Chapter 4

Establishing a Reproducible and Sustainable Analysis Workflow

Jivesh Ramduny, Mélanie Garcia, and Clare Kelly

Abstract

Getting started on any project is often the hardest thing—and when it comes to starting your career in research, just figuring out *where* and *how* to start can seem like an insurmountable challenge. This is particularly true at this moment—when there are so many programming languages, programs, and systems that are freely available to neuroimaging researchers, and even more guides, tutorials, and courses on how to use them. This chapter is intended to set you off on the right foot as you get stuck into the task of learning to work with large neuroimaging data. We will cover a number of processes, systems, and practices that you should adopt to help ensure that your work is efficient, your processing steps traceable and repeatable, your analyses and findings reproducible, and your data and processing scripts amenable to sharing and open science. While this chapter is aimed at those getting started, it will also be of use to established researchers who want to streamline their processes and maximize robustness and reproducibility of their neuroimaging analyses. Finally, this chapter is also intended to help make neuroimaging work practices and processes more environmentally sustainable by reducing demands on computational resources through better planning, efficiency, and awareness of resource use.

Key words Reproducibility, BIDS, Docker, Python, Sustainability

1 Why Establish a Reproducible Workflow?

In the wake of the "replication crisis" in science [1, 2] reproducibility has become a cornerstone of neuroimaging research. The term *reproducibility* [*see* Glossary] refers to the ability to obtain the same results as a prior study, using procedures that are closely matched with those used in the original research [3]. As a result of this increased emphasis on reproducibility, a whole set of disciplinary norms have been instituted. It is now routinely expected that researchers will share not only their data and derivatives (to the extent that data protection regulations permit) but also their analysis code, so that others may reproduce their findings with the same

Authors Jivesh Ramduny and Mélanie Garcia have equally contributed to this chapter.

Robert Whelan and Hervé Lemaître (eds.), *Methods for Analyzing Large Neuroimaging Datasets*, Neuromethods, vol. 218, https://doi.org/10.1007/978-1-0716-4260-3_4, © The Author(s) 2025

data, or attempt to replicate findings using different data. Disorganized or idiosyncratically named data and esoterically written and uncommented code are of little use to anyone, including yourself, when you inevitably return to a project after a break (e.g., to address peer reviewers' comments). Putting a reproducible workflow in place from the outset will help ensure that your data and code are both reproducible and useful to yourself and to other researchers.

The practices we outline in this chapter may seem like a considerable investment at first—particularly when students are also just beginning to learn about their research topic itself. There can be an understandable urge to "just get stuck in", but, we can confirm—from personal experience—that investments made now will reap many benefits in the future. In contrast, cutting corners now may lead to heartache (and extra work) down the line. Not only will other researchers thank you for adopting these practices—but future "you" will also appreciate it!

Note
Time invested now in learning reproducible research practices will reap benefits throughout your career. Believe it or not, you'll never have more time to learn than as a PhD student!

1.1 FAIR Principles

Based on the recognition that data reuse is central not only to reproducibility but also to maximizing the value of research, a set of best-practice guidelines have been developed to maximize the usability of such data. These are known as the FAIR principles [*see* Resources]. The FAIR principles [*see* Glossary] prescribe characteristics of data and digital objects to maximize their reuse by the scientific community—that is, to maximize data sharing, exploration, reuse and deposition by parties other than the original researcher [4]. The application of the FAIR principles for neuroimaging data has been extensively and accessibly documented by ReproNIM [*see* Resources]—we recommend that you take time to explore their excellent module on Data and FAIR Principles. In brief, the FAIR principles require that data are:

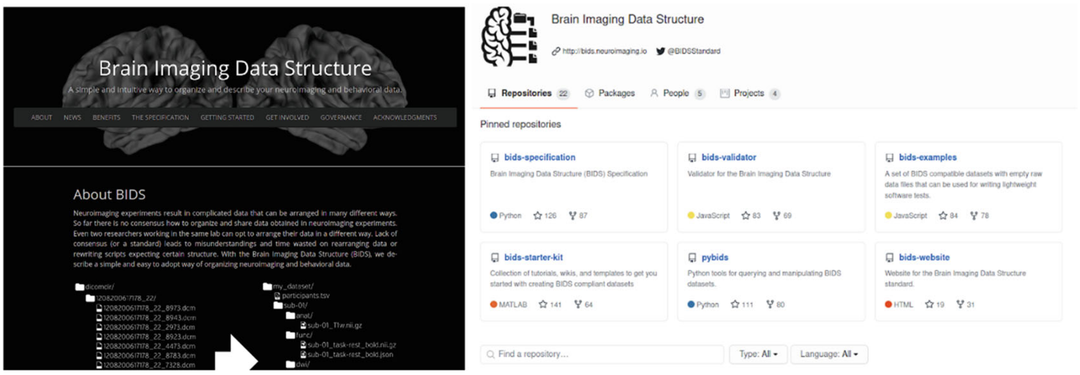
- **F**indable: Data and supplementary materials have sufficiently rich metadata [*see* Glossary] and a unique and persistent identifier (PID).
- **A**ccessible: Data are deposited in a trusted and accessible repository [*see* Glossary]. Both the data and metadata are accessible and downloadable via platforms such as OpenNeuro, Open Science Framework, github, amongst others [*see* Resources].

- **I**nteroperable: Data and metadata use a formal, agreed-upon and shared language or format such as the BIDS standard (explained in more detail below).
- **R**eusable: Data are described with clear and understandable attributes, and there should be a clear and acceptable license for reuse (e.g., CC0 public domain).

In the rest of this chapter, we outline how you can build reproducibility into your workflow. Our guide is intended to get you started, rather than to be exhaustive, so we also recommend that you also build on these basics by exploring other guides [5, 6] and resources (e.g., ReproNIM). Much of what we outline is simply good practice for keeping your data organized and maximally reusable for yourself and your collaborators, but it will become very important when you reach the point of publishing your study. If you wish to share your data using a public platform such as OpenNeuro (increasingly the norm for the field), then you must curate your dataset to comply with the FAIR principles and the BIDS standard. Next, we'll take a look at what this means.

2 Working with the BIDS Ecosystem

BIDS stands *for Brain Imaging Data Structure* [*see* Glossary]. It is a standard for the organization of neuroimaging datasets that follows FAIR principles and facilitates both data reuse and automated processing by open science data analysis pipelines [5]. The BIDS standard specifies machine-readable directory structure, filenames, file formats, and metadata for various neuroimaging modalities. The need for these standards arose as a result of increasing demand for data sharing in the field, and the difficulties created when such data are idiosyncratically named, organized, and formatted. The BIDS specifications, initially developed by those working with the OpenfMRI (now OpenNeuro) data repository [*see* Resources], are consensus-based and community-driven, and leverage existing conventions in the neuroimaging community (Fig. 1). They emphasize **simplicity**, **readability**, and **accessibility**. BIDS is now widely accepted as the standard for the field and includes specifications for modalities beyond MRI, including EEG and MEG [*see* Chapter 6]. There is an active BIDS community that welcomes involvement [*see* Resources]. Databases such as OpenNeuro.org, LORIS, COINS, XNAT, SciTran, and others will accept and export datasets organized according to BIDS, and some open-source software such as fMRIPrep, C-PAC, and MRIQC, works only or optimally with BIDS data [*see* Resources]. Making the BIDS standard a key feature of your reproducible workflow is therefore a very important first step!



About BIDS

Neuroimaging experiments result in complicated data that can be arranged in many different ways. So far there is no consensus how to organize and share data obtained in neuroimaging experiments. Even two researchers working in the same lab can opt to arrange their data in a different way. Lack of consensus (or a standard) leads to misunderstandings and time wasted on rearranging data or rewriting scripts expecting certain structure. With the Brain Imaging Data Structure (BIDS), we describe a simple and easy to adopt way of organizing neuroimaging and behavioral data.

```
dicomdir/
    1208200617178_22/
        1208200617178_22_8973.dcm
        1208200617178_22_8943.dcm
        1208200617178_22_2973.dcm
        1208200617178_22_8923.dcm
        1208200617178_22_4473.dcm
        1208200617178_22_8783.dcm
        1208200617178_22_7328.dcm
```

```
my_dataset/
    participants.tsv
    sub-01/
        anat/
            sub-01_T1w.nii.gz
        func/
            sub-01_task-rest_bold.nii.gz
            sub-01_task-rest_bold.json
        dwi/
```

bids-specification	Brain Imaging Data Structure (BIDS) Specification	Python	☆ 126 · 87
bids-validator	Validator for the Brain Imaging Data Structure	JavaScript	☆ 83 · 69
bids-examples	A set of BIDS compatible datasets with empty raw data files that can be used for writing lightweight software tests.	JavaScript	☆ 84 · 78
bids-starter-kit	Collection of tutorials, wikis, and templates to get you started with creating BIDS compliant datasets	MATLAB	☆ 141 · 64
pybids	Python tools for querying and manipulating BIDS datasets.	Python	☆ 111 · 80
bids-website	Website for the Brain Imaging Data Structure standard.	HTML	☆ 19 · 31

Fig. 1 BIDS

```
project/
└── subject
    └── session
        └── datatype
```

Fig. 2 BIDS dataset hierarchical organization

Let's take a more detailed look at how to build a BIDS dataset, and how to create and run compatible applications called BIDS apps [*see* Glossary]. From the outset, we should say that BIDS is extremely well documented (e.g., https://bids.neuroimaging.io/; https://github.com/bids-standard). Here, we highlight some of the most important features and resources for further information.

2.1 BIDS-Structured Data

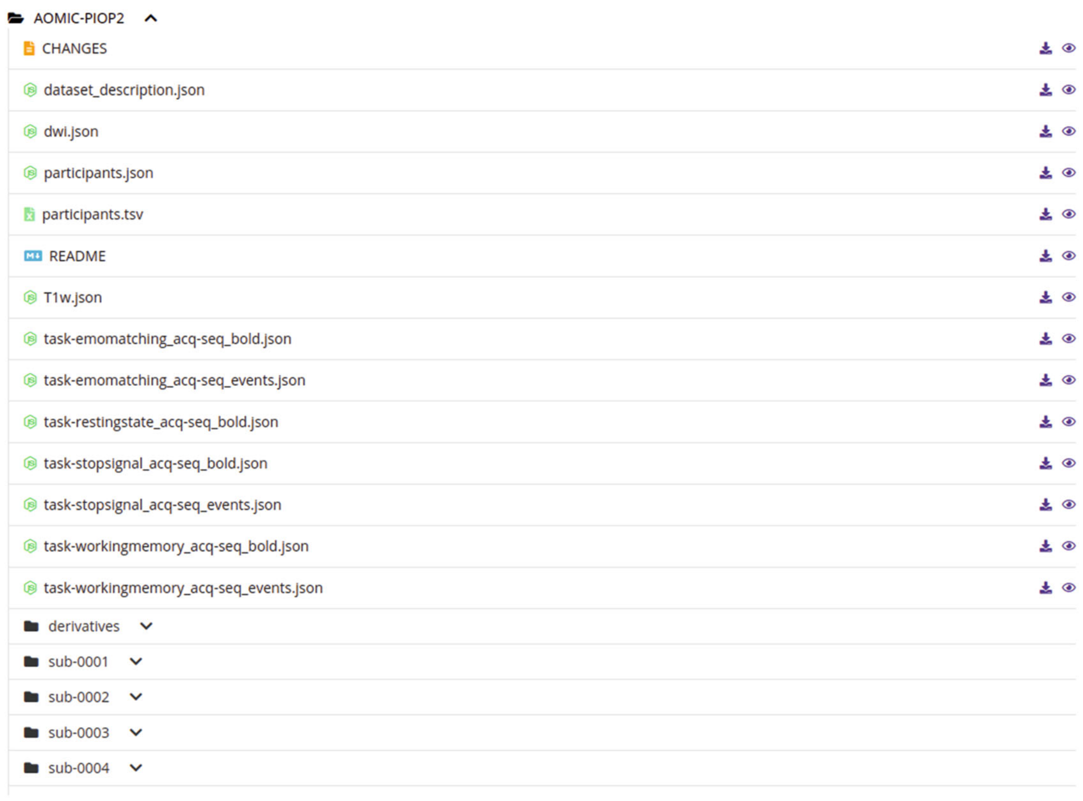
BIDS datasets are hierarchically organized to contain both modality-agnostic and modality-specific files. The directory organization follows four main levels of hierarchy, shown in Fig. 2.

The main directory "project" contains all of the project files while sub-directories contain subject-level data, which is further organized according to session (if data were collected over multiple sessions) and modality (e.g., anatomical, diffusion-weighted, functional, etc.). The session folder is not necessary when the study contains only one session of data per participant.

Let's illustrate BIDS in more detail using an example: the AOMIC-PIOP2 dataset [7], which can be downloaded from the OpenNeuro database [see Chapter 2].

The AOMIC-PIOP2 dataset is already BIDS-structured. Once you have downloaded the data, if you look inside the main folder (*see* Fig. 3), you will find:

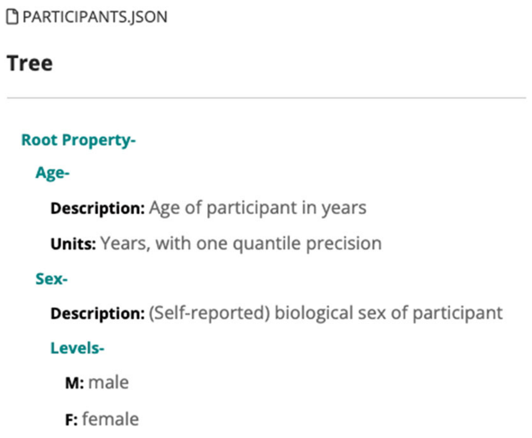
- **README**: The first file to open in order to quickly understand what the dataset contains; for instance, we can learn about the data types/modalities, and whether the dataset contains raw, preprocessed or processed data.



AOMIC-PIOP2
CHANGES
dataset_description.json
dwi.json
participants.json
participants.tsv
README
T1w.json
task-emomatching_acq-seq_bold.json
task-emomatching_acq-seq_events.json
task-restingstate_acq-seq_bold.json
task-stopsignal_acq-seq_bold.json
task-stopsignal_acq-seq_events.json
task-workingmemory_acq-seq_bold.json
task-workingmemory_acq-seq_events.json
derivatives
sub-0001
sub-0002
sub-0003
sub-0004

Fig. 3 Files and folders in the main folder of the AOMIC-PIOP2 dataset

- **dataset_description.json**: A json format text file [*see* Glossary] containing metadata on the dataset, including details such as the name, the authors, the version, a reference to a paper describing the data, etc.
- **participants.tsv**: A tab-delimited file (.tsv format—column separators are tabulations) that contains at least one column labelled "participant_id", which provides a unique identifier for each subject (i.e., person or animal) in the dataset. Further columns may contain other relevant information on each participant (e.g., s, gender, age, clinical status etc.). The first row of the file provides the column names.
- **participants.json**: A json text file that provides a legend for each column of participants.tsv. In the "Tree" section, the "Root Property" list can be expanded. Clicking on an item in the list (e.g., "Age" will expand the label to provide a Description and Units etc.). *See* Fig. 4.
- **CHANGES**: A timeline of all the updates/changes to the dataset.



PARTICIPANTS.JSON

Tree

Root Property-

Age-

Description: Age of participant in years

Units: Years, with one quantile precision

Sex-

Description: (Self-reported) biological sex of participant

Levels-

M: male

F: female

Fig. 4 Screenshot of an expanded Participants.json list

- There are also several modality-specific files:
  - dwi.json: Metadata on the diffusion-weighted MRI data,
  - T1w.json: Metadata on the T1-weighted MRI data.
  - There are several metadata files providing information on the fMRI, scans labelled according to experimental condition (emomatching, restingstate, stopsignal, etc.). Some of these (. . ._bold.json) provide information on the MRI scanning parameters themselves, such as slice timing information. For task-based fMRI there json files (and the experimental task events, such as response accuracy type (“correct”, “incorrect”, “miss”) (. . ._events.json):
    - task-emomatching_acq-seq_bold.json,
    - task-emomatching_acq-seq_events.json,
    - task-restingstate_acq-seq_bold.json,
    - task-stopsignal_acq-seq_bold.json,
    - task-stopsignal_acq-seq_events.json,
    - task-workingmemory_acq-seq_bold.json,
    - task-workingmemory_acq-seq_events.json,
- **sub-X folders**: There is one folder per participant, containing raw data (and potentially participant metadata in json files) (*see* Fig. 5).
- a **derivatives** directory [*see* Glossary]: This contains processed data. For each type of processing, the organization of the corresponding subdirectory mirrors that of the main folder: there is a group.tsv file containing information on the participants whose files were processed, and participants’ folders similarly named sub-X (*see* Fig. 6). The organization of each subject’s folder and of the folder **derivatives** is specific to each study.

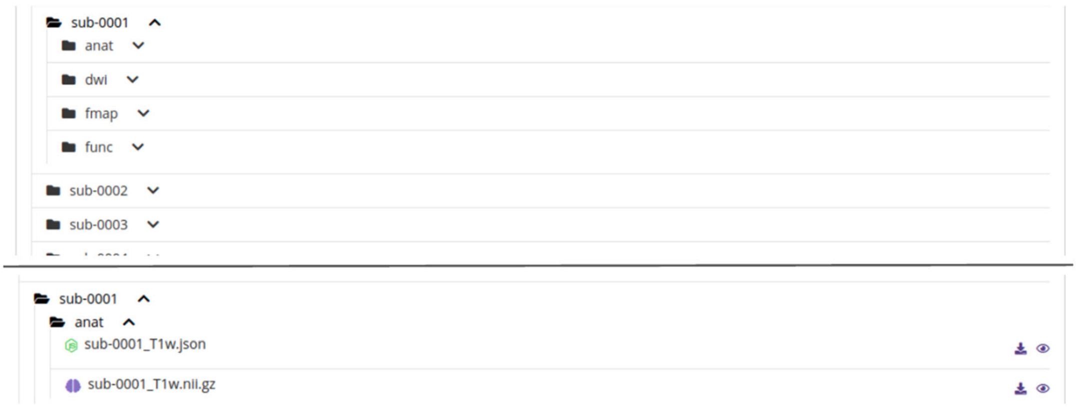


Fig. 5 The contents of each participant's folder "sub-X" of the AOMIC-PIOP2 dataset: (**a**) on the top panel, there are different subdirectories for every data modality (e.g., anat—anatomical; dwi—diffusion weighted imaging; func—functional); (**b**) the bottom panel shows the contents of the anat subdirectory

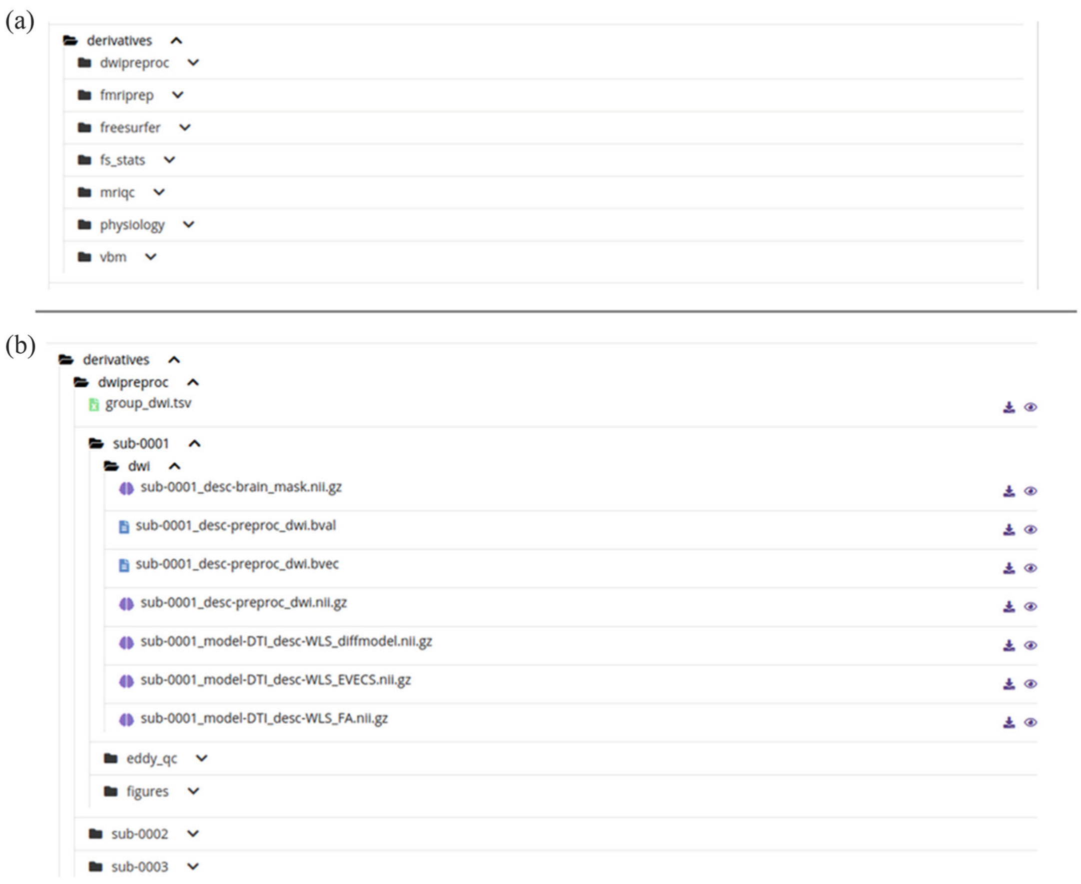

(a)
derivatives
dwipreproc
fmriprep
freesurfer
fs_stats
mriqc
physiology
vbm
(b)
derivatives
dwipreproc
group_dwi.tsv
sub-0001
dwi
sub-0001_desc-brain_mask.nii.gz
sub-0001_desc-preproc_dwi.bval
sub-0001_desc-preproc_dwi.bvec
sub-0001_desc-preproc_dwi.nii.gz
sub-0001_model-DTI_desc-WLS_diffmodel.nii.gz
sub-0001_model-DTI_desc-WLS_EVECS.nii.gz
sub-0001_model-DTI_desc-WLS_FA.nii.gz
eddy_qc
figures
sub-0002
sub-0003

Fig. 6 Contents of the "derivatives" folder of the AOMIC-PIOP2 dataset: (**a**) as the top-panel shows, preprocessed data derivatives are organized according to the process/program applied (e.g., dwipreproc); (**b**) the bottom panel shows the contents of one subdirectory of data processed using dwipreproc

2.2 Implementing BIDS

If you are working with Open Science data, obtained from a platform such as OpenNeuro, it is likely to already have a BIDS organization. However, if you are working with newly acquired data, or preexisting data in your lab, you may have to implement BIDS organization yourself.

This is easiest to do with new data. For MRI data, converting from DICOM (typical scanner output) to BIDS is relatively straightforward as there are BIDS converters [*see* Glossary]. The BIDS website provides a full list of conversion software for a variety of data modalities.

Note
Implement BIDS organization for your data from the beginning (ideally, when extracted/imported from the scanner). This is best practice and will save the harder work of having to convert your data later, when you want to share it.

If you have preexisting data in, for example, NIFTI format, complying with BIDS specifications is likely to be a matter of data organization/structure, relabelling, and the creation of the required .json files, tasks that can all be scripted (e.g., with python). A complete list of BIDS specifications for how a dataset should be organized are provided on the BIDS website. In addition, the BIDS Starter Kit [*see* Resources] provides a very good explanation of the folder and file names and formats, as well as pieces of code to create your own .tsv [*see* Glossary] or .json files [*see* Glossary] in Matlab, Python and R. The starter kit also includes a general template for a BIDS-dataset, and examples of BIDS-organized data.

Finally, once you have created your dataset, you should verify that it fully complies with BIDS structure, using the online **BIDS-validator** tool [*see* Resources]. To use the tool (which currently only supports Google Chrome and Mozilla Firefox), you browse to your data folders—the tool will check whether your file organization meets BIDS criteria. If you do not have a supported browser on your machine, you can use the command line version working with Node.js, the bids_validator Python package or the bids/validator Docker image. All these methods are described in detail on the **BIDS-validator** github pages.

2.3 Using BIDS Apps to Work with Data

Standardizing the organization of neuroimaging datasets has greatly accelerated the development of open-source data analysis packages and tools. A variety of tools have been developed to facilitate procedures from data handling and manipulation (e.g., PyBIDS and BIDS-Matlab) to fMRI data analysis (e.g., fMRIprep).

A particularly useful template of reproducible apps—called BIDS apps [*see* Resources]—has been developed by the BIDS community [8]. A **BIDS App** "*is an analysis pipeline that takes* ***a BIDS formatted dataset as input*** *and has* ***all of its dependencies packaged within a container*** *[see Glossary] (Docker or Singularity)*". This feature—the capture of all the dependencies of an app within a container—means that the app is not dependent on any software outside the container [*see* Chapter 3]. This removes what was traditionally a huge headache for users: the requirement to ensure that all dependencies were installed, that the version (i.e., release/revision number) installed was the correct one, etc.). Containers dramatically increase the reproducibility of studies by enabling users—and therefore reusers—to record and fix the version of an app (and all its dependencies) used in an analysis [*see* Chapter 3].

Before we dig in, let's quickly review some Docker usage. If you are not familiar with Docker, the BIDS community recommends that you view this tutorial (https://neurohackweek.github.io/docker-for-scientists/) and this video (https://www.slideshare.net/chrisfilo1/docker-for-scientists) on Docker and Singularity. The official "get started" Docker tutorial may help you too [*see* Resources]. **You will need to install Docker or Singularity before getting started with BIDS Apps**.

When you download an app using Docker (known as a *Docker image*), the entrypoint for interaction with that BIDS app is set to a preset script called /run.py. This script defines the required parameters for the BIDS app, which will vary by app but will include:

– **bids_dir**: The directory (full path) on your computer containing the input dataset organized according to BIDS standard.

– **output_dir**: The directory (full path) on your computer where the output files should be stored. If you are running a group level analysis, this folder should be prepopulated with the results of the participant-level analysis.

– **analysis_level**: Level of the analysis that will be performed (e.g., participant, group). Multiple participant level analyses can be run independently (in parallel).

– **--participant_label**: The label(s) of the participant(s) that should be analyzed. The label corresponds to sub-<participant_label> from the BIDS spec (so you should not include the "sub-" prefix). If this parameter is not provided, all the participants in the dataset will be analyzed. Multiple participants can be specified with a space-separated list.

In addition to these required parameters, you may add as many optional parameters as needed and appropriate to the app you are running. These parameters are visible in the script run.py of the BIDS app.

2.3.1 Worked Example

Let's illustrate how to run such a BIDS app with an example. In this example, we want to run **fMRIPrep**, which is a BIDS app that runs a customizable preprocessing pipeline for structural and functional MRI data analysis (*see*: https://fmriprep.org/en/stable/workflows.html). Here, we will run it on participants 0001 and 0002 of the AOMIC-PIOP2 dataset (which we downloaded in the previous example, Subheading 2.1).

Step 1 Get the fMRIPrep BIDS app.

In a shell on your laptop, computer, or server, run:

```
docker pull nipreps/fmriprep:latest
```

Here we call Docker's action `pull`, which finds the `latest` version of the Docker image `nipreps/fmriprep` on the DockerHub platform [*see* Resources] and copies it to your machine (this may take several minutes). The Docker image `nipreps/fmriprep` contains the scripts and installations for all the dependencies (specified in the Dockerfile) to be able to run the fMRIPrep pipeline under various conditions. For this reason, Docker images can be quite large (several GB).

If you want to use a specific version of a BIDS app (e.g., to reproduce analyses reported in a paper and performed with an older version of the BIDS app), you will find it on the DockerHub platform. For instance, if we want to use the version 1.5.10 of fMRIPrep, we can check if it is still openly shared in https://hub.docker.com/r/nipreps/fmriprep/tags. We can then adapt our docker command to pull this version instead of the latest: `docker pull nipreps/fmriprep:1.5.10`.

Step 2 Run fMRIPrep on specific participants.

The generic command line to run fMRIprep is as follows:

```
docker run -it --rm -v <freesurfer_licenses>:/opt/freesurfer/
license.txt -v <bids_dir>:/data:ro -v <output_dir>:/out -v
<temporary_dir>:/tmpnipreps/fmriprep:latest /data /out <ana-
lysis_level> -w /tmp --participant_label <label1 label2 la-
bel3>
```

Here, we are calling Docker's `run` action, which launches a container as a new instance of the *nipreps/fmriprep:latest* image. All fMRIPrep's dependencies are installed in the container.

Following `docker run`, we see several options are passed to the command:

`-it` is the combination of two options, -i and -t, that are the single character versions of --interactive (which keeps STDIN open even if not attached—here STDIN is the terminal window you are using to input the commands) and --tty (which allocates a pseudo-TTY connected to the container's STDIN). These two options are often passed when running `docker run` because most programs need to be run in -it mode.

`--rm` is another optional parameter that specifies that after the container exits (for instance, after the fMRIPrep pipeline is completed), the container will be automatically removed from your machine.

`-v` makes it possible for the container to interact with your machine filesystem. This is required because while a container uses computational and memory resources on your machine, it runs *independently* of the machine filesystem (it has its own filesystem which is defined in the Dockerfile). If we want the BIDS app to run on a BIDS dataset located on our machine, then we need to create "volumes" or "bind mounts" between the filesystems of our machine and of the container. In the command above, we use `-v` to specify three volumes:

- We link `<bids_dir>` on our machine with the directory `/data` in the container, specifying the mode `:ro` for read-only, to avoid saving files inside the BIDS directory. So everything in `<bids_dir>` will be accessible for reading from the container, in its directory `/data`.
- We link `<output_dir>` on our machine (a directory where the outputs will be saved) with the directory `/out` of the container. No mode is specified, so the default read-and-write mode is applied.
- We link `<temporary_dir>` on our machine (a directory where we want intermediate outputs of the pipeline to be stored) with the directory `/tmp` of the container. No mode is specified, so the default read-and-write mode is applied.

Note
To know what parameters you should specify when calling "docker run", check the documentation of the BIDS app you are using, or ask the creator!

For example, fmriprep uses freeSurfer tools, which require a license to run. To obtain a freeSurfer license, simply register for free at https://surfer.nmr.mgh.harvard.edu/registration.html.

Like all the BIDS apps, `nipreps/fmriprep` also takes parameters specific to the fMRIPrep pipeline like `-w`, a single character version of the parameter —work-dir, which specifies the directory where intermediate outputs should be stored. Here, we are passing */tmp* as a value for *-w*, which will save the intermediate results locally on our machine in *<temporary_dir>* thanks to the volume specified with the *-v* option.

A full list of the parameters you can pass to the BIDS app *nipreps/fmriprep* is given here: https://fmriprep.org/en/stable/usage.html#execution-and-the-bids-format.

Now we understand the basic ingredients of a call to a Docker container, let's look at a fully specified example, working with the AOMIC-PIOP2 dataset. In our case, the AOMIC-PIOP2 data is locally stored in /home/melanie/, so we replace the directory placeholders such as *<bids_dir>* with our specific directories, we specify "participant" for the analysis_level and provide our participant list. This gives us the command:

```
docker run -it --rm -v /home/melanie/AOMIC-PIOP2:/data:ro -v
-v $HOME/.licenses/freesurfer/license.txt:/opt/freesurfer/li-
cense.txt
/home/melanie/AOMIC-PIOP2/derivatives/fmriprep_test:/out -v /
home/melanie/AOMIC-PIOP2/derivatives/fmriprep_test/tmp:/tmp
nipreps/fmriprep:latest /data /out participant -w /tmp --
participant_label 0001 0002
```

Running this command will launch fMRIPrep on participants 0001 and 0002, store the intermediate results locally in the folder ***/home/melanie/AOMIC-PIOP2/derivatives/fmriprep_test/tmp*** and the final results in ***/home/melanie/AOMIC-PIOP2/derivatives/fmriprep_test***.

If you do not specify participants using the option—participant_label, the program will process all the subjects in the BIDS dataset.

Step 3 Run a group level analysis.

Once the participant-level analysis is completed for all participants (or the ones selected) in your dataset, you may need to run the analysis at the group-level:

```
docker run -it --rm -v /home/melanie/AOMIC-PIOP2:/data:ro -v /
home/melanie/AOMIC-PIOP2/derivatives/fmriprep_test:/out  -v /
home/melanie/AOMIC-PIOP2/derivatives/fmriprep_test/tmp:/tmp
nipreps/fmriprep:latest /data /out group -w /tmp
```

Notice that the command is identical, apart from the specification of analysis_level. Here, we omit a participant list, because we want all participants to be included.

This example shows how easy it is to work with your BIDS data using BIDS apps. There are many different BIDS apps [*see* Resources]. Once you have established your own analysis pipeline, you may even consider creating your own BIDS app. A template is provided here: https://github.com/BIDS-Apps/example.

More information on how to run BIDS apps, especially on HPC clusters, is provided (http://bids-apps.neuroimaging.io/tutorial/) and there is a very helpful FAQ (https://bids-standard.github.io/bids-starter-kit/apps.html). Don't forget to join the BIDS community (https://bids.neuroimaging.io/get_involved.html)!

3 Getting Started with Python Notebooks

If you have followed the steps above, you will now have some familiarity with the BIDS ecosystem, you'll have implemented BIDS structure for your data, and you'll have run your first analysis using a BIDS app. Now it's time to work with your data!

When performing reproducible research, one key recommendation is to use *open-source* tools, software, and programming languages, where possible. By using open-source tools, we remove the barrier of access to expensive, proprietary tools, and maximize the opportunity for any researcher to repeat our analyses, regardless of their resources. Python is an open-source programming language that has recently gained popularity across a broad spectrum of disciplines including neuroscience, psychology, biomedical science, and beyond, due to its intuitive platform and substantial computational capability for basic and complex analyses. While python can be used in traditional scripts, like other programming languages, **Python notebooks** offer a particularly useful way to write reproducible and portable code. Python notebooks make use of Markdown—a markup language that allows easy plain-text formatting (e.g., for adding comments, headings, bold, italics, hyperlinks, etc.). Notebooks are also commonly used with another open-source program for statistical analysis, R.

Python can be installed either alone or using a platform like Anaconda. Installing Jupyter Notebook will enable notebook use on your own local computer, but, alternatively, Google Colaboratory offers an easy web-based platform for creating and running Python notebooks, for which no local installation of Python is required. Google Colab also offers free access to (carbon neutral) GPUs and facilitates the storage and sharing of code using Google Drive. For large projects or analyses of large neuroimaging volumes, however, it is preferable to have your own local installation of Python, since Google Colab has resource limitations (12GB RAM) and may not be scalable for very large datasets. For analysis of smaller datasets, or less resource-demanding analyses of time series or behavioral data, this resource limitation should not be a problem. The example notebook included with this book chapter has been written using Jupyter Notebook (installed locally), but it can also easily be used in Google Colab.

Before we take a look at our example, there are several Python packages that you are likely to use frequently for reproducible neuroimaging and behavioral analyses. These are very well documented and supported by tutorials that are easily found online. These include [*see* Resources]:

- numpy: Numerical package for scientific and arithmetic computing,
- pandas: Supports data manipulation and analysis, particularly useful for reading data in csv/xls format,
- nilearn: Package supporting neuroimaging analyses of structural and functional volumetric data. Includes tools for voxelwise statistical analyses, multi-voxel pattern analysis (MVPA), GLMs, clustering and parcellation, etc.
- scikit-learn: Tools for machine learning, including classification, model selection, and dimensionality reduction,
- https://nltools.org: Neuroimaging package for fMRI data analyses (e.g., resting-state, task-based, movie-watching) that incorporates code from nilearn and scikit-learn,
- statsmodels: Package to build statistical models and perform statistical tests,
- pingouin: Simple statistical functions and graphics,
- matplotlib and seaborn: Tools for data visualization.

3.1 Writing and Sharing Code

As you begin your analysis, it is important to think about future **code reuse and sharing**—so that other researchers (including "future you") can reproduce your analyses and findings. Because neuroimaging analyses are often complex in nature, with many different parameters, each with a number of possible settings, it can be very difficult to reproduce a workflow or set of results if code/scripts are not provided with the published paper. For this reason, when writing code, one of the most important things to think about is whether someone else will be able to look at your code, understand what is being done, and reproduce or adapt the analyses themselves.

The idea of code sharing can be anxiety-inducing—researchers often fear that their code will contain errors that may be identified by more experienced researchers and programmers. While this is an understandable and common fear, practice helps. Learning to write reproducible code is just like learning a second language or a musical instrument—if you don't practice regularly, you will not master the ability to speak fluently or play coherently. If you have never written Python code before, it can be daunting to start. We are not going to cover the basics and mechanics here, since there are many excellent resources available online. If you are completely new to Python, you might want to consider some of the beginner's tutorials and guides (https://wiki.python.org/moin/BeginnersGuide/Programmers) linked to on the Python wiki (https://wiki.python.org/moin/), or one of the MOOCS offered by Coursera, edX, Udemy, and others. In what follows, we're going to highlight some of the most important things you should know. We have also created an **example notebook** (https://osf.io/fye48) to accompany this chapter, which illustrates some of the concepts and practices we outline below.

First, there are some basic good practices that all programmers can follow to ensure their code is well-structured and clean (avoid "spaghetti code"), clear (without ambiguous or confusing variable names), and concise (avoiding unnecessary loops). You should check out this excellent guide from the Alan Turing Institute (https://the-turing-way.netlify.app/reproducible-research/reproducible-research.html), python best practices (https://towardsdatascience.com/5-python-best-practices-every-python-programmer-should-follow-3c92971ed370), and, although very detailed, you should also familiarize yourself with the official Python Style Guide (https://peps.python.org/pep-0008/), including best practice naming conventions (https://peps.python.org/pep-0008/#naming-conventions).

Beyond following best practices for programming style, the best way to ensure usability of your code is to **#comment** it. Commenting means adding short (one-line) explanatory notes to your code, preceded by a designated comment symbol—the # symbol is used for this purpose in many coding languages, including Python. The comment notes explain the intent of the code, or summarize what the code does or refers to. Any time you write code, you should try to comment it as much as possible—not just for others but for "future you". We are all susceptible to the myth that we will successfully recall the "what" and "why" of our code at a later date, but the unfortunate truth is that even highly experienced programmers return to uncommented code after a period of time and can't decipher it. To save heartache down the line, you should carefully comment your code as you write it. One useful recommendation from Kirstie Whitaker of The Alan Turing Institute is that comments should represent about 40% of your code. In our example notebook (https://osf.io/fye48), we provide lots of examples of how to comment code. Because notebooks offer an easy means for separation of code and mark-up (plain-text formatting), they allow for detailed annotation of your analyses, which will maximize reproducibility and reuse.

Note Always comment your code, to make it accessible to your (future) self and others! By allowing for markdown plain-text sections as well as code, notebooks have made this easier and tidier than ever.

Another important coding skill is **debugging**—figuring out why your code is not working or is producing the wrong output. As with code-writing in general, debugging is a skill that develops over time and involves a lot of trial-and-error (trying something, seeing if it works, trying something else), combined with some intrepid Googling skills. Believe it or not, the default way for even quite experienced programmers to understand errors in their code is to copy/paste the error message into Google! This works because it is very likely that your error has been encountered and posted about before—a quick Google search often reveals the solution, on platforms such as Stack Overflow [*see* Resources].

Note
Even the best programmers use Google to help debug and find solutions to errors!

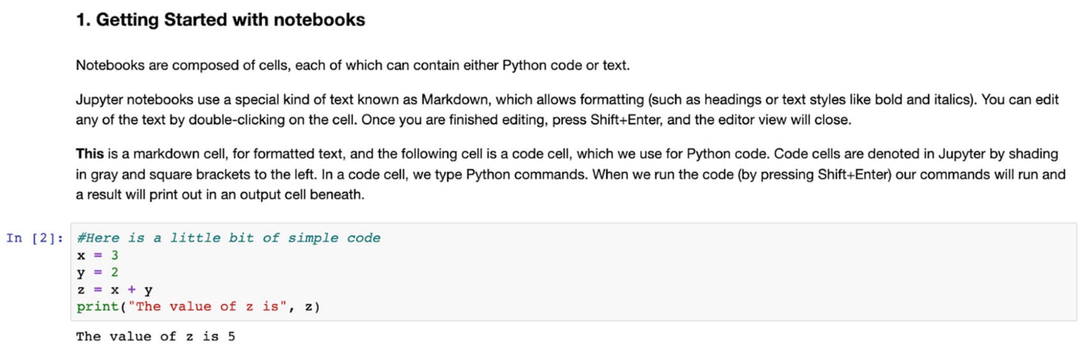
3.2 Planning and Implementing Your Analysis

When it comes to implementing a reproducible and sustainable analysis pipeline, it is important to carefully consider and select the preprocessing steps for your raw imaging data, so that this step can be run **only once**. This is because it is data preprocessing—steps such as motion correction and normalization to standard space—that consume the majority of computational resources and therefore energy consumption. Thankfully, the development of standardized preprocessing pipelines such as fMRIPrep [9], C-PAC [10]), and NiPreps, amongst others [*see* Resources], has the promise of not only harmonizing analyses and increasing their reproducibility, but also increasing the attractiveness of sharing and using preprocessed rather than raw data—a crucial next step in reducing the carbon footprint of neuroimaging.

Our example notebook (https://osf.io/fye48) works with data preprocessed using fMRIPrep, described above. The example works with time series extracted using a standard functional parcellation of the brain into Regions of Interest (ROIs). Working with ROI time series, rather than voxelwise data, saves both time and computational resources, because the dimensionality of the data is considerably reduced—from ~200,000 voxels × *n_timepoints* to 268 (ROIs) × *n_timepoints*. One drawback of this approach is the fine detail offered by high resolution data may be lost—the specific approach adopted and level of dimensionality required should be selected based on the goals of your analysis.

Here, we highlight some of the features of the notebook.

1. Notebook basics: For those who are completely new to Python notebooks, we introduce and demonstrate Markdown and code cells (Fig. 7).
2. Examples of good coding practice—commenting, variable names, and style (Fig. 8).
3. Using neuroimaging packages to work with voxelwise and time series data (Fig. 9).
4. An example functional connectivity analysis with data from the AOMIC-PIOP2 dataset (Fig. 10).



1. Getting Started with notebooks

Notebooks are composed of cells, each of which can contain either Python code or text.

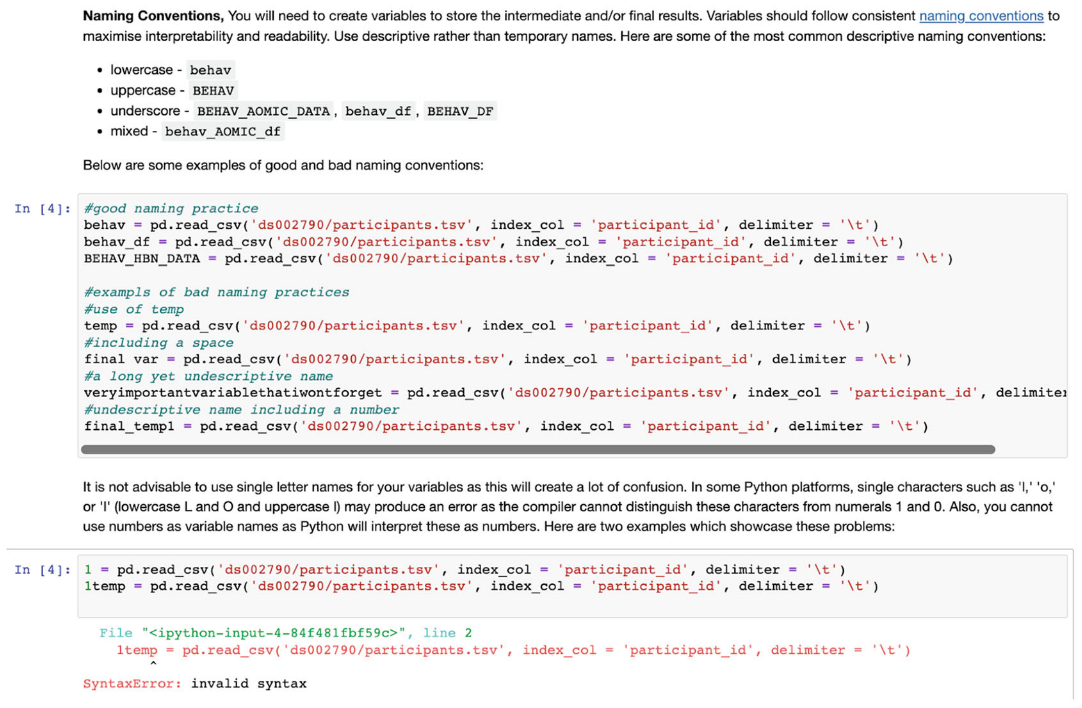
Jupyter notebooks use a special kind of text known as Markdown, which allows formatting (such as headings or text styles like bold and italics). You can edit any of the text by double-clicking on the cell. Once you are finished editing, press Shift+Enter, and the editor view will close.

This is a markdown cell, for formatted text, and the following cell is a code cell, which we use for Python code. Code cells are denoted in Jupyter by shading in gray and square brackets to the left. In a code cell, we type Python commands. When we run the code (by pressing Shift+Enter) our commands will run and a result will print out in an output cell beneath.

```
In [2]: #Here is a little bit of simple code
        x = 3
        y = 2
        z = x + y
        print("The value of z is", z)

        The value of z is 5
```

Fig. 7 Code and markdown cells in a Python notebook



Naming Conventions, You will need to create variables to store the intermediate and/or final results. Variables should follow consistent naming conventions to maximise interpretability and readability. Use descriptive rather than temporary names. Here are some of the most common descriptive naming conventions:

- lowercase - `behav`
- uppercase - `BEHAV`
- underscore - `BEHAV_AOMIC_DATA`, `behav_df`, `BEHAV_DF`
- mixed - `behav_AOMIC_df`

Below are some examples of good and bad naming conventions:

```
In [4]: #good naming practice
        behav = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')
        behav_df = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')
        BEHAV_HBN_DATA = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')

        #exampls of bad naming practices
        #use of temp
        temp = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')
        #including a space
        final var = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')
        #a long yet undescriptive name
        veryimportantvariablethatiwontforget = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimite
        #undescriptive name including a number
        final_temp1 = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')
```

It is not advisable to use single letter names for your variables as this will create a lot of confusion. In some Python platforms, single characters such as 'l,' 'o,' or 'I' (lowercase L and O and uppercase I) may produce an error as the compiler cannot distinguish these characters from numerals 1 and 0. Also, you cannot use numbers as variable names as Python will interpret these as numbers. Here are two examples which showcase these problems:

```
In [4]: 1 = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')
        1temp = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')

          File "<ipython-input-4-84f481fbf59c>", line 2
            1temp = pd.read_csv('ds002790/participants.tsv', index_col = 'participant_id', delimiter = '\t')
             ^
        SyntaxError: invalid syntax
```

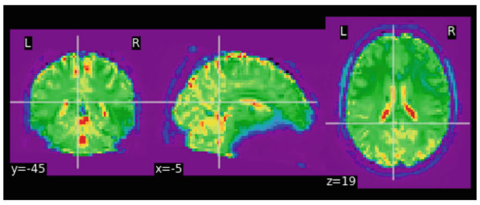
Fig. 8 Some examples of good and bad coding practices

Once we have the resulting 3D nifti image, we can plot it with different functions such as `view_img` and `plot_epi`. While both functions plot segments of an EPI image, the former provides an interactive HML viewer of the image and the latter provides a static cut of the image as shown below.

```
In [13]: from nilearn.plotting import plot_epi

         #plot the 3D nifti image in a static format
         plot_epi(func_3D)

Out[13]: <nilearn.plotting.displays.OrthoSlicer at 0x7fd384fc97c0>
```



We can also create a brain mask from the 3D nifti image of a participant using the `compute_epi_mask` function as part of the `nilearn` package. Once you generate a brain mask, you can easily plot it as an ROI using `plot_roi`.

```
In [14]: from nilearn.masking import compute_epi_mask
         from nilearn.plotting import plot_roi

         #create a brain mask using the 3D nifti image of the first participant and plot it as an ROI
         mask_img = compute_epi_mask(func_3D)
         plot_roi(mask_img, func_3D)
```

Fig. 9 Working with neuroimaging packages and data

8. Computing Whole-Brain Functional Connectivity (Approach 1)

```
Now that we have the preprocessed fMRI timeseries using the Shen 268 parcellation, we can derive the FC matrix of
each participant by computing the Pearson's correlation coefficient (PCC) between all possible ROI pairs to construct
a 268 x 268 FC matrix. The resultant correlation values represent the connectivity strengths (i.e., edges) between
two ROIs (i.e., nodes).
```

```
In [34]: all_subs_RSFC = []

         for sub in range(len(clean_timeseries)):

             #compute FC matrix of each participant using np.corrcoef
             timeseries_2d = clean_timeseries[sub]
             RSFC = np.corrcoef(timeseries_2d.T)
             #append the FC matrix of all participants to a list
             all_subs_RSFC.append(RSFC)

         all_subs_RSFC_3D = np.array(all_subs_RSFC)
         #compute the mean correlation across the participant to produce a symmetrical 268 x 268 FC matrix
         mean_correlations = np.mean(all_subs_RSFC_3D, axis = 0).reshape(all_subs_RSFC_3D.shape[1], all_subs_RSFC_3D.shape[1])
         #plot the symmetrical 268 x 268 FC matrix with the colorbar showing the correlation ranging from -1 to 1
         plotting.plot_matrix(mean_correlations, vmax = 1, vmin = -1, colorbar = True, title = 'Whole-brain FC')

Out[34]: <matplotlib.image.AxesImage at 0x7f939881df10>
```

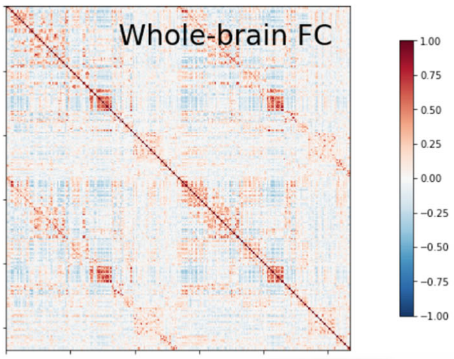


Fig. 10 Performing an analysis with the AOMIC-PIOP2 dataset

4 Thinking About Sustainability for Your Workflow

The twin climate and biodiversity crises constitute the greatest challenge that we and our planet have ever faced. It is a challenge of our own making—the unprecedented changes in our climate and devastating destruction of ecosystems are a direct result of human behavior and the social, political, and economic systems and structures we have created. Addressing the planetary crisis—by mitigating our changing climate, stalling biodiversity loss, restoring nature, and learning to thrive within planetary boundaries—requires urgent and collective action at all levels of society.

The trouble is that most of us don't know where or how to start. As neuroimagers, neuroscientists, psychologists, researchers, academics, we doubt what contribution we could possibly make in our professional lives to addressing our planetary crisis. We worry that we don't have the right expertize, that we need to change too much about what we do, or that any small changes we make won't have an impact on such a huge problem. But in reality, there's lots we can do—as recent papers from Aron et al. [11], and Rae et al. [12] have compellingly outlined. The actions range from reducing air travel (e.g., for conferences), teaching or doing research on the crisis, being aware of the environmental impact and sustainability of helium extraction, engaging in advocacy and activism, including nonviolent direct action, to simply talking about the planetary crisis with your colleagues, friends, and family. Relevant to the goals of this chapter, one of the simplest and most immediate things we can all do is to reduce the energy usage and associated environmental impact of our data analyses. Another important action neuroimagers at all career stages can take is to better inform themselves about the origins and consequences of the planetary crisis, as well as potential solutions. In addition to the papers by Aron et al. [11], and Rae et al. [12], which review environmental issues in neuroimaging and neuroscience research, scientific computing, and our field's conferences, and propose practical steps towards sustainability, we recommend recent articles by Keifer & Summers [13] and Zak et al. [14]. Looking beyond our field to the broader picture, we strongly recommend Naomi Klein's *This Changes Everything* [15], Jason Hickel's *Less is More—How Degrowth Will Save the World* [16], and the podcasts Drilled (https://www.drilledpodcast.com/drilled-podcast/), Scene on Radio (fifth season—The Repair: http://www.sceneonradio.org/the-repair/), and Upstream (https://www.upstreampodcast.org/).

The advice contained in this chapter is intended to set you off on the right foot in terms of implementing an efficient and effective workflow that minimizes unnecessary resource use. In the next chapter, we cover Git, which can also help with this. You might also want to actively monitor your and your lab's energy use. Just a few years ago, the task of measuring the resource demands and carbon footprint of neuroimaging processing pipelines would have been exceedingly difficult. Luckily, in 2020, the Organization for Human Brain Mapping Sustainability and Environmental Action Special Interest Group (SEA-SIG, https://ohbm-environment.org/) was launched, in recognition of the need to reduce the environmental impact of the organization, its conference, and its members. As one of its first actions, the SEA-SIG established a working group focused on assessing the environmental impact of neuroimaging processing pipelines. The group has developed several carbon tracker toolboxes, based on existing utilities (Code Carbon, EIT) that monitor CPU and GPU resource use during data processing [*see* Resources]. For example, you can monitor the carbon footprint of your BIDS app by following this excellent tutorial (https://github.com/sebastientourbier/tutorial_carbonfootprint_neuropipelines). There are also guidelines (https://github.com/nikhil153/fmriprep/blob/carbon-trackers/singularity/carbon_trackers_readme.md) for measuring the impact of the way you use fMIPrep, and if you use Deep Learning models, this carbon tracker (https://github.com/lfwa/carbontracker/) can help you perform energy-optimized model training and selection procedures. Facing up to the planetary crisis is a challenging task that is made easier when you are part of a community. Finding like-minded folks to meet, talk, and take action with is one of the best things you can do to support yourself. Whether it's in your university, local community, you'll find that climate action is easier when you can draw inspiration, motivation, solidarity, support, information and resources from others, and give back in return. For neuroimagers, a global community like the SEA-SIG is particularly relevant. The SEA-SIG Neuroimaging Research Pipeline WG have shared their work on quantifying the carbon footprint of neuroimaging tools and their vision for making our processing pipelines greener by improving their reproducibility and sustainability (https://neuropipelines.github.io/)—a vision we support, share, and express throughout this chapter.

Note

We can all make concern for the climate and biodiversity crisis part of who we are and how we work. All actions matter, and those that scale to collective action and social norms are the most powerful.

References

1. Munafò MR, Nosek BA, Bishop DVM, Button KS, Chambers CD, du Sert NP, Simonsohn U, Wagenmakers E-J, Ware JJ, Ioannidis JPA (2017) A manifesto for reproducible science. Nat Hum Behav 1:0021. https://doi.org/10.1038/s41562-016-0021
2. Poldrack RA, Baker CI, Durnez J, Gorgolewski KJ, Matthews PM, Munafò MR, Nichols TE, Poline J-B, Vul E, Yarkoni T (2017) Scanning the horizon: towards transparent and reproducible neuroimaging research. Nat Rev Neurosci 18:115–126. https://doi.org/10.1038/nrn.2016.167
3. Goodman SN, Fanelli D, Ioannidis JPA (2016) What does research reproducibility mean? Sci Transl Med 8:341ps12. https://doi.org/10.1126/scitranslmed.aaf5027
4. Wilkinson MD, Dumontier M, Aalbersberg IJJ, Appleton G, Axton M, Baak A, Blomberg N, Boiten J-W, da Silva Santos LB, Bourne PE, Bouwman J, Brookes AJ, Clark T, Crosas M, Dillo I, Dumon O, Edmunds S, Evelo CT, Finkers R, Gonzalez-Beltran A, Gray AJG, Groth P, Goble C, Grethe JS, Heringa J, 't Hoen PAC, Hooft R, Kuhn T, Kok R, Kok J, Lusher SJ, Martone ME, Mons A, Packer AL, Persson B, Rocca-Serra P, Roos M, van Schaik R, Sansone S-A, Schultes E, Sengstag T, Slater T, Strawn G, Swertz MA, Thompson M, van der Lei J, van Mulligen E, Velterop J, Waagmeester A, Wittenburg P, Wolstencroft K, Zhao J, Mons B (2016) The FAIR guiding principles for scientific data management and stewardship. Sci Data 3:160018. https://doi.org/10.1038/sdata.2016.18
5. Gorgolewski KJ, Poldrack RA (2016) A practical guide for improving transparency and reproducibility in neuroimaging research. PLoS Biol 14:e1002506. https://doi.org/10.1371/journal.pbio.1002506
6. Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013) Ten simple rules for reproducible computational research. PLoS Comput Biol 9:e1003285. https://doi.org/10.1371/journal.pcbi.1003285
7. Snoek L, van der Miesen MM, Beemsterboer T, van der Leij A, Eigenhuis A, Steven Scholte H (2021) The Amsterdam open MRI collection, a set of multimodal MRI datasets for individual difference analyses. Sci Data 8:85. https://doi.org/10.1038/s41597-021-00870-6
8. Gorgolewski KJ, Alfaro-Almagro F, Auer T, Bellec P, Capotă M, Chakravarty MM, Churchill NW, Cohen AL, Craddock RC, Devenyi GA, Eklund A, Esteban O, Flandin G, Ghosh SS, Guntupalli JS, Jenkinson M, Keshavan A, Kiar G, Liem F, Raamana PR, Raffelt D, Steele CJ, Quirion P-O, Smith RE, Strother SC, Varoquaux G, Wang Y, Yarkoni T, Poldrack RA (2017) BIDS apps: improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods. PLoS Comput Biol 13:e1005209. https://doi.org/10.1371/journal.pcbi.1005209
9. Esteban O, Markiewicz CJ, Blair RW, Moodie CA, Isik AI, Erramuzpe A, Kent JD, Goncalves M, DuPre E, Snyder M, Oya H, Ghosh SS, Wright J, Durnez J, Poldrack RA, Gorgolewski KJ (2019) fMRIPrep: a robust preprocessing pipeline for functional MRI. Nat Methods 16:111–116. https://doi.org/10.1038/s41592-018-0235-4
10. Craddock C, Sharad S, Brian C, Ranjeet K, Satrajit G, Chaogan Y, Li Q, Daniel L, Vogelstein J, Burns R, Stanley C, Mennes M, Clare K, Adriana D, Castellanos F, Michael M (2013) Towards automated analysis of connectomes: the configurable pipeline for the analysis of connectomes (C-PAC). Front Neuroinform 7. https://doi.org/10.3389/conf.fninf.2013.09.00042
11. Aron AR, Ivry RB, Jeffery KJ, Poldrack RA, Schmidt R, Summerfield C, Urai AE (2020) How can neuroscientists respond to the climate emergency? Neuron 106:17–20. https://doi.org/10.1016/j.neuron.2020.02.019
12. Rae CL, Farley M, Jeffery KJ, Urai AE (2022) Climate crisis and ecological emergency: why they concern (neuro)scientists, and what we can do. Brain Neurosci Adv 6:23982128221075430. https://doi.org/10.1177/23982128221075430
13. Keifer J, Summers CH (2021) The neuroscience community has a role in environmental conservation. eNeuro 8.:ENEURO.0454-20.2021. https://doi.org/10.1523/ENEURO.0454-20.2021
14. Zak JD, Wallace J, Murthy VN (2020) How neuroscience labs can limit their environmental impact. Nat Rev Neurosci 21:347–348. https://doi.org/10.1038/s41583-020-0311-5
15. Klein N (2015) This changes everything: capitalism vs. the climate, First Simon&Schuster trade paperback edition. Simon & Schuster Paperbacks, New York
16. Hickel J (2020) Less is more: how degrowth will save the world. Penguin Random House