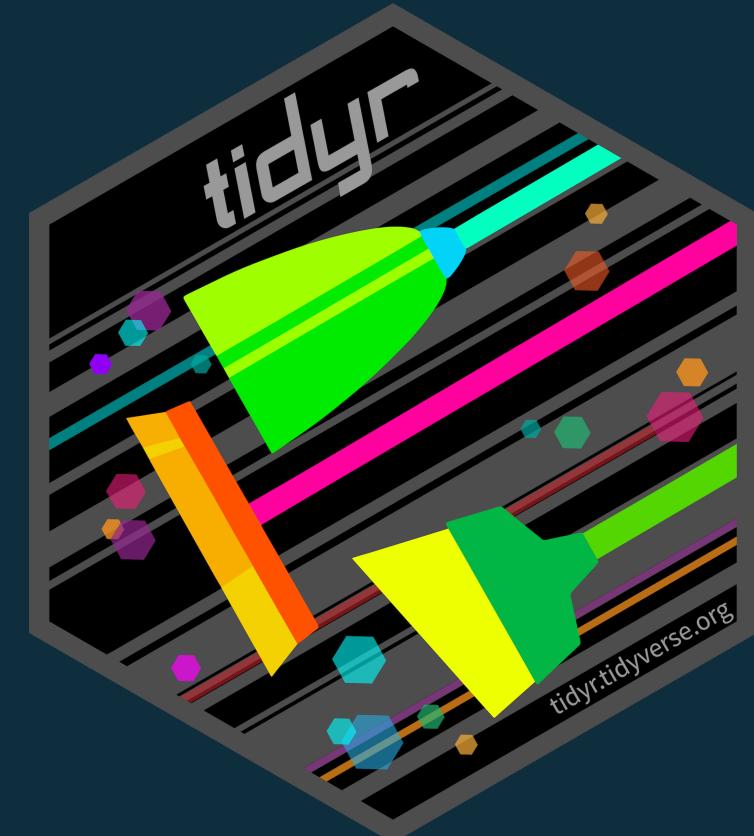


Tidy your data using `tidyverse`

Introduction to R - Day 2

Instructor: Selina Baldauf

Freie Universität Berlin - Theoretical Ecology



2021-08-01 (updated: 2023-02-26)

What is tidy data?

What is tidy data?

“**TIDY DATA** is a standard way of mapping the meaning of a dataset to its structure.”

-HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

each column a variable

id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

each row an observation

Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

Illustration from the [Openscapes](#) blog *Tidy Data for reproducibility, efficiency, and collaboration* by Julia Lowndes and Allison Horst

What is tidy data?

Let's look at some examples

Tidy

id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

Non-tidy

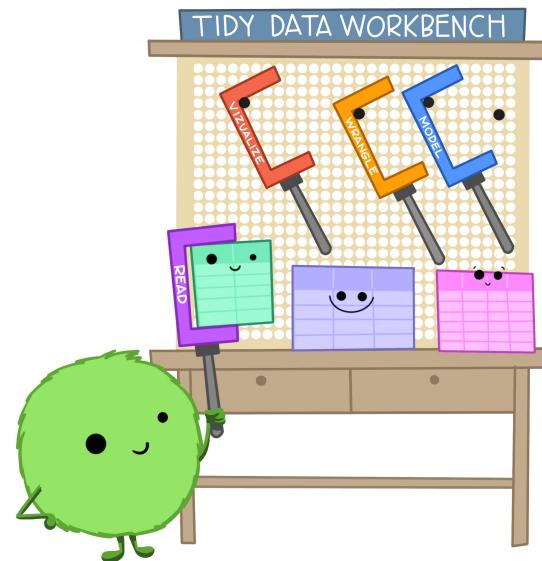
floof	max	cat	donut	merlin	panda
gray	black	orange	gray	black	calico
<hr/>					
gray	black	orange	calico		
floof	max	cat	panda		
donut	merlin				

Sometimes **raw data** is non-tidy because its structure is optimized for data entry or viewing rather than analysis.

Why tidy data?

The main advantages of **tidy** data is that the **tidyverse** packages are built to work with it.

When working with tidy data,
we can use the **same tools** in
similar ways for different datasets...



...but working with untidy data often means
reinventing the wheel with **one-time**
approaches that are **hard to iterate or reuse**.

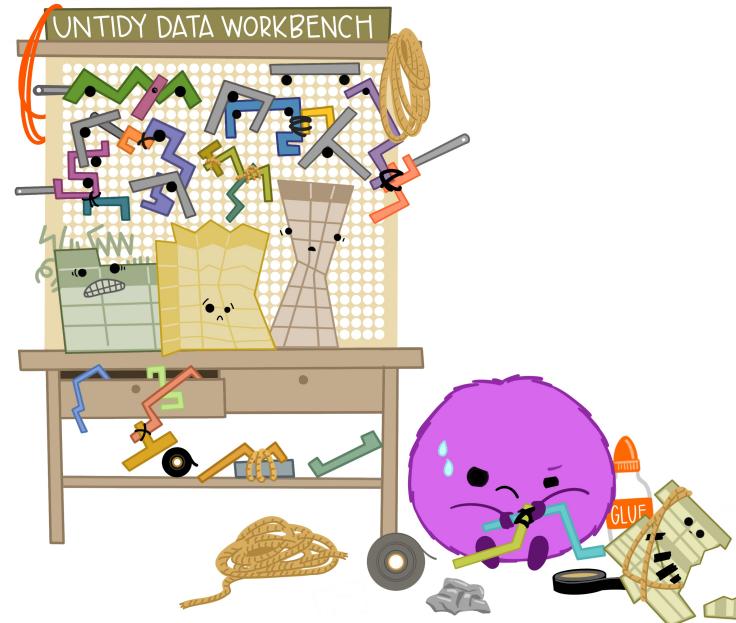


Illustration from the [Openscapes blog](#) *Tidy Data for reproducibility, efficiency, and collaboration* by Julia Lowndes and Allison Horst

Example

Let's go back to the city data set from earlier:

```
cities_tbl
```

```
## # A tibble: 10 × 4
##   city           population area_km2 country
##   <chr>          <dbl>     <dbl> <chr>
## 1 Istanbul      15100000     2576 Turkey
## 2 Moscow        12500000     2561 Russia
## 3 London         9000000     1572 UK
## 4 Saint Petersburg 5400000    1439 Russia
## 5 Berlin         3800000      891 Germany
## 6 Madrid         3200000      604 Spain
## 7 Kyiv           3000000      839 Ukraine
## 8 Rome            2800000     1285 Italy
## 9 Bucharest      2200000      228 Romania
## 10 Paris          2100000      105 France
```

This already looks pretty tidy.

Same data different format

```
cities_untidy
```

```
## # A tibble: 2 × 11
##   type      Turke...¹ Russi...² UK_Lo...³ Russi...⁴ Germa...⁵ Spain...⁶ Ukrai...⁷ Italy...⁸ Roman...⁹ Franc...˟
##   <chr>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 populat... 1.51e7  1.25e7  9000000  5400000  3800000  3200000  3000000  2800000  2200000
## 2 area_km2   2.58e3  2.56e3   1572     1439     891     604     839     1285     228     105
## # ... with abbreviated variable names ¹Turkey_Istanbul, ²Russia_Moscow, ³UK_London,
## #   ⁴Russia_Saint_Petersburg, ⁵Germany_Berlin, ⁶Spain_Madrid, ⁷Ukraine_Kyiv,
## #   ⁸Italy_Rome, ⁹Romania_Bucharest, ˟France_Paris
```

What's not tidy here?

- Each row has multiple observations
- At the same time, each observation is split across multiple rows
- Country and city variable are split into multiple columns
- Country and city variable values are united to one value

Let's tidy this data using functions from the `tidyverse` package!

pivot_longer()

One variable split into multiple columns can be solved with pivot_longer

```
## # A tibble: 2 × 11
##   type      Turkey_Istanbul Russi…¹ UK_Lo…³ Russi…⁴ Germa…⁵ Spain…⁶ Ukrai…⁷ Italy…⁸ Roman…⁹ Franc…×
##   <chr>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 populat… 1.51e7    1.25e7    9000000  5400000  3800000  3200000  3000000  2800000
## 2 area_km²  2.58e3    2.56e3    1572     1439     891      604      839      1285
## # ... with abbreviated variable names ¹Turkey_Istanbul, ²Russia_Moscow, ³UK_London,
## #   ⁴Russia_Saint Petersburg, ⁵Germany_Berlin, ⁶Spain_Madrid, ⁷Ukraine_Kyiv,
## #   ⁸Italy_Rome, ⁹Romania_Bucharest, ¹France_Paris
```

```
step1 <- pivot_longer(
  cities_untidy,                      # the tibble
  cols = Turkey_Istanbul:France_Paris, # the columns to pivot from:to
  names_to = "location",               # name of the new column
  values_to = "value")                # name of the value column
```

```
## # A tibble: 20 × 3
##   type      location          value
##   <chr>      <chr>           <dbl>
## 1 population Turkey_Istanbul 15100000
## 2 population Russia_Moscow  12500000
## 3 population UK_London      9000000
## 4 population Russia_Saint Petersburg 5400000
## # ... with 16 more rows
```

pivot_longer()

One variable split into multiple columns can be solved with pivot_longer

Another way to select the columns to pivot:

```
step1 <- pivot_longer(  
  cities_untidy,           # the tibble  
  cols = !type,            # All columns except type  
  names_to = "location",   # name of the new column  
  values_to = "value")     # name of the value column
```

```
## # A tibble: 20 × 3  
##   type      location       value  
##   <chr>     <chr>        <dbl>  
## 1 population Turkey_Istanbul 15100000  
## 2 population Russia_Moscow  12500000  
## 3 population UK_London    9000000  
## 4 population Russia_Saint Petersburg 5400000  
## # ... with 16 more rows
```

separate()

Multiple variable values that are united into one can be separated using `separate`

```
## # A tibble: 20 × 3
##   type      location       value
##   <chr>     <chr>        <dbl>
## 1 population Turkey_Istanbul 15100000
## 2 population Russia_Moscow   12500000
## # ... with 18 more rows
```

```
step2 <- separate(
  step1,                               # the tibble
  location,                            # the column to separate
  sep = "_",                           # the separator
  into = c("country", "city")) # names of new columns
```

```
## # A tibble: 20 × 4
##   type      country city       value
##   <chr>     <chr>   <chr>     <dbl>
## 1 population Turkey  Istanbul  15100000
## 2 population Russia  Moscow   12500000
## # ... with 18 more rows
```

The opposite function exists as well and is called `unite`. Check out `?unite` for details.

`pivot_wider()`

One observation split into multiple rows can solved with `pivot_wider`

```
## # A tibble: 20 × 4
##   type      country city        value
##   <chr>     <chr>   <chr>      <dbl>
## 1 population Turkey  Istanbul  15100000
## 2 population Russia  Moscow   12500000
## # ... with 18 more rows
```

```
step3 <- pivot_wider(
  step2,                               # the tibble
  names_from = type,                   # the variables
  values_from = value)                # the values
```

```
## # A tibble: 10 × 4
##   country city        population area_km2
##   <chr>   <chr>      <dbl>       <dbl>
## 1 Turkey  Istanbul  15100000     2576
## 2 Russia  Moscow   12500000     2561
## 3 UK      London    9000000      1572
## 4 Russia  Saint Petersburg 5400000     1439
## 5 Germany Berlin   3800000      891
## # ... with 5 more rows
```

All steps in 1

We can also use a pipe to do all these steps in one:

```
cities_tidy <- cities_untidy %>%
  pivot_longer(Turkey_Istanbul:France_Paris,
    names_to = "location",
    values_to = "values") %>%
  separate(location,
    sep = "-",
    into = c("country", "city")) %>%
  pivot_wider(names_from = type,
    values_from = values)
```

Now you

Task 3: Tidying data (30 min)

Find the task description [here](#)