MY472 – Week 2: The Shape of Data

Friedrich Geiecke

MY 472: Data for Data Scientists

4 October 2021

Course website: lse-my472.github.io
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 visualisation
9. Creating and managing databases
10. Interacting with online databases
11. Cloud computing
Plan for today

- Datasets, “tidy data”, and reshaping data in R
- Guided coding session
- Some good coding practices
Plan for today

- 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-normalised” data
  - Probably not best form to store data
The difference between a table and a dataset

This is a table:

<table>
<thead>
<tr>
<th></th>
<th>Lost</th>
<th>Won</th>
</tr>
</thead>
<tbody>
<tr>
<td>Challenger</td>
<td>266</td>
<td>60</td>
</tr>
<tr>
<td>Incumbent</td>
<td>32</td>
<td>106</td>
</tr>
</tbody>
</table>

This is a (partial) dataset:

<table>
<thead>
<tr>
<th>district</th>
<th>incumbf</th>
<th>wonseatf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Carlow Kilkenny Challenger</td>
<td>Lost</td>
</tr>
<tr>
<td>2</td>
<td>Carlow Kilkenny Challenger</td>
<td>Lost</td>
</tr>
<tr>
<td>5</td>
<td>Carlow Kilkenny Incumbent</td>
<td>Won</td>
</tr>
<tr>
<td>100</td>
<td>Donegal South West Challenger</td>
<td>Lost</td>
</tr>
<tr>
<td>459</td>
<td>Wicklow Incumbent</td>
<td>Won</td>
</tr>
<tr>
<td>464</td>
<td>Wicklow Challenger</td>
<td>Lost</td>
</tr>
</tbody>
</table>
“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

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

```r
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:

<table>
<thead>
<tr>
<th>country</th>
<th>year</th>
<th>cases</th>
<th>country</th>
<th>1999</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>1999</td>
<td>745</td>
<td>Afghanistan</td>
<td>745</td>
<td>2666</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>2000</td>
<td>2666</td>
<td>Brazil</td>
<td>37737</td>
<td>80488</td>
</tr>
<tr>
<td>Brazil</td>
<td>1999</td>
<td>37737</td>
<td>China</td>
<td>212258</td>
<td>213766</td>
</tr>
<tr>
<td>Brazil</td>
<td>2000</td>
<td>80488</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>China</td>
<td>1999</td>
<td>212258</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>China</td>
<td>2000</td>
<td>213766</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(table4)
What else can go wrong?

Datasets where observations are scattered across multiple rows:

table2

```r
#> # 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:

<table>
<thead>
<tr>
<th>country</th>
<th>year</th>
<th>type</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>1999</td>
<td>cases</td>
<td>745</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>1999</td>
<td>population</td>
<td>19987071</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>2000</td>
<td>cases</td>
<td>2666</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>2000</td>
<td>population</td>
<td>20595360</td>
</tr>
<tr>
<td>Brazil</td>
<td>1999</td>
<td>cases</td>
<td>37737</td>
</tr>
<tr>
<td>Brazil</td>
<td>1999</td>
<td>population</td>
<td>172006362</td>
</tr>
<tr>
<td>Brazil</td>
<td>2000</td>
<td>cases</td>
<td>80488</td>
</tr>
<tr>
<td>Brazil</td>
<td>2000</td>
<td>population</td>
<td>174504898</td>
</tr>
<tr>
<td>China</td>
<td>1999</td>
<td>cases</td>
<td>212258</td>
</tr>
<tr>
<td>China</td>
<td>1999</td>
<td>population</td>
<td>1272915272</td>
</tr>
<tr>
<td>China</td>
<td>2000</td>
<td>cases</td>
<td>213766</td>
</tr>
<tr>
<td>China</td>
<td>2000</td>
<td>population</td>
<td>1280428583</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>country</th>
<th>year</th>
<th>cases</th>
<th>population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>1999</td>
<td>745</td>
<td>19987071</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>2000</td>
<td>2666</td>
<td>20595360</td>
</tr>
<tr>
<td>Brazil</td>
<td>1999</td>
<td>37737</td>
<td>172006362</td>
</tr>
<tr>
<td>Brazil</td>
<td>2000</td>
<td>80488</td>
<td>174504898</td>
</tr>
<tr>
<td>China</td>
<td>1999</td>
<td>212258</td>
<td>1272915272</td>
</tr>
<tr>
<td>China</td>
<td>2000</td>
<td>213766</td>
<td>1280428583</td>
</tr>
</tbody>
</table>

**table2**
A brief history of reshaping in R

`stats::reshape`: The “classic” method

```r
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

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

```r
## S3 method for class 'data.frame'
melt(data, id.vars, measure.vars,
    variable.name = "variable", ..., na.rm = FALSE,
    value.name = "value", factorsAsStrings = TRUE)
```

```r
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

```r
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

▶ 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

- Datasets, “tidy data”, and reshaping data in R
- Guided coding session
- Some good coding practices
Good practices in scientific computing

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 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 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 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. 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. 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