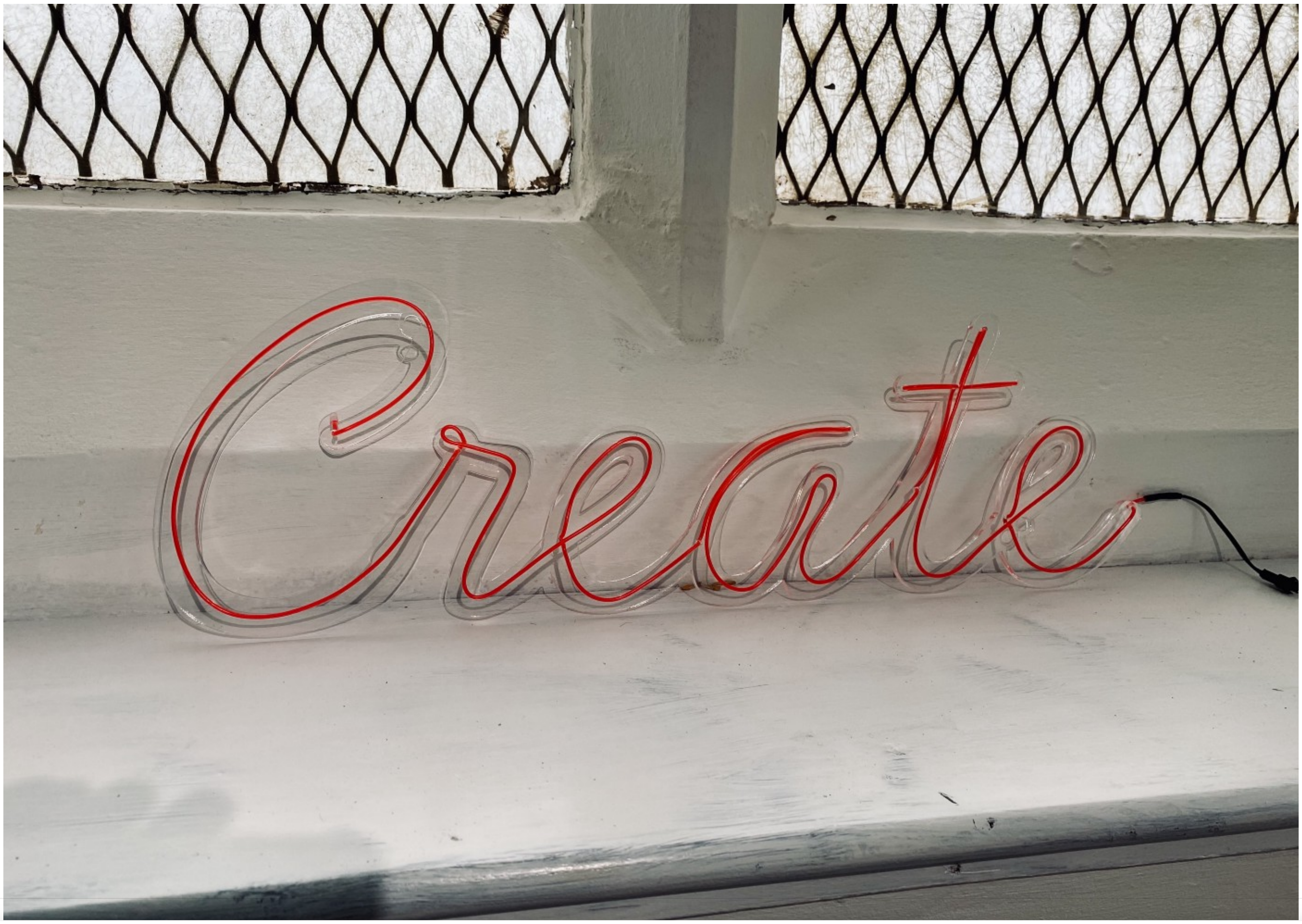
Published in Towards Data Science

Luke Gloege, Ph.D. Follow
Jul 4, 2021 · 6 min read ★ · Listen

Save

How to Create Xarray Datasets

Defining a dataset from scratch



`Xarray` is an open-source Python package for working with labeled multi-dimensional datasets. It is indispensable when working with NetCDF formatted data, which is common in the Earth science community.

Opening a NetCDF file with `xarray` is straightforward.

```
ds = xr.open_dataset("/path/to/file.nc")
```

However, creating a `xarray` object that you can save to a NetCDF file requires a little more work. In this post, I will go over how to do just that. I will first go over how to convert a `pandas` DataFrame to a `xarray` Dataset and then go over how to create a dataset from scratch.

Table of contents

1. Pandas DataFrame to Xarray Dataset

Starting with a DataFrame, you can directly convert it to a Dataset.

`to_xarray()` to make it into a `xarray` object.

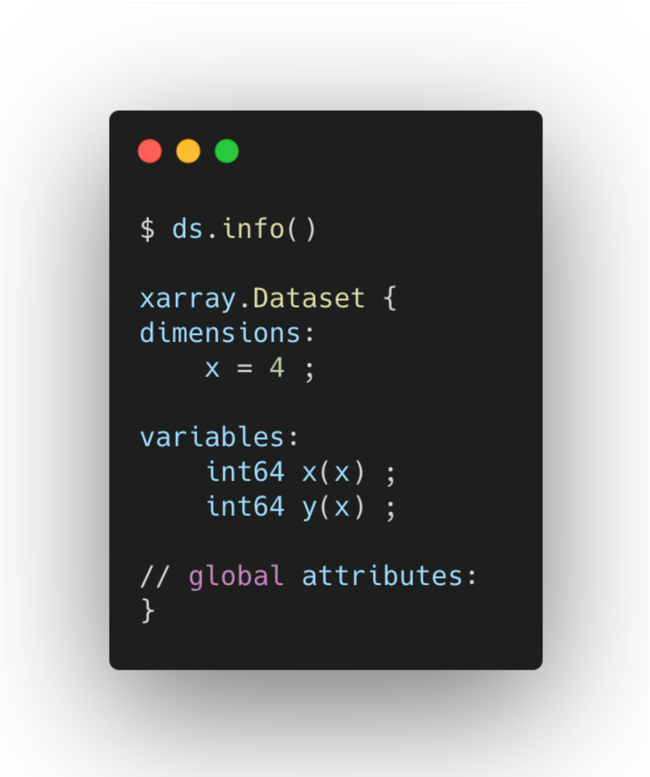
```python
import pandas as pd
import xarray as xr

# create dataFrame
df = pd.DataFrame({'x':[1,2,3,4],
                   'y':[9.8,19.6,29.4,39.2]}).set_index('x')

# convert to dataset
ds = df.to_xarray()
```

df_to_ds.py hosted with ❤ by **GitHub** **view raw**

This dataset isn't formatted very well yet. Here is the `ds.info()` output:



```
$ ds.info()

xarray.Dataset {
dimensions:
    x = 4 ;

variables:
    int64 x(x) ;
    int64 y(x) ;

// global attributes:
}
```

image by author

Ideally, `info()` should tell you some metadata about the variables and the dataset as a the data.

In the next subsection, we will add some metadata to this dataset.

For climate and forecasting (cf), there is a standard convention for metadata. A discussion of this is beyond the scope of this post, but here are two resources for those that are interested:

- cfconventions.org
- overview of cf conventions

Please note that this post is meant to illustrate a concept and thus I may deviate from conventional standards.

1.1 How do I add metadata and change variable names?

With the object created we can begin to add metadata and change variable/coordinate names.

Dictionary `key:value` pairs are used to rename variables/coordinates and for adding attributes. Here is the basic syntax:

- **rename variable**: `ds.rename({“old_name” : “new_name”})`
- **var attribute:** `ds[“var”].attrs = {“attribute_name”:”attribute_value”}`
- **global attribute:** `ds.attrs = {“attribute_name”:”attribute_value”}`

The code snippet below first renames the coordinates and variables and then creates variable and global attributes.

(**Note: xarray does not have an `inplace=True` option like pandas.**)

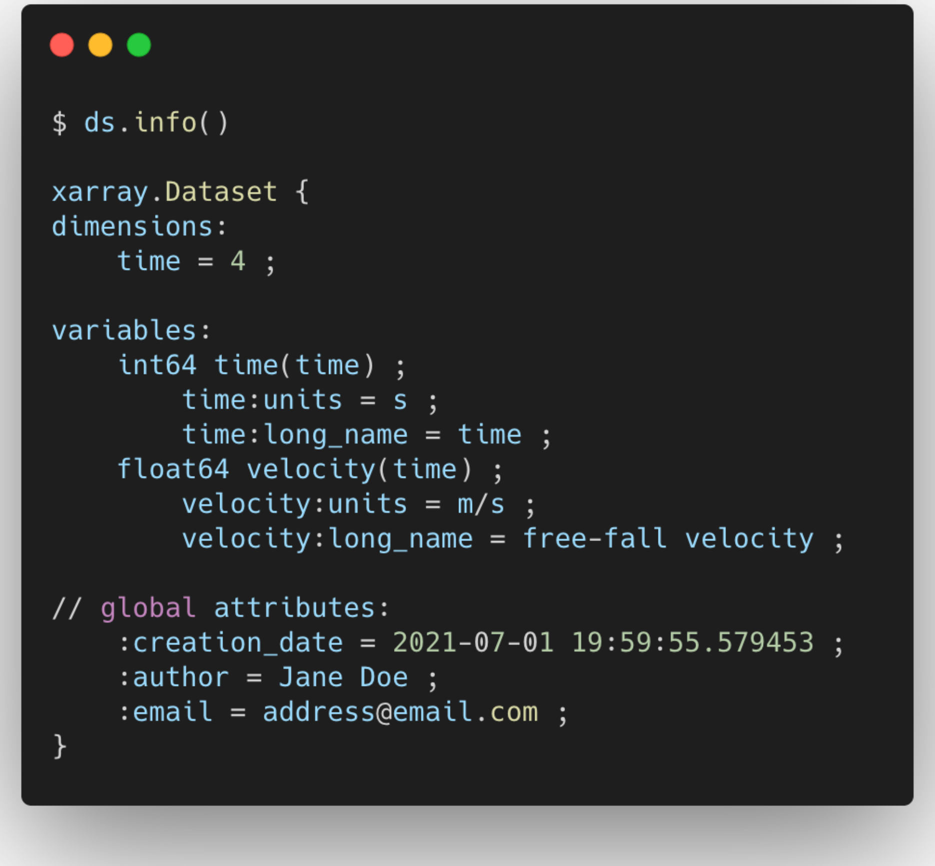
```
# change variable names
ds = ds.rename({'x':'time', 'y':'velocity'})

# variable attributes
ds['y'].attrs = {'units':'m/s', 'long_name':'free-fall velocity'}
ds['x'].attrs = {'units':'s', 'long_name':'time'}

# global attributes
ds.attrs = {'creation_date':datetime.now(), 'author':'Jane Doe', 'email':'address@email
```

change_dataset.py hosted with ❤ by **GitHub** **view raw**

Here is the `ds.info()` output after adding some metadata



```
$ ds.info()

xarray.Dataset {
dimensions:
	time = 4 ;

variables:
	int64 time(time) ;
		time:units = s ;
		time:long_name = time ;
	float64 velocity(time) ;
		velocity:units = m/s ;
		velocity:long_name = free-fall velocity ;

// global attributes:
	:creation_date = 2021-07-01 19:59:55.579453 ;
	:author = Jane Doe ;
	:email = address@email.com ;
}
```

image by author

1.2 Create a multi-dimensional dataset

For multiple dimensions, simply include a list of dimensions in `set_index()` . Then you can add metadata and change variables the same way as before

1.3 How do I save my data?

Once you are satisfied with your changes, you can save them to a NetCDF file:

```
ds.to_netcdf('/save/to/this/path/file.nc')
```

In this next section, I will go over creating a dataset from scratch. This is useful if you have a large `NumPy` array or list you want to save as a NetCDF.

2. Create Xarray dataset from scratch

The following syntax is used to create a dataset with `xarray` :

```
ds = xr.Dataset(data_vars, coords, attrs)
```

A complete dataset consists of three dictionaries:

- `data_vars` : The key is the variable name and value is a tuple consisting of `(dimensions, data, variable_attributes)`
  `- dimensions` → list of names
  `- data` → the data can be in any format (`Pandas` , `Numpy` , list, etc.)
  `- variable_attributes` → **optional** dictionary of attributes
- `coords` : this defines the coordinates and their attributes
- `attrs` : **optional** dictionary of global attributes

```python
from datetime import datetime

# define data with variable attributes
data_vars = {'velocity':(['time'], [9.8,19.6,29.4,39.2],
                         {'units': 'm/s',
                          'long_name':'free-fall velocity'})}

# define coordinates
coords = {'time': (['time'], [1,2,3,4])}

# define global attributes
attrs = {'creation_date':datetime.now(),
         'author':'Jane Doe',
         'email':'address@email.com'}

# create dataset
ds = xr.Dataset(data_vars=data_vars,
                coords=coords,
                attrs=attrs)
```

create_dataset.py hosted with ❤️ by **GitHub** **view raw**

data_vars: defines the variable `velocity` with one dimension, `time`, and includes two attributes, `units` an `long_name`.

coords: defines the `time` coordinate, whose dimension is also named time.

attrs: defines the global attributes, `creation_data`, `author`, and `email`

2.2 Dataset with multiple coordinates

If your data is multi-dimensional it will typically have multiple coordinates.

Here, I create a dataset where the variable has two dimensions, `x` and `y`, and two coordiantes, `lat` and `lon`.

In the previous example, the coordinate and dimension were the same name. This example

```
# create data
data = np.random.randn(2, 3)

# create coords
rows = [1,2]
cols = [1,2,3]

# put data into a dataset
ds = xr.Dataset(
    data_vars=dict(
        variable=(["x", "y"], data)
    ),
    coords=dict(
        lon=(["x"], rows),
        lat=(["y"], cols),
    ),
    attrs=dict(description="coords with vectors"),
)
```

create_multi_ds.py hosted with ❤ by **GitHub** **view raw**

2.3 Dataset with coordinates matrices

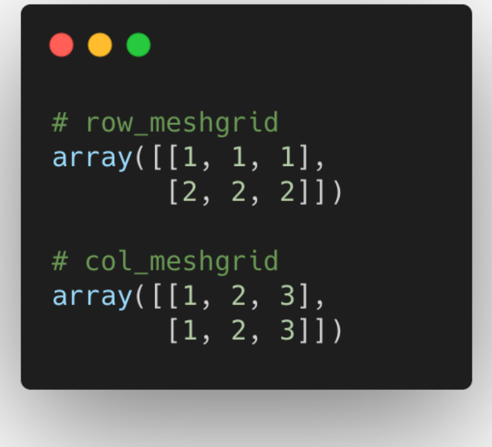
In the previous example, each coordinate was represented by a vector. However, there are instances where it makes sense to represent the coordinate as a matrix. This is common with Earth system models on a non-regular grid.

Let's take the previous example and turn each coordinate into a matrix using `np.meshgrid()`

```
# row and column vectors
rows = [1,2]
cols = [1,2,3]

# create meshgrids
row_meshgrid, col_meshgrid = np.meshgrid(rows, cols, indexing='ij')
```

create_meshgrid.py hosted with ❤ by **GitHub** **view raw**



```
# row_meshgrid
array([[1, 1, 1],
       [2, 2, 2]])

# col_meshgrid
array([[1, 2, 3],
       [1, 2, 3]])
```

image by author

Now let's create a dataset using the coordinates represented by matrices.

```
# create data
data = np.random.randn(2, 3)

# create coords
rows = [1,2]
cols = [1,2,3]
row_meshgrid, col_meshgrid = np.meshgrid(rows, cols, indexing='ij')

# put data into a dataset
ds = xr.Dataset(
    data_vars=dict(
        variable=(["x","y"], data)
    ),
    coords=dict(
        row=(["x","y"], row_meshgrid),
        col=(["x","y"], col_meshgrid),
    ),
    attrs=dict(description="coords with matrices"),
)
```

cases representing the coordinate as a vector makes more sense.

2.4 Dataset with coordinate vectors and matrices

Coordinates can be a mix of matrices and vectors. You just need to make sure that all the dimensions in your variables are accounted for in your coordinates.

Let's create a dataset with three dimensions, `time`, `lat`, and `lon`. The spatial dimensions (`lat`, `lon`) will be each be represented by matrices and `time` will be a vector.

```
# create data
data = np.random.randn(2, 3, 3)

# create coordinates
rows = [1,2]
cols = [1,2,3]
row_meshgrid, col_meshgrid = np.meshgrid(rows, cols, indexing='ij')
time = pd.date_range("2000-01-01", periods=3)

# put data into a dataset
ds = xr.Dataset(
    data_vars=dict(
        variable=(["x","y","time"], data, {"units":"m/s"})
    ),
    coords=dict(
        row=(["x","y"], row_meshgrid),
        col=(["x","y"], col_meshgrid),
        time=(["time"], time)
    ),
    attrs=dict(description="coords with matrices"),
)
```

create_3d_ds.py hosted with ❤️ by **GitHub** **view ra**

3. Final thoughts

dataset from scratch.

I covered two ways to represent coordinates: vectors or matrices. My advice is to keep the coordinates as a vector whenever possible. Using matrices to store coordinate information is common if the data is not on a regular grid.

If you get an error creating a dataset, the first thing to check is the dimensions. It is easy to put the dimensions out of order.

My other piece of advice is that even though attributes are optional when creating a dataset, it is good practice to always include them so you and other people can easily interpret the dataset. I prefer to add attributes after creating the dataset object. This is a personal preference that I think improves readability.

Thanks for reading and I am happy to help troubleshoot any issues

Join Medium with my referral link — Luke Gloege, Ph.D.

As a Medium member, a portion of you goes to writers you read, and you get full access to eve

lukegloege.medium.com

104

Open in app

Get started

About Help Terms Privacy

Get the Medium app