adnipy Documentation

Release 0.1.0

Maximilian Cosmo Sitter

Oct 31, 2019

Contents:

1 adnipy **1**
- 1.1 Credits . . . 1

2 Installation **3**
- 2.1 Stable release . . . 3
- 2.2 From sources . . . 3

3 Usage **5**

4 adnipy **7**
- 4.1 adnipy package . . . 7

5 Contributing **15**
- 5.1 Types of Contributions . . . 15
- 5.2 Get Started! . . . 16
- 5.3 Pull Request Guidelines . . . 16
- 5.4 Tips . . . 17
- 5.5 Deploying . . . 17

6 Credits **19**
- 6.1 Development Lead . . . 19
- 6.2 Contributors . . . 19

7 History **21**
- 7.1 0.0.1 (2019-09-05) . . . 21
- 7.2 0.1.0 (2019-10-25) . . . 21

8 Indices and tables **23**

Python Module Index **25**

Index **27**

CHAPTER 1

adnipy

Process ADNI study data with adnipy.

Adnipy is a python package designed for working with the ADNI database. It also offers some handy tools for file operations.

- Free software: MIT license
- Documentation: https://adnipy.readthedocs.io

1.1 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install adnipy, run this command in your terminal:

```
$ pip install adnipy
```

This is the preferred method to install adnipy, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for adnipy can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/mcsitter/adnipy
```

Or download the tarball:

```
$ curl  -OL https://github.com/mcsitter/adnipy/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use adnipy in a project:

```
import adnipy
```

CHAPTER 4

adnipy

4.1 adnipy package

4.1.1 Submodules

4.1.2 adnipy.adni module

Pandas dataframe extension for ADNI.

class adnipy.adni.**ADNI**(*pandas_dataframe*)
Bases: object

Methods

drop_dynamic(self)	Remove images which are dynamic.
groups(self[, grouped_mci])	Create a dataframe for each group and save it to a csv file.
longitudinal(self)	Keep only longitudinal data.
rid(self)	Add a roster ID column.
standard_column_names(self)	Rename dataframe columns to module standard.
standard_dates(self)	Change type of date columns to datetime.
standard_index(self[, index])	Process dataframes into a standardized format.
timepoints(self[, second])	Extract timepoints from a dataframe.

drop_dynamic(*self*)
Remove images which are dynamic.

Drops all rows, in which the Description contains 'Dynamic'.

Returns

pd.DataFrame All images that are not dynamic.

groups(*self*, *grouped_mci=True*)
Create a dataframe for each group and save it to a csv file.

Parameters

grouped_mci [bool, default True] If true, 'LMCI' and 'EMCI' are treated like 'MCI'. However, the original values will stills be in the output.

Returns

dict Dictionnairy with a dataframe for each group.

longitudinal (*self*)

Keep only longitudinal data.

This requires an 'RID' or 'Subject ID' column in the dataframe. Do not use if multiple images are present for a single timepoint.

Parameters

images [pd.DataFrame] This dataframe will be modified.

Returns

pd.DataFrame A dataframe with only longitudinal data.

See also:

drop_dynamic

rid (*self*)

Add a roster ID column.

Will not work if 'RID' is already present or 'Subject ID' is missing.

Returns

pd.DataFrame Dataframe with a 'RID' column.

Examples

```
>>> subjects = {"Subject ID": ["100_S_1000", "101_S_1001"]}
>>> collection = pd.DataFrame(subjects)
>>> collection
Subject ID
0  100_S_1000
1  101_S_1001
>>> collection.rid()
Subject ID   RID
0  100_S_1000  1000
1  101_S_1001  1001
```

standard_column_names (*self*)

Rename dataframe columns to module standard.

This function helps when working with multiple dataframes, since the same data can have different names. It will also call *rid()* on the dataframe.

Returns

pd.DataFrame This will have standardized columns names.

See also:

rid

Examples

```
>>> subjects = pd.DataFrame({"Subject": ["101_S_1001", "102_S_1002"]})
>>> subjects
    Subject
0  101_S_1001
1  102_S_1002
>>> subjects.standard_column_names()
Subject ID   RID
0  101_S_1001  1001
1  102_S_1002  1002
```

```
>>> images = pd.DataFrame({"Image": [100001, 100002]})
>>> images
    Image
0  100001
1  100002
>>> images.standard_column_names()
Image ID
0    100001
1    100002
```

standard_dates(*self*)
Change type of date columns to datetime.

Returns

pd.DataFrame Dates will have the appropriate dtype.

standard_index(*self*, *index=None*)
Process dataframes into a standardized format.

The output is easy to read. Applying functions the the output may not work as expected.

Parameters

index [list of str, default None] These columns will be the new index.

Returns

pd.DataFrame An easy to read dataframe for humans.

timepoints(*self*, *second='first'*)
Extract timepoints from a dataframe.

Parameters

second [{'first' or 'last'}, default 'first'] 'last' to have the latest, 'first' to have the earliest values for timepoint 2.

4.1.3 adnipy.adnipy module

Process ADNI study data with adnipy.

adnipy.adnipy.**drop_dynamic**(*images*)
Remove images which are dynamic.

Drops all rows, in which the Description contains 'Dynamic'.

Parameters

images [pd.DataFrame] This dataframe will be modified.

Returns

pd.DataFrame All images that are not dynamic.

adnipy.adnipy.**get_matching_images**(*left*, *right*)
Match different scan types based on closest date.

The columns 'Subject ID' and 'SCANDATE' are required.

Parameters

left [pd.DataFrame] Dataframe containing the tau scans.

right [pd.DataFrame] Dataframe containing the mri scans.

Returns

pd.DataFrame For each timepoint there is a match from both inputs.

adnipy.adnipy.**groups**(*collection*, *grouped_mci=True*)
Create a dataframe for each group and save it to a csv file.

Parameters

collection [pd.DataFrame] DataFrame has to have a Group column.

grouped_mci [bool, default True] If true, 'LMCI' and 'EMCI' are treated like 'MCI'. However, the original values will stills be in the output.

Returns

dict Dictionnairy with a dataframe for each group.

adnipy.adnipy.**longitudinal**(*images*)
Keep only longitudinal data.

This requires an 'RID' or 'Subject ID' column in the dataframe. Do not use if multiple images are present for a single timepoint.

Parameters

images [pd.DataFrame] This dataframe will be modified.

Returns

pd.DataFrame A dataframe with only longitudinal data.

See also:

drop_dynamic

adnipy.adnipy.**read_csv**(*file*)
Return a csv file as a pandas.DataFrame.

Recognizes missing values used in the ADNI database.

Parameters

file [str, pathlib.Path] The path to the .csv file.

Returns

pd.DataFrame Returns the file as a dataframe.

See also:

standard_column_names

standard_dates

standard_index

adnipy.adnipy.**rid**(*collection*)
Add a roster ID column.

Will not work if 'RID' is already present or 'Subject ID' is missing.

Parameters

collection [pd.DataFrame] This dataframe will be modified.

Returns

pd.DataFrame Dataframe with a 'RID' column.

Examples

```
>>> collection = pd.DataFrame({"Subject ID": ["100_S_1000", "101_S_1001"]})
>>> collection
   Subject ID
0  100_S_1000
1  101_S_1001
>>> rid(collection)
   Subject ID   RID
0  100_S_1000  1000
1  101_S_1001  1001
```

adnipy.adnipy.**standard_column_names**(*dataframe*)

Rename dataframe columns to module standard.

This function helps when working with multiple dataframes, since the same data can have different names. It will also call *rid()* on the dataframe.

Parameters

dataframe [pd.DataFrame] This dataframe will be modified.

Returns

pd.DataFrame This will have standardized columns names.

See also:

rid

Examples

```
>>> subjects = pd.DataFrame({"Subject": ["101_S_1001", "102_S_1002"]})
>>> subjects
      Subject
0  101_S_1001
1  102_S_1002
>>> standard_column_names(subjects)
   Subject ID   RID
0  101_S_1001  1001
1  102_S_1002  1002
```

```
>>> images = pd.DataFrame({"Image": [100001, 100002]})
>>> images
    Image
0  100001
1  100002
>>> standard_column_names(images)
   Image ID
0    100001
1    100002
```

adnipy.adnipy.**standard_dates**(*dataset*)

Change type of date columns to datetime.

Parameters

dataset [pd.DataFrame] This dataframe will be modified.

Returns

pd.DataFrame Dates will have the appropriate dtype.

adnipy.adnipy.**standard_index**(*df*, *index=None*)
Process dataframes into a standardized format.

The output is easy to read. Applying functions the the output may not work as expected.

Parameters

df [pd.DataFrame] This dataframe will be modified.

index [list of str, default None] These columns will be the new index.

Returns

pd.DataFrame An easy to read dataframe for humans.

adnipy.adnipy.**timedelta**(*old*, *new*)
Get timedelta between timepoints.

Parameters

old [pd.DataFrame] This is the older dataframe.

new [pd.DataFrame] This is the newer dataframe.

Returns

pd.Series The content will be timedelta values. Look into numpy for more options.

adnipy.adnipy.**timepoints**(*df*, *second='first'*)
Extract timepoints from a dataframe.

Parameters

df [pd.DataFrame] This dataframe will be used as a base.

second [{'first' or 'last'}, default 'first'] 'last' to have the latest, 'first' to have the earliest values for timepoint 2.

4.1.4 adnipy.data module

Process data created in Matlab.

adnipy.data.**image_id_from_filename**(*filename*)
Extract image ID of single ADNI .nii filename.

Images from the ADNI database have a specific formatting. Using regular expressions the image ID can be extracted from filenames.

Parameters

filename [str] It must contain the Image ID at the end.

Returns

numpy.int64 Image as a integer.

Examples

```
>>> image_id_from_filename("*_I123456.nii")
123456
```

4.1.5 Module contents

Process ADNI study data with adnipy.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/mcsitter/adnipy/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

adnipy could always use more documentation, whether as part of the official adnipy docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mcsitter/adnipy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *adnipy* for local development.

1. Fork the *adnipy* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/adnipy.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv adnipy
$ cd adnipy/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 adnipy tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 3.5, 3.6 and 3.7. Check https://travis-ci.org/mcsitter/adnipy/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ $ py.test tests.test_adnipy
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Maximilian Cosmo Sitter <msitter@smail.uni-koeln.de>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.0.1 (2019-09-05)

- First release on GitHub.
- First release on PyPI.

7.2 0.1.0 (2019-10-25)

- Improved documentation.
- Added pandas dataframe class extension for ADNI

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

adnipy, 13
adnipy.adni, 7
adnipy.adnipy, 9
adnipy.data, 12

Index

A

ADNI (*class in adnipy.adni*), 7
adnipy (*module*), 13
adnipy.adni (*module*), 7
adnipy.adnipy (*module*), 9
adnipy.data (*module*), 12

D

drop_dynamic() (*adnipy.adni.ADNI method*), 7
drop_dynamic() (*in module adnipy.adnipy*), 9

G

get_matching_images() (*in module adnipy.adnipy*), 9
groups() (*adnipy.adni.ADNI method*), 7
groups() (*in module adnipy.adnipy*), 10

I

image_id_from_filename() (*in module adnipy.data*), 12

L

longitudinal() (*adnipy.adni.ADNI method*), 8
longitudinal() (*in module adnipy.adnipy*), 10

R

read_csv() (*in module adnipy.adnipy*), 10
rid() (*adnipy.adni.ADNI method*), 8
rid() (*in module adnipy.adnipy*), 10

S

standard_column_names() (*adnipy.adni.ADNI method*), 8
standard_column_names() (*in module adnipy.adnipy*), 11
standard_dates() (*adnipy.adni.ADNI method*), 9
standard_dates() (*in module adnipy.adnipy*), 11
standard_index() (*adnipy.adni.ADNI method*), 9
standard_index() (*in module adnipy.adnipy*), 12

T

timedelta() (*in module adnipy.adnipy*), 12
timepoints() (*adnipy.adni.ADNI method*), 9
timepoints() (*in module adnipy.adnipy*), 12