Reproducible data processing:
Strategies to make your research auditable, scalable, and reproducible

Jule Krüger, Ph.D.
Center for Political Studies, Institute for Social Research
and Advanced Research Computing, Information Technology Services,
University of Michigan
Process your research data in a way that

- Is self-documenting (to minimize active time spent on documentation)
- Can scale (multiple data sources, collaborators, project years, etc.)
- Automates updates (as inputs, source code or constants change)
- Facilitates auditability and reproducibility (for yourself and others)
Overview of data processing principles

1. Conceptualize your project as tasks
2. Organize your project directory by task
3. Separate input data, source code, manual work, and outputs
4. Use Makefiles to document targets and dependencies
5. Separate manually set constants from your code
6. Apply self-explanatory naming conventions
7. Version control your work and publish a public repository
1. Break your data science project into TASKS

Example research data workflow:

import ➔ clean ➔ estimate ➔ apply estimates ➔ visualize
2. Organize your project directory into tasks

- Structure your project directory intentionally to represent this workflow
- Each task gets its own directory
2. Organize your project directory into tasks

<table>
<thead>
<tr>
<th>Name</th>
<th>Date Modified</th>
<th>Size</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>.git</td>
<td>12/17/21, 9:26 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>.gitignore</td>
<td>11/16/20, 10:18 AM</td>
<td>21 bytes</td>
<td>Document</td>
</tr>
<tr>
<td>apply-estimates</td>
<td>11/15/20, 9:46 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>clean</td>
<td>11/14/20, 5:45 PM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>estimate</td>
<td>11/16/20, 11:23 AM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>import</td>
<td>11/14/20, 11:33 AM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>read</td>
<td>11/5/20, 9:21 AM</td>
<td>--</td>
<td>Folder</td>
</tr>
<tr>
<td>README.md</td>
<td>12/17/21, 9:25 PM</td>
<td>2 KB</td>
<td>Mark...wn File</td>
</tr>
<tr>
<td>visualize</td>
<td>11/16/20, 11:22 AM</td>
<td>--</td>
<td>Folder</td>
</tr>
</tbody>
</table>
3. For each task, separate input data, source code, manual work and output files

- **src**: code scripts (R, Python files, etc.)
- **input**: Raw input files, to be read from src/. Not in every task.
- **hand**: Manually generated stuff (training data, constants, maps, dictionaries, etc.)
- **output**: Output files written from src/ (processed data, results, graphs, tables, reports, etc.)
Example data flow

```
my-research-project-name/
  import/
    input/raw-data.txt
    src/read-raw-data.R
    output/imported-data.csv
  clean/
    hand/mapping-values.csv
    src/clean-imported-data.R
    output/clean-data.csv
```
4. Write a Makefile for each task

- GNU Make runs and compiles code files in src/ directory
- Clearly identifies dependencies (code scripts, input files, manual work) and targets (output files) → self-documenting
- It observes the timestamps of each dependency to determine when to re-run and update a target → automation
- Combine with a library to parse arguments into your code scripts (*argparse*) to track file names exclusively in the Makefile
Example Makefile for a data cleaning task

```bash
#!/usr/bin/env make -f

.PHONY: all

all: output/my-clean-data.csv

go: 

output/my-clean-data.csv: 
  src/clean-imported-data.R 
  ../import/output/imported-data.csv 
  Makefile 
  Rscript --vanilla $< 
  --inputfile=../import/output/imported-data.csv 
  --outputfile=$@

#end.
```
5. Separate constants from your code

- Constants are parameters that do not change within one version of analysis.
- Can be identified in yaml code, csv files, dictionaries, or another format of your choice.
  - Read yaml code into R with the `yaml` library, in Python with `PyYAML`.
- Go in the hand/ directory to identify their manual work nature.
- When a constant changes, update the files in hand/ → facilitates automation.
Example constants file in yaml code

```
1. # hand/CONSTANTS.yaml
2.
3. project_start_date: 2021-01-01
4.
5. project_end_date: 2021-06-30
6.
7. #end.
```
Putting it all together:

Makefile + R code + argparse + CONSTANTS.yaml
#!/usr/bin/env make -f

.PHONY: all

all: \
    output/my-clean-data.csv

output/my-clean-data.csv: \
    src/clean-imported-data.R \
    ../import/output/imported-data.csv \
    hand/CONSTANTS.yaml \
    Makefile
    Rscript --vanilla $< \
        --inputfile=../import/output/imported-data.csv \
        --CONSTANTS=hand/CONSTANTS.yaml \
        --outputfile=@@

#end.

# src/clean-imported-data.R

library(argparse)
library(yaml)

parser <- ArgumentParser()
parser$add_argument("--inputfile", type="character")
parser$add_argument("--CONSTANTS", type="character")
parser$add_argument("--outputfile", type="character")
arguments <- parser$parse_args()

CONSTANTS <- yaml.load_file(arguments$CONSTANTS)

data <- read.csv(arguments$inputfile, header = TRUE, 
    sep = '|', stringsAsFactors = FALSE)

data$date <- as.Date(data$date, format="%Y-%m-%d")

# subset to period of observation
obs_period <- data[data$date >= CONSTANTS$project_start_date & 
    data$date <= [CONSTANTS$project_end_date, ]]

write.table(obs_period, arguments$outputfile, row.names = FALSE, 
    sep = '|', quote = FALSE)

print(paste("done printing study subset to", arguments$outputfile))

# end of Rscript.

# hand/CONSTANTS.yaml

project_start_date: 2021-01-01
project_end_date: 2021-06-30
#end.
6. Adopt self-explanatory naming conventions

<table>
<thead>
<tr>
<th>Directories</th>
<th>Tasks/data they contain</th>
<th>import, clean, write</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data files</th>
<th>What they contain</th>
<th>data-raw.txt, data-clean.csv, bg-time-period-subset.pdf</th>
</tr>
</thead>
</table>

|-------------|-------------|-------------------------------------------------------------|

<table>
<thead>
<tr>
<th>Objects in your code</th>
<th>What they are</th>
<th>days, months, obs_period, time_period_subset</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th>What they do</th>
<th>CleanRawData, PlotDaysPerMonth</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Constants</th>
<th>What they are</th>
<th>project_start_date, project_end_date</th>
</tr>
</thead>
</table>
7. Version control all of your work

- git, svn (subversion)
- commit small increments often to log the iterations of your work
- Generates a log of everything that was done → self-documenting
- Ability to revert back in time anytime
- Publish your project directory to a hosting platform of your choice (GitHub, GitLab, Bitbucket, etc.)
  - (you might prefer a cleaned up copy without the logs, e.g., a clean branch)
Benefits of principled data processing

- Provides transparency about how you set up your project and manipulated the data
- Quickly conveys the role of each project file based on its name and where it is located in the project directory
- Clearly distinguishes between computation and manual work, making both accessible to project-unrelated individuals
- Enables sensitivity analysis of modeling parameters and coding decisions

→ Facilitates peer review and reproducibility
An example project directory

https://github.com/juleka/JPR-LVM-SVAC

Shares replication data, code and supplementary information for joint research with Ragnhild Nordås, Assistant Professor, Department for Political Science, University of Michigan