5. Directory and Data Structure

Data Management
Spring & Summer 2018
OSIPP, Osaka U

Shuhei Kitamura

Goal

Get to know about the directory and data structure!

Outline

Data cleaning

- Directory structure
- Data structure

Data analysis

Typical questions you may have...

How many directories and subdirectories do I need?

How should I name them?

How should I structure them?

- A directory for each project?
- A directory for each data?

Main directories and workflow

Main directories are:

- (a) A directory for data cleaning.
- (b) A directory for analyzing data and writing paper & slides

Workflow:

- In (a), put raw files in **input**, clean them, and produce cleaned data in **output**.
- Import cleaned data from **output** in (a) into **input** in (b), process the data (e.g., collapse and merge), analyze them, and produce tables and figures in **output** in (b).
- Read tables and figures to produce a paper and slides in **output** in (b).

If all data are almost clean, you can skip (a).

A directory for data cleaning

Since you may use the same data for several projects, it is useful to have a separate directory just for cleaning those data.

- You do not need to export all relevant files for another project.
- You can easily find relevant code.
- You can easily share code and data.

A directory for data cleaning (cont.)

- An ideal directory structure for data cleaning may look like:

```
data_todai-asahi
    code
    input
        - raw
    temp
    output
```

- If there is any data which require simple changes by hand, put the original data in `raw` and the outcome in `input`. If you do not have such data, you do not create the `raw` directory itself.
- You import input data from `input` and produce cleaned data in `output`. `temp` is used for storing temporary files.

Data structure

Cross-sectional data at the county level in the U.S.

county	state	cnty_pop	state_pop	region
36037	NY	3817735	43320903	1
36038	NY	422999	43320903	1
36039	NY	324920	.	1
36040	.	143432	43320903	1
.	VA	.	7173000	3
37001	VA	3228290	7173000	4

Source: Code and Data for the Social Sciences

Can you spot what's wrong with the data?

Data structure (cont.)

county	state	cnty_pop	state_pop	region
36037	NY	3817735	43320903	1
36038	NY	422999	43320903	1
36039	NY	324920	.	1
36040	.	143432	43320903	1
.	VA	.	7173000	3
37001	VA	3228290	7173000	4

- County id missing for one observation.
  - Do we really need that observation?
- State name and state_pop missing.
  - Guessing from the data, 36040 might be located in NY.
  - State population for 36039 might be 43320903.
- There are two regions for VA $\rightarrow$ Most likely wrong.

Data structure (cont.)

If data look like this, you cannot use such data.

- Maybe you understand why the data look like this now, can you still recall the reasons a couple of years later? Probably not.
- Can you share such data to others? Probably not.

Data structure (cont.)

A better data structure:

county	state	population
36037	NY	3817735
36038	NY	422999
36039	NY	324920
36040	NY	143432
37001	VA	3228290

state	population	region
NY	43320903	1
VA	7173000	3

- Now we have two data (tables): a county table and a state table. They are called a *relational database.*
  - No missing value.
  - The table is self-documenting. For example, it's clear to anyone that population in the state table means state population.
- Each table has a *key* (i.e., "county" in the county table, and "state" in the state table).
  - The key should not contain any missing value.
  - It should not be duplicated.

Data structure (cont.)

A relational database:

county	state	population
36037	NY	3817735
36038	NY	422999
36039	NY	324920
36040	NY	143432
37001	VA	3228290

state	population	region
NY	43320903	1
VA	7173000	3

- A table can contain a *foreign key* (i.e., "state" in the county table and "region" in the state table).
- Data stored in this form are considered normalized.

  - In contrast, if e.g. the county table contains variables that are not properties of countries, the data are *not* normalized.
- We need data cleaning just for making normalized data.

  - You merge, make new variables (e.g. log of population), etc. when you analyze data.

Data structure (cont.)

A relational database:

county	state	population
36037	NY	3817735
36038	NY	422999
36039	NY	324920
36040	NY	143432
37001	VA	3228290

state	population	region
NY	43320903	1
VA	7173000	3

- You do not need to think about how you will use the data when you prepare normalized data. Make the data as if you will release to a broad group of users with differing needs.
  - These tables may be used in other projects in future.
- If you want panel data, you may need county-year-level variables as a separate database.
  - The rule is the same: the key should be unique and non-missing.
  - Btw, what is the key in such panel data?

Data cleaning

Let's clean Todai-Asahi Data and make a panel dataset.

- Publicly available comprehensive panel survey of all candidates for the Japanese Upper and Lower House elections between 2003 and 2016.

First, fork the following directory from my repository, then clone it to your local machine.

```
data_todai-asahi
    code
    input
        - codebooks
        - raw
    temp
    output (Add this folder manually, which is not included in the forked repository.)
```

`input` contains the Todai-Asahi data from 2003 to 2014.
Our final goal is to make a panel dataset using those data.

A directory for data analysis

You should have a separate single directory for a project.

- You can easily find relevant code.
- You can easily share code and data.
- Journals may ask you to provide replication code and data. (Even if not, replication files should easily be available to the public unless privacy, non-disclosure statements, ethical reasons, etc. prevent it).

A directory for data analysis

- An ideal directory structure for data cleaning may look like:

```
proj_todai-asahi
    build_code
    build_input
        - raw
    build_temp
    build_output
    analysis_code
    analysis_temp
    analysis_output
```

- If there is any data which require simple changes by hand, put the original data in `raw` and the outcome in `input`. If you do not have such data, you do not create the `raw` directory.

Data analysis

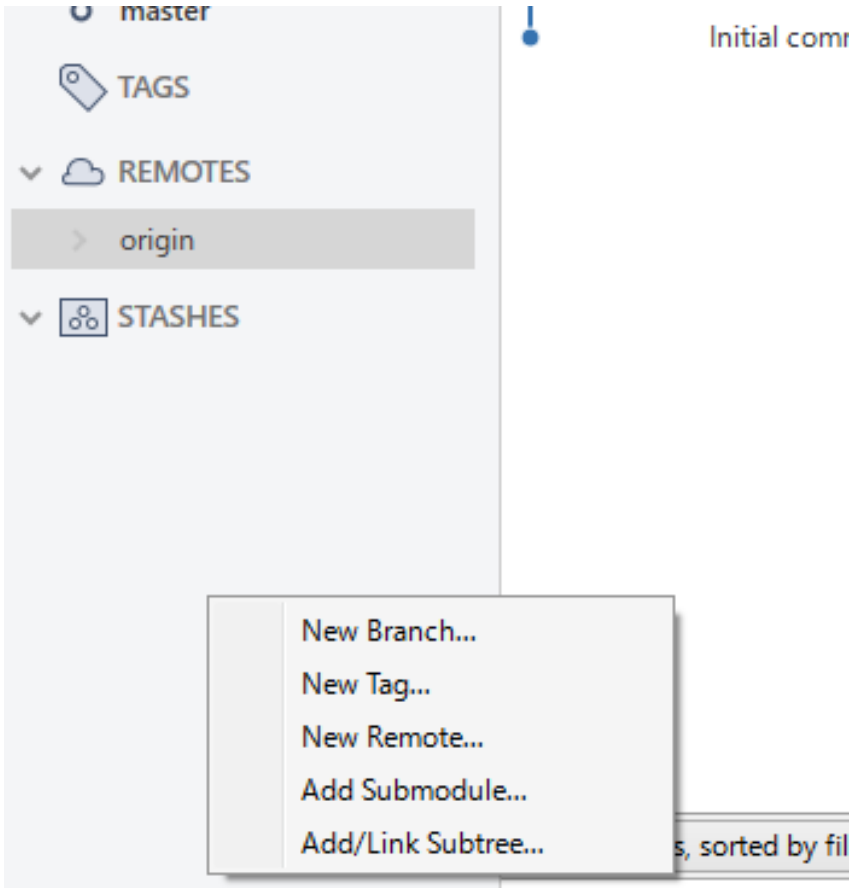
Let's analyze the Todai-Asahi data.

First, fork the following directory from my repository.

proj_todai-asahi

- ~~build_code~~
- build_input (Add this folder manually.)
  - ~~raw~~
- ~~build_temp~~
- ~~build_output~~
- analysis_code
- ~~analysis_temp~~
- analysis_output (Add this folder manually.)

Submodules



- Let's make a submodule in `build_input`.
- Open SourceTree.
- Assuming that you are currently at the master branch, right click on the left tab, click "Add Submodule".

Submodules (cont.)

- Write the path to your remote repo “data_todai-asahi” in Source Path / URL.
- Write “build_code/data_todai-asahi” in Local Relative Path.
  - You need to make a “data_todai-asahi” folder within the “build_code” folder before doing so.
- Click OK.
- You should be able to see “Submodules” to the left.
- Push “build_input/data_todai-asahi” and “.gitmodules”.

Update submodules

- Since SourceTree does not seem to automatically update submodules, you need to do it manually.
- Under “build_code”, you may find “data_todai-asahi”.
- Right click. $\rightarrow$ Click “Open data_todai-asahi”. This opens a new tab.
- In the new tab, pull the update.
    - If you just want to check the update, use fetch instead.