Pandas

Abstract

This proposal seeks funding for the continued maintenance and development of Pandas (McKinney (2010)), a Python library providing high-performance, easy-to-use data structures. Pandas is a foundational library in the Scientific Python Ecosystem, providing the most widely-used data structures for heterogenous, tabular data, and is the most-used Python tag on StackOverflow (Robinson (2019)). Pandas documentation averages over 1,000,000 unique vistors per month.

The proposal consists of library maintenance, improving the extension array interface, a native extension type for text data, and documentation tooling improvements.

1 Goals

1.1 Library Maintenance

Pandas is a large library with many users. We have many open issues (about 3,000) and Pull Requests (about 100). Keeping up with the stream of updates is time consuming, but important to the project. We would like to fund time dedicated specifically to maintenance.

Dedicated maintenance time, including periodic reviews of the open issue backlog, will result in higher average quality of the open issues. We hope this will increase the growth rate of new contributors, as the barrier to contributing is lowered.

We expect to see the number of open issues and pull requests decline.

See Library Maintenance for how we plan to achieve this goal.

1.2 Extension Types

Pandas Extension Array interface enables working with arrays of data that aren't limited to NumPy's data types and in-memory data model. Pandas provides several extension types (categorical, datetime with timezone, interval), and third-party libraries implementing the interface can define their own custom data types.

Many parts of pandas still don't fully handle such extension types, leading to special cases in the code (hurting maintainability), places where the data is unintentionally converted to a NumPy array (with possible performance implications) and missing functionalities for those data types. These problems are especially pronounced for nested data.

We'd like to improve the handling of extension types throughout the library, making their behavior more consistent with the handling of NumPy arrays. We'll do this by cleaning up pandas' internals and adding new methods to the extension array interface. The goal is to better enable new extension types (such as a native string data type, see below) and to ensure that the existing types such as the integer data type with missing values support can be a full replacement for the default numpy-based type.

To measure the outcome of this item, we expect that the number of open extension-array-related issues to decline, and that the number of special cases in pandas' internals to be minimized.

See Extension Types for how we plan to achieve this goal.

1.3 Native String Data Type

Currently, pandas stores text data in an `object`-dtype NumPy array. The current implementation has two primary drawbacks:

1. `object`-dtype is not specific to strings: any Python object can be stored in an `object`-dtype array, not just strings.
2. Storing an array of Python strings as an `object`-dtype array is not memory efficient. The NumPy memory model isn't especially well-suited to variable width text data.

To solve the first issue, we would like to implement a new extension type for string data. This will initially be opt-in, with users explicitly requesting `dtype="string"`. Over time, we'd like to provide a migration path to users so that the string extension type is used by default for text data.

To solve the second issue (memory efficiency), we'll explore alternative in-memory array libraries (for example, Apache Arrow). Using a library whose data model is better-suited to text data should result in lower memory usage when loaded into pandas. We also expect that operations (for example `Series.str.upper`) will be faster.

The second issue with pandas' current string implementation will be considered complete when a `string`-dtype array is no longer backed by a NumPy array of Python string objects.

1.4 Documentation Validation

To improve the quality and consistency of Pandas documentation, we've developed tooling to check docstrings in a variety of ways. Every docstring is checked for

1. Consistency: Ensuring that every docstring has the same components (parameters, return values, notes, examples, etc.). Ensuring that each section is formatted correctly.
2. Completeness: Ensuring that every function parameter is documented.
3. Correctness: Ensuring that the examples run correctly.

When a user submits a Pull Request to pandas, their changes to the documentation are automatically checked and informative error messages notify them of any issues.

Like many other projects, pandas uses the numpydoc standard for writing docstrings. With the collaboration of the numpydoc maintainers, we'd like to move the project-agnostic tooling we've written to a package outside of pandas, which any numpydoc-using project can benefit from.

If possible, we'd like this project to be undertaken by a member of an unrepresented minority, with mentorship provided by the pandas maintaiers. This project primarily requires experience with *using* pandas, NumPy, and related libraries, rather that deep knowledge of pandas' internals. Pandas has a diverse community, just not at the maintainer level (yet).

1.5 Performance Monitoring

Pandas uses airspeed velocity to monitor for performance regressions. ASV itself is a fabulous tool, but requires some additional work to be integrated into an open source project's workflow.

The asv-runner GitHub organization, currently made up of pandas maintainers, provides tools built on top of ASV. We have a physical machine for running a number of project's benchmarks and tools managing the benchmark runs and reporting on results.

We'd like to fund improvements and maintenance of these tools to

- Be more reliable. Currently, they're maintained on the nights and weekends when a maintainer has free time. This occasionally results in days or weeks of downtime.
- Tune the system to improve benchmark stability
- Report performance regressions to a project rather than relying on maintainers manually checking for regressions.

- Build a GitHub bot to request ASV runs *before* a Pull Request is merged. The benchmarks are too expensive to run as part of every commit. Running on-demand from a maintainer provides a nice balance.
- Pay for hosting a dedicated benchmark server. A maintainer's basement isn't the most stable environment for a machine.

This project has an impact beyond pandas. We run the benchmarks for many foundational libraries in the Scientific Python Ecosystem including xarray, Dask, scikit-image, scikit-learn, and PyMC3. Each of these projects would benefit from improvements made to the tooling.

2 Work plan

2.1 Library Maintenance

Maintainers funded by this grant would be expected to

1. Promptly triage newly opened issues.
2. Promptly review new pull requests.
3. Ensure conversations on open issues are progressing and not waiting on maintainer feedback.
4. Ensure that Pull Requests are not going stale waiting for maintainer feedback or contributor responses to feedback.
5. Periodically review the issue backlog to find and close issues that are duplicates or no longer relevant. We should ensure that open issues are clearly described and scoped.
6. Review and address performance regressions.
7. Engage in discussions on the pandas mailing list.

This item requires some familiarity with pandas codebase, community, and workflow. We hope to draw from pandas' current pool of maintainers and contributors, or pandas (unofficial) mentorship program to find people with the necessary skills and experience.

2.2 Extension Types

This item is somewhat open-ended. New issues will be discovered as extension arrays are adopted by the pandas community. That said, there are a few concrete starting points.

2.3 Native String Data Type

The two aspects of this item (`string` extension type and an alternative in-memory array library) can largely proceed in parallel.

We will implement `StringDtype`, a new `ExtensionDtype`, whose scalar type is `str`. The array for that type will be `StingArray`, a new `ExtensionArray`. Because the first aspect of this item is just providing a new user-API for the same functionality, this extension type will be relatively straightforward to implement. It can closely follow the `PandasArray` implementation, but limit the allowed data types to just `object`. The constructor and mutation methods may limit the allowed scalar values to just be instances of `str`.

Initially, `StringArray` need not have any API-breaking changes. We will make it opt-in

```
>>> s = pd.Series(['a', 'b', 'c'], dtype='string')
>>> s
0    a
1    b
2    c
dtype: string
>>> s.array
StringArray(['a', 'b', 'c'])
```

Over time, we might consider deprecating the current behavior of inferring an untyped array of strings to be `object` dtype, in favor of `StringDtype`.

The second component, updating the string extension array to use an alternative in-memory array library, will be a larger effort. First, we'll need to decide which array library to use (Apache Arrow, awkward-array, or some other library). Second, we'll need to do the actual implementation.

If Apache Arrow is chosen, many string algorithms will need to be implemented, preferable in the Apache Arrow C++ library so that other languages binding to the C++ library benefit as well. This is within scope for Apache Arrow[1].

2.4 Documentation Validation

This item may be implemented by anyone familiar with using pandas. Experience with the pandas codebase is not required.

The implementer will need to coordinate with many new contributors, as documentation improvements are the most common type of contribution by first-time contributors.

The implementer will start by improving our existing tooling within the pandas repository, to clean up the rough edges of the current system.

We will collaborate with the numpydoc maintainers and other numpydoc users to find a suitable home for the checks (within numpydoc itself, or a separate package).

[1] See https://issues.apache.org/jira/browse/ARROW-555

The numpydoc maintainers have expressed interest in docstring validation being part of numpydoc: https://github.com/numpy/numpydoc/issues/213.

2.5 Performance Monitoring

This item does not require familiarity with pandas.

Work can largely be broken into two buckets: maintenance and feature development. These two buckets can proceed in parallel.

For developing a GitHub bot, we'll Heroku to respond to GitHub events.

3 Existing Support

Pandas currently has about 1.6 full-time equivalent maintainers.

- Anaconda Inc funds Tom Augspurger (50% FTE) and Brock Mendel (100% FTE)
- Ursa Labs funds Joris van den Bossche (10% FTE).
- NumFocus Small Development Grant for "Improving and modernizing the introductory *Getting Started* pages of the pandas documentation" ($5,000)

An up-to-date list of current members and institutional parters can be found in our governance documents.

References

McKinney, Wes. 2010. "Data Structures for Statistical Computing in Python." In *Proceedings of the 9th Python in Science Conference*, edited by Stéfan van der Walt and Jarrod Millman, 51–56.

Robinson, David. 2019. "Why Is Python Growing so Quickly?" *StackOverflow Blog*. StackOverflow. https://stackoverflow.blog/2017/09/14/python-growing-quickly/.