dachar Documentation

Release 0.1.0

Elle Smith

Jan 12, 2022

CONTENTS:

CHAPTER

ONE

DACHAR (PRON. "DAY-CAR")

The "dachar" package (pronounced "day-car", like René Descartes, a founder of modern science and philosophy) is a python library used to capture and analyse the *character* of scientific data sets. We typically focus on data sets held in the Earth System Grid Federation (ESGF) catalogues.

ESGF data sets are usually defined by the following characteristics:

- an identifier (string) that consists of an ordered set of facet values with a version identifier
- a single 2D or 3D geophysical variable over multiple time steps
- represented in one or more NetCDF files

Examples ESGF data sets are:

- **CMIP5**: `cmip5.output1.MPI-M.MPI-ESM-LR.decadal1995.mon.land.Lmon.r5i1p1.v20120529`
- **CORDEX**: `cordex.output.AFR-44.DMI.ECMWF-ERAINT.evaluation.r1i1p1.HIRHAM5.v2.day.uas.v20140804`
- Free software: BSD
- Documentation: https://dachar.readthedocs.io.

1.1 Features

There are three main stages to the characterisation process:

1. **Scan**: Scan all data sets and write a character file (JSON).
2. **Analysis**: Define *populations* of data sets (that might be processed together) and analyse each *population* to identify irregularities when comparing with other members of the population. Write the results of the analysis (JSON).
3. **Define Fixes**: Suggest fixes required to individual data sets to overcome the irregularities. Write the required fixes to a new set of files (JSON).

See below for using the cli to scan, analyse, propose fixes and process fixes. Character, analysis, fix and fix proposal records are stored on elasticsearch indices. Creating, deleting and writing to indices is described below. The elastic api token must be set in `etc/roocs.ini` in order to do these actions.

CHAPTER

TWO

CHARACTERISING

2.1 Scanning

```
$ dachar scan <project> -l <location>
```

e.g. `dachar scan c3s-cmip6 -l ceda`. This will scan all c3s-cmip6 datasets.

There are 2 different scanning modes available - either quick or full. Use `-m full` or `-m quick`. Quick scans can be overwritten with full scans using `-m full-force`.

Use `dachar scan -h` to see the options available for scanning specific datasets.

2.2 Analysing

To analyse populations of datasets. The sample id identifies the population to analyse.

```
$ dachar analyse -s <sample-id> <project> -l <location>
```

Using the flag *-f* will overwrite existing analysis records for the sample id.

2.3 Proposing Fixes

Analysis will automatically prpose fixes if any are found, however, if fixes are identified by another source they can be proposed.

There are different way of proposing fixes

1. By providing a JSON file of the fix. More than one JSON file can be provided.

```
$ dachar propose-fixes -f <json_file>,<json_file2>,<json_file3>
```

2. By providing a JSON template and a list of datasets that the fix should be proposed for.

```
$ dachar propose-fixes -t <json_template> -d <dataset_list>
```

See the directory `tests/test_fixes/decadal_fixes` for examples.

Note that if CMIP6 fixes are intended to be used for CDS datasets - the ds ids for the datasets must start with `c3s-cmip6` instead of `CMIP6`.

2.4 Processing Fixes

To publish or reject proposed fixes use:

```
$ dachar process-fixes -a process
```

This can also be used as:

```
$ dachar process-fixes -a process -d <dataset-id>,<dataset-id>
```

to process specific fixes.

To withdraw existing fixes, use:

```
$ dachar process-fixes -a withdraw -d <dataset-id>,<dataset-id>
```

To publish all fixes use:

```
$ dachar process-fixes -a publish-all
```

To reject all fixes use:

```
$ dachar process-fixes -a reject-all
```

In this case you will be prompted to give a reason for rejection. This will be applied to all fixes.

CHAPTER

THREE

ADDING TO ELASTICSEARCH

When a new version of the index is being created:

1. A new index must be created with new date. This can be done by creating an empty index or cloning the old one. Creating an empty index will just make a new index with the date of creation and update the alias to point to it if desired. Cloning creates a new index with the date of creation, fills it with all documents from the old index and updates the alias to point to it if desired.
2. It can then be populated either with all documents in local store or one document at a time.

3.1 Cloning an index

To create an index with today's date and populate it with all documents from another index.

```
$ python dachar/index/cli.py clone -i <index-to-create> -c <index-to-clone>
```

e.g. `python dachar/index/cli.py clone -i fix -c roocs-fix-2020-12-21`

To update the alias to point to this new index, provide the *-u* flag.

```
$ python dachar/index/cli.py clone -i <index-to-create> -c <index-to-clone> -u
```

3.2 Creating an empty index

To create an empty index with today's date.

```
$ python dachar/index/cli.py create -i <index-to-create>
```

e.g. `python dachar/index/cli.py create -i fix`

To update the alias to point to this new index, provide the *-u* flag.

```
$ python dachar/index/cli.py create -i <index-to-create> -u
```

3.3 Deleting an index

To delete an index.

```
$ python dachar/index/cli.py delete -i <index-to-delete>
```

e.g. `python dachar/index/cli.py delete -i roocs-fix-2020-12-21`

3.4 Populating an index from a local json store

Popluate an elasticsearch index with the contents of a local store.

```
$ python dachar/index/cli.py populate -s <store> -i <index-to-populate>
```

Store must be one of fix, fix-proposal, analysis or character.

e.g. `python dachar/index/cli.py populate -s fix -i roocs-fix-2020-12-21`

3.5 Adding one document to an existing index

To add one document from any file path to a store

```
$ python dachar/index/cli.py add-document -f <file-path> -d <drs-id> -i <index>
```

drs-id is what the id is called in the index i.e. either dataset_id (for fix, character and fix proposal store) or sample_id (for the analysis store)

e.g. `python dachar/index/cli.py add-document -f /path/to/doc.json -d c3s-cmip6.ScenarioMIP.INM.INM-CM5-0.ssp245.r1i1p1f1.Amon.rlds.gr1.v20190619 -i roocs-fix-2020-12-21`

CHAPTER

FOUR

CREDITS

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

- Cookiecutter: https://github.com/audreyr/cookiecutter
- cookiecutter-pypackage: https://github.com/cedadev/cookiecutter-pypackage

CHAPTER

FIVE

INSTALLATION

5.1 Stable release

To install dachar, run this command in your terminal:

```
$ pip install dachar
```

This is the preferred method to install dachar, 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.

5.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/ellesmith88/dachar
```

Or download the tarball:

```
$ curl  -OL https://github.com/ellesmith88/dachar/tarball/master
```

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

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

CHAPTER

SIX

USAGE

To use dachar in a project:

```
import dachar

For information on the configuration options available in daops, see: https://roocs-
↪utils.readthedocs.io/en/latest/configuration.html#dachar
```

CHAPTER

SEVEN

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:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/ellesmith88/dachar/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.

7.1.2 Fix Bugs

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

7.1.3 Implement Features

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

7.1.4 Write Documentation

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

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ellesmith88/dachar/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 :)

7.1.6 Get Started!

Ready to contribute? Here's how to set up `dachar` for local development.

#. Fork the `dachar` repo on GitHub. #.

   Clone your fork locally:

   $ git clone git@github.com:your_name_here/dachar.git

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

   $ mkvirtualenv dachar $ cd dachar/ $ python setup.py develop

2. Create a branch for local development:

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

   Now you can make your changes locally.

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

   $ flake8 dachar tests $ python setup.py test or py.test $ tox

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

4. 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

5. Submit a pull request through the GitHub website.

7.2 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.md.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/ellesmith88/dachar/pull_requests and make sure that the tests pass for all supported Python versions.

7.3 Tips

To run a subset of tests:

$ py.test tests.test_dachar

7.4 Deploying

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

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

Travis will then deploy to PyPI if tests pass.

CHAPTER

EIGHT

CREDITS

8.1 Developers

- Elle Smith eleanor.smith@stfc.ac.uk

8.2 Contributors

None yet. Why not be the first?

CHAPTER

NINE

HISTORY

0.1.0 (2020-03-26)

- First release on PyPI.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search