The Journal of Open Source Software

dantro: a Python package for handling, transforming, and visualizing hierarchically structured data

Yunus Sevinchan[1], **Benjamin Herdeanu**[1, 2], **and Jeremias Traub**[1]

1 Institute of Environmental Physics, Heidelberg University, Germany **2** Heidelberg Graduate School for Physics, Heidelberg University, Germany

DOI: 10.21105/joss.02316

Software

- Review
- Repository
- Archive

Editor: Lorena Mesa

Reviewers:

- @Chilipp
- @mdpiper

Submitted: 20 May 2020
Published: 23 August 2020

Summary

Researchers, especially those relying on computer simulations, frequently generate and analyze large amounts of data. With recent increases in computational capacity, the demand to efficiently and reliably process the generated data is growing accordingly. To address these needs, one powerful approach is to streamline the data handling, transformation, and visualization procedures into a *data processing pipeline*. Akin to the Continuous Integration pipelines widely used in modern software engineering, a data processing pipeline implements an automated sequence of predefined, yet dynamically configurable operations. Ultimately, using such a data processing pipeline can greatly improve the efficiency, reliability, and reproducibility of the scientific workflow.

However, we observed two challenges that impede the implementation and use of data processing pipelines in modeling-based research. First, data is often hierarchically structured, typically because it represents some underlying modularization in the investigated model. Second, the data may be semantically heterogeneous: this means that it may include many different data structures (e.g., numerical array-like data of different sizes, configuration files, metadata), or data that becomes meaningful only after processing. These properties make it difficult to handle data uniformly, thus greatly complicating the automatization efforts needed when constructing a processing pipeline.

`dantro` – from *data* and *dentro* (Greek for *tree*) – is a Python package that overcomes these problems; it does so by providing a uniform interface that handles hierarchically structured and semantically heterogeneous data. `dantro` is built around three main features, which constitute the stages of the data processing pipeline:

- **handling:** loading heterogeneous data into a tree-like data structure and providing a uniform interface for it
- **transformation:** efficiently performing arbitrary operations on the data
- **visualization:** creating a visual representation of the processed data

While plenty of established Python packages exist for each of these stages (e.g., `numpy` (van der Walt, Colbert, & Varoquaux, 2011), `xarray` (Hoyer & Hamman, 2017), `h5py` (Collette, 2013), `dask` (Dask Development Team, 2016; Rocklin, 2015), and `matplotlib` (Hunter, 2007)), coupling them into a processing pipeline can be difficult, especially when high generality and flexibility is desired. With `dantro`'s uniform data handling interface, the interoperability between these packages is simplified, while additionally allowing pipeline-specific specializations (see details section for more information). Furthermore, `dantro` is designed for a configuration-based specification of all operations via YAML configuration files (Ben-Kiki, Evans, & Net, 2009). Thus, once the pipeline is set up, it can be controlled entirely via these configuration files and without requiring any code changes. Taken together, the unique feature of `dantro` is that it conveniently combines the capabilities of many packages

and defines a data processing pipeline without having to take care of interfacing between the many involved packages.

Importantly, `dantro` is meant to be *integrated* into projects. While this integration process creates a one-time overhead during the setup of the pipeline, we believe that it offers more generality and supplies a wider feature set compared to any ready-to-use pipeline. To achieve a deep integration, the data structures provided by `dantro` define a shared interface and offer many possibilities to specialize and extend them to the requirements and data structures of the project it is integrated in. Effectively, the processing pipeline becomes a part of the project, growing and developing alongside it.

`dantro` was developed as part of the Utopia project (Riedel, Herdeanu, Mack, Sevinchan, & Weninger, 2020; Sevinchan et al., 2020), a modeling framework for complex and evolving systems. Within this project, `dantro` is used for automated, configuration-based data processing and plotting. While `dantro` was devised with simulation-based applications in mind, it is general enough to be used in all domains that require the handling, transformation, and visualization of data, also offering use cases for statistical or exploratory data analysis.

The `dantro` package is released under the LGPLv3+ and is available from the Python Package Index (PyPI) at `pypi.org/project/dantro` and from `conda-forge`. Its documentation, including integration instructions and examples, can be found at `dantro.readthedocs.io`.

Details

Corresponding to the three stages comprising a data processing pipeline, `dantro` is built upon three main modules: the data tree, a data transformation framework, and a plotting framework.

The Data Tree

The data tree is a representation of hierarchically structured data. Therein, `dantro` employs *data groups* and *data containers*, which constitute the branching points and the leaves of the tree, respectively. Abstract base classes are used to define the shared interface for all data types used in the tree.

Furthermore, `dantro` provides a set of mixin classes, which can be used to specialize a container or group for the data they are expected to hold. As an example, a group might hold containers that represent sequential simulation output; the group could then be specialized to represent a time series, offering the corresponding data selection capabilities. Another example is the `GraphGroup`, which generates a `networkx` (Hagberg, Schult, & Swart, 2008) graph object from the data available in the group. For numerical data, containers are available that specialize in holding `numpy` arrays or `xarray` data structures; these containers behave in the same way as the underlying arrays while additionally adhering to the `dantro` interface. As long as groups and containers concur to the interface of the data tree, they can be customized in any way that suits the data structures of the project `dantro` is used in.

At the root of the data tree is the `DataManager`, which is associated with a data directory; it takes care of loading data into the tree, forming a bridge between the data in memory and that inside the directory. The `DataManager` can be extended with a set of readily-available loaders for different file formats. For loading HDF5 files (The HDF Group, 1997), which are widely used to handle high volumes of hierarchically organized data, it uses `h5py` to load the file into the data tree while retaining its internal structure. `dantro` furthermore provides a *data proxy* construct, which acts as a placeholder for any form of data, effectively allowing it to delay the loading of the data until it becomes necessary. For numerical data, `dantro` combines this with the capabilities of the `dask` package to perform operations on data that cannot be loaded into memory at the same time.

The Data Transformation Framework

The data transformation framework relies on the uniform interface of the data tree, which allows it to perform arbitrary transformations on the data. This includes but is not limited to aggregating statistical measures, filtering data, or performing calculations using different entities from the data tree.

To achieve this, a pipeline user specifies a directed acyclic graph (DAG) of operations that are to be carried out on the data. These operations can be chosen from a set of predefined or user-specified operations, but in principle include all operations available in Python. Moreover, transformations can involve arbitrary objects in the data tree and file-based caching of operations can be used to alleviate repeating long computations, thus speeding up the data transformation procedure. Following `dantro`'s configuration-based approach, specification of the DAG and its arguments can happen entirely via YAML configuration files.

The Plotting Framework

The visualization of the potentially transformed data concludes the data processing pipeline. The `dantro` plotting framework consists of two main data structures: the `PlotManager` and the *plot creators*.

The `PlotManager` implements a configuration-based interface for defining which plots are to be created and further consolidates all overarching features. For example, the data tree is made available, the DAG framework is invoked, or plot configuration inheritance and composition are managed. Furthermore, using the `paramspace` package (Sevinchan, 2020), one can easily declare plot configurations with varying parameters, subsequently resulting in multiple output files.

Given the configuration and the data, the individual plot creators then focus on the actual visualization using a plotting backend. Currently, `dantro` provides plot creators that are based on `matplotlib`. Plotting procedures are defined as separate functions that use the familiar `matplotlib` interface. On top of that, these creators allow the user to specify plot aesthetics (e.g., style sheet, RC parameters) via the plot configuration and simplify the creation of animations. For instance, to generically represent high-dimensional data, `dantro` integrates `xarray`'s plotting capabilities and uses the animation feature to make one additional visualization dimension available. Aside from the integrated plotting functions, custom plotting functions can be easily defined and benefit from the same infrastructure.

As part of a pipeline, additional plot creators can be defined that use a different plotting backend or focus on some specific visualization aspects. As an example, one intriguing addition to the set of plot creators would be to utilize the `altair` framework (VanderPlas et al., 2018) where plots can be specified via the Vega-Lite visualization grammar (Satyanarayan, Moritz, Wongsuphasawat, & Heer, 2017). Such a declarative plotting approach would be well-suited for the visualization stage of a `dantro`-based data processing pipeline, further simplifying and generalizing the bridge between transformed data and its visualization.

Acknowledgments

We would like to thank Unai Fischer Abaigar, Daniel Lake, and Julian Weninger for their contributions to `dantro`. We thank Maria Blöchl for comments on an earlier version of this manuscript. We are grateful to Kurt Roth for feedback on the manuscript and his support during the development of this project.

References

Ben-Kiki, O., Evans, C., & Net, I. döt. (2009). YAML Ain't Markup Language, v1.2. Retrieved from https://yaml.org/spec/1.2/spec.html

Collette, A. (2013). *Python and HDF5*. O'Reilly Media.

Dask Development Team. (2016). *Dask: Library for dynamic task scheduling*. Retrieved from https://dask.org

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15). Pasadena, CA USA.

Hoyer, S., & Hamman, J. (2017). Xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1). doi:10.5334/jors.148

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

Riedel, L., Herdeanu, B., Mack, H., Sevinchan, Y., & Weninger, J. (2020). Utopia: A comprehensive and collaborative modeling framework for complex and evolving systems. *Journal of Open Source Software*. doi:10.21105/joss.02165

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In K. Huff & J. Bergstra (Eds.), *Proceedings of the 14th python in science conference* (pp. 130–136). doi:10.25080/majora-7b98e3ed-013

Satyanarayan, A., Moritz, D., Wongsuphasawat, K., & Heer, J. (2017). Vega-lite: A grammar of interactive graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*. doi:10.1109/TVCG.2016.2599030

Sevinchan, Y. (2020). *Paramspace v2.4.1*. Zenodo. doi:10.5281/zenodo.3826470

Sevinchan, Y., Herdeanu, B., Mack, H., Riedel, L., & Roth, K. (2020). Boosting group-level synergies by using a shared modeling framework. In *Lecture notes in computer science* (pp. 442–456). Springer International Publishing. doi:10.1007/978-3-030-50436-6_32

The HDF Group. (1997). Hierarchical data format version 5. Retrieved from http://www.hdfgroup.org/HDF5

VanderPlas, J., Granger, B., Heer, J., Moritz, D., Wongsuphasawat, K., Satyanarayan, A., Lees, E., et al. (2018). Altair: Interactive statistical visualizations for python. *Journal of Open Source Software*. doi:10.21105/joss.01057

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, *13*(2), 22–30. doi:10.1109/MCSE.2011.37