Developing and using a set of core tools in R

Developing toolsets in the context of complex and tidy data structures

Brooke Anderson and Rachel Severson

What we'll talk about

1. Data structures in R
2. The tidyverse approach
3. Some examples of our favorite tools

Data structures

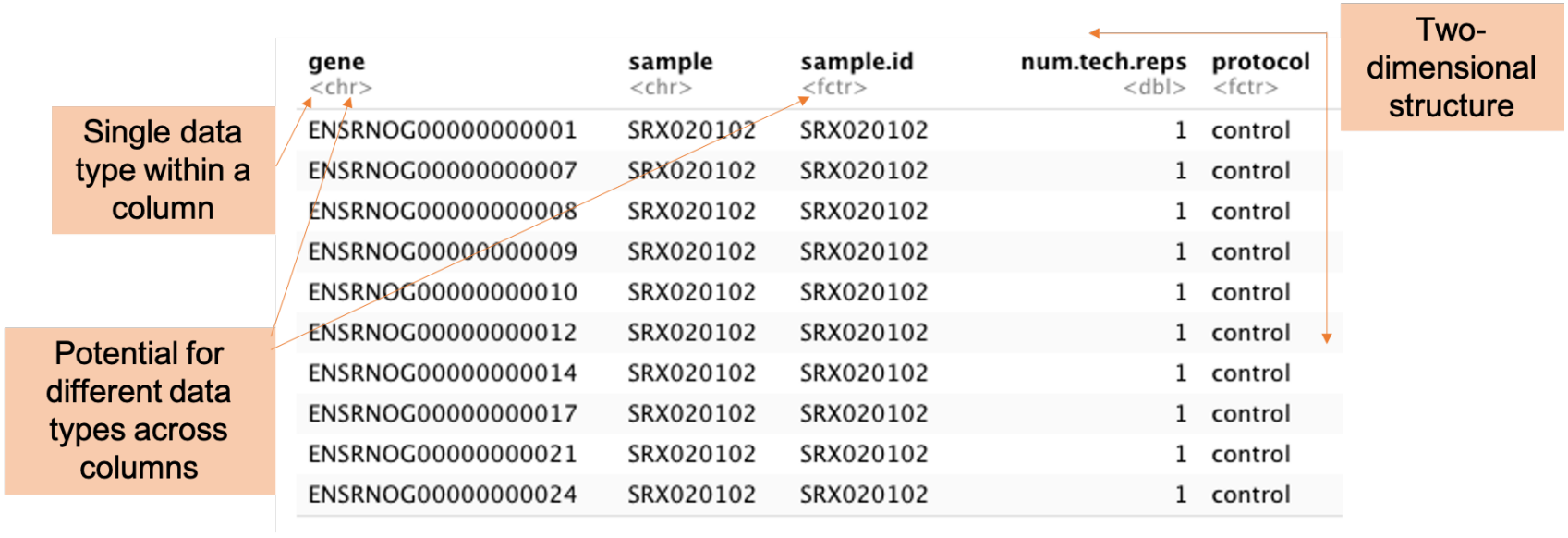
Where do you keep your data?

To work with data in R, you typically read it into memory.

Data structures help define the format in which you store your data.

Types of data structures in R: Simple

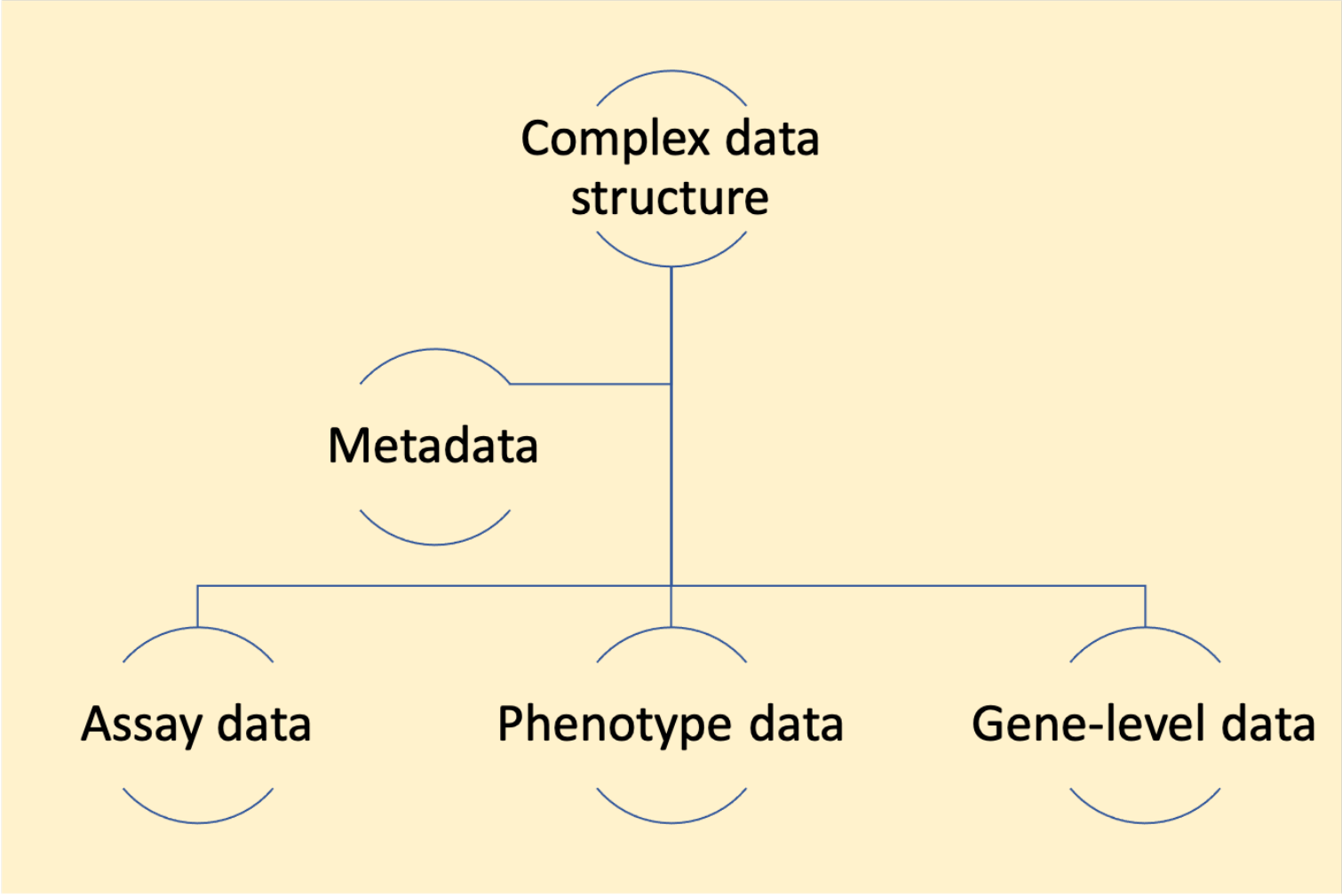
One simple data structure in R is the **dataframe**. This is the structure used extensively in the **tidyverse approach**.



gene <chr>	sample <chr>	sample.id <fctr>	num.tech.reps <dbl>	protocol <fctr>
ENSRNOG00000000001	SRX020102	SRX020102	1	control
ENSRNOG00000000007	SRX020102	SRX020102	1	control
ENSRNOG00000000008	SRX020102	SRX020102	1	control
ENSRNOG00000000009	SRX020102	SRX020102	1	control
ENSRNOG00000000010	SRX020102	SRX020102	1	control
ENSRNOG00000000012	SRX020102	SRX020102	1	control
ENSRNOG00000000014	SRX020102	SRX020102	1	control
ENSRNOG00000000017	SRX020102	SRX020102	1	control
ENSRNOG00000000021	SRX020102	SRX020102	1	control
ENSRNOG00000000024	SRX020102	SRX020102	1	control

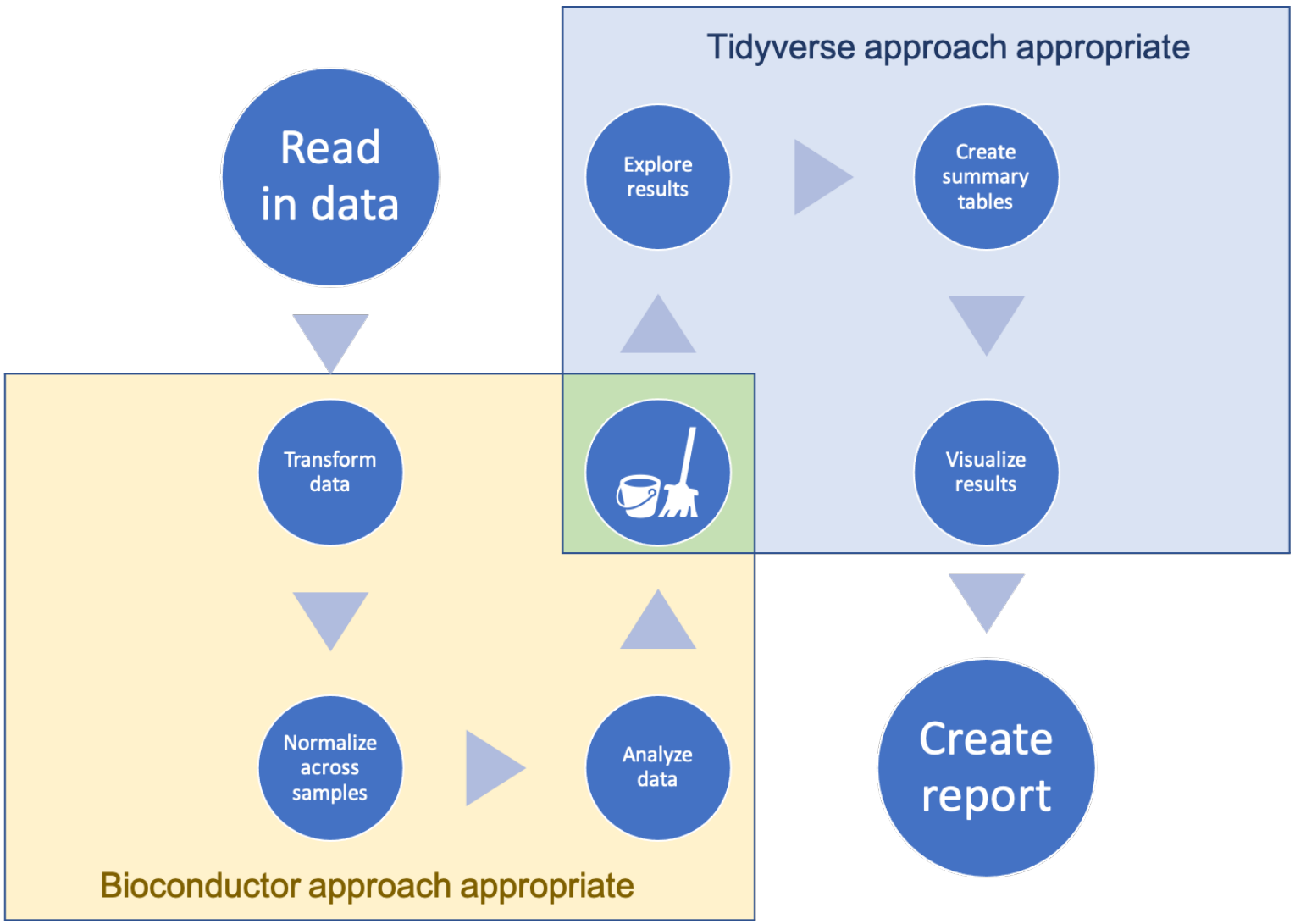
Types of data structures in R: Complex

More complex data structures in R are all, at heart based on lists. This format allows each object to collect different pieces of data, with different types and dimensions.



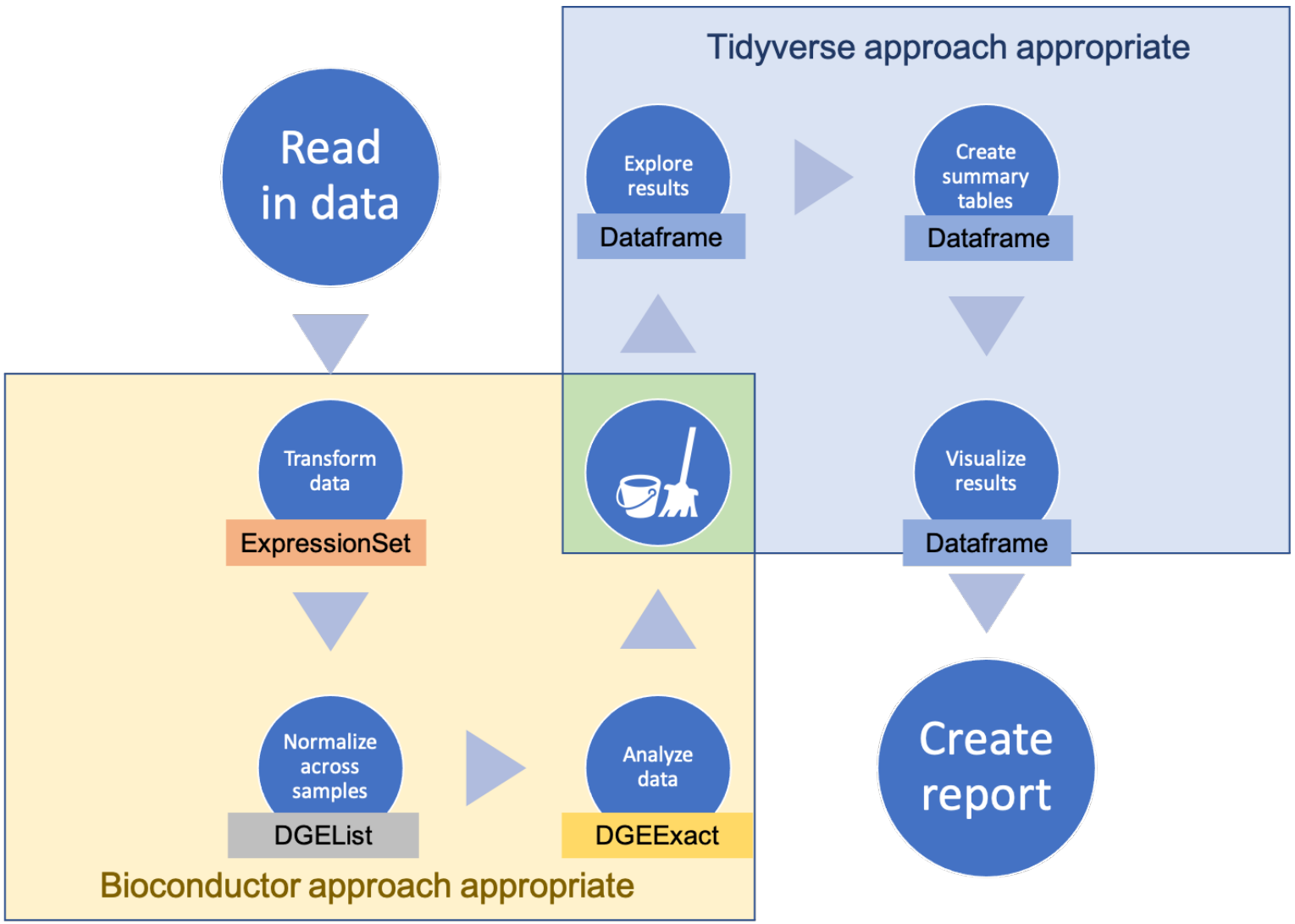
Data structures across a workflow

Work with research data will typically require a series of steps for pre-processing, analysis, exploration, and visualization. Collectively, these form a **workflow** for the data.



Data structures across a workflow

You can move your data among different structures across a workflow, including from more complex data to simpler data structures.



Tidyverse approach

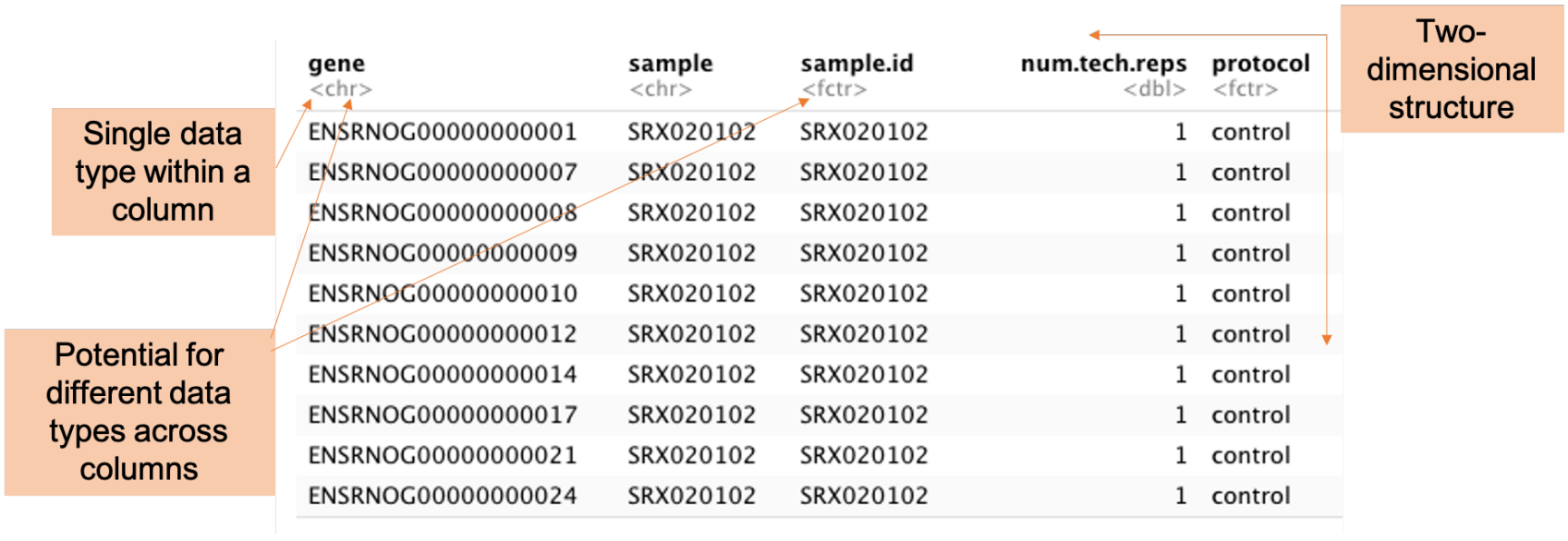
Tidyverse

Tidyverse is an *opinionated* collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Source: tidyverse.org

Tidyverse data structure

The key data structure in the tidyverse approach is the **dataframe**:

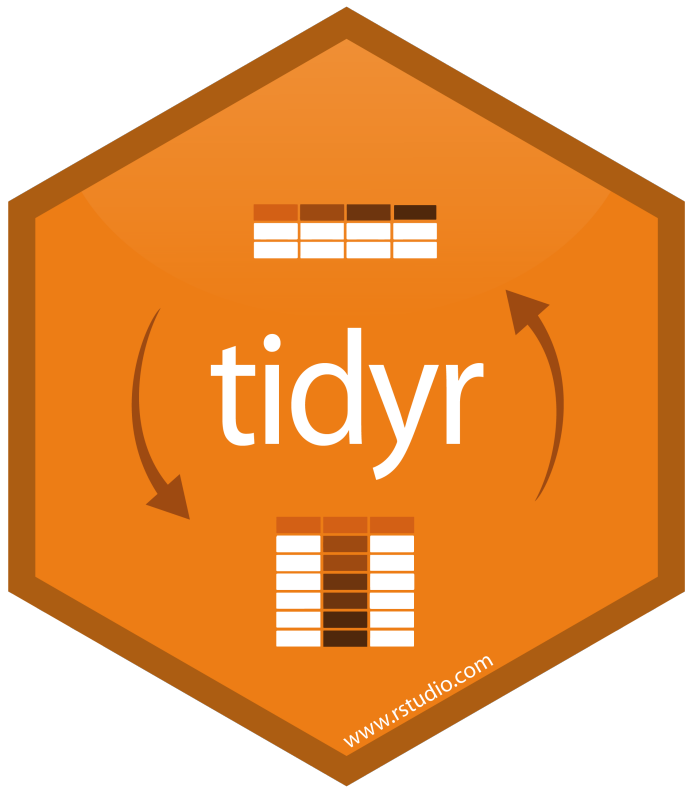


gene <chr>	sample <chr>	sample.id <fctr>	num.tech.reps <dbl>	protocol <fctr>
ENSRNOG00000000001	SRX020102	SRX020102	1	control
ENSRNOG00000000007	SRX020102	SRX020102	1	control
ENSRNOG00000000008	SRX020102	SRX020102	1	control
ENSRNOG00000000009	SRX020102	SRX020102	1	control
ENSRNOG00000000010	SRX020102	SRX020102	1	control
ENSRNOG00000000012	SRX020102	SRX020102	1	control
ENSRNOG00000000014	SRX020102	SRX020102	1	control
ENSRNOG00000000017	SRX020102	SRX020102	1	control
ENSRNOG00000000021	SRX020102	SRX020102	1	control
ENSRNOG00000000024	SRX020102	SRX020102	1	control

Tidy data

1. Each variable is a column; each column is a variable.
2. Each observation is a row; each row is an observation.
3. Each value is a cell; each cell is a single value.

Messy data is any other arrangement of the data.



Source: Tidy data vignette

Tidy data

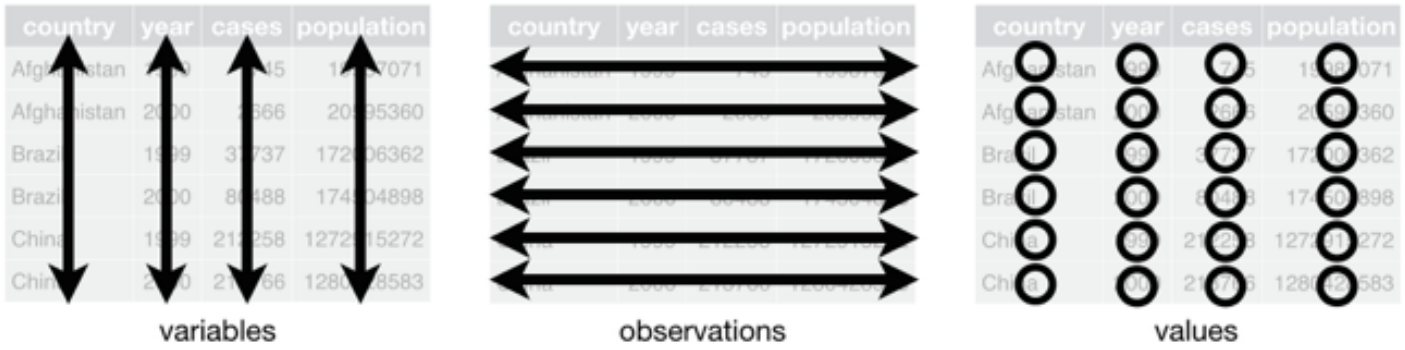


Figure 5.1: The following three rules make a dataset tidy: variables are columns, observations are rows, and values are cells.

Source: R for Data Science

Tidy data

> "Happy families are all alike; every unhappy family is unhappy in its own way."
> — Leo Tolstoy

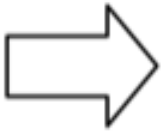
> "Tidy datasets are all alike, but every messy dataset is messy in its own way."
> — Hadley Wickham

Source: R for Data Science

Tidy data

- Data in column names

id	bp1	bp2
A	100	120
B	140	115
C	120	125



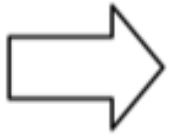
id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

Source: R for Data Science

Tidy data

- Many variables in column names

id	x_1	y_2
A	1	2
B	3	4
C	5	6



id	name	number	value
A	x	1	1
A	y	2	2
B	x	1	3
B	y	2	4
C	x	1	5
C	y	2	6

Source: R for Data Science

Piping

Linking together modular tidyverse verbs in a pipeline using the **pipe** symbol can allow you to write human-readable, sentence-like code.

Since most tidyverse functions take a data frame as their first argument, piping is an alternative to nesting or reassigning at each step.

Piping example: storms dataset

```
library(dplyr)

head(storms, 3)
```

```
## # A tibble: 3 x 13
##   name   year month   day  hour   lat  long status
##   <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <fct>
## 1 Amy    1975     6    27     0  27.5   -79 tropical de
## 2 Amy    1975     6    27     6  28.5   -79 tropical de
## 3 Amy    1975     6    27    12  29.5   -79 tropical de
## # i 2 more variables: tropicalstorm_force_diameter <int>
## #   hurricane_force_diameter <int>
```

Nesting

```
count(
  summarise(
    group_by(
      filter(storms, year == 2022), name),
    start_month = min(month)),
  start_month, name = "n_storms")
```

```
## # A tibble: 6 x 2
##   start_month n_storms
##         <dbl>    <int>
## 1           6        2
## 2           7        1
## 3           8        1
## 4           9        6
## 5          10        5
## 6          11        1
```

Reassignment

```
storms2 <- filter(storms, year == 2022)
storms3 <- group_by(storms2, name)
storms4 <- summarise(storms3, start_month = min(month))
count(storms4, start_month, name = "n_storms")
```

```
## # A tibble: 6 x 2
##   start_month n_storms
##         <dbl>    <int>
## 1           6        2
## 2           7        1
## 3           8        1
## 4           9        6
## 5          10        5
## 6          11        1
```

Piping

```
storms %>%
  filter(year == 2022) %>%
  group_by(name) %>%
  summarise(start_month = min(month)) %>%
  count(start_month, name = "n_storms")
```

```
## # A tibble: 6 x 2
##   start_month n_storms
##         <dbl>    <int>
## 1           6        2
## 2           7        1
## 3           8        1
## 4           9        6
## 5          10        5
## 6          11        1
```

Tidyverse approach

The tidyverse approach is built on the use of a common structure for storing data—almost all functions take data in this structure and almost all return data in this structure.

In other words, it is built on the idea of a **common interface** across all its functions.

Advantage of the tidyverse approach

This is similar to Legos: there are set dimensions for all blocks, so they can easily snap together in any order:

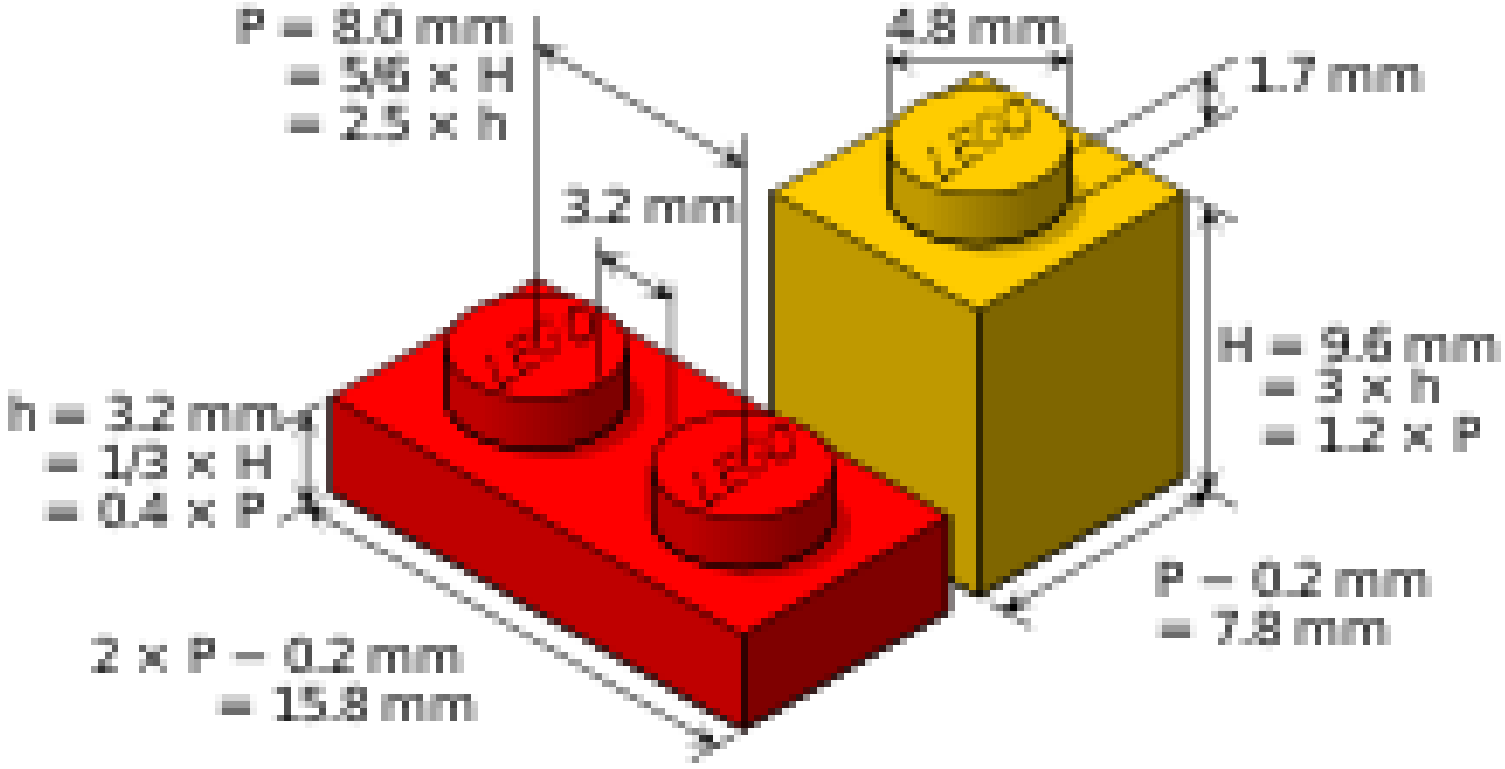


Image source:
https://en.wikipedia.org/wiki/Lego#/media/File:Lego_dimensions.svg

Advantage of the tidyverse approach

The common interface idea turns out to be very powerful. It allows you to not have to rely on large functions that do a lot all at once.

Instead, this idea allows for lots of **small functions** that each do **one small thing**, but that can be chained together in lots of different configurations to do very flexible and powerful things.

Advantage of the tidyverse approach

It is hard to overemphasize how powerful this approach is. Simple tools that can be connected together in different ways can be used to create very complex things:

Image source: *Architectural Digest*

Advantage of the tidyverse approach

This allows you to learn a single set of tools—most of which can be learned in a few months.

These work across all your data, as long as it's in a **tidy dataframe structure** while you're working on it.

By contrast, if you use a variety of data structures, you often have to learn different tools (functions) for each data structure, rather than being able to use a single set of tools for all your data.