PHO Data Management Guidelines

Callum Arnold

January 17, 2019

Reproducible Work

What is reproducible work?

- Contains relevant code, including which packages were used
- Contains enough text, either via `markdown` or comment to be able to understand what the purpose of each code is
    - Ideally integrates code and results, along with text, into a single document

Why is it important?

Simply put, all code and work has mistakes and bugs. If you are able to

Structuring a project

This is the structure that I find works for me. You may want to find a variation on it that works for you, but the basic premise of keeping repositories self-contained should remain.

```
proj/
 data/
 docs/
 figs/
 funs/
 out/
 cleaning.R
 analysis.R
```

As you can see, the project repository contains separate directories that you can use to store different file types. Importantly, the analysis and cleaning files are stored in the project root, allowing easy use of relative paths over explicit paths e.g. `read_csv(here('data', 'data_file.csv'))` rather than `read_csv('C:/Users/owner/Documents/Repos/my_project/data/data_file.csv')`. The reason why we prefer relative paths is that they allow projects to be used by multiple people without the need to re-write code. If you change computer, or the project is opened by another person, the code will break as they will not have the same directory structure as the computer that the code was created on.

Note: the example above used an `R` package called `here_here`, calling the function `here()`. Similar solutions may exist for other languages, and you should try and find them for the language of your choice.

`data/`

An important idea is that you should treat your data as read-only. You and your team have likely worked hard to collect the data and it's easy to make a changes along the way that you either forget about, or need to reverse. As most projects span a long time between the data collection and analysis stages, if that happens to you it will take a lot of work to figure out exactly which changes you are interested in reversing etc. To save yourself this hassle, and help make your work reproducible, once the data is collected it should not be edited; all the work should happen in your code, allowing it to be easily checked.

Other subdirectories

- `docs/`: this contains the output documents. For example, if you are using `R Markdown` to create a pdf via `LaTeX`, you could place them here.
- `figs/`: this contains the functions you write and might want to reference. The idea is to create functions so that can give code a meaningful name. It also helps if you need to repeat a code chunk multiple times, especially if you need to edit it at some point, as you can just call the function rather than typing it out each time.
- `out/`: this contains files that are produced from the original data e.g. cleaned data files. You can then call them in your analysis scripts.
- `figs/`: this contains figures that may be generated from your scripts.

Importantly, if you follow the principle that your `data/` files are read-only, all of the files in these directories (with the exception of `funs/`) *should* be reproducible and could be deleted at any time without concern of generating them again. In order to revert to previous figures and output versions, you will need to be able to track changes in your code. This is where a *version control system* like `git`, which we will discuss in the next section.

Key Points - Use a version control system such as `git` to track changes in your code. - Data isn't touched one collected: - Do all *data munging* within your program i.e. no editing the excel spreadsheets!!! - Never set explicit file paths if you can avoid it e.g. `setwd()` - Try and use a package that allows you to set relative paths e.g. `here_here` in `R`. This allows the project to be passed to someone else in its entirity and the code won't break because they don't have the same folder names and set up as you (also if you work on multiple computers/OS)

Git

Think of it as tracked changes for your code. When working on a project by yourself, it's important to be able to go back to previous versions if you make a mistake and can't remember all the steps you went through since your last stable version.

- SourceTree

Branching

Notebooks

- Many examples of different kinds of notebooks
- Jupyter useful as you can see the results immediately integrated within the document
  - Has a system to allow other statistical and programming languages to run via kernels
    - Other systems don't allow this level of integration, so will not be explored in much detail

Jupyter notebooks

This section will give you a brief overview of what a Jupyter notebook is and how to use them, but if you would like a more detailed understanding, please read the official documentation.

Jupyter notebooks are run on `python`, though additional things can be downloaded to allow you to use your programming language of choice. For examples of what you can do with Jupyter notebooks, click here, and here for a collection of neat and applied notebooks.

- Download the full anaconda distribution i.e. not miniconda

    - Be sure to choose `Python 3.x`, not `Python 2.x`, as it's the newer version and is fowards-compatible.
    - Be sure to only install for one user, not the whole system
    - Be sure to select `Add Anaconda to my PATH environment variable` under Advanced Options
    - Be sure to install `Anaconda` to the `H:\` drive on your computer, as this is where your data lives. To do this you will need to manually edit the installation path within the `anaconda` installer wizard, otherwise it will end up in the `C:\` drive
        - This is OK if you are able to store data on this drive, and therefore can create your repositories within the `C:\` drive.
        - Worst case scenario you can use the command `cd "H:/..."` at the top of the notebook to specify the relevant path to your data, but this is bad practise for the reasons mentioned above.
- Use `kernels` to connect your programming language of choice with python and the notebook
    - To see how to get a particular language to work in Jupyter Notebooks, please click on the appropriate language:
        - Stata
        - SAS
        - R
        - Other kernels
- Visualization for data exploration:
    - plotly (https://towardsdatascience.com/the-next-level-of-data-visualization-in-python-dd6e99039d5e)

Creating a notebook

You can either open up the anaconda navigator and then Jupyter notebooks, or open Jupyter notebooks directly. Once open, navigate to the directory you would like to create the notebook in (*If you are using a version control system like git, then you should be within the project's repository*)

Select the **New** button in the top right corner, and then select the language you would like to program in (*this assumes that you have downloaded an appropriate* `kernel` *if you would like to use a language other than* `python`)

Installing the Stata Kernel

The instructions for installing the `stata_kernel` are based from the original documentation here. It should work with `Stata 12` (we have tested it). If these instructions do not work for you, it may be that there has been an update to the `kernel`, at which point, please refer to the original documentation linked above.

Open a command prompt (Windows) / terminal (linux/mac) and type/copy-paste the following commands, pressing enter after each line

- `pip install stata_kernel`
- `python -m stata_kernel.install`

Windows-specific steps

In order to let `stata_kernel` talk to Stata, you need to link the Stata Automation library:

1. In the installation directory (most likely `C:\Program Files (x86)\Stata12` or similar), right-click on the Stata executable, for example, `StataSE.exe` (this will just show as `StataSE`, but is listed as an application). Choose `Create Shortcut`. Placing it on the Desktop is fine.
2. Right-click on the newly created `Shortcut to StataSE.exe`, choose `Properties`, and append`/Register` to the end of the Target field. So if the target is currently `"C:\Program Files\Stata12\StataSE.exe"`, change it to `"C:\Program Files\Stata12\StataSE.exe" /Register` (note the space before `/`). Click OK.
3. Right-click on the updated `Shortcut to StataSE.exe`; choose Run as administrator.
4. Enter your CIHS details

Installing the SAS kernel

*_This has not yet been tested here at PHO. The instructions for installing the `sas_kernel` are based from the original documentation here_*

Open a command prompt (Windows) / terminal (linux/mac) and type/copy-paste the following commands, pressing enter after each line. First we need to install a dependecy called `saspy` that helps the kernel connect `SAS` to `python`

- `pip install saspy`
- `pip install sas_kernel`

You should now see something like this.

```
Available kernels:
python3
    /home/sas/anaconda3/lib/python3.5/site-packages/ipykernel/resources
sas        /home/sas/.local/share/jupyter/kernels/sas
```

Now verify that the SAS Executable is correct

- find the `sascfg.py` file – it is currently located in the install location (see above) `[install location]/site-packages/saspy/sascfg.py`. To query `pip` for the location of the file, type `pip show saspy`. Failing that,

this command will search the OS for the file location: `find / -name sascfg.py`
- edit the file with the correct path the SAS executable and include any options you wish it include in the SAS invocation. See examples in this file

Connecting R with Jupyter

If you are hoping to make nice documents and reproducible work using `R`, I would highly recommend that you use the `R Markdown` or `R Notebook` through `RStudio` application instead. However, if you would prefer Jupyter, then please read on.

It is possible to download an `R kernel`, much like for `Stata` and `SAS`, but it can be a bit fickle, so a different approach is described below. It is important to note that with this method you are installing a fresh version of `R`, so you will not have access to the packages you have previously installed - you will need to reinstall them in this `R` environment, which could be done within a Jupyter notebook.

Open a command prompt (Windows) / terminal (Linux/Mac) and type/copy-paste the following commands, pressing enter after each line

- `conda install r-essentials r-igraph`
- `Rscript -e 'install.packages("languageserver")'`

If you would rather install an `R kernel` than a fresh install of `R` within the `anaconda` distribution, you can follow the instructions here. The advantage of this is that it allows the notebook to access previously installed packages as they are not running off a fresh version of `R`.

Connecting other kernels

To see a full list of `kernels` available for Jupter, along with the appropriate documentation and installation instructions, follow this link.

Additional resources

This document only touches on enough to get you up an running with reproducible work. However, to become fully proficient you will need to delve deeper into the material - trust me, it'll make your life easier in the long run. Here are a few places to start for each section, many of which were the basis for the systems I implement and advocate for here.

Project structure

https://nicercode.github.io/blog/2013-04-05-projects/

https://tomwallis.info/2014/01/16/setting-up-a-project-directory/

https://medium.freecodecamp.org/why-you-need-python-environments-and-how-to-manage-them-with-conda-85f155f4353c

This takes a deeper look into how to manage `python` environments with anaconda, and how this affects your project structures. This is useful if you are work with `python 2.x` and `python 3.x`, but also allows your to ensure old code won't get broken when modules are updated as each module is specific to the environment it is downloaded in.

Git

https://happygitwithr.com/

I cannot emphasise this enough: this is genuinely **the best resource** I have come across for explaining how to set up and organise a project with `git`. Whilst it is aimed at `R` users, there is a large amount of cross-over, so read it regardless of the language you use.

https://medium.freecodecamp.org/how-not-to-be-afraid-of-git-anymore-fe1da7415286

This helps you understand the nuts-and-bolts of `git` by learning to use the command line, rather than an application like SourceTree.