Raster To DataFrame Documentation

Release 0.2.1

Martin Black

Feb 13, 2019

Contents:

CHAPTER 1

Raster To DataFrame

A simple python module that converts a raster to a Pandas DataFrame.

```
from rastertodataframe import raster_to_dataframe

raster_path = '/some/gdal/compatible/file.tif'
vector_path = '/some/ogr/compatible/file.geojson'

# Extract all image pixels (no vector).
df = raster_to_dataframe(raster_path)

# Extract only pixels the vector touches and include the vector metadata.
df = raster_to_dataframe(raster_path, vector_path=vector_path)
```

- Free software: MIT license
- Documentation: https://rastertodataframe.readthedocs.io.

1.1 Features

- Convert any GDAL compatible raster to a Pandas DataFrame.
- Optionally, if any OGR compatible vector file is given, only pixels touched by the vector are extracted from the raster. The output DataFrame includes these pixels as well as any attributes from the vector file.

1.2 Installation

```
pip install rastertodataframe
```

- A working GDAL/OGR installation is required. This is best accomplished with conda.

```
conda install -c conda-forge numpy gdal geopandas pandas pyproj
```

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Raster To DataFrame, run this command in your terminal:

```
$ pip install rastertodataframe
```

This is the preferred method to install Raster To DataFrame, 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.

2.2 From sources

The sources for Raster To DataFrame can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/mblack20/rastertodataframe
```

Or download the tarball:

```
$ curl  -OL https://github.com/mblack20/rastertodataframe/tarball/master
```

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

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

CHAPTER 3

Usage

To use Raster To DataFrame in a project:

```python
from rastertodataframe import raster_to_dataframe

raster_path = '/some/gdal/compatible/file.tif'
vector_path = '/some/ogr/compatible/file.geojson'

# Extract all image pixels (no vector).
df = raster_to_dataframe(raster_path)

# Extract only pixels the vector touches and include the vector metadata.
df = raster_to_dataframe(raster_path, vector_path=vector_path)
```

CHAPTER 4

rastertodataframe

4.1 rastertodataframe package

Top-level package for Raster To DataFrame.

4.1.1 Submodules

rastertodataframe.rastertodataframe module

`rastertodataframe.rastertodataframe.`**`raster_to_dataframe`**`(`*raster_path*, *vector_path=None*`)`

Convert a raster to a Pandas DataFrame.

Parameters

- **`raster_path`** (*`str`*) – Path to raster file.
- **`vector_path`** (*`str`*) – Optional path to vector file. If given, raster pixels will be extracted from features in the vector. If None, all raster pixels are converted to a DataFrame.

Returns

Return type pandas.core.frame.DataFrame

rastertodataframe.tiling module

Utils for reading a GDAL Dataset in small tiles.

`rastertodataframe.tiling.`**`tiles`**`(`*ras*, *size=256*`)`

Generator return a raster array in tiles.

Parameters

- **`ras`** (*`gdal.Dataset`*) – Input raster.

- **size** (*int*) – Size of window in pixels. One value required which is used for both the x and y size. E.g 256 means a 256x256 window.

Yields *np.ndarray* – Raster array in form [band][y][x].

`rastertodataframe.tiling.`**`windows`**(*ras*, *size=256*)
Generator for raster window size/offsets.

Parameters

- **ras** (*gdal.Dataset*) – Input raster.
- **size** (*int*) – Size of window in pixels. One value required which is used for both the x and y size. E.g 256 means a 256x256 window.

Yields *tuple[int]* – 4 element tuple containing the x size, y size, x offset and y offset of the window.

rastertodataframe.util module

`rastertodataframe.util.`**`burn_vector_mask_into_raster`**(*raster_path*, *vector_path*, *out_path*, *vector_field=None*)
Create a new raster based on the input raster with vector features burned into the raster. To be used as a mask for pixels in the vector.

Parameters

- **raster_path** (*str*) –
- **vector_path** (*str*) –
- **out_path** (*str*) – Path for output raster. Format and Datatype are the same as `ras`.
- **vector_field** (*str or None*) – Name of a field in the vector to burn values from. If None, all vector features are burned with a constant value of 1.

Returns Single band raster with vector geometries burned.

Return type gdal.Dataset

`rastertodataframe.util.`**`get_epsg`**(*data*)
Get the EPSG code from an input.

Parameters **data** (*gdal.Dataset or ogr.DataSource or gpd.GeoDataFrame*) –

Returns

Return type int

`rastertodataframe.util.`**`get_pixels`**(*ras*, *mask*, *mask_val=None*)
Get pixels from a raster (with optional mask).

Parameters

- **ras** (*np.ndarray*) – Array of raster data in the form [bands][y][x].
- **mask** (*np.ndarray*) – Array (2D) of zeroes to mask data.
- **mask_val** (*int*) – Value of the data pixels in the mask. Default: non-zero.

Returns Array of non-masked data.

Return type np.ndarray

`rastertodataframe.util.`**`get_raster_band_names`**(*raster*)
Obtain the names of bands from a raster. The raster metadata is queried first, if no names a present, a 1-index list of band_N is returned.

Parameters **`raster`** (*`gdal.Dataset`*) –

Returns

Return type list[str]

`rastertodataframe.util.`**`open_raster`**(*path*, *read_only=True*)

Open a raster using GDAL.

Parameters

- **`path`** (*`str`*) – Path of file to open.
- **`read_only`** (*`bool`*) – File mode, set to False to open in "update" mode.

Returns

Return type GDAL dataset

`rastertodataframe.util.`**`open_vector`**(*path*, *with_geopandas=False*, *read_only=True*)

Open a vector dataset using OGR or GeoPandas.

Parameters

- **`path`** (*`str`*) – Path to vector file.
- **`with_geopandas`** (*`bool`*) – Set to True to open with geopandas, else use OGR.
- **`read_only`** (*`bool`*) – If opening with OGR, set to False to open in "update" mode.

Returns

Return type GeoDataFrame if `with_geopandas` else OGR datsource.

`rastertodataframe.util.`**`same_epsg`**(*data1*, *data2*)

Check sets of data have the same EPSG.

Parameters

- **`data1`** (*`gdal.DataSet or ogr.DataSource or gpd.GeoDataFrame`*) –
- **`data2`** (*`gdal.DataSet or ogr.DataSource or gpd.GeoDataFrame`*) –

Returns

Return type bool

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/mblack20/rastertodataframe/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mblack20/rastertodataframe/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 :)

5.2 Get Started!

Ready to contribute? Here's how to set up *rastertodataframe* for local development.

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

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

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

   ```
   $ mkvirtualenv rastertodataframe
   $ cd rastertodataframe/
   $ 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 rastertodataframe 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.

5.3 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 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/mblack20/rastertodataframe/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_rastertodataframe
```

5.5 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 6

Credits

6.1 Development Lead

- Martin Black <mblack@posteo.de>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.1 (2019-02-13)

- Add support for single band rasters.

7.2 0.2.0 (2018-07-12)

- Implement tiling to reduce memory use for large rasters.

7.3 0.1.3 (2018-07-09)

- Remove dependencies to fix non-building installs.

7.4 0.1.2 (2018-07-09)

- Fix creation of temporary files on windows.

7.5 0.1.1 (2018-07-08)

- All logic implement with unit tests. Prepare for PyPI release.

7.6 0.1.0 (2018-07-07)

- Project started.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index