Flywheel Tools

Data Curation and Manipulation on the Flywheel Platform

Tinashe M. Tapera, MSc Matthew Cieslak, PhD
Azeez Adebimpe, PhD Max Bertolero, PhD
Theodore Satterthwaite, PhD

Introduction

As the field of neuroimaging continues to grow, datasets being collected have seen similar growth in volume and variety. This expansion provides researchers with novel opportunities to answer questions using much larger sample sizes, but also presents challenges to reproducibility. Researchers carrying out different tasks on different operating systems, while attempting to reproduce an analysis, can often end up with disparate and irreplicable results (Vaccarino et al. 2018). Two major advances in the field offer ways of improving reproducibility: imaging databases, and Brain Imaging Data Structure (BIDS).

Imaging databases have have become powerful software tools for storing, accessing, and analyzing imaging data in a reproducible manner. They reduce considerable computational and storage overhead by ensuring data is stored and structured efficiently. Without the technical experience and expertise, finding appropriate resources for storage space and computational efficiency can be a challenging and laborious task. Importantly, many imaging databases provide at least two critical features for improving reproducibility. The first is data provenance, allowing all changes made to a dataset to be tracked and recorded at all times. The second is metadata provenance, allowing fine grain tracking of who applied changes, what software was used, and when. Notable examples of these databases include COINS (Landis et al. 2016), XNAT (Herrick et al. 2016), LORIS (Vaccarino et al. 2018), and others ((Book et al. 2016), (Rogovin et al. 2020), (Vaccarino et al. 2018), (Poldrack and Gorgolewski 2017)). Flywheel, the focus of this paper, is a recent addition to this list. In addition to the fundamental features above, Flywheel's model also strongly emphasizes the use of flexible data storage, prepackaged data analysis pipelines, and highly accessible provenance for both data and metadata.

In addition to the rise of database options, the neuroimaging community has also developed data standards — most notably the Brain Imaging Data Structure (BIDS) — to improve scientific reproducibility. BIDS is an open source standard

for neuroimaging data storage that specifies how files should be named and how directories should be organized. In addition, BIDS' directory structure allows users to read and interpret both data and metadata clearly and easily (K. J. Gorgolewski et al. 2016). The benefits of adopting BIDS include minimizing curation efforts, as datasets can be easily understood, operated on, and shared within and between sites; reduced usage error, as important metadata are explicitly declared in each file's name and path; and access to software tools and analysis pipelines, called BIDS apps, that can automatically configure themselves to the data as necessary, since the metadata is machine-readable.

BIDS was initially developed at Stanford in 2015, and through open-source community participation, it has grown into a globally recognised neuroimaging standard. The BIDS Steering Group publishes an official *BIDS Specification* for the community, which defines the current accepted practices and schemas. Development of this BIDS Specification is governed by both the democratically elected Steering Group, as well as input from the neuroimaging community. This open governance strategy allows the BIDS specification to be quickly updated to meet the needs of the latest imaging techniques and sequences, and adopted instantly within the community. Imaging databases like Flywheel are able to adopt BIDS standards and BIDS apps as core functionalities that are reliable, flexible to change, and serve to increase reproducibility. A wide array of BIDS apps are available in the BIDS ecosystem – notably, the BIDS validator is a common tool for validating datasets after they have been curated into BIDS (C. Gorgolewski et al. 2020).

To further reduce error and increase reproducibility, researchers can choose to curate their datasets into BIDS programmatically with version-controlled software. The most common BIDS curation tool is `HeuDiConv` **CITE heudiconv website? they don't have a DOI**, (shorthand for *Heuristic DICOM Converter*). `HeuDiConv` curates data using simple heuristics that map DICOM files to BIDS paths and file names. These heuristics are defined in machine-readable Python scripts, allowing them to be version controlled for reproducibility. For the typical user, BIDS curation with `HeuDiConv` follows a simple, yet highly flexible workflow using `HeuDiConv` commands and the BIDS validator tool. For the typical user, the first step is to scrape DICOM metadata from the dataset. Having done this, the user must examine the metadata to find scan parameters that discriminate between groups of files. These discriminating parameters can be used to map DICOMs to the desired filename and path the user expects the scan should be assigned to in BIDS. This mapping procedure is designed programmatically in a *heuristic script* using Python. Discriminations are specified using boolean logic. After creating the heuristic, the user then uses `HeuDiConv` functions to convert DICOMs to NIfTI files and apply the mapping.

After this process, the user can make use of the BIDS validator separately, to ensure the data is in correct BIDS format.

`HeuDiConv` establishes a robust and reproducible workflow for BIDS curation that has been widely adopted throughout the neuroimaging community. However,

although BIDS is supported on Flywheel, the platform's functionality only offers automated BIDS curation of incoming datasets, without the option to effectively and flexibly curate data using a `HeuDiConv`-like workflow. While Flywheel comes equipped with minimal utility for automated curation into BIDS, its limitations have led to BIDS curation delays in critical scenarios, such as when importing legacy data that adhered to previous standards, importing novel modalities that are yet to be defined in the BIDS specification, or importing datasets with significant errors that cannot be handled by the automated system.

In order to meet the need for a more reproducible and reliable BIDS curation workflow on Flywheel, this paper introduces the `FlywheelTools` array of packages: a software toolkit designed to implement a more flexible BIDS curation workflow on the Flywheel platform, and expand on current Flywheel functionality for addressing BIDS curation. In this paper, we describe the Flywheel platform, BIDS, and the software design approach in further detail in the Methods section. In the Results section, we demonstrate a reproducible BIDS curation workflow on Flywheel. `FlywheelTools` brings an array of powerful functionality for curating data on the Flywheel database and interacting with BIDS data on the platform, whilst maintaining the simplicity and usability of the common `HeuDiConv` workflow and expanding on Flywheel's BIDS functionality.

Methods

The FlywheelTools toolkit allows users to follow a reproducible workflow for BIDS curation and audit of their data. This workflow typically includes: inspection of sequences collected during a study; design of a curation schema; implementation of curation schema; curated data inspection and export; and finally audit of data and analyses.

Programming Languages & Technologies

Flywheel Tools is built primarily in Python 3.6 (Van Rossum and Drake 2009) in order to leverage its highly accessible SDK. Additionally, R 3.4.1 (R Core Team 2019) is used for HTML report generation. For reproducibility and workflow management, Flywheel Tools' modules are packaged in version-controlled software containers built and managed in Docker (Merkel 2014). Lastly, the Flywheel Tools package relies on users adopting the Brain Imaging Data Structure (BIDS) to curate their data. BIDS has rapidly evolved to become the field standard directory standard in the neuroimaging community for reproducible data organization (K. J. Gorgolewski et al. 2016). Importantly, BIDS has a large community of participants contributing to its development and adoption, and proposals for BIDS schema go through rigorous community testing and approval before being added. This leads to a trustworthy common framework for data curation shared amongst researchers. Software developers leverage this ubiquity by creating analysis and processing pipelines that can operate directly on BIDS

data sets as inputs, enhancing reproducibility and interoperability on various software platforms.

Flywheel

Flywheel is a data management and analysis platform for research, which lends itself well to neuroimaging. The platform focuses heavily on collaborative and reproducible science. User-facing components of the platform itself are the web User Interface (UI), the Software Development Kits (SDK), and the Application Programming Interface (API) (**Figure 1**).

Flywheel Web UI

The web UI is accessible through any modern web browser and is the primary method of interacting with Flywheel data. Through this point-and-click interface, users are able to upload, view, download, and analyse data with ease and simplicity. However, accomplishing tasks with many repetitive steps or a large number of subjects/sessions to iterate over can be tiresome and error-prone. Alongside knowledge of navigating the web UI, many users also make use of the SDK to manipulate and analyse data programmatically.

Flywheel API & SDK

Flywheel's database utilises MongoDB for data storage and access, meaning that all Flywheel data is represented by hierarchical relationships between document objects. This allows users to create and store complex structures with ease, and query data rapidly (Banker 2011). In order to access this data, Flywheel uses a RESTful Application Programming Interface (REpresentational State Transfer) (Biehl 2016), and hence each document or data object is accessible through a a specific URL that a web browser or SDK can access by requesting the data and waiting for a response from the server. The Flywheel Python SDK provides a powerful interface for inspecting and manipulating data through this API. By standardising this underlying data model into Pythonic objects, the flywheel SDK is effectively an object relationship mapper, similar to the popular SQLAlchemy software.

Flywheel Data Model

Objects in Flywheel's data model follow a specific hierarchical structure (**Figure 2**) — at the top level is the Flywheel *instance*, a process running that serves the API to users (for example, a neuroimaging center). Within the Flywheel instance, there are multiple *groups*, which are typically labs or research units that collaborate on one or more *projects*. Each project object can have one or many *subjects* (i.e. participants), and each subject can have one or many *sessions* (i.e. scanning visits). Within a session, there may be one or many *acquisition* objects which represent the scanning sequence collected during a particular scan or examination (e.g. sMRI, rs-fMRI, DWI), and under each acquisition is

attached the data file associated with the sequence (e.g. a NIfTI file or DICOM). Note that a file can be attached to any object type, and each object can have metadata associated with it. Hence, a subject object may have demographic metadata associated to that participant, and the subject may also have a text file attached to it (such as clinical data). A notable exception to the hierarchical structure rule is the analysis object, which behaves in much the same way as others but can be a child object of any other object, allowing researchers to create analyses of entire projects, for example, each with their own associated metadata and files (such as inputs and outputs).

Abstracting this data model in Python results in simple hierarchical objects, each with methods for handling metadata and files, and methods for accomplishing object-specific tasks like traversing the hierarchical structure or running analyses. Flywheel Tools' modules make use of this data model to accomplish a wide range of tasks.

Flywheel Gears

Flywheel encourages the use of pre-packaged computational workflows, called *gears*. Gears are run by virtual machines/containers using Docker and hence are version-controlled and software/platform agnostic. Gears can accomplish tasks such as data manipulation, pre-processing, analysis, and summarisation. In addition to the multitude of gears available on the platform, users are able to package their own software in a gear and use it for running analysis workflows on their Flywheel data, via the web UI or SDK. The complexity and frequency of the task suggests whether to accomplish a task using the web UI, programmatically using the SDK, or by wrapping it as a workflow into a gear. Gears can consume existing Flywheel data, such as images or file attachments, as inputs to the workflow, and can be created with clickable configuration options. Once a workflow has completed running, Flywheel collects any files remaining in the container's pre-defined output directory and attaches them to a resulting analysis object. The resultant files of a gear (such as an HTML report or tabulated data) can be viewed on the Flywheel UI, downloaded to disk for further data sharing or analysis, or be used as input to a subsequent gear.

Results

Flywheel Tools is implemented using the Flywheel SDK to enable easy inspection, curation, validation, and audit of Flywheel data through a handful of user-friendly gears and command-line interfaces. The first module of the package is called `fw-heudiconv`, and is largely inspired by the popular HeuDiConv Python package. `fw-heudiconv` is a multi-part toolbox for reprodicible curation of neuroimaging data into BIDS on Flywheel. The second module, `flaudit`, is a tool for accomplishing a complete audit of a Flywheel project, giving users a broad overview of the key elements of their data set.

`fw-heudiconv`

The first tool, `fw-heudiconv`, is a multi-purpose command-line interface and Flywheel gear designed for comprehensive BIDS curation on Flywheel. It is designed to be intuitive, flexible, and reproducible.

Architecture & Design

`fw-heudiconv` is inspired in large part by the Heuristic Dicom Converter (HeuDiConv) package, and shares much of its design practices. In order to curate data into BIDS, `fw-heudiconv` first considers Dicom data to be the "ground truth" data, and builds its curation approach using data in the Dicoms' headers. Ultimately, `fw-heudiconv` only has permission to manipulate metadata attached to NIfTI files, in the "Info: BIDS" field, which ensures that curation can be repeated from the stage of Dicom ingress reliably and safely.

`fw-heudiconv` can be downloaded as a Python command-line interface from the Python Package Index using `pip`, and is available as a point-and-clickable gear on the Flywheel UI. The gear is managed by Docker containerisation, meaning that versioning is reliable and reproducible. There are a number of commands available in `fw-heudiconv`, and each of them starts by querying data from Flywheel. Users can filter their queries, so as to operate on an entire Flywheel project, a subset of subjects, or a subset of sessions. Notably each command has the ability to safely test and evaluate its effects without manipulating metadata on Flywheel or writing data to disk (using the `--dry-run` option). In particular, there are five commands users can access:

1. `fw-heudiconv-tabulate` The tabulate tool is used to parse and extract DICOM header information in a project (or within a filtered subset of that project) and tabulate this data for the user to examine with ease. By collecting DICOM header information into tabular format, the tabulate tool gives users a comprehensive overview of the different scanning sequences that have been collected in the query, including their sequence parameters. Additionally, users have the option to limit the tabulation to a unique combination of common DICOM header fields, which significantly decreases complexity of the table. The table output by this command can be written to a local disk if run from the command line, or is saved in the output section of a Flywheel gear if run on Flywheel.

2. `fw-heudiconv-curate` The curate tool is used to curate a dataset on Flywheel into BIDS format. Much like HeuDiConv, curation is done through the use of a heuristic, a Python file which programmatically defines the templates of a range of BIDS valid filenames, and defines the boolean logic that would assign a given scanning sequence to each template. This boolean logic is based on the sequence information users find in the tabulation of sequences, and all fields available in the Dicom header can be used to parse out which template a particular file can be assigned to. Additionally, the curate tool can be used

to manipulate BIDS metadata that may need to be added to the dataset. The process of curation only manipulates BIDS metadata of NIfTI files, and hence can be repeated or updated at any time at the user's discretion.

3. `fw-heudiconv-export` The export tool is used to export a BIDS dataset on Flywheel to disk. The tool is primarily used as a tool for other gears and scripts to quickly and easily extract their BIDS data into the workspace of their analysis pipeline. It is also able to export BIDS data from Flywheel to a local disk.

4. `fw-heudiconv-validate` The validate tool is a wrapper around the popular BIDS Validator package **CITE?** and is use to check if the applied curation results in a BIDS-valid dataset. After exporting a data set with `fw-heudiconv-export`, the validate tool runs the BIDS validator on the dataset and returns the result and verbose description of the errors and warnings given by the BIDS Validator. Additionally, the results of the validator can be tabulated for easy inspection. On Flywheel, `fw-heudiconv-validate` also displays a green check mark in the analysis tab for successful validation, and a red check mark otherwise, allowing for quick visual inspection of BIDS curation status for each session.

5. `fw-heudiconv-clear` The clear tool is used to clear BIDS information cleanly and safely from the project or subjects and sessions queried. This can be useful when overwriting existing BIDS data doesn't overwrite all of the BIDS fields previously available. These persistent fields can be removed by running `fw-heudiconv-clear` before re-curating.

Heuristic File

The heuristic file is a Python file used as input to the `fw-heudiconv-curate` command. The file instructs `fw-heudiconv` on how to programmatically sort and parse through each acquisition object in Flywheel, and assign it to a BIDS-valid naming template. This is done by checking attributes of a list of `seqInfo` objects — which are generated from each DICOM's header information — against user-defined boolean rules. For example, if a T1-weighted image is present in a dataset, the user could define a string with a BIDS-valid naming template for this type of file, such as:

```
t1w = 'subject-{Subject}_session-{Session}_T1w.nii.gz'
```

Where the `Subject` and `Session` portions are expected to be automatically generated for each subject and session in the dataset. After a DICOM's `SeriesDescription` field is added to the `seqInfo`'s `SeriesDescription` attribute, the user can create simple boolean rules to check if the string 'T1w' is in the `SeriesDescription`. If such a rule is met, this acquisition and its NIfTI file will be assigned to the T1-weighted image naming template. The NIfTI file will ultimately have this BIDS naming added to its metadata, and be named correctly when exported to a filesystem.

In addition to setting naming templates, the heuristic file can also be used to hardcode and assign metadata in BIDS. These data are hard-coded into the file object's metadata on Flywheel, and are assigned by using specially reserved functions and keywords in `fw-heudiconv`. For example, the heuristic file can be used to point fieldmap scans to their intended sequences using a list:

```
IntendedFor = {

  fieldmap1: ['sub-{SubjectLabel}_{SessionLabel}_task-rest_bold.nii.gz']

}`
```

By reserving select keywords for functions and metadata, heuristic files become versatile tools for defining and manipulating a wide array of metadata in Flywheel BIDS curation.

Importantly, because this heuristic file plain text Python code, users are able to version control their files using Git, and shared with other tools such as Github. Finally, when run on Flywheel as a gear, the heuristic file is automatically attached as an input to the analysis object created by `fw-heudiconv-curate`, allowing users to easily access the version history of their curation attempts.

Curation Workflow

For most users, the curation workflow follows the sequence detailed above (**Figure 3**). After ingress of a batch of DICOMs from a scan, Flywheel's automated utility gears convert the DICOMs to NIfTI files. Users can then begin running `fw-heudiconv-tabulate` to gather the information stored in the DICOM headers necessary for creating a heuristic. Once the tabulation has been completed, the output file can be opened by any program that can read tabular data. Users at this stage can begin creating a heuristic file and running `fw-heudiconv-curate`, using the `--dry-run` flag to test the heuristic changes incrementally with informative logging. When satisfied, users can simply remove the `--dry-run` flag to apply the changes. The user can then use `fw-heudiconv-validate` to run the BIDS validator on the dataset, or start over by removing all BIDS metadata with `fw-heudiconv-clear`.

If `fw-heudiconv` is run from the Flywheel UI, each of the commands is available as a Flywheel gear. This option can be beneficial for data provenance as all of a gear's commands and inputs, as well as outputs and log files, are stored and attached to each gear run.

`flaudit`

The second module is a Flywheel project auditor, aptly named `flaudit`. The module is intended to give Flywheel users a broad understanding of their entire Flywheel project, by summarizing the available data and illustrating analysis workflows. The output of this module, a portable HTML report, presents this

information using a number of visualizations built in R Markdown using HTML, Javascript, and GGplot, in two main sections.

Architecture & Design

Using similar internal machinery to `fw-heudiconv-tabulate`, `flaudit` loops over existing data in a project and tabulates information about scanning sequences, BIDS metadata, and gear analyses that have been run. These 3 tables are saved internally and then passed as input to an R markdown script that generates an interactive HTML report. The data are also saved as output for the user to access and analyze in their software of choice.

Report Section 1: Overview

The overview section provides a numerical overview of sequences, BIDS data, gear runs, and gear runtimes.

The first visualization uses the sequence data input to create a word cloud visualizing the names of the different sequences acquired across the entire Flywheel dataset, where the size of the word corresponds to the frequency of the sequence collected. This visual is accompanied by a bar chart and an interactive table that users can search to compare values (**Figures 4 and 5**).

Next, using the BIDS metadata input, the report provides an interactive tree viewer to examine BIDS curation. In the tree, the nodes branch out from the project to show each sequence acquisition. For each acquisition, if the data has been curated into BIDS, the node itself can also branch out to show a BIDS name template, demonstrating what BIDS name that sequence has been given. Hovering over the BIDS name will display the number of subjects whose data have been named as such (**Figure 6**).

Using the gear analysis data as input, the section lastly enumerates the gear analyses that have been run successfully on any session within the project, and enumerates the runtimes for these processes. The results are visualized in a series of bar charts (**Figure 7**).

Report Section 2: Project Completion

As an optional input, `flaudit` allows users to specify a *template subject*, a subject from the Flywheel project who serves as an exemplar for other subjects to be compared against. This subject may have been the researcher's first choice for testing BIDS curation and analysis pipelines, perhaps due to the subject's high data quality or data completeness. Researchers can compare others to this *template subject*, to determine if other subjects in the project also meet this high standard. The project completion section comprises of three interactive tables.

In the first table, it is assumed that the *template subject* has a complete number of sequences in their dataset. These sequences are listed as columns in the table.

Each subsequent row is a subject in the project, and each value in the table is a boolean value (*complete* or *incomplete*) indicating if that subject has each sequence. The table is searchable, meaning that users can simply filter each column for "incomplete" to learn which subjects do not have the same data as the template (**Figure 8**). Likewise, the second table illustrates the completeness of BIDS data for each other subject in comparison to the template (**Figure 9**). In this case, rows indicate subjects while columns indicate each BIDS naming template.Lastly, the third table illustrates completeness of gear analyses. In this case, researchers can use this table to assert that all other subjects in the project have had at least one run of each of the analysis pipelines that the *template subject* has had. Hence, in this table, each column specifies an analysis run. For version uniformity, this comparison is sensitive to pipeline versions, and so the version of a pipeline that was used for each subject must match that of the *template subject* (**Figure 10**).

Discussion

As the neuroimaging community embraces Big Data and the various platforms available for storage and analysis, it is becoming increasingly important for researchers to eschew the *ad hoc* analysis procedures previously run on a single machine or cluster. Instead, cloud-based platforms like Flywheel provide opportunities for more reproducible, reliable, and scalable science. Flywheel Tools provides software that maximizes these opportunities on the Flywheel platform.

Figures

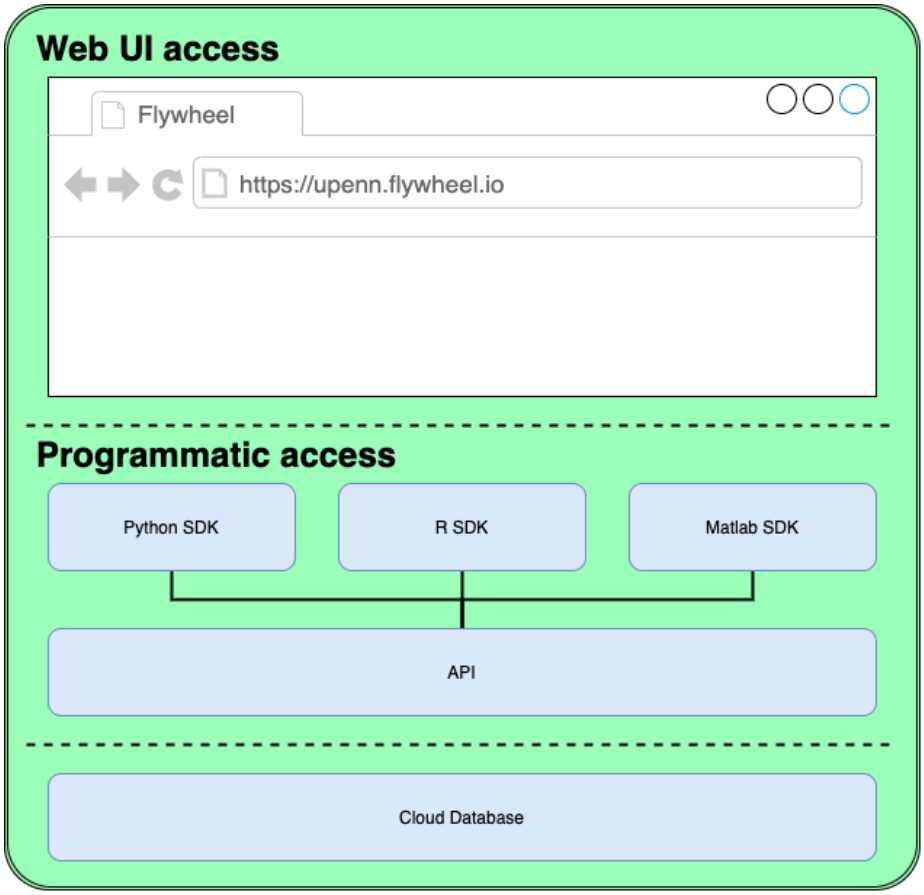


Figure 1: Flywheel Basic Architecture

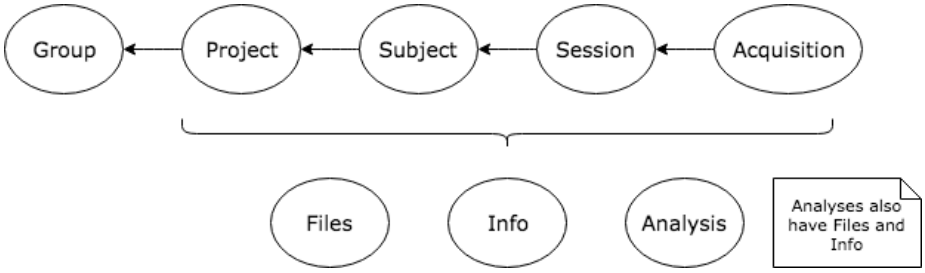


Figure 2: Flywheel Data Model

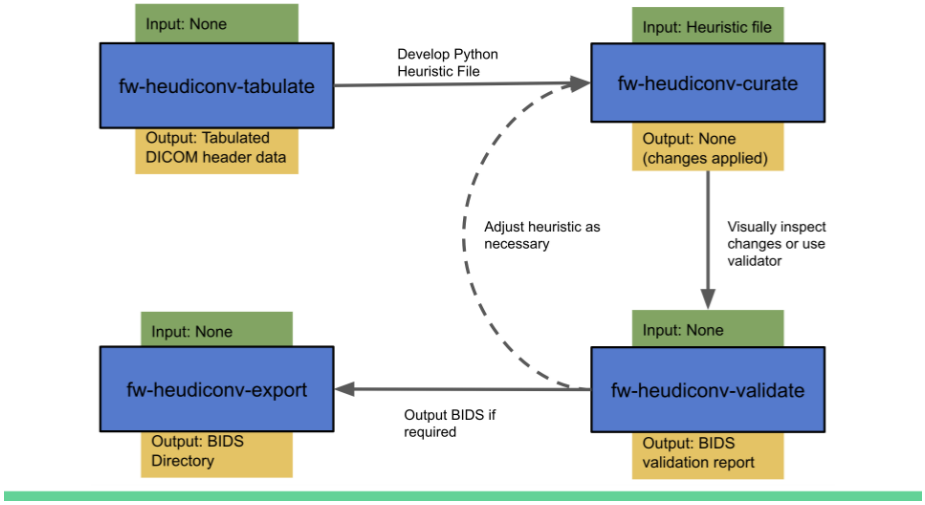


Figure 3: Flywheel Tools Workflow

We visualise the above frequencies below:

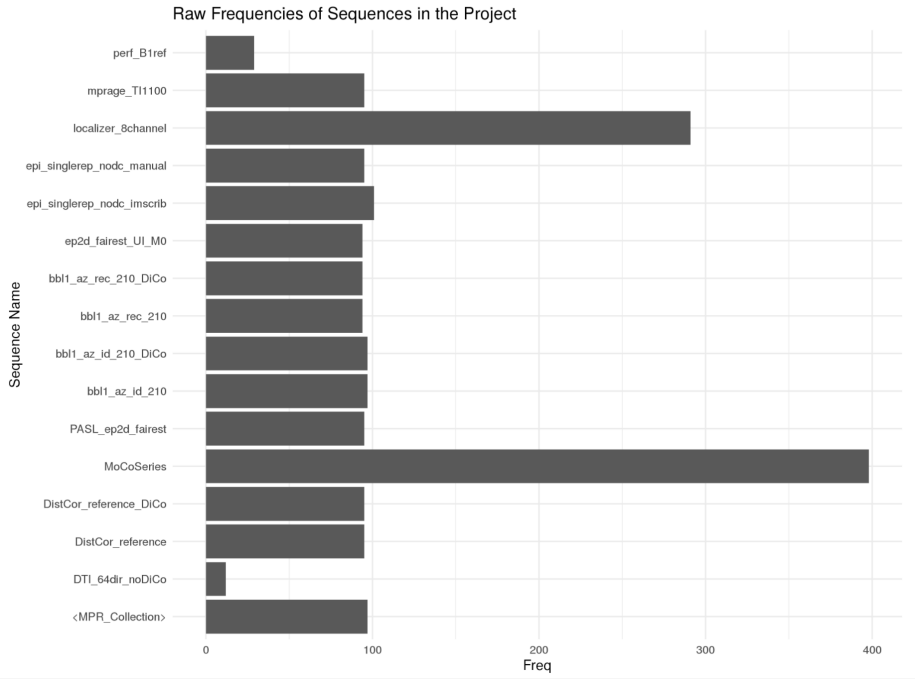


Figure 4: Plot Enumerating Available Sequences in a Flywheel Project

Show 10 entries Search:

	series_description	Freq
1	MoCoSeries	398
2	localizer_8channel	291
3	epi_singlerep_nodc_imscrib	101
4	<MPR_Collection>	97
5	bbl1_az_id_210	97
6	bbl1_az_id_210_DiCo	97
7	DistCor_reference	95
8	DistCor_reference_DiCo	95
9	PASL_ep2d_fairest	95
10	epi_singlerep_nodc_manual	95

Showing 1 to 10 of 16 entries Previous 1 2 Next

Figure 5: Interactive Table of Available Sequences in a Flywheel Project

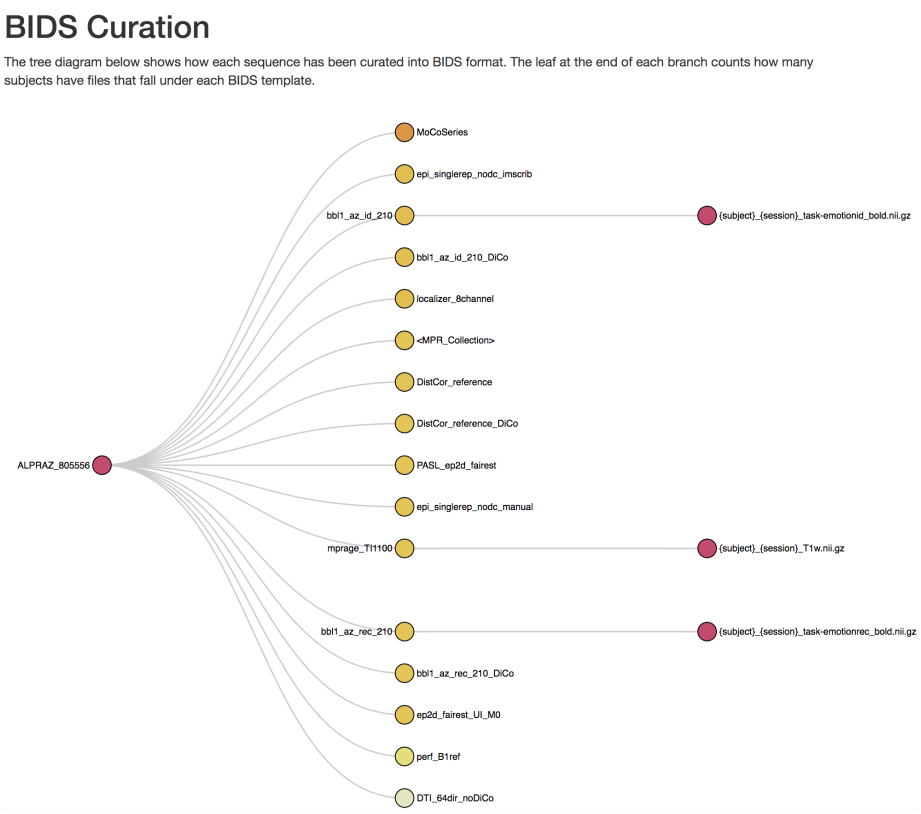


Figure 6: Interactive Tree Diagram Illustrating BIDS Curation

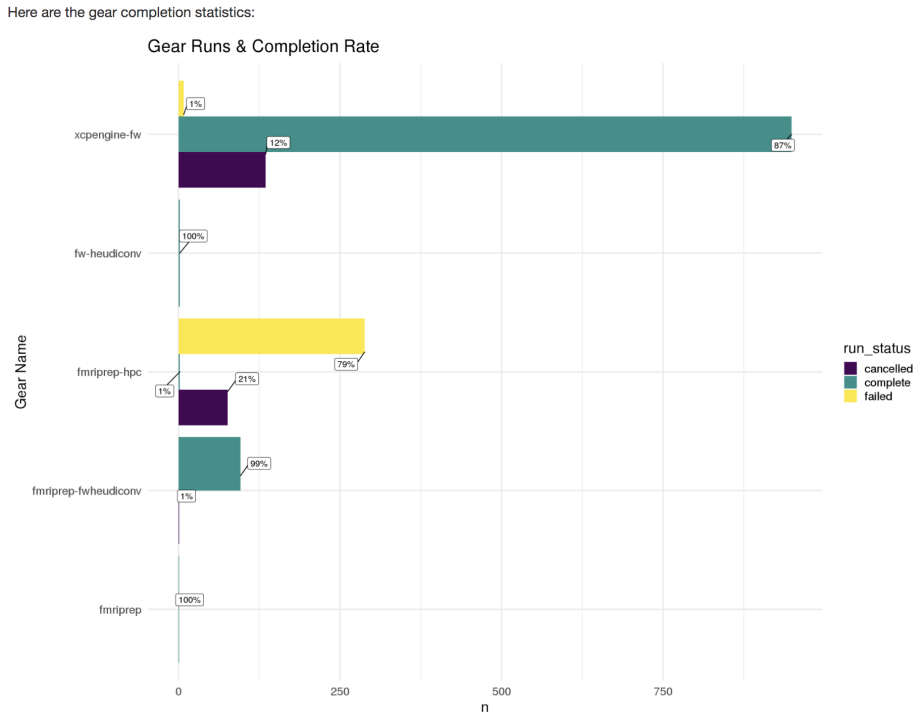
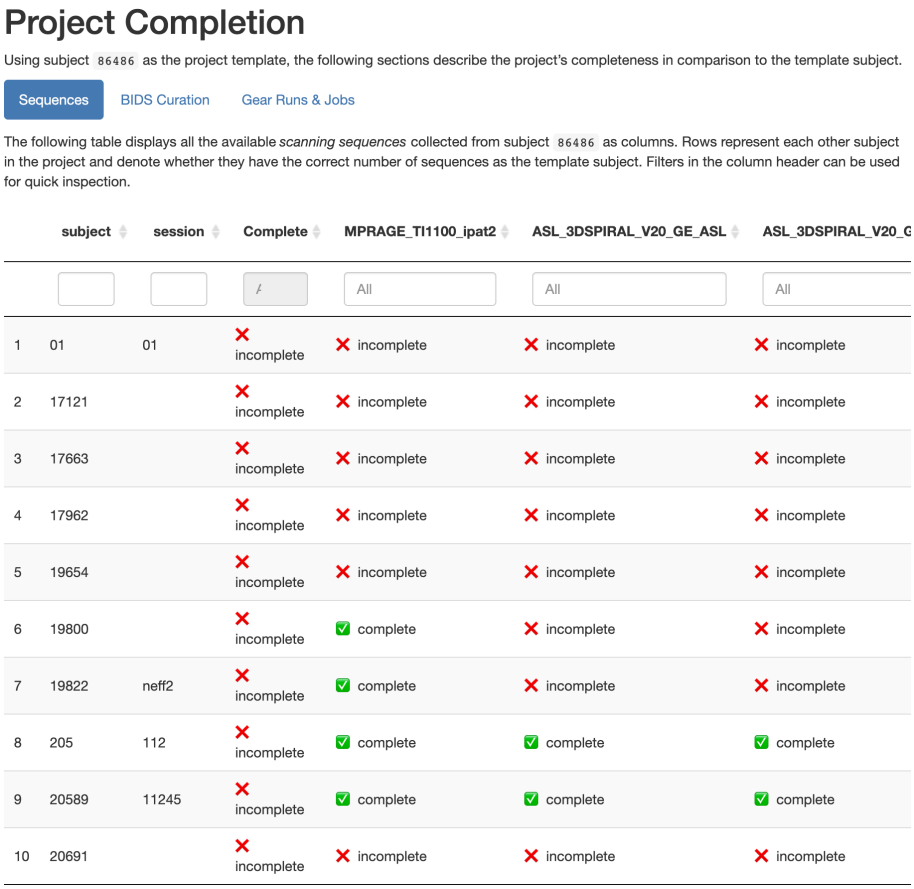


Figure 7: Plot Enumerating Gear Runs in a Flywheel Project



Project Completion

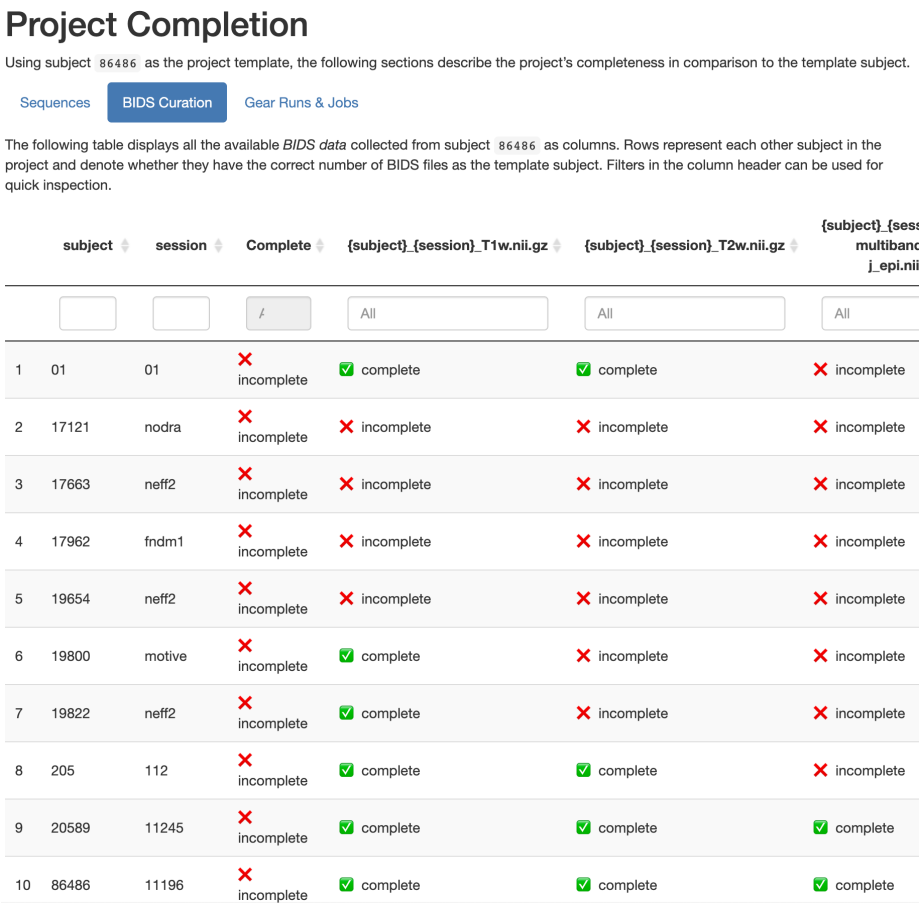
Using subject `86486` as the project template, the following sections describe the project's completeness in comparison to the template subject.

Sequences | BIDS Curation | Gear Runs & Jobs

The following table displays all the available *scanning sequences* collected from subject `86486` as columns. Rows represent each other subject in the project and denote whether they have the correct number of sequences as the template subject. Filters in the column header can be used for quick inspection.

	subject	session	Complete	MPRAGE_TI1100_ipat2	ASL_3DSPIRAL_V20_GE_ASL	ASL_3DSPIRAL_V20_G
			/	All	All	All
1	01	01	✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
2	17121		✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
3	17663		✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
4	17962		✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
5	19654		✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
6	19800		✗ incomplete	✓ complete	✗ incomplete	✗ incomplete
7	19822	neff2	✗ incomplete	✓ complete	✗ incomplete	✗ incomplete
8	205	112	✗ incomplete	✓ complete	✓ complete	✓ complete
9	20589	11245	✗ incomplete	✓ complete	✓ complete	✓ complete
10	20691		✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete

Figure 8: Sequence Completeness Compared to the Template Subject in a Flywheel Project



Project Completion

Using subject `86486` as the project template, the following sections describe the project's completeness in comparison to the template subject.

Sequences | **BIDS Curation** | Gear Runs & Jobs

The following table displays all the available *BIDS data* collected from subject `86486` as columns. Rows represent each other subject in the project and denote whether they have the correct number of BIDS files as the template subject. Filters in the column header can be used for quick inspection.

	subject	session	Complete	{subject}_{session}_T1w.nii.gz	{subject}_{session}_T2w.nii.gz	{subject}_{ses multiband j_epi.nii
			/	All	All	All
1	01	01	✗ incomplete	✓ complete	✓ complete	✗ incomplete
2	17121	nodra	✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
3	17663	neff2	✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
4	17962	fndm1	✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
5	19654	neff2	✗ incomplete	✗ incomplete	✗ incomplete	✗ incomplete
6	19800	motive	✗ incomplete	✓ complete	✗ incomplete	✗ incomplete
7	19822	neff2	✗ incomplete	✓ complete	✗ incomplete	✗ incomplete
8	205	112	✗ incomplete	✓ complete	✓ complete	✗ incomplete
9	20589	11245	✗ incomplete	✓ complete	✓ complete	✓ complete
10	86486	11196	✗ incomplete	✓ complete	✓ complete	✓ complete

Figure 9: BIDS Completeness Compared to the Template Subject in a Flywheel Project

	subject	session	Complete	fw-heudiconv	flaudit
	All	All	All	All	All
1	01	01	✗ incomplete	✓ complete	✗ incomplete
2	19822	neff2	✗ incomplete	✓ complete	✗ incomplete
3	19977	10781	✗ incomplete	✓ complete	✗ incomplete
4	205	112	✗ incomplete	✓ complete	✗ incomplete
5	20589	11245	✗ incomplete	✓ complete	✗ incomplete
6	86486	11196	✓ complete	✓ complete	✓ complete
7	95257	11191	✗ incomplete	✓ complete	✗ incomplete
8	BIDSTESTING	reproin	✗ incomplete	✓ complete	✗ incomplete
9	exampleSubject	exampleSession2	✗ incomplete	✗ incomplete	✗ incomplete
10	sub-01	ses-1	✗ incomplete	✗ incomplete	✗ incomplete

Figure 10: BIDS Completeness Compared to the Template Subject in a Flywheel Project

Bibliography

Banker, Kyle. 2011. *MongoDB in Action*. USA: Manning Publications Co.

Biehl, Matthias. 2016. *RESTful API Design*. Vol. 3. API-University Press.

Book, Gregory A, Michael C Stevens, Michal Assaf, David C Glahn, and Godfrey D Pearlson. 2016. "Neuroimaging Data Sharing on the Neuroinformatics Database Platform." *Neuroimage* 124: 1089–92.

Gorgolewski, Chris, Nell Hardcastle, Teal Hobson-Lowther, David Nishikawa, Ross Blair, Stefan Appelhoff, Suyash, et al. 2020. *Bids-Standard/Bids-Validator: 1.4.3* (version 1.4.3). Zenodo. https://doi.org/10.5281/zenodo.3688707.

Gorgolewski, Krzysztof J, Tibor Auer, Vince D Calhoun, R Cameron Craddock, Samir Das, Eugene P Duff, Guillaume Flandin, et al. 2016. "The Brain Imaging Data Structure, a Format for Organizing and Describing Outputs of Neuroimaging Experiments." *Scientific Data* 3 (1): 1–9.

———, et al. 2016. "The Brain Imaging Data Structure, a Format for Organizing and Describing Outputs of Neuroimaging Experiments." *Scientific Data* 3 (1): 1–9.

Herrick, Rick, William Horton, Timothy Olsen, Michael McKay, Kevin A Archie, and Daniel S Marcus. 2016. "XNAT Central: Open Sourcing Imaging Research Data." *NeuroImage* 124: 1093–96.

Landis, Drew, William Courtney, Christopher Dieringer, Ross Kelly, Margaret King, Brittny Miller, Runtang Wang, Dylan Wood, Jessica A Turner, and

Vince D Calhoun. 2016. "COINS Data Exchange: An Open Platform for Compiling, Curating, and Disseminating Neuroimaging Data." *NeuroImage* 124: 1084–88.

Merkel, Dirk. 2014. "Docker: Lightweight Linux Containers for Consistent Development and Deployment." *Linux J.* 2014 (239).

Poldrack, Russell A, and Krzysztof J Gorgolewski. 2017. "OpenfMRI: Open Sharing of Task fMRI Data." *Neuroimage* 144: 259–61.

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rogovin, O, Y Zhao, S Chen, Z Wang, O Papaemmanouil, SD Van Hooser, and others. 2020. "NDI: A Platform-Independent Data Interface and Database for Neuroscience Physiology and Imaging Experiments."

Vaccarino, Anthony L, Moyez Dharsee, Stephen Strother, Don Aldridge, Stephen R Arnott, Brendan Behan, Costas Dafnas, et al. 2018. "Brain-CODE: A Secure Neuroinformatics Platform for Management, Federation, Sharing and Analysis of Multi-Dimensional Neuroscience Data." *Frontiers in Neuroinformatics* 12: 28.

Van Rossum, Guido, and Fred L. Drake. 2009. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.