Dynamics of Youth

DATA HANDBOOK

Neha Moopen

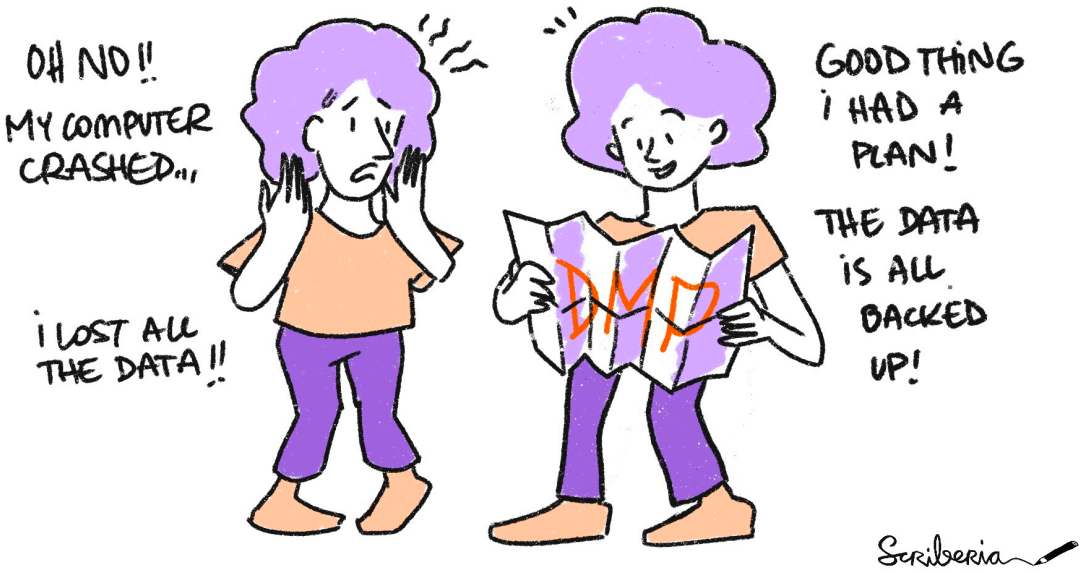
2024-08-06

Table of contents

Welcome!

Figure 1: This illustration is created by Scriberia with The Turing Way community. Used under a CC-BY 4.0 licence. DOI: 10.5281/zenodo.3332807

Data Management Plans



What Is A Data Management Plan?

A Data Management Plan (DMP) is a formal document that describes your data and outlines all aspects of managing your data - both during and after your project.

Moreover, it is a *living* document that can you can revise and update as needed.

Why Should You Write A DMP?

Writing a DMP provides an opportunity to reflect on your data, particularly how you organize and manage it. It nudges you to think about how to make your RDM more *concrete* and *actionable*. This creates efficiency and more value for your data.

When Should You Write A DMP?

Working on a DMP at the start of your project will ensure that you are better informed of best practices in RDM and prepared to implement them. That being said, you can also write a DMP can during the project or when it's completed.

DMPonline & DMP Templates

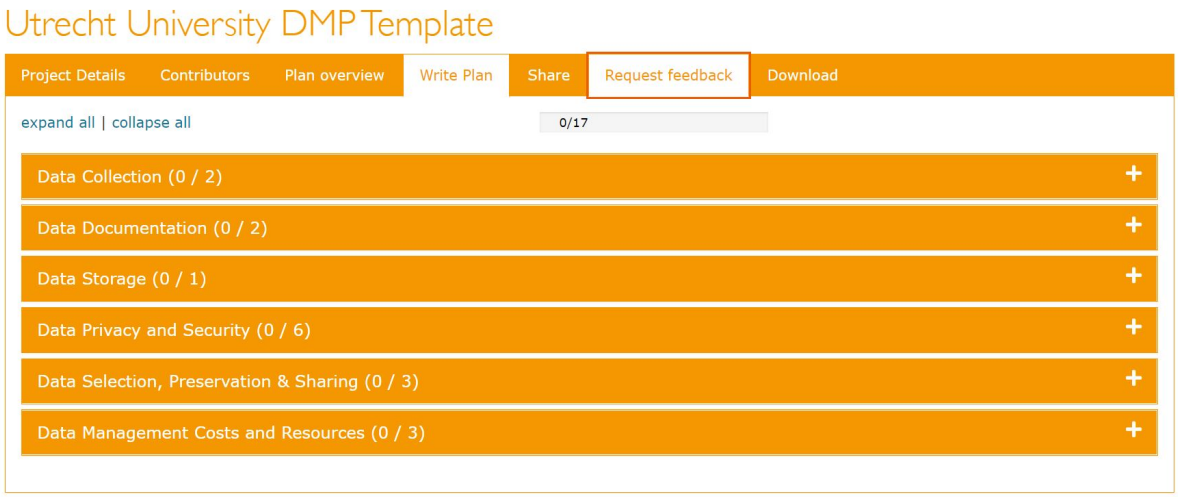
DMPonline is a tool that helps you create and maintain DMPs. With DMPonline, you can:

- register and sign in with your institutional credentials,
- write and collaborate on (multiple) DMPs,
- share DMPs or switch their visibility between private and public,
- request feedback from RDM Support,
- download DMPs in various formats.

DMPonline offers DMP templates from various institutions and funders, including:

- Utrecht University
- UMC Utrecht
- NWO
- ZonMw
- ERC
- Horizon 2020
- Horizon Europe

These templates also contain example answers and guidance.



Tips

!!! note “Tips”

```
- Contact your DoY data manager! They can (co)write your DMP and/or review it.
- If the DoY data manager is unavailable, you can still request feedback from RDM Support.
```

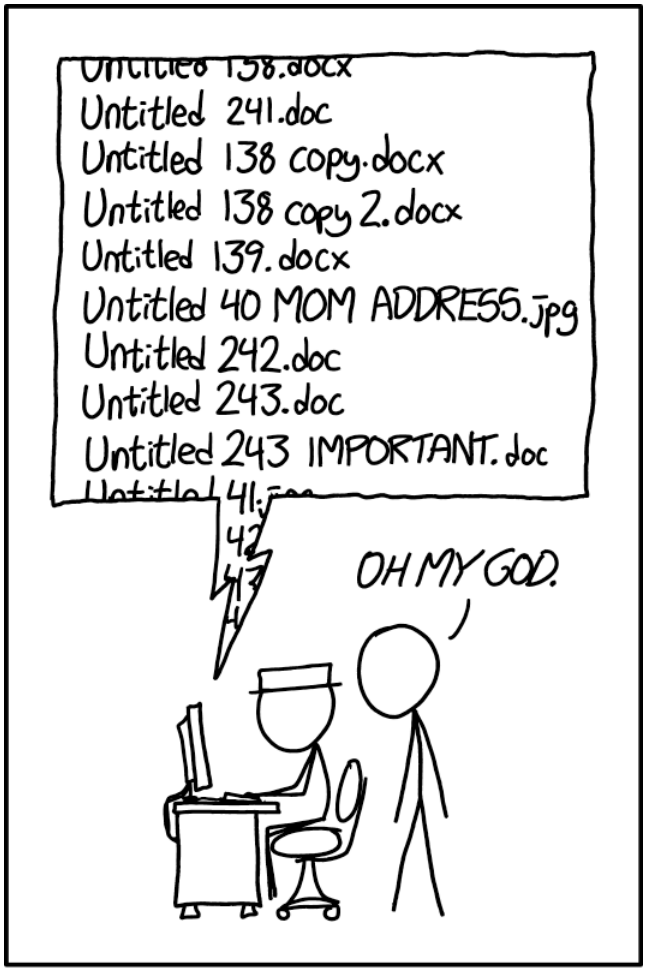
Resources

- Create your DMP online
- Data management planning
- Learn to write your DMP (online training)

References

1. https://www.uu.nl/en/research/research-data-management/guides/data-management-planning
2. https://www.kuleuven.be/rdm/en/faq/faq-dmp
3. https://rdm.uva.nl/en/planning/data-management-plan/data-management-plan.html
4. https://www.uu.nl/en/research/research-data-management/tools-services/tool-to-create-your-dmp-online.html

Naming Conventions



Documents - xkcd. Used under a CC BY-NC 2.5 license.

What Is A Naming Convention?

A naming convention is a set of rules for naming things. You can apply it to things like folders, files, and variables.

Why Should I Apply A Naming Convention?

Names that are informative and useful for machines and humans are a step toward efficient data management and reproducible research. The more consistent and meaningful the name, the easier it will be to locate and identify things, understand what they contain, and (re)use them.

When Should I Apply A Naming Convention?

Aim to select and implement a naming convention at the beginning of a project. If you want to retroactively apply a naming convention, there are several tools for bulk renaming.

The entire research team should agree on and adopt a naming convention. Document the choice of naming convention in the DMP, so others can refer to and grasp it quickly.

Popular Naming Conventions

Instead of developing a naming convention from scratch, you can start with one that is already being used in programming and software development communities:

Naming Covention	Example	Description
original name	`an awesome name`	N/A
snake_case	`an_awesome_name`	All words are lowercase and separated by an underscore (`_`)
kebab-case	`an-awesome-name`	All words are lowercase and separated by a hyphen (`-`)
PascalCase	`AnAwesomeName`	All words are capitalized. Spaces are not used.

Naming Covention	Example	Description
camelCase	`anAwesomeName`	The first word is lowercase, the remaining words are capitalized. Spaces are not used.

Human-Readable Names

You can tailor naming conventions like `snake_case` and `PascalCase` to suit your project and workflow. Determine what information is relevant (or not) to create meaningful names and how you can string this information together. Don't forget to document this in your DMP!

!!! note "Elements for Human-Readable Names"

```
Names should be =<25 characters long and can include:

- Date of creation/update (`YYYY-MM-DD` or `YYYYMMDD`)
- Description of content, like type of data
- Initials of creator/reviewer
- Project number or acronym
- Location/coordinates
- Version number (like `v2` or v2.2`)
```

Machine-Readable Names

When names are machine-readable, they can be efficiently processed by computers and software. This makes it easier to search for files and run operations that involve programming like extracting information from file names or working with regular expressions.

!!! note "Avoid"

```
- Spaces
- Special characters like `$`, `@`, `%`, `#`, `&`, `*`, `!`, `/`, `\`
- Punction characters like `,`, `:`, `;`, `?`, `'`, `"`
- Accented characters
```

A Note on Numbering, Dates, Versioning

- Append numbers to the beginning of a name to enable sorting according to a logical structure. Use multiple digits like `01` or `001`.
- Dates should follow the ISO 8601 standard which is either `YYYY-MM-DD` or `YYYYMMDD`. Append dates to the beginning of names to enable sorting in chronological order.
- Specify versions using ordinal numbers (1,2,3) for major revisions and decimals for minor changes (1.1, 1.2, 2.1, 2.2). Alternatively, you can specify versions with multiple digits like v01 and v02.

Renaming files

The following tools enable renaming in bulk:

- Bulk Rename Utility (Windows, free)
- Renamer (MacOS, paid)
- NameChanger, (MacOS, free)
- GPRename (Linux, free)

References

1. https://en.wikipedia.org/wiki/Naming_convention
2. https://help.osf.io/article/146-file-naming
3. https://rdm.elixir-belgium.org/file_naming.html
4. https://khalilstemmler.com/blogs/camel-case-snake-case-pascal-case/
5. https://dev.to/chaseadamsio/most-common-programming-case-types-30h9
6. https://rdmkit.elixir-europe.org/data_organisation http://dataabinitio.com/?p=987
7. https://dmeg.cessda.eu/Data-Management-Expert-Guide/2.-Organise-Document/File-naming-and-folder-structure
8. https://annakrystalli.me/rrresearchACCE20/filenaming-view.html

Data Pipelining

A data pipeline is a series of (automated) actions that ingests raw data from various sources and moves the data to a destination for storage and (eventual) analysis.

Benefits of a data pipeline include:

- Time saved by automating the boring stuff!
- Reduced mistakes.
- Tasks broken down into smaller steps.
- Reproducibility!

When do I need a data pipeline?

Here's a rule of thumb, just as an example:

If you have a task that needs to occur >= 3 times, you could think about automating it.

If automation is not possible, think about how you can make the task as efficient as possible.

How can I implement a data pipeline? Some examples for inspiration

- If you data collection tools have APIs, they can be leveraged to extract data.
- For example, Qualtrics has the qualtRics R package & pyQualtrics Python library which contain functions to automate exporting surveys.
- If APIs are not available, you could use R/Python to automate the use of an internet browser using the RSelenium package / Selenium library. Imagine automating the clicks and typing of going to a specific website, logging in, clicking the download button.
- You can use Windows Task Scheduler / cron / the taskscheduleR R package / cronR to schedule your scripts to run automatically, on a recurring basis as well (if needed).

- You can also send emails with R & Python! Consider if you've ever had to contact participants because you noticed something wrong with their incoming data. You could implement these data checks with a script and automatically draft and send emails (from a template) to those participants who were flagged as having issues with their data.

QualtRics R package

```
library(readr)
library(qualtRics)

qualtrics_api_credentials(api_key = "YOUR-QUALTRICS-API-KEY",
                          base_url = "YOUR-QUALTRICS-BASE-URL",
                          overwrite = TRUE,
                          install = TRUE)

readRenviron("~/.Renviron")

surveys <- all_surveys()

survey_results <- fetch_survey(surveyID = surveys$id[2], # you can also replace surveys$id[2]
                               verbose = TRUE)

write_csv(survey_results, paste0("path/to/folder/", format(Sys.time(), "%d-%m-%Y-%H.%M"), "_s
```

taskscheduleR package

```
library(taskscheduleR)

scheduled_script <- "path/to/folder/myscript.R"

## run script once within 120 seconds

taskscheduler_create(taskname = "extract-data-once", rscript = scheduled_script,
                     schedule = "ONCE", starttime = format(Sys.time() + 120, "%H:%M"))

## Run every 5 minutes, starting from 10:40

taskscheduler_create(taskname = "extract-data-5min", rscript = scheduled_script,
                     schedule = "MINUTE", starttime = "10:40", modifier = 5)
```

```
## delete tasks

taskscheduler_delete("extract-data-once")
```

Codebooks

A codebook is an example of data-level metadata.

The purpose of a codebook or data dictionary is to explain what all the variable names and values in your spreadsheet really mean.

Information to include in a codebook includes:

- Variable Names
- Readable Variable Name
- Measurement Units
- Allowed Values
- Definition Of The Variable
- Synonyms For The Variable Name (Optional)
- Description Of The Variable (Optional)
- Other Resources

See: https://help.osf.io/article/217-how-to-make-a-data-dictionary

codebook R package

```
library(qualtRics)
library(readr)
library(dplyr)
library(codebook)
library(writexl)

surveys <- all_surveys()

survey_results <- fetch_survey(surveyID = surveys$id[2], # you can also replace surveys$id[2]
                               verbose = TRUE)

survey_results <- select(survey_results, -c(1:17))

# survey_questions() retrieves a data frame containing questions and question IDs for a surve
survey_questions <- survey_questions(surveyID = surveys$id[2])
```

```
survey_questions <- select(survey_questions, -c(1, 4))
survey_questions <- slice(survey_questions, -1)

# generate codebook

codebook <- codebook_table(survey_results)

codebook <- rename(codebook, qname = name)

codebook <- full_join(survey_questions, codebook, by = "qname")

write_xlsx(codebook, "documentation/codebook-demo.xlsx")
```

The `labelled` R package can also do something similar.

References