xndframes Documentation

Release v0.1.0+23.ga2b4491.dirty

Anderson Banihirwe

Jul 24, 2018

Contents:

CHAPTER 1

Installation

1.1 Stable release

To install xndframes, run this command in your terminal:

```
$ pip install xndframes
```

This is the preferred method to install xndframes, 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.

1.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/Quansight/xndframes
```

Or download the tarball:

```
$ curl  -OL https://github.com/Quansight/xndframes/tarball/master
```

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

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

CHAPTER 2

Usage

Xndframes is meant to provide a set of Pandas ExtensionDType/Array implementations backed by xnd

This document describes how to use the methods and classes provided by `xndframes`.

We will assume that the following packages have been imported.

```
In [1]: import xndframes as xf

In [2]: import pandas as pd
```

2.1 Pandas Integration

So far, xndframes implements `XndframesArray`. `XndframesArray` satisfies pandas extension array interface, which means that it can safely be stored inside pandas's Series and DataFrame.

```
In [3]: s = ["Pandas", "NumPy", "xnd", "SciPy", None, "CuPy", None, "Keras", "Numba"]

In [4]: packages = xf.XndframesArray(s)

In [5]: type(packages)
Out[5]: xndframes.base.XndframesArray

In [6]: print(packages.data)
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\xnd(['Pandas', 'NumPy', 'xnd', 'SciPy', None,
↪'CuPy', None, 'Keras', 'Numba'], type='9 * ?string')

In [7]: ser = pd.Series(packages)

In [8]: ser
Out[8]:
0    Pandas
1     NumPy
```

(continues on next page)

(continued from previous page)

```
2        xnd
3      SciPy
4       None
5       CuPy
6       None
7      Keras
8      Numba
dtype: xndframes[9 * ?string]

In [9]: vals = list(range(9))

In [10]: values = xf.XndframesArray(vals)

In [11]: ser2 = pd.Series(values)

In [12]: ser2
Out[12]:
0    0
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
dtype: xndframes[9 * int64]

In [13]: df = pd.DataFrame({"packages": packages, "id": values})

In [14]: df.head()
Out[14]:
  packages id
0   Pandas  0
1    NumPy  1
2      xnd  2
3    SciPy  3
4     None  4

In [15]: df
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[15
↪
  packages id
0   Pandas  0
1    NumPy  1
2      xnd  2
3    SciPy  3
4     None  4
5     CuPy  5
6     None  6
7    Keras  7
8    Numba  8

In [16]: df.describe()
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪
       packages  id
```

(continues on next page)

(continued from previous page)

```
count          7   9
unique         7   9
top        Keras   8
freq           1   1
```

Most pandas methods that make sense should work. The following section will call out points of interest.

```
In [17]: packages.shape
Out[17]: (9,)

In [18]: packages.unique()
\\\\\\\\\\\\\\Out[18]: <xndframes.base.XndframesArray at 0x7fa954a4ef98>

In [19]: packages.unique().data
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[19]:␣
↪xnd(['Pandas', 'NumPy', 'xnd', 'SciPy', None, 'CuPy', 'Keras', 'Numba'], type='8 * ?
↪string')

In [20]: packages.isna()
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪array([False, False, False, False,  True, False,  True, False, False])

In [21]: df.info()
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 2 columns):
packages    7 non-null xndframes[9 * ?string]
id          9 non-null xndframes[9 * int64]
dtypes: xndframes[9 * ?string](1), xndframes[9 * int64](1)
memory usage: 224.0 bytes
```

2.1.1 Indexing

If your selection returns a scalar, you get back a string.

```
In [22]: ser[0]
Out[22]: 'Pandas'

In [23]: df.loc[2, "packages"]
\\\\\\\\\\\\\\\\\\Out[23]: 'xnd'
```

2.1.2 Missing Data

xnd uses *None* to represent missing values. Xndframes does the same.

```
In [24]: df.isna()
Out[24]:
   packages     id
0     False  False
1     False  False
2     False  False
3     False  False
```

(continues on next page)

(continued from previous page)

```
4      True  False
5     False  False
6      True  False
7     False  False
8     False  False

In [25]: df.dropna()
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪
  packages id
0   Pandas  0
1    NumPy  1
2      xnd  2
3    SciPy  3
5     CuPy  5
7    Keras  7
8    Numba  8
```

CHAPTER 3

API

Xndframes provides one extension type, :class: *XndframesArray*

class *String Array*

3.1 Methods

Various methods that are useful for pandas. When a Series contains a XndframesArray, calling the Series method will dispatch to these methods.

`XndframesArray.`**`take`**`(`*indices*, *allow_fill=False*, *fill_value=None*`)`

Take elements from an array.

Parameters

indices [sequence of integers] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*. * False: negative values in *indices* indicate position indices

from the right (the default). This is similar to `numpy.take()`.

- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used. For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if necessary.

Returns

ExtensionArray

Raises

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contain negative values other than `-1` and *allow_fill* is True.

Notes

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignemnt, with a *fill_value*. See Also ——— numpy.take pandas.api.extensions.take

`XndframesArray.`**`unique`**`()`

Compute the ExtensionArray of unique values.

Returns

uniques [ExtensionArray]

`XndframesArray.`**`isna`**`()`

Boolean NumPy array indicating if each value is missing. This should return a 1-D array the same length as 'self'.

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/Quansight/xndframes/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

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

4.2 Get Started!

We strongly recommend using a *conda* based development setup. Ready to contribute? Here's how to set up *xndframes* for local development.

1. Fork the *xndframes* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/xndframes.git
   ```

3. Install your local copy into a *conda* environment. Assuming you have *conda* installed, this is how you set up your fork for local development:

   ```
   $ cd xndframes/
   $ conda env create -f ci/environment.yml
   $ conda activate xndframes
   $ python setup.py develop
   ```

4. Create a branch for local development:

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

   Now you can make your changes locally.

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

   ```
   $ flake8 xndframes tests
   $ python setup.py test or py.test
   $ tox
   ```

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

6. 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
   ```

7. Submit a pull request through the GitHub website.

4.3 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.rst.
3. The pull request should work for Python 3.6+. Check https://travis-ci.org/Quansight/xndframes/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_xndframes
```

4.5 Deploying

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

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

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

History

5.1 0.1.0 (2018-06-16)

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index