MY472 – Day 2:
The Shape of Data

Pablo Barberá & Akitaka Matsuo

MY 472: Data for Data Scientists

October 2, 2018

Course website: lse-my472.github.io

Course outline

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

Housekeeping

- Seminar allocation and rules
- Assignment 2 (to be marked) to be distributed on Thursday. Due on October 19
- Assessment criteria
- Any questions?

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
- Data frames and lists in R
- Data and datasets:
  - “tidy” data
  - reshaping data
- Reshaping data in R
- Good coding practices
- Functions and loops in R

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
  - Often confounds variables with their values

The difference between a table and a dataset

This is a table:

	Lost	Won
Challenger	266	60
Incumbent	32	106

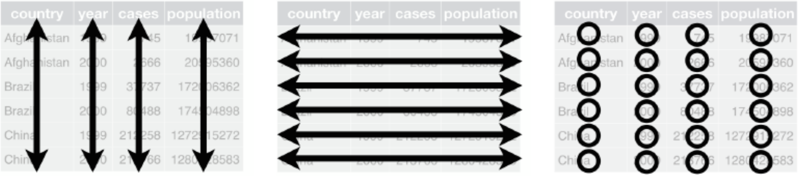
This is a (partial) dataset:

	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



variables observations values

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 gather those columns into a new pair of variables:

```
table4a %>%
  gather(‘1999‘, ‘2000‘, key = "year", value = "cases")
#> # A tibble: 6 x 3
#>   country     year   cases
#>   <chr>       <chr>  <int>
#> 1 Afghanistan 1999     745
#> 2 Brazil      1999   37737
#> 3 China       1999  212258
#> 4 Afghanistan 2000    2666
#> 5 Brazil      2000   80488
#> 6 China       2000  213766
```

What is happening here?

We switched from wide to long format:

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 spread those rows into a new pair of columns:

```
table2 %>%
    spread(key = type, value = 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	key	value
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

(But in practice we will mostly use the reshape2 package)

Good (enough) 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 should I waste my time?

- Replication is a key part of science:
  - Keep good records of what you did so that others can understand it
- "Yourself from 3 months ago doesn't answer emails"
  - More efficient research: avoid retracing own steps
  - Your future self will be grateful

General principles:

1. Good documentation: README and comments
2. Modularity with structure modularity — keep steps separated
3. Parsimony (without being too smart)
4. Track changes

Summary of good practices

1. Safe and efficient data management
2. Well-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 that indicate direction: `labour` instead of `party` or `V322`; `voted` vs `turnout`
  - 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-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. Functions should not be more than 20 lines long
  3. 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 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 are produced in scripts and compiled directly into manuscript with LaTeX

Examples

Replication materials for my 2014 PA paper:

- Code on GitHub
- Code and Data

John Myles White's ProjectTemplate R package.

Replication materials for Leeper 2017:

- Code and data