

MY472 – Data for Data Scientists

Week 2: Tabular data

Thomas Robinson

<https://lse-my472.github.io/>

Course outline

1. Introduction
2. Tabular data
3. Data visualisation
4. Textual data
5. HTML, CSS, and scraping static websites
6. (Reading week)
7. XML, RSS, and scraping non-static website
8. Working with APIs
9. Creating and managing databases
10. Interacting with online databases
11. Cloud computing

Plan for today

- ▶ “Tidy data” and reshaping data in R
- ▶ Excursus: Some good practises for code in (research) projects
- ▶ Coding

Plan for today

- ▶ “Tidy data” and reshaping data in R
- ▶ Excursus: Some good practises for code in (research) projects
- ▶ Coding

Shapes of data

- ▶ One very common form of data is tabular
- ▶ Examples of other forms: Raw texts, key-value and array structures such as JSON files
- ▶ This week: How to organise and process tabular data (in R)
- ▶ Helpful to think about an ideal format of tabular data first

Comparing data structures

	tweet	tags	lists	list_descriptions
1	I think MY472 is the best course ever	LSE, MY472	asds_tweets, publicity	my thoughts, to share

```
{
  "tweet" : {
    "message" : "I think MY472 is the best course ever",
    "tags" : ["LSE", "MY472"],
    "lists" : [
      {
        "name" : "asds_tweets",
        "description" : "my thoughts"
      },
      {
        "name" : "publicity",
        "description" : "to share"
      }
    ]
  }
}
```

"Tidy" data (Hadley Wickham)

Three rules:

1. Each variable must have its own column
2. Each observation must have its own row
3. Each value must have its own cell

country	year	cases	population
Afghanistan	1999	765	15467071
Afghanistan	2000	766	20095360
Brazil	1999	30737	17206362
Brazil	2000	80488	174004898
China	1999	210258	1272015272
China	2000	210256	128000583

variables

country	year	cases	population
Afghanistan	1999	765	15467071
Afghanistan	2000	766	20095360
Brazil	1999	30737	17206362
Brazil	2000	80488	174004898
China	1999	210258	1272015272
China	2000	210256	128000583

observations

country	year	cases	population
Afghanistan	1999	765	15467071
Afghanistan	2000	766	20095360
Brazil	1999	30737	17206362
Brazil	2000	80488	174004898
China	1999	210258	1272015272
China	2000	210256	128000583

values

Section based on <https://r4ds.had.co.nz/tidy-data.html>

What can go wrong?

Datasets where columns represent values of a variable:

```
table4a
```

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

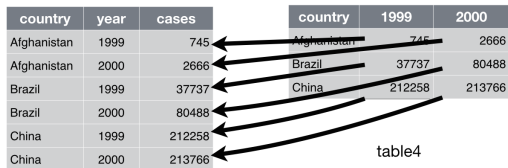

How to fix it?

We need to **pivot** those columns into a new pair of variables:

```
table4a %>%  
  pivot_longer(c('1999', '2000'), names_to = "year", values_to = "cases")  
#> # A tibble: 6 x 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
```

What is happening here?

We switched from **wide** to **long** format:



The diagram illustrates the transformation of data from a wide format to a long format. On the right is a wide table with columns for country, 1999, and 2000. On the left is a long table with columns for country, year, and cases. Arrows indicate the mapping of data from the wide table to the long table.

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

What else can go wrong?

Datasets where observations are scattered across multiple rows:

```
table2
```

```
#> # A tibble: 12 x 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
```

```
#> # ... with 6 more rows
```

How to fix it?

We need to **pivot** those rows into a new pair of columns:

```
table2 %>%  
  pivot_wider(names_from = type, values_from = count)  
#> # A tibble: 6 x 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
```

What is happening here?

We switched from **long** to **wide** format:

country	year	type	count
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

Storing tabular data

Common file formats for storing tabular data:

- ▶ Comma-separated values (.csv) – ubiquitous and simple
 - ▶ Each *line* is an observation
 - ▶ Each variable value is separated by a comma
- ▶ Application specific (proprietary) formats (.dta, .sav, etc.)
 - ▶ Can allow for richer representations including meta-data
 - ▶ More complex, and not necessarily human-readable

Often choice is dictated by the source (and size) of the data

- ▶ Packages like `haven` allow for reading in non-csv formats in R

Plan for today

- ▶ “Tidy data” and reshaping data in R
- ▶ Excursus: Some good practises for code in (research) projects
- ▶ Coding

- ▶ Before we continue with examples of processing tabular data in R, it is helpful to spend some time early in this course with a brief discussion of good coding practises
- ▶ Based on Nagler (1995) “Coding Style and Good Computing Practices” (PS) and Wilson *et al* (2017) “Good Enough Practices in Scientific Computing” (PLOS Comput Biol)

Good practices in scientific computing

Why care?

- ▶ Yourself
 - ▶ Much lower chance of unnoticed bugs
 - ▶ Future self will be grateful: "Yourself from 3 months ago doesn't answer emails"
 - ▶ More efficient research, avoid retracing own steps
- ▶ Others
 - ▶ Keep good records of what you did so that others can understand it
 - ▶ **Replication** is a key part of science

Summary of some good practices

1. Safe and efficient data management
2. Well organised and documented code
3. Organised collaboration
4. One project = one repository
5. Track changes
6. Manuscripts as part of the analysis

1. Data management

- ▶ Save raw data as originally generated
- ▶ Create the data you would like to see, e.g.
 - ▶ If possible, open and non-proprietary formats such as .csv
 - ▶ Informative variable names instead of V322
 - ▶ Informative file names that contain metadata: e.g. 05-alaska.csv instead of state5.csv
- ▶ Record all steps used to process data and store intermediate data files if computationally intensive (easier to rerun parts of a data analysis pipeline)
- ▶ Separate data manipulation from data analysis
- ▶ Prepare README with codebook of all variables
- ▶ Periodic backups (or Dropbox, Google Drive, etc.)
- ▶ Sanity checks: Summary statistics after data manipulation

2. Well organised and documented code

- ▶ Number scripts based on execution order
 - e.g. 01-clean-data.R, 02-recode-variables.R, 03-run-regression.R, 04-produce-figures.R...
- ▶ Write an explanatory note at the start of each script
 - Author, date of last update, purpose, inputs and outputs, other relevant notes
- ▶ Rules of thumb for modular code
 1. Any task you run more than once should be a function (with a meaningful name!)
 2. Many functions can be relatively short
 3. Can separate functions from execution (e.g. in functions.R file and then use `source(functions.R)` to load functions into current environment
- ▶ Try to keep it simple rather than too clever
- ▶ Add informative comments before blocks of code

3. Organised collaboration

- ▶ Create a README file with an overview of the project: Title, brief description, contact information, structure of folder
- ▶ Shared to-do list with tasks and deadlines
- ▶ Choose one person as corresponding author / point of contact / note taker
- ▶ Split code into multiple scripts to avoid simultaneous edits
- ▶ GitHub, ShareLatex, Overleaf, Google Docs, etc. to collaborate in writing of manuscript

4. One project = one repository

Logical and consistent folder structure:

- ▶ code or src for all scripts
- ▶ data for raw data
- ▶ temp for temporary data files
- ▶ output or results for final data files and tables
- ▶ figures or plots for figures produced by scripts
- ▶ manuscript for text of paper
- ▶ docs for any additional documentation

5 & 6. Track changes; producing manuscript

- ▶ Ideally: Use version control (e.g. Git/GitHub)
- ▶ Manual approach: Keep dates versions of code & manuscript, and a `changelog` file with list of changes
- ▶ Dropbox also has some basic version control built-in
- ▶ Avoid typos and copy&paste errors: Tables and figures can be produced in scripts and compiled directly into manuscript with \LaTeX

Examples

Replication materials for Pablo Barberá's 2014 *Political Analysis* paper:

- ▶ [Code on GitHub](#)
- ▶ [Code and Data](#)

John Myles White's [ProjectTemplate](#) R package.

Another example of replication materials, Thomas Leeper (2017):

- ▶ [Code and data](#)

Plan for today

- ▶ “Tidy data”, and reshaping data in R
- ▶ Excursus: Some good practises for code in (research) projects
- ▶ Coding

Coding

- ▶ 01-conditionals-loops-functions.Rmd
- ▶ 02-processing-data.Rmd