Pybids: Python tools for manipulation and analysis of BIDS datasets

Tal Yarkoni[1], Alejandro de la Vega[1], Elizabeth DuPre[2], Oscar Esteban[3], Yaroslav O. Halchenko[4], Michael Hanke[5], Valerie Hayot-Sasson[6], Alexander Ivanov[7], Gregory Kiar[2], Christopher Markiewicz[3], Quinten McNamara[1], Dmitry Petrov[8], Taylor Salo[9], Dylan Nielson[10], Jean-Baptiste Poline[2], Russell Poldrack[3], Krzysztof Gorgolewski[3]

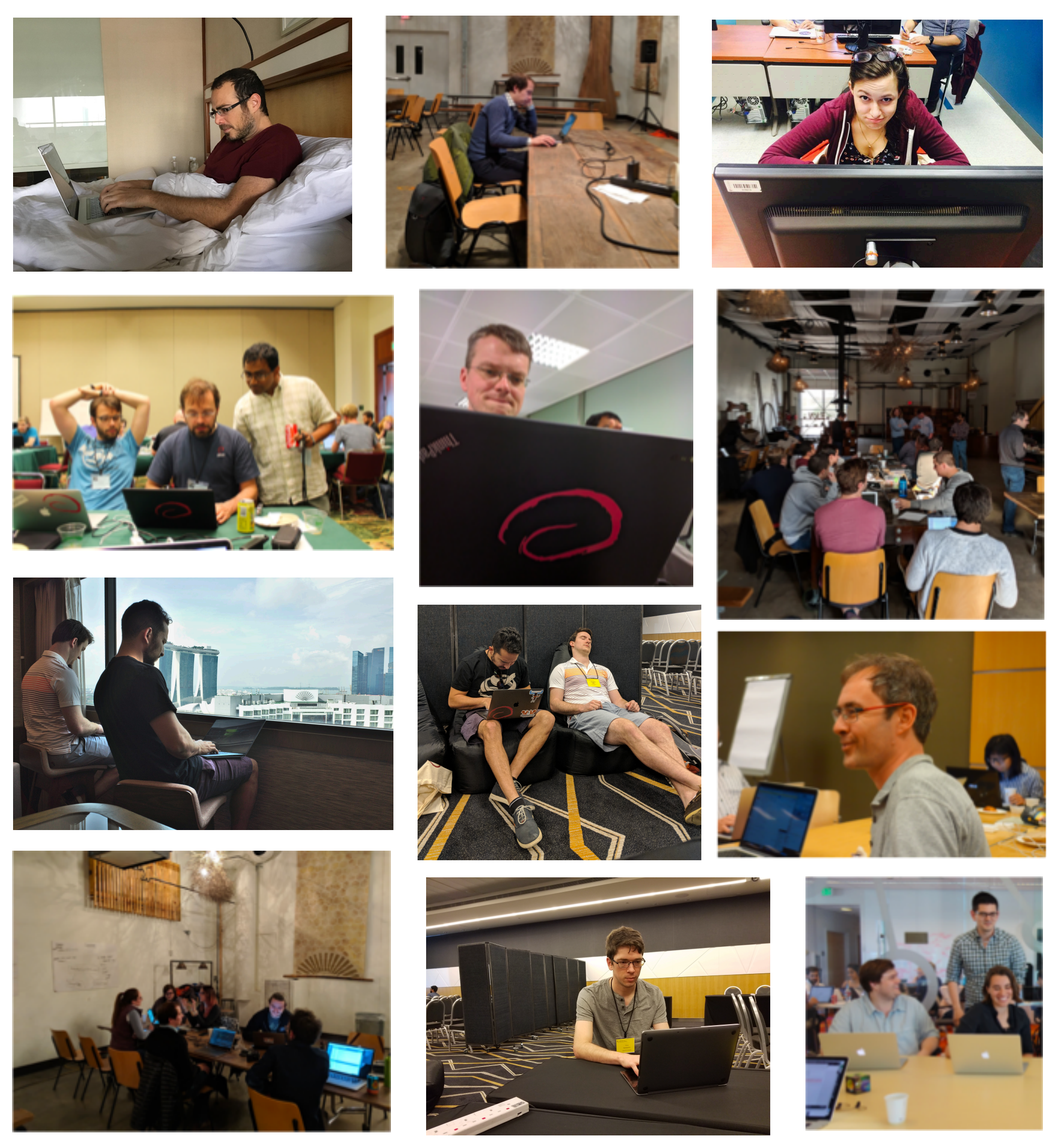
[1]University of Texas at Austin, [2]McGill University, [3]Stanford University, [4]Dartmouth College, and several other fine institutions

Introduction

- Efforts to standardize the representation of neuroimaging datasets have recently converged on the Brain Imaging Data Structure (BIDS; Gorgolewski et al., 2016)--a relatively simple specification that has already been adopted by hundreds of researchers around the world
- To maximize the utility of this common standard, it is important to develop easy-to-use tools that facilitate programmatic interaction with, and manipulation of, BIDS-compliant datasets.
- Here we describe a new open-source Python package—"pybids"--that provides powerful tools for querying BIDS datasets and constructing complex statistical analysis pipelines.

Methods

We wrote a bunch of code



About pybids

Where can I get it?

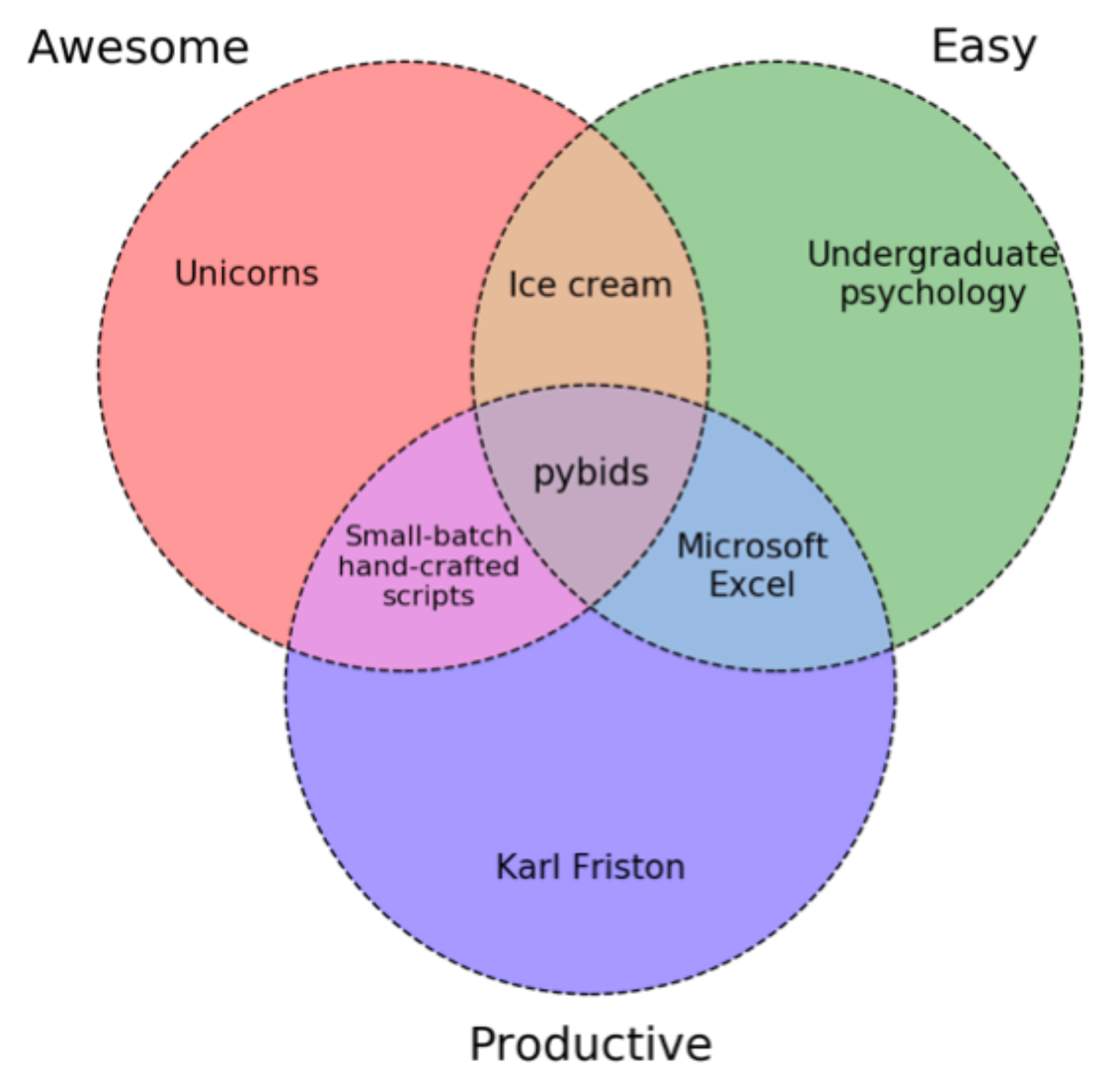
https://github.com/incf/pybids

Hipsters please point your mobile devices at the magic square on the right

What does it do?

- Makes it much easier to work with BIDS-compliant datasets in Python
- Querying module ("grabbids") makes it easy to construct and execute complex queries that might otherwise require extensive scripting.
- Loading/extraction of all variables (task events, physiological recordings, behavioral measures, etc.) found in BIDS projects--optionally returned as pandas DataFrames
- A working implementation of the forthcoming BIDS-Model specification, which provides a simple, machine-readable way to represent complex statistical models that can potentially be fitted using a variety of fMRI analysis packages
- Partial auto-generation of methods sections
- ~~Hundreds~~ ~~dozens~~ three other features

Why should I use it?



How do I use it?

Initialize a BIDS project

```
from bids import BIDSLayout
layout = BIDSLayout(bids_dir)
```

Simple but flexible querying

```
# Get list of unique subjects
layout.get_subjects()
```

```
['01', '02', '03', '04', …]
```

```
# Get filenames of all BOLD timeseries for subject '04'
layout.get(type='bold', subject='04', return_type='file')
```

```
['sub-04_task-mixedgamblestask_run-01_bold.nii.gz',
 'sub-04_task-mixedgamblestask_run-02_bold.nii.gz', …]
```

```
# Get metadata for an image
target = 'sub-04_ses-1_task-rest_run-1_bold.nii.gz'
layout.get_metadata(target)
```

```
{'EchoTime': 0.017, 'EffectiveEchoSpacing': 0.0003, 'PhaseEncodingDirection':
'j-', 'RepetitionTime': 3.0, …}
```

Easy access to variables in BIDS projects

```
# Load all run-level variables (e.g., experimental events,
# physio recordings, etc.) in the BIDS project
events = layout.get_collections(level='run')
events.to_df()
```

	index	duration	task	onset	run	subject	PTval	RT	gain	loss	parametric gain	respcat	respnum	response	trial_type
0	1	3	mixedgamblestask	0.0	1	01	5.15	0.000	20.0	15.0	-0.139	-1.0	0.0	hello	parametric gain
1	49	3	mixedgamblestask	4.0	1	01	6.12	1.793	18.0	12.0	-0.189	1.0	2.0	no	parametric gain
2	97	3	mixedgamblestask	8.0	1	01	-4.85	1.637	10.0	15.0	-0.389	0.0	3.0	hello	parametric gain
3	193	3	mixedgamblestask	18.0	1	01	18.16	1.316	34.0	16.0	0.211	1.0	1.0	yes	parametric gain
4	241	3	mixedgamblestask	24.0	1	01	13.05	1.670	18.0	5.0	-0.189	1.0	1.0	NaN	parametric gain

Model specification and design matrix construction

Pybids supports the BIDS-Model specification; here's an example JSON file specifying a simple first-level model:

```
{"name":"my_model","blocks":[{"name":"run","level":"run","transformations":
[{"name":"scale","input":"RT"},{"name":"factor","input":"trial_type"}],
"model":{"variables":["parametric gain","parametric loss","RT"]},
"auto_contrasts":true}]}
```

We can pass this specification directly to pybids, and have it easily construct design matrices, contrast specifications, etc.—potentially after applying complex transformations to the variables included in the design matrix.

```
from bids import Analysis
analysis = Analysis(layout, model.json)
# Sets up the analysis
analysis.setup()
# Retrieve the design matrix for subject 1's first run
analysis['run'].get_design_matrix(subject='01', run=1)
```

	task	type	subject	modality	run	PTval	RT	gain	loss	parametric gain	respcat	respnum
0	mixedgamblestask	bold	01	func	1	5.15	-3.530433	20.0	15.0	-0.139	-1.0	0.0
1	mixedgamblestask	bold	01	func	1	5.15	-3.530433	20.0	15.0	-0.139	-1.0	0.0
2	mixedgamblestask	bold	01	func	1	5.15	-3.530433	20.0	15.0	-0.139	-1.0	0.0
3	mixedgamblestask	bold	01	func	1	5.15	-3.530433	20.0	15.0	-0.139	-1.0	0.0
4	mixedgamblestask	bold	01	func	1	5.15	-3.530433	20.0	15.0	-0.139	-1.0	0.0

And more...

There are ~~thousands~~ ~~hundreds~~ at least three other things you can do with pybids. More information at **https://github.com/incf/pybids**

Overflow space for comment box below

Comments, suggestions, requests, sketches, haiku, and other expressions of [dis]satisfaction go in this box

…or report what ails you at https://github.com/incf/pybids/issues

Supported by NIH award R01MH109682
For further information or reprints (why???), email tyarkoni@utexas.edu