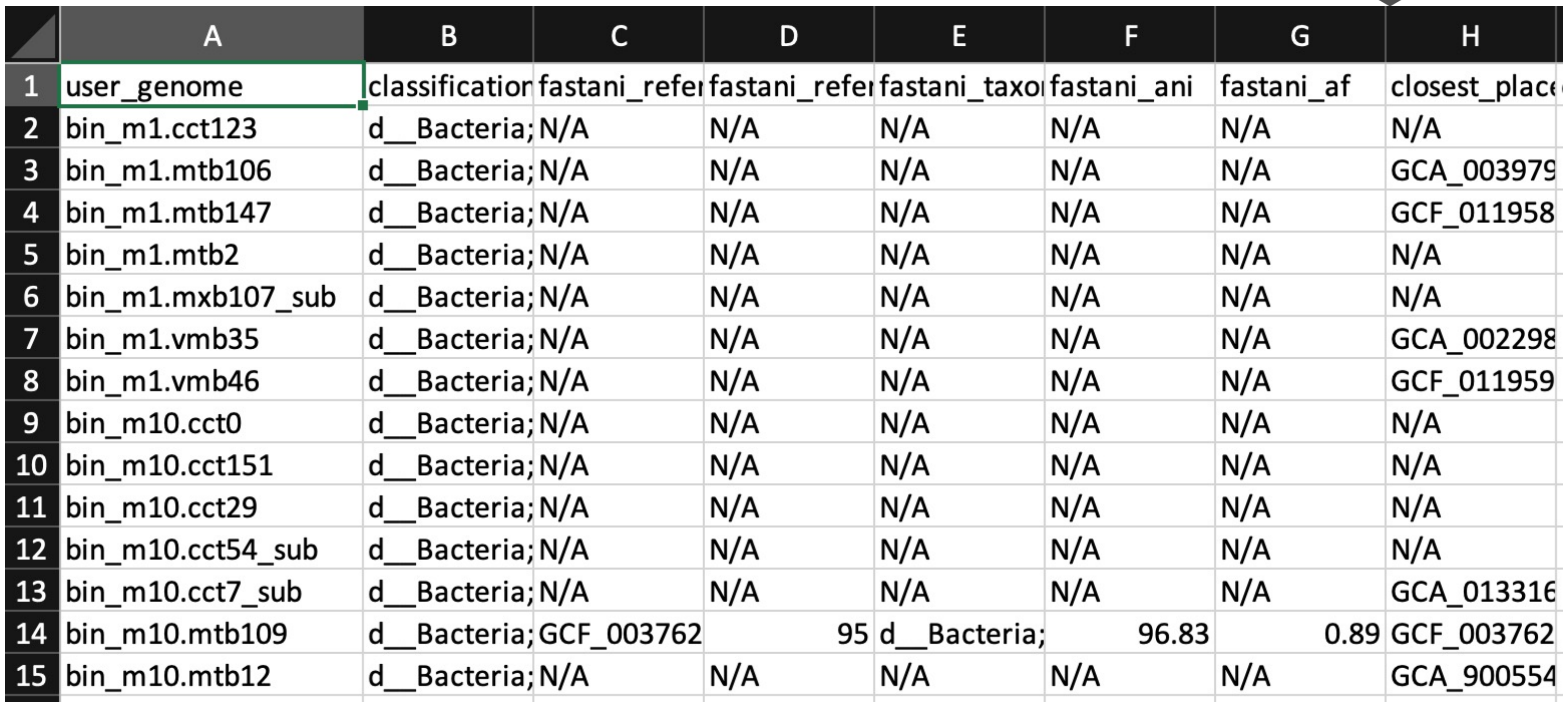
# 3. Joining tables

# The problem



| | A | B | C |
|---|---|---|---|
| 1 | genome | completeness | contamination |
| 2 | bin_m1.cct123 | 99.23 | 0.35 |
| 3 | bin_m1.mtb106 | 93.55 | 0.05 |
| 4 | bin_m1.mtb147 | 96.29 | 4.36 |
| 5 | bin_m1.mtb2 | 89.97 | 0.88 |
| 6 | bin_m1.mxb107_sub | 36.3 | 15.58 |
| 7 | bin_m1.vmb35 | 96.43 | 0.95 |
| 8 | bin_m1.vmb46 | 95.49 | 1.69 |
| 9 | bin_m10.cct0 | 87.8 | 0.81 |
| 10 | bin_m10.cct151 | 89.81 | 2.75 |
| 11 | bin_m10.cct29 | 61.97 | 1.59 |
| 12 | bin_m10.cct54_sub | 91.21 | 0 |
| 13 | bin_m10.cct7_sub | 85.71 | 1.16 |
| 14 | bin_m10.mtb109 | 97.55 | 1.13 |
| 15 | bin_m10.mtb12 | 91.78 | 3.02 |

+



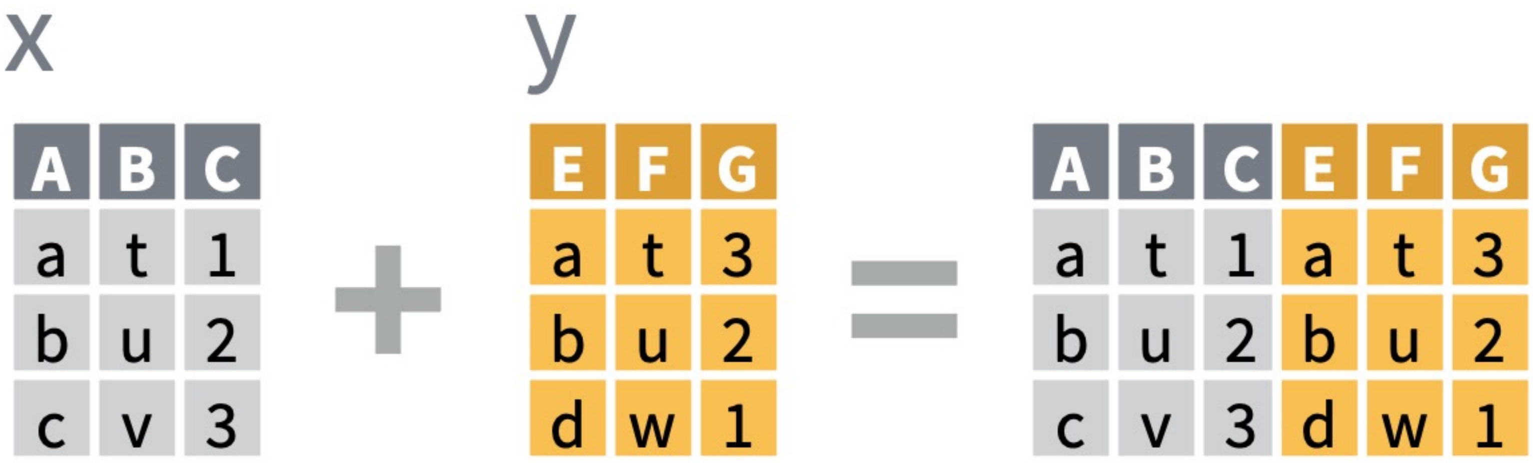
| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | user_genome | classificatior | fastani_refer | fastani_refer | fastani_taxo | fastani_ani | fastani_af | closest_place |
| 2 | bin_m1.cct123 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | N/A |
| 3 | bin_m1.mtb106 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | GCA_003979 |
| 4 | bin_m1.mtb147 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | GCF_011958 |
| 5 | bin_m1.mtb2 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | N/A |
| 6 | bin_m1.mxb107_sub | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | N/A |
| 7 | bin_m1.vmb35 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | GCA_002298 |
| 8 | bin_m1.vmb46 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | GCF_011959 |
| 9 | bin_m10.cct0 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | N/A |
| 10 | bin_m10.cct151 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | N/A |
| 11 | bin_m10.cct29 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | N/A |
| 12 | bin_m10.cct54_sub | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | N/A |
| 13 | bin_m10.cct7_sub | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | GCA_013316 |
| 14 | bin_m10.mtb109 | d__Bacteria; | GCF_003762 | 95 | d__Bacteria; | 96.83 | 0.89 | GCF_003762 |
| 15 | bin_m10.mtb12 | d__Bacteria; | N/A | N/A | N/A | N/A | N/A | GCA_900554 |

- What if they are not ordered in the same way?
- What if tables do not have the same number of rows?

# The simplest, but non-ideal way

- Simplest way: **bind_cols()**

- **N.B.** does not match tables by ID, so you have to ensure that both tables are sorted correctly!



x

| A | B | C |
|---|---|---|
| a | t | 1 |
| b | u | 2 |
| c | v | 3 |

+

y

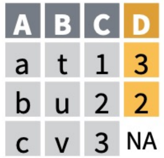
| E | F | G |
|---|---|---|
| a | t | 3 |
| b | u | 2 |
| d | w | 1 |

=

| A | B | C | E | F | G |
|---|---|---|---|---|---|
| a | t | 1 | a | t | 3 |
| b | u | 2 | b | u | 2 |
| c | v | 3 | d | w | 1 |

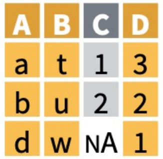
# Relational joining of tables

- We can do this a bit smarter by matching values in the rows between two tables (relational)
- E.g. say that both tables have the same grouping variable:
  - sample_id
  - genome_name
  - etc.



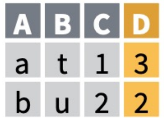
| A | B | C | D |
|---|---|---|---|
| a | t | 1 | 3 |
| b | u | 2 | 2 |
| c | v | 3 | NA |

**left_join()**



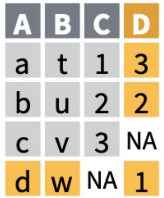
| A | B | C | D |
|---|---|---|---|
| a | t | 1 | 3 |
| b | u | 2 | 2 |
| d | w | NA | 1 |

**right_join()**



| A | B | C | D |
|---|---|---|---|
| a | t | 1 | 3 |
| b | u | 2 | 2 |

**inner_join()**



| A | B | C | D |
|---|---|---|---|
| a | t | 1 | 3 |
| b | u | 2 | 2 |
| c | v | 3 | NA |
| d | w | NA | 1 |

**full_join()**

# Relational joining of tables

Example: **left_join(x** = dataframe1,

**y** = dataframe2,

**by** = ”col name” OR c(“col name1”,“colname2”),

**)**

- If both tables have the same column name for grouping, **by** = “column name”

- If both tables have different names (e.g. x = grouping_var_x & y = grouping_var_y), then use a vector: **by** = c(“grouping_var_x”,“grouping_var_y”)

- If you have multiple tables, you can pipe %>% multiple ***_join()** together!

# Exercise time!

- See the exercises in "Exercises.R"

# Summary

- Tidy data is consistent, easier to learn to manipulate, and faster
- Dplyr has some really nice tools for manipulating tables to g
  - pivot_longer() and pivot_wider()
  - spread() and gather()
- Joining tables in a relational fashion saves time, and is less prone to error!