adnipy Documentation

Release 0.1.0

Maximilian Cosmo Sitter

Apr 29, 2021

Contents:

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

Dataframe deals with ADNI data.

This class presents methods, which are designed to work with data from the ADNI database.

DATES = ['Acq Date', 'Downloaded', 'EXAMDATE', 'EXAMDATE_bl', 'update_stamp', 'USERD

INDEX = ['Subject ID', 'Image ID']

MAPPER = {'ASSAYTIME': 'TAUTIME', 'Acq Date': 'SCANDATE', 'Image': 'Image ID', 'Im

drop_dynamic()

Remove images which are dynamic.

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

Returns All images that are not dynamic.

Return type pd.DataFrame

groups(*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 Dictionnairy with a dataframe for each group.

Return type dict

longitudinal()

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 A dataframe with only longitudinal data.

Return type pd.DataFrame

See also:

drop_dynamic()

rid()

Add a roster ID column.

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

Returns Dataframe with a 'RID' column.

Return type pd.DataFrame

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.adni.rid()
   Subject ID   RID
0  100_S_1000  1000
1  101_S_1001  1001
```

standard_column_names()

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 This will have standardized columns names.

Return type pd.DataFrame

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.adni.standard_column_names()
"VISCODE2" not included.
   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.adni.standard_column_names()
"VISCODE2" not included.
   Image ID
0    100001
1    100002
```

standard_dates()
Change type of date columns to datetime.

Returns Dates will have the appropriate dtype.

Return type pd.DataFrame

standard_index(*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 An easy to read dataframe for humans.

Return type pd.DataFrame

timepoints(*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.**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 For each timepoint there is a match from both inputs.

Return type pd.DataFrame

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 Returns the file as a dataframe.

Return type pd.DataFrame

See also:

standard_column_names(), standard_dates(), standard_index()

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

Return type pd.Series

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 Image as a integer.

Return type numpy.int64

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, 3.7 and 3.8. 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

Index