Hoyer, S and Hamman, J J 2017 xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software,* 5: 10, DOI: https://doi.org/10.5334/jors.148

SOFTWARE METAPAPER

xarray: N-D labeled Arrays and Datasets in Python

Stephan Hoyer[1,2] and Joseph J. Hamman[3]

[1] The Climate Corporation, San Francisco, CA, US

[2] Google Research, Mountain View, CA, US

[3] Department of Civil and Environmental Engineering, University of Washington, Seattle, WA, US

Corresponding author: Stephan Hoyer
(shoyer@google.com)

xarray is an open source project and Python package that provides a toolkit and data structures for N-dimensional labeled arrays. Our approach combines an application programing interface (API) inspired by pandas with the Common Data Model for self-described scientific data. Key features of the xarray package include label-based indexing and arithmetic, interoperability with the core scientific Python packages (e.g., pandas, NumPy, Matplotlib), out-of-core computation on datasets that don't fit into memory, a wide range of serialization and input/output (I/O) options, and advanced multi-dimensional data manipulation tools such as group-by and resampling. xarray, as a data model and analytics toolkit, has been widely adopted in the geoscience community but is also used more broadly for multi-dimensional data analysis in physics, machine learning and finance.

Keywords: Python; pandas; netCDF; multidimensional; data, data handling; data analysis

(1) Overview

Introduction

Python has emerged as a leading programing language for both the physical sciences and data sciences. At the core of modern scientific computing and analysis in Python are the NumPy [14] and SciPy [22] packages, which provide a robust N-dimensional array object and the fundamental operations required for science and engineering applications. Much of the success of Python in data science and business analytics is due to pandas [16], which introduced intuitive and fast tabular data analysis tools to Python, inspired by R's `data.frame` [19]. The pandas `DataFrame` and `Series` objects provide unparalleled analysis tools for data alignment, resampling, grouping, pivoting, and aggregation in Python.

xarray implements data structures and an analytics toolkit for multi-dimensional labeled arrays strongly inspired by pandas. While pandas includes a data structure called the `Panel` for three dimensional data, its fixed rank design make it unsuitable for applications that require arbitrary rank arrays. Additionally, many of the features that make the pandas `DataFrame` and `Series` objects so useful, are not fully available on the `Panel`. Our approach with xarray adopts Unidata's self-describing Common Data Model on which the network Common Data Form (netCDF) is built [20, 7]. NetCDF provides a well-defined data model for labeled N-dimensional array-oriented scientific data analysis.

xarray builds on top of, and seamlessly interoperates with, the core scientific Python packages, such as NumPy, SciPy, Matplotlib [13], and pandas. xarray provides a range of backends for serialization and input/output(IO), including the Pickle, netCDF, OPeNDAP (read-only), GRIB1/2 (read-only), and HDF file formats. Leveraging the dask parallel computing library [21], xarray can optionally perform efficient parallel, out-of-core analysis on datasets that are too large to fit into memory. Finally, xarray interfaces with existing domain-specific packages such as UV-CDAT [25], Iris [17], and Cartopy [18].

Purpose: Your data has labels; you should use them

Scientific data is inherently labeled. For example, time series data includes timestamps that label individual periods or points in time, spatial data has coordinates (e.g. longitude, latitude, elevation), and model or laboratory experiments are often identified by unique identifiers. **Figure 1** provides an example of a labeled dataset. In this case the data is a map of global air temperature from a numeric weather model. The labels on this particular dataset are time (e.g. "2016-05-01"), longitude (x-axis), and latitude (y-axis).

Unlabeled, N-dimensional arrays of numbers (e.g., NumPy's ndarray) are the most widely used data structure in scientific computing. However, they lack a meaningful representation of the metadata associated

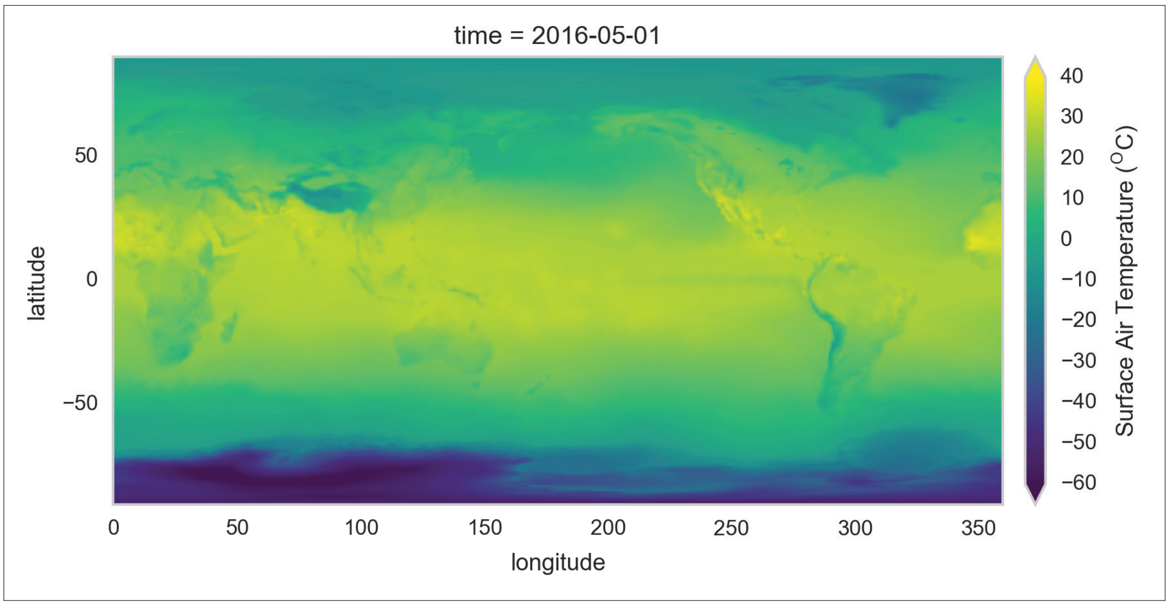

Figure 1: An example of a multidimensional labeled array. This figure (map) is showing the global surface air temperature for May 1, 2016 from ERA-Interim Reanalyis [11]. The map is labeled with the array's coordinates: longitude and latitude.

with their data. Implementing such functionality is left to individual users and domain-specific packages. As a result, programmers frequently encounter pitfalls in the form of questions like "is the time axis of my array in the first or third index position?" or "does my array of timestamps still align with my data after resampling?".

The core motivation for developing xarray was to provide labeled data tools for N-dimensional arrays that render such questions moot. Every operation in xarray both relies on and maintains the consistency of labels.

NetCDF

The network Common Data Form is a collection of self-describing, machine-independent binary data formats and software tools. These data formats and tools facilitate the creation, access, and sharing of scientific data stored in N-dimensional arrays, along with metadata describing the contents of each array [20]. NetCDF has become very popular in the geoscience community, and there are existing libraries for reading and writing netCDF in many programming languages, including C, Fortran, Python, Java, Matlab, and Julia.

The principal data structure in the netCDF data model is the dataset. Each netCDF dataset contains dimensions, variables, and attributes, each of which are identified by a hierarchy of unique names. The dataset and variable objects may contain attributes that describe the contents, units, history, or other metadata of the object. Standardized conventions, such as the Climate and Forecast (CF) Conventions [12], allow for the associations of coordinate variables with dimensions.

Implementation and architecture

NetCDF forms the basis of the xarray data model and provides a natural and portable serialization format. Building on netCDF, xarray features two main data structures: the `DataArray` and the `Dataset`. The API for these data structures is summarized in the following sections and in **Figure 2**.

DataArray

The `DataArray` is xarray's implementation of a labeled, multi-dimensional array. It has several key properties:

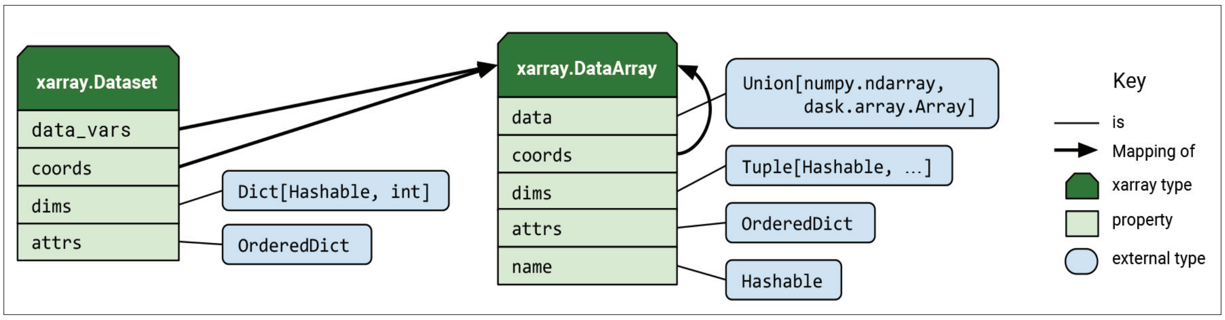

Figure 2: An overview of xarray's main data structures. Types are annotated using Python 3 style type hints [23]. "Mapping of" denotes an ordered mapping with values of the given type.

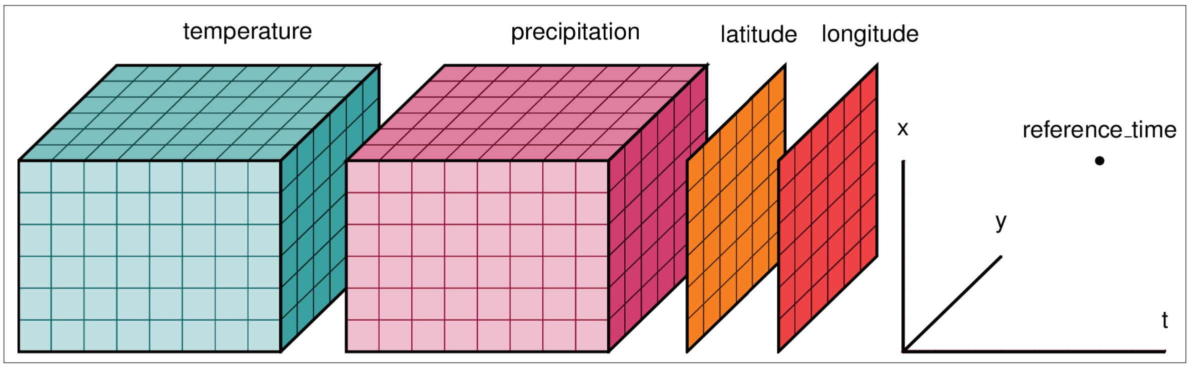

Figure 3: An example of how a dataset (netCDF or xarray) for a weather forecast might be structured. This dataset has three dimensions, time, y, and x, each of which is also a one-dimensional coordinate. *Temperature* and *precipitation* are three-dimensional data variables. Also included in the dataset are two-dimensional coordinates *latitude* and *longitude*, having dimensions y and x, and *reference time*, a zero-dimensional (scalar) coordinate.

- `data`: N-dimensional array (NumPy or dask) holding the array's values,
- `coords`: dict-like container of arrays (coordinates) that label each point (e.g., 1-dimensional arrays of numbers, `datetime` objects or strings),
- `dims`: dimension names for each axis [e.g., (`'time'`, `'latitude'`, `'longitude'`)],
- `attrs`: `OrderedDict` holding arbitrary metadata (e.g. units or descriptions), and
- `name`: an arbitrary name for the array.

xarray uses `dims` and `coords` to enable its core metadata-aware operations. Dimensions provide names that xarray uses instead of the axis argument found in many NumPy functions. Coordinates are ancillary variables used to enable fast label based indexing and alignment, building on the functionality of the pandas `Index`. `DataArray` objects also can have a name and can hold arbitrary metadata in the form of their `attrs` property, which can be used to further describe data (e.g. by providing units). Names and attributes are strictly for users and user-written code; in general xarray makes no attempt to interpret them, and propagates them only in unambiguous cases. In contrast, xarray does interpret and persist coordinates in operations that transform xarray objects.

Dataset

The `Dataset` is xarray's multi-dimensional equivalent of a `DataFrame`. It is a dict-like container of labeled arrays (`DataArrays`) with aligned dimensions. It is designed as an in-memory representation of a netCDF dataset. In addition to the dict-like interface of the dataset itself, which can be used to access any `DataArray` in a `Dataset`, datasets have four key properties:

- `data_vars`: `OrderedDict` of `DataArray` objects corresponding to data variables,
- `coords`: `OrderedDict` of `DataArray` objects intended to label points used in `data_vars` (e.g., 1-dimensional arrays of numbers, `datetime` objects or strings),
- `dims`: dictionary mapping from dimension names to the fixed length of each dimension (e.g., `{'x': 6, 'y': 6, 'time': 8}`), and
- `attrs`: `OrderedDict` to hold arbitrary metadata pertaining to the dataset.

`DataArray` objects inside a `Dataset` may have any number of dimensions but are presumed to share a common coordinate system. Coordinates can also have any number of dimensions but denote constant/independent quantities, unlike the varying/dependent quantities that belong in data. **Figure 3** illustrates these concepts for an example `Dataset` containing meteorological data.

Core xarray Features

xarray includes a powerful and growing feature set. The following list highlights some of the key features available in xarray. The xarray documentation [2] includes a complete description of available features and their usage.

- *Label-based indexing*: Similarly to pandas objects, xarray objects support both integer- and label-based lookups along each dimension. However, xarray objects also have named dimensions, so you can optionally use dimension names instead of relying on the positional ordering of dimensions.
- *Arithmetic*: arithmetic between xarray objects vectorizes based on dimension names, automatically looping (broadcasting) over each distinct dimension. This eliminates the need to insert dummy dimensions of size one to facilitate broadcasting, a common pattern with NumPy.
- *Aggregation*: calculation of statistics (e.g. sum) along a dimension of an xarray object can be done by dimension name instead of an integer axis number.
- *Alignment*: xarray supports database-like join operations for combining xarray objects along common coordinates.
- *Split-apply-combine*: xarray includes N-dimensional grouped operations implementing the split-apply-combine strategy [24].

- *Resampling and rolling window operations*: Utilizing the efficient resampling methods from pandas and rolling window operations from Bottleneck [15], xarray offers a robust set of resampling and rolling window operations along a single dimension.
- *Plotting*: xarray plotting functionality is a thin wrapper around the popular Matplotlib library. xarray uses the syntax and function names from Matplotlib whenever possible, resulting in a seamless transition between the two.
- *Missing Data*: xarray smoothly handles missing data in all operations, including arithmetic, alignment and aggregation.
- *Interactivity with pandas*: xarray objects seamlessly to convert to and from pandas objects to interact with the rest of the PyData ecosystem.
- *Serialization and I/O*: xarray supports direct serialization and I/O to several file formats including pickle, netCDF, OPeNDAP (read-only), GRIB1/2 (read-only), and HDF by integrating with third-party libraries. Additional serialization formats for 1-dimensional data are available through pandas.
- *Out-of-core computation*: xarray's data structures can be backed by dask [21] instead of NumPy to support parallel and streaming computation on data that does not fit into memory, up to 100s of GB or TBs in size. Such large datasets ("big data") are increasingly prevalent in science.

Quality control

xarray is provided with a large test suite comprised of over 1,500 unit tests. These tests cover the core xarray functionality as well as features facilitated by optional dependencies. The unit tests are executed automatically on the TravisCI (Linux) [5] and Appveyor (Windows) [1] continuous integration systems. A selection of sample data is also distributed with the source code, allowing users to reproduce any examples in the xarray documentation.

(2) Availability

Operating system

Linux, Windows and Mac OS X.

Programming language

Python, versions 2.7, 3.4 and later.

Additional system requirements

None.

Dependencies

xarray is implemented in pure Python and relies on compiled dependencies for speed.

- NumPy: 1.7 or later
- pandas: 0.15.0 or later
- netcdf4-python: (optional) used for reading and writing netCDF files
- SciPy: (optional) used as a fallback for reading/writing netCDF3
- Pydap: (optional) used as a fallback for accessing OPeNDAP
- h5netcdf: (optional) an alternative library for reading and writing netCDF4 files that does not use the netCDF-C libraries
- PyNIO: (optional) for reading GRIB1/2 and other geoscience specific file formats
- Bottleneck: (optional) speeds up NaN-skipping and rolling window aggregations by a large factor
- cyordereddict: (optional) speeds up most internal operations with xarray data structures
- Dask: (optional) required for out-of-core parallel computation
- Matplotlib: (optional) required for plotting
- Cartopy: (optional) required for plotting maps
- seaborn: (optional) additional plotting functionality

List of contributors

- Alex Kleeman, The Climate Corporation
- Alistair Miles, Wellcome Trust Centre for Human Genetics, University of Oxford
- Andreas Hilboll, Institute of Environmental Physics, University of Bremen
- Anna Kuznetsova, The Climate Corporation
- Antony Lee, Department of Physics, University of California, Berkeley
- Bas Hoonhout, Deltares, The Netherlands
- Benjamin Root, Atmospheric and Environmental Research, Inc.
- Benoit Bovy, Department of Astrophysics, Geophysics and Oceanography, University of Lige
- Chun-Wei Yuan, Institute for Health Metrics and Evaluation, University of Washington
- Christoph Deil, Max Planck Institute for Nuclear Physics
- Clark Fitzgerald, Department of Statistics, University of California, Davis
- Dean Pospisil, Institute for Learning and Brain Sciences, University of Washington
- Edward Richards, Scripps Institution of Oceanography, University of California, San Diego
- Erik Welch, Continuum Analytics
- Eugene Brevdo, Google Research
- Fabien Maussion, Institute of Atmospheric and Cryospheric Sciences, University of Innsbruck
- Filipe Fernandes, Universidade de So Paulo
- Guido Imperiale, Legal & General
- Hauser Mathias, Institute for Atmospheric and Climate Science, ETH Zurich
- Igor Babuschkin, School of Physics and Astronomy, University of Manchester
- Jeffrey Gerard, The Climate Corporation
- Jeremy McGibbon, Department of Atmospheric Sciences, University of Washington
- Jonathan Chambers, UCL Energy Institute, London UK
- Julia Signell, Department of Civil and Environmental Engineering, Princeton University
- Joseph Hamman, National Center for Atmospheric Research
- Johnnie Gray, University College London

- Jonathan Helmus, Environmental Science Division, Argonne National Laboratory
- Kelsey Jordahl, Planet Labs
- Maciek Swat, University of Pennsylvania
- Marco Zhlke, Brockmann Consult GmbH
- Markel Garca-Dez, Catalan Institute of Climate Sciences
- Markus Gonser
- Maximilian Roos, Sixty Capital
- Mike Graham, Edgestream Partners
- Nikolay Koldunov, Climate Service Center Germany
- Peter Cable, Raytheon
- Phillip J. Wolfram, Fluid and Solid Mechanics (T-3), Theoretical Division, Los Alamos National Laboratory
- Rafael Guedes, MetOcean Solutions Ltd
- Robin Wilson, Geography & Environment, University of Southampton, UK
- Ryan Abernathey, Department of Earth and Environmental Sciences, Columbia University
- Scott Sinclair, Satellite Applications and Hydrology Group, Dept. Civil Engineering, University of KwaZulu-Natal
- Sébastien Celles, Thermal Science and Energy Department, Poitiers Institute of Technology (IUT)
- Spencer A. Hill, Program in Atmospheric and Oceanic Sciences, Princeton University
- Stefan Pfenninger, Department of Environmental Systems Science, ETH Zrich
- Stephan Hoyer, Google Research
- Steven E. Pav, Gilgamath Consulting
- Takeshi Kanmae, National Institute of Polar Research, Japan
- Thomas Kluyver, Computational Modeling Group, University of Southampton
- Todd Small, The Climate Corporation
- Valliappa Lakshmanan, Google
- Yves Delley, Department of Physics, ETH Zurich

Software location

Archive

Name: Zenodo
Persistent identifier: https://doi.org/10.5281/zenodo.264282
License: Apache, v2.0
Publisher: Zenodo
Version published: 0.9.1
Date published: January 30, 2017

Code repository

Name: GitHub
Identifier: http://github.com/pydata/xarray
License: Apache, v2.0
Date published: January 30, 2017

Language

English.

(3) Reuse potential

xarray was written in a modular, objected-oriented way, to build upon and extend the core scientific Python libraries in a domain-agnostic fashion. The xarray documentation is complete with a wide range of examples and a number of tutorials that use real-world datasets that are available in the xarray repository. We have intentionally avoided including domain-specific functionality in the library, leaving that to third party libraries. It has been widely adopted in the geoscience community [e.g. 6, 10, 9], but has also been used in physics [e.g. 3], time series analytics [8], and finance. The core xarray data structures (the `DataArray` and the `Dataset`) are extensible through subclassing or the preferred approach of composition. We also provide an extensible high-level accessor interface to allow users to implement domain specific methods on xarray data objects.

xarray is developed and supported by a team of volunteers. The primary avenue for user support is *StackOverflow* [4], with the "xarray-python" tag. Additionally, we use GitHub for a bug tracker (https://github.com/pydata/xarray/issues) and maintain the "xarray" mailing list on Google Groups (https://groups.google.com/forum/#!forum/xarray).

Acknowledgements

Initial development of xarray was supported by The Climate Corporation. We thank Matthew Rocklin and Jim Crist for their assistance integrating xarray with Dask, and Todd Small, Francisco Alvarez and Fabien Maussion for their feedback on early drafts of this manuscript.

Competing Interests

The authors have no competing interests to declare.

References

1. **Appveyor.** https://ci.appveyor.com. Accessed: 2015-06-12.
2. **xarray documentation.** http://xarray.pydata.org. Accessed: 2017-01-30.
3. pycalphad: Computational thermodynamics. http://pycalphad.readthedocs.io. Accessed: 2015-06-12.
4. **Stack overflow.** http://stackoverflow.com/questions/tagged/python-xarray. Accessed: 2015-06-12.
5. Travis CI – test and deploy your code with confidence. https://travis-ci.org. Accessed: 2015-06-12.
6. xgcm: General circulation model postprocessing with xarray. http://xgcm.readthedocs.io. Accessed: 2015-06-12.
7. **Brown, S A, Folk, M, Goucher, G, Rew, R** and **Dubois, P F** 1993 Software for Portable Scientific Data Management. *Computers in Physics*, 7(3):304. DOI: https://doi.org/10.1063/1.4823180
8. **Cesium Development Team** 2016 *Cesium: Open-Source Machine Learning for Time Series Analysis.*
9. **Dawson, A** 2016 eofs: A library for eof analysis of meteorological, oceanographic, and climate data. *Journal of Open Research Software*, 4(1).
10. **Dawson, A, Irving, D, Filipe** and **Russo, A** 2016 windspharm: Version 1.5.0.
11. **Dee, D P, Uppala, S M, Simmons, A J, Berrisford, P, Poli, P, Kobayashi, S, Andrae, U, Balmaseda, M A, Balsamo, G, Bauer, P, Bechtold, P, Beljaars, A C M, van de Berg, L, Bidlot, J, Bormann, N, Delsol, C, Dragani, R, Fuentes, M, Geer, A J, Haimberger, L, Healy, S B,**

Hersbach, H, Hlm, E V, Isaksen, L, Kllberg, P, Khler, M, Matricardi, M, McNally, A P, Monge-Sanz, B M, Morcrette, J J, Park, B K, Peubey, C, de Rosnay, P, Tavolato, C, Thpaut, J N and **Vitart, F** 2011 The era-interim reanalysis: configuration and performance of the data assimilation system. *Quarterly Journal of the Royal Meteorological Society*, 137(656):553–597. DOI: https://doi.org/10.1002/qj.828

12. **Eaton, B, Gregory, J, Drach, B, Taylor, K, Hankin, S, Caron, J, Signell, R, Bentley, P, Rappa, G** and **Höck, H** et al 2003 NetCDF Climate and Forecast (CF) metadata conventions.
13. **Hunter, J D** 2007 Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.*, 9(3):90–95. DOI: https://doi.org/10.1109/MCSE.2007.55
14. **Jones, E, Oliphant, T, Peterson, P,** et al. 2001 SciPy: Open source scientific tools for Python. [Online; accessed 2015-11-06].
15. **Goodman, K** *Bottleneck*: *Fast NumPy array functions written in Cython*, 2016.
16. **McKinney, W** 2010 Data Structures for Statistical Computing in Python. In van der Walt, S and Millman, J (eds.), *Proceedings of the 9th Python in Science Conference*, pages 51–56.
17. **Met Office.** *Iris:* A *Python library for analysing and visualising meteorological and oceanographic data sets*. Exeter, Devon, v1.2 edition, 2010–2013.
18. **Met Office.** *Cartopy: a cartographic python library with a matplotlib interface*. Exeter, Devon, 2010–2015.
19. **R Core Team** 2013 *R:* A *Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
20. **Rew, R** and **Davis, G** 1990 NetCDF: an interface for scientific data access. *IEEE Comput. Grap. Appl.*, 10(4):76–82, Jul. DOI: https://doi.org/10.1109/38.56302
21. **Rocklin, M** 2015 Dask: Parallel computation with blocked algorithms and task scheduling. In Huff K and Bergstra, J (eds), *Proceedings of the 14th Python in Science Conference*, pages 130–136.
22. **van der Walt, S, Colbert, S C** and **Varoquaux, G** 2011 The NumPy Array: A Structure for Efficient Numerical Computation. *Comput. Sci. Eng.*, 13(2): 22–30, Mar. DOI: https://doi.org/10.1109/MCSE.2011.37
23. **van Rossum, G, Lehtosalo, J** and **Langa, L** 2016 PEP 484 – type hints. https: //www.python.org/dev/peps/pep-0484/. Accessed: 01–24.
24. **Wickham, H** 2011 The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software*, 40(1), DOI: https://doi.org/10.18637/jss.v040.i01
25. **Williams, D N, Doutriaux, C, Fries, S, Lipsa, D, Painter, J, McEnerney, J, Chaudhary, A, Jhaveri, S, Maxwell, T, Durack, P J,** et al. 2016 UV-CDAT 2.4.1.

How to cite this article: Hoyer, S and Hamman, J J 2017 xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software,* 5: 10, DOI: https://doi.org/10.5334/jors.148

Published: 07 September 2016 **Accepted:** 23 February 2017 **Published:** 05 April 2017