Ensuring the continued growth of 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 improvements to pandas documentation and performance monitoring tooling.

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.

We expect to see the number of open issues and pull requests decline while maintainers are dedicating additional time to maintenance.

We also expect to increase the growth-rate of new contributors to the project. As maintenance time cleans up the issue backlog, the average quality of the outstanding open issues will rise. This will lower the barrier to contributing.

See Library Maintenance for how we will achieve this goal.

1.2 Extension Types

Pandas Extension Array interface[1] enables working with arrays of data that aren't limited to NumPy's data types and in-memory data model. Pandas

[1] https://pandas.pydata.org/pandas-docs/stable/development/extending.html#extension-types.

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 will 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, leading to confusion and bugs when doing dtype-specific operations.
2. Storing an array of Python strings as an `object`-dtype array is not memory efficient. The NumPy memory model isn't well-suited to variable-width text data.

To solve the first issue, we'll implement a new extension type, `StringArray`, specifically for text data. With `StringArray`, users can write clearer and more correct code when working with text data. This sub-item will be considered complete when a `StringArray` is available in a released version of pandas.

To solve the second issue (memory efficiency), we'll change `StringArray` to be backed by an alternative in-memory array, rather than a NumPy array of Python strings. The alternative backing array will give better memory-efficiency, letting users work with larger datasets in memory and enable faster throughput in string operations. This sub-item will be considered complete when `StringArray` is backed by an alternative array, and (at least the most common) string operations can be applied to that array.

See Native String Data Type for how we plan to achieve this goal.

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 by our continuous integration system. Any issues are logged with informative error messages describing the issue with the change.

We would like to improve the quality of this tooling by fixing bugs, catching more issues, improving feedback, and improving the documentation of the tooling.

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 maintainers. This project primarily requires experience with *using* pandas, NumPy, and related libraries, rather that deep knowledge of pandas' internals.

See Documentation Validation for how we plan to achieve this goal.

1.5 Performance Monitoring

Pandas uses airspeed velocity[2] to monitor for performance regressions. airspeed velocity itself is a fabulous tool, but requires some additional work to be integrated into an open source project's workflow.

The asv-runner[3] GitHub organization, currently made up of pandas maintainers, provides tools built on top of airspeed velocity. 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

[2] See https://asv.readthedocs.io/en/stable/.
[3] See https://github.com/asv-runner.

- Report performance regressions to a project rather than relying on maintainers manually checking for regressions.
- 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
- Build a GitHub bot to request airspeed velocity 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.

See Performance Monitoring for how we plan to achieve this goal.

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.
8. Mentoring contributors, especially from those from underrepresented groups, who would like to become maintainers.

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.

We expect this to take about 1 FTE over the course of the grant.

2.2 Extension Types

This work item will require some familiarity with pandas' internals. We plan to more deeply integrate extension arrays into pandas, so that all columns in a `DataFrame` are backed by an `ExtensionArray`. This will simplify pandas internals and algorithms, improving the maintainability of the project.

Additionally, changing every column to be backed by an `ExtensionArray` will surface bugs in pandas internals, algorithms, and shortcomings of the extension array interface. We will fix those issues, and every extension type, including 3rd-party implementations, will benefit.

Finally, we will improve test coverage for extension arrays for nested data. This will uncover issues that are specific to having "scalar" values of a dtype be collections themselves.

We expect this work to take about 1/4 FTE.

2.3 Native String Data Type

The two aspects of this item (a new `StringArray` and an alternative array library backing `StringArray`) can largely proceed in parallel.

We will implement a new extension type dedicated to string data. Initially, this implementation will still be backed by an object-dtype NumPy array of Python strings. This work will be mostly self-contained, requiring few changes to the broader pandas library.

The second component, updating the string extension type to use an alternative backing 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 of `StringArray` to be backed by the alternative array. Finally, because we aren't using Python strings anymore, we may need to re-implement basic string algorithms (like `str.upper`) to work on the alternative array library's memory.

We expect this work to take about ½ FTE.

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 start by improving our existing tooling within the pandas repository, to clean up the rough edges of the current system before exposing it to other packages. Some aspects of the tooling will have made assumptions that only work for pandas and will need to be made project-agnostic.

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). The numpydoc maintainers have expressed interest in docstring validation being part of numpydoc[4].

The implementer will need to coordinate with many new contributors to pandas, who often start contributing by improving pandas' documentation.

We expect this to take about ½ FTE.

2.5 Performance Monitoring

This item does not require deep familiarity with pandas.

We will prioritize reporting performance regressions. When a performance regression is detected, we will use the GitHub API to notify the project which commit caused the slowdown[5].

Meanwhile, we'd also like to improve the stability and reliability of the nightly benchmark runner. We need monitoring tools for when the Airflow service running the benchmarks goes down, and possibly alternative hosting for the physical machine[6].

Finally, implementing the feature to let GitHub maintainers request benchmark runs will require writing a GitHub bot to respond to requests. This will require communication with the benchmark machine to ensure that a maintainer-requested benchmark does not run a the same time as another benchmark run (either maintainer-requested or scheduler nightly run).

We expect this to take ¼ FTE.

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.

[4] See https://github.com/numpy/numpydoc/issues/213.

[5] https://github.com/asv-runner/asv-watcher has a promising proof of concept.

[6] The basement of a house with a three-year-old who enjoys pushing glowing buttons is not a safe environment for a server.

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/.