

MY472 – Week 2: The Shape of Data

Friedrich Geiecke

MY 472: Data for Data Scientists

5 October 2020

Course website: lse-my472.github.io

Course outline

1. Introduction to data
2. The shape of data
3. HTML and CSS
4. Using data from the Internet
5. Working with APIs
6. (Reading week)
7. Textual data
8. Data visualization
9. Creating and managing databases
10. Interacting with online databases
11. Cloud computing

Plan for today

- ▶ Administration and logistics
- ▶ Datasets, “tidy data”, and reshaping data in R
- ▶ Guided coding session
- ▶ Some good coding practices

First assignment

- ▶ Will be distributed on Monday 5 October, 2pm
- ▶ Due Friday 16 October, 2pm

Assessment criteria

- ▶ **70–100: Very Good to Excellent (Distinction)**
 - ▶ Perceptive, focused use of a good depth of material with a critical edge. Original ideas or structure of argument.
- ▶ **60–69: Good (Merit)**
 - ▶ Perceptive understanding of the issues plus a coherent well-read and stylish treatment though lacking originality.
- ▶ **50–59: Satisfactory (Pass)**
 - ▶ A “correct” answer based largely on lecture material. Little detail or originality but presented in adequate framework. Small factual errors allowed.
- ▶ **30–49: Unsatisfactory (Fail)**
- ▶ **0–29: Unsatisfactory (Bad fail)**
 - ▶ Based entirely on lecture material but unstructured and with increasing error component. Concepts are disordered or flawed. Poor presentation. Errors of concept and scope or poor in knowledge, structure and expression.

Plan for today

- ▶ Administration and logistics
- ▶ Datasets, “tidy data”, and reshaping data in R
- ▶ Guided coding session
- ▶ Some good coding practices

What is a *dataset*?

- ▶ A dataset is a “rectangular” formatted table of data in which all the values of the same variable must be in a single column
- ▶ A dataset is not
 - ▶ The results of tabulating a dataset
 - ▶ Any set of summary statistics on a dataset
 - ▶ A series of relational tables
- ▶ Many of the datasets we use have been artificially reshaped in order to fulfill this criterion of rectangularity
 - ▶ This means “non-normalized” data
 - ▶ Probably not best form to store data

The difference between a table and a dataset

This is a table:

	Lost	Won
Challenger	266	60
Incumbent	32	106

This is a (partial) dataset:

		district	incumbf	wonseatf
1	Carlow	Kilkenny	Challenger	Lost
2	Carlow	Kilkenny	Challenger	Lost
5	Carlow	Kilkenny	Incumbent	Won
100	Donegal	South West	Challenger	Lost
459		Wicklow	Incumbent	Won
464		Wicklow	Challenger	Lost

"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	18215	19987071
Afghanistan	2000	18666	20095360
Brazil	1999	31737	17206362
Brazil	2000	80498	17404898
China	1999	212258	1272015272
China	2000	213796	128000583

variables

country	year	cases	population
Afghanistan	1999	18215	19987071
Afghanistan	2000	18666	20095360
Brazil	1999	31737	17206362
Brazil	2000	80498	17404898
China	1999	212258	1272015272
China	2000	213796	128000583

observations

country	year	cases	population
Afghanistan	1999	18215	19987071
Afghanistan	2000	18666	20095360
Brazil	1999	31737	17206362
Brazil	2000	80498	17404898
China	1999	212258	1272015272
China	2000	213796	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
```

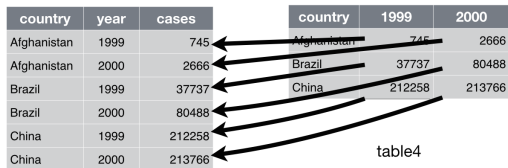
How to fix it?

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
```

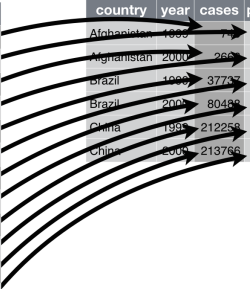
How to fix it?

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

A brief history of reshaping in R

stats::reshape: The “classic” method

```
reshape(data, varying = NULL, v.names = NULL, timevar = "time",
        idvar = "id", ids = 1:NROW(data),
        times = seq_along(varying[[1]]),
        drop = NULL, direction, new.row.names = NULL,
        sep = ".",
        split = if (sep == "") {
          list(regexp = "[A-Za-z][0-9]", include = TRUE)
        } else {
          list(regexp = sep, include = FALSE, fixed = TRUE)}
)
```


A brief history of reshaping (cont.)

reshape2: First update

```
melt(data, ..., na.rm = FALSE, value.name = "value")
```

```
## S3 method for class 'data.frame'
```

```
melt(data, id.vars, measure.vars,  
      variable.name = "variable", ..., na.rm = FALSE,  
      value.name = "value", factorsAsStrings = TRUE)
```

```
dcast(data, formula, fun.aggregate = NULL, ..., margins = NULL,  
       subset = NULL, fill = NULL, drop = TRUE,  
       value.var = guess_value(data))
```

A brief history of reshaping (cont.)

tidyr: Current (tidyverse) iteration

```
pivot_longer(  
  data,  
  cols,  
  names_to = "name",  
  names_prefix = NULL,  
  names_sep = NULL,  
  names_pattern = NULL,  
  names_ptypes = list(),  
  names_transform = list(),  
  names_repair = "check_unique",  
  values_to = "value",  
  values_drop_na = FALSE,  
  values_ptypes = list(),  
  values_transform = list(),  
  ...  
)  
pivot_wider(  
  data,  
  id_cols = NULL,  
  names_from = name,  
  names_prefix = "",  
  names_sep = "_",  
  names_glue = NULL,  
  names_sort = FALSE,  
  names_repair = "check_unique",  
  values_from = value,  
  values_fill = NULL,  
  values_fn = NULL,  
  ...  
)
```

Plan for today

- ▶ Administration and logistics
- ▶ Datasets, “tidy data”, and reshaping data in R
- ▶ Guided coding session
- ▶ Some good coding practices

Markdown files

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

Plan for today

- ▶ Administration and logistics
- ▶ Datasets, “tidy data”, and reshaping data in R
- ▶ Guided coding session
- ▶ Some good coding practices

Good practices in scientific computing

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 organized and documented code
3. Organized 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 wish to see in the world
 - ▶ Open, non-proprietary formats e.g. .csv
 - ▶ Informative variable names instead of V322
 - ▶ Recode missing values to NA
 - ▶ 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 organized 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 should not be more than ca. 20 lines long
 3. Can separate functions from execution (e.g. in functions.r file and then use `source(functions.r)` to load functions to current environment
 4. Errors should be corrected when/where they occur
- ▶ Keep it simple and don't get too clever
- ▶ Add informative comments before blocks of code

3. Organized 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. 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.

Replication materials for Leeper 2017:

- ▶ [Code and data](#)