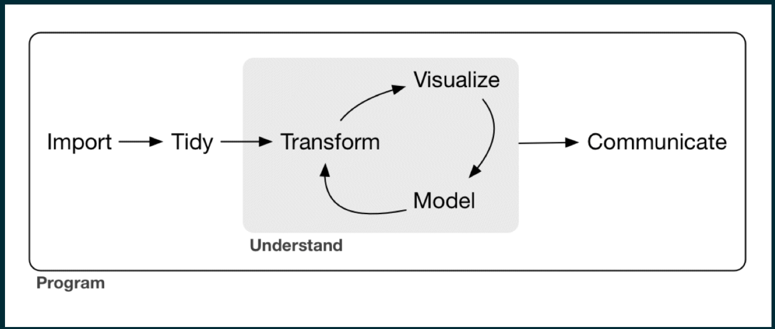
INTRODUCTION TO DATA SCIENCE WITH R

Session 6b: Tidying data

Ina Bornkessel-Schlesewsky

January 20, 2023

RECALL



- **import** data (into R)
- **tidy** data
  - bring it into a consistent format that can be used for multiple purposes (each column = variable; each row = observation)
  - lets you focus on understanding the data rather than which format you need
- **transform** data
  - e.g. focus on observations of interest (such as those from a particular location), create new variables (such as speed from distance and time), compute summary statistics
- **visualise** data
  - essential for understanding
- **model** data
  - use (statistical) models to answer your questions about the data
- **communicate** insights

DATA STRUCTURE

- Most datasets are made up of rows and columns
- Many ways to structure the same data (see figure)
- Datasets are collections of values (either numbers or strings); these belong to a *variable* and an *observation*
- “A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units.”
- “An observation contains all values measured on the same unit (like a person, or a day, or a race) across attributes.” (Wickham, 2014)

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

Table 1: Typical presentation dataset.

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

Table 2: The same data as in Table 1 but structured differently.

figure from Wickham (2014)

Wickham, H. (2014). Tidy Data. Journal of Statistical Software, 59(10), 1 - 23. doi:http://dx.doi.org/10.18637/jss.v059.i10

DATA STRUCTURE

- The same dataset but with observations in rows and variables in columns

person	treatment	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

figure from Wickham (2014)

TIDY DATA

CHARACTERISTICS OF TIDY DATA:*

- Each variable has its own column
- Each observation has its own row
- Each value has its own cell

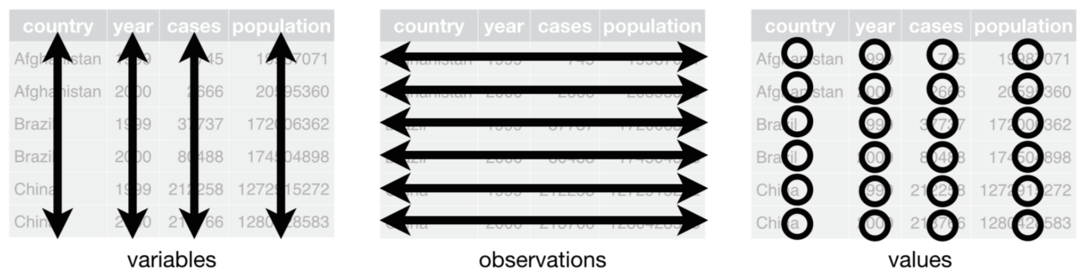


Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

figure from R4DS

DIFFERENT DATA FORMATS

- The three data frames on the next slide show example data provided in the `tidyr` package, which is part of the `tidyverse`.

> "all display the number of TB cases documented by the World Health Organization in Afghanistan, Brazil, and China between 1999 and 2000"

(from `?tidyr::table1`)

- Which of these tables is tidy?

If you want to inspect the data yourself, you can access them via `table1`, `table2` and `table3`.

DIFFERENT DATA FORMATS

```
[1] "table1"
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

```
[1] "table2"
# A tibble: 6 × 4
  country      year type            count
  <chr>       <int> <chr>           <int>
1 Afghanistan  1999 cases             745
2 Afghanistan  1999 population   19987071
3 Afghanistan  2000 cases            2666
4 Afghanistan  2000 population   20595360
5 Brazil       1999 cases           37737
6 Brazil       1999 population  172006362
```

```
[1] "table3"
# A tibble: 6 × 3
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

DIFFERENT DATA FORMATS

- If you said `table1`, you were right!

- Another possibility is for data to be spread across two data frames
- See the next slide for an example (`table4a` and `table4b` from `tidyr`)

DIFFERENT DATA FORMATS

```
[1] "table4a"
```

```
# A tibble: 3 × 3
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

```
[1] "table4b"
```

```
# A tibble: 3 × 3
  country         `1999`     `2000`
* <chr>            <int>      <int>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

ADVANTAGES TO WORKING WITH TIDY DATA

1. Having one consistent format for data makes it easier to learn the tools required for analysis (which can have a certain uniformity). The `tidyverse` packages, for example, are designed to work with tidy data (who would have thought! 🤣)
2. It is advantageous for variables to be placed in columns because this caters to R's vectorised nature. (Most R-functions work with vectors of values.)

DATA ARE OFTEN UNTIDY

COMMON PROBLEMS

1. Variables are spread across multiple columns
2. Observations are spread across multiple rows

THE SOLUTION

- functions `pivot_longer()` and `pivot_wider()` in `tidyr`!

Note: this doesn't mean that non-tidy data are "bad". There can be many reasons for why a dataset is in a non-tidy format, e.g. ease of data entry if this is being done manually.

PIVOT TO LONGER

Common problem: column names are values of a variable rather than variables

Example: `table4a`

```
# A tibble: 3 × 3
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Solution: *pivot* these columns to new variables, rendering the dataset longer

We need:

- the columns with values as names (`1999` and `2000`)
- the name of the variable to move the column names to (`year`)
- the name of the variable to move the column values to (`cases`)

PIVOT TO LONGER

```
table4a  |>
  pivot_longer(c(`1999`,`2000`), names_to = "year", values_to = "cases")
```

```
# A tibble: 6 × 3
  country     year  cases
  <chr>       <chr> <int>
1 Afghanistan 1999    745
2 Afghanistan 2000   2666
3 Brazil      1999  37737
4 Brazil      2000  80488
5 China       1999 212258
6 China       2000 213766
```

PIVOT TO LONGER

Figure 12.2: Pivoting `table4` into a longer, tidy form.

from R4DS

Exercise: try doing the same thing with `table4b`!

EXCURSUS 1: JOINING TABLES

We can easily join the longer versions of `table4a` and `table4b` using `left_join()` (more on joining operations later):

```
tidy4a <- table4a %>%
  pivot_longer(c(`1999`, `2000`),
               names_to = "year",
               values_to = "cases")

tidy4b <- table4b %>%
  pivot_longer(c(`1999`, `2000`),
               names_to = "year",
               values_to = "populatior

left_join(tidy4a, tidy4b)
```

```
# A tibble: 6 × 4
  country     year   cases population
  <chr>       <chr>  <int>      <int>
1 Afghanistan 1999     745   19987071
2 Afghanistan 2000    2666   20595360
3 Brazil      1999   37737  172006362
4 Brazil      2000   80488  174504898
5 China       1999  212258 1272915272
6 China       2000  213766 1280428583
```

EXCURSUS 2: ADDITIONAL TOOLS FOR CLEANING DATA

- the {janitor} package includes a number of useful functions for cleaning data
- one of these is `clean_names()`, which cleans up problematic variable names (e.g. names with spaces, starting with a digit etc.)

```
table4a |>
  janitor::clean_names()
```

```
# A tibble: 3 × 3
  country      x1999  x2000
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

PIVOT TO WIDER

- `pivot_wider()` is the counterpart of `pivot_longer()` which you need when observations are spread across multiple rows such as in `table2`
- here, the table needs to be made wider

```
# A tibble: 6 × 4
  country      year type            count
  <chr>       <int> <chr>           <int>
1 Afghanistan  1999 cases             745
2 Afghanistan  1999 population   19987071
3 Afghanistan  2000 cases            2666
4 Afghanistan  2000 population   20595360
5 Brazil       1999 cases           37737
6 Brazil       1999 population 172006362
```

PIVOT TO WIDER

To tidy `table2` we need:

- the column to take variables from (`type`)
- the column to take values from (`count`)

```
table2 |>
  pivot_wider(names_from = type, values_from = count)
```

```
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

PIVOT TO WIDER

Figure 12.2: Pivoting `table4` into a longer, tidy form.

from R4DS

WHAT'S UP WITH TABLE3?

```
# A tibble: 6 × 3
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

MULTIPLE VARIABLES IN ONE COLUMN

- in `table3`, the `rate` column contains both cases and population
- to deal with this problem, we can use the `separate()` function
- it allows us to easily split a column according to a delimiting character (here, the "/")
- note how `separate` is clever enough to correctly guess the delimiting character – it looks for a non-alphanumeric character by default (to specify it manually, use `sep = "/"`)

```
table3 |>
  separate(rate, into = c("cases","population"))
```

```
# A tibble: 6 × 4
  country      year cases  population
  <chr>       <int> <chr>  <chr>
1 Afghanistan  1999 745    19987071
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

MULTIPLE VARIABLES IN ONE COLUMN

- by default, `separate` retains the original column type (character in this case)
- we can ask it to try to convert to a more suitable type using the `convert` parameter

```r
table3 |>
  separate(rate, into = c("cases","population"), convert = TRUE)
```

```
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

RESOURCES

- Chapter 6: Data tidying of R4DS
- Chpter 21: Joins of R4DS provides more information on joining data frames, which is beyond the scope of this course