01_xarray_intro

August 13, 2021

1 Xarray

In this lesson we are going to learn about the `Xarray` library - one of the most useful libraries available to us as earth/ocean/atmospheric researchers.

1.0.1 credit

This lesson is from the Geohackweek tutorial on Xarray (https://geohackweek.github.io/nDarrays/), and Abernathy's book: (https://earth-env-data-science.github.io/lectures/xarray/xarray_intro.html).

1.0.2 Think of Xarray as a bit like numpy, except that it has a better 'understanding' of the dimensions of your data.

1.1 Multidimensional Arrays

(not as scary as they sound)

1.1.1 Like a table in Excel but with more than two dimensions

Let's start by drawing some 2D arrays (familiar spreadsheets) on the board. List a bunch of types of data that might fit in that format.

Now extend this idea: what are common examples of 3D arrays? What type of ocean/earth/atmos data might fit this format? Can we see how they are easily related to 2D arrays?

[1]:
```
# list some types of N-d arrays that you migh work with

# A movie is a time series of 2D images, so is a 3D dataset

# A series of satellite images is a bit like a movie too, so is also 3D

# Output from numerical simulation might be 3D or 4D even
```

2 What is Xarray?

Xarray is a python library that brings the convienient features of Pandas to higher dimensions. It was initally developed by people at the Climate Corporation for dealing with climate data. It turns out it's very useful in much of our work.

Like Pandas, Xarray let's us do operations on named dimensions. It keeps a lot of the 'metadata' associted with the actually data that makes plotting, calculating, and grouping much easier.

It also is a great way to load data formats that we will see a lot in our work, in particular NetCDFs (more on this later).

features:

the `DataArray` is Xarray's standard data type: a labeled, multi-dimensional array

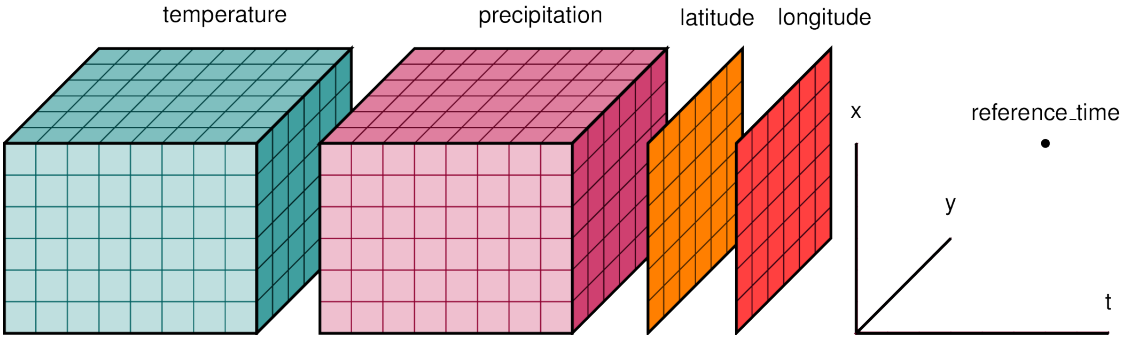
the `DataArray` has these key properties:

- `data`: N-dimensional array (NumPy or dask) holding the array's values, i.e. your actual data,
- `dims`: dimension names for each axis, just the names, like 'latitude' or 'longitude' or 'time'
- `coords`: dictionary-like container of arrays that label each point, i.e. the actual values of each axis like time or latitue or something
- `attrs`: ordered dictionary holding metadata, or 'attributes', like the data units, person who collected, any of that stuff

The second main data structure is the `Dataset` - which you can think of as a group of `DataArray`, like perhaps a grid that has a temperature `DataArray`, a salinity `DataArray`, and an oxygen `DataArray` that are all from the same instument or model and belong together. These can be housed within one `Dataset`. We will show this later

3 When to use Xarray?

- if your data are multidimensional, like lat, lon, x, y, z, time ...
- if your data are on a regular grid (output from a model)
- if your data are contained in a `.nc` netcdf file.



4 What can Xarray do for *me*?

Think about how you would want to build a data structure. Let's go through on the board all the things that should go along with your data. For concretness let's think about the temperature at all depths and points in the mid-Atlantic Bight, from a model say. What do we need to know? We can name all those vectors, but wouldn't it be easier if they all lived with the data itself??

5 Build a simple DataArray

That is maybe a lot of complex info. Let's start from the basic basics and build up our own `DataArray` using Xarray to see how all it's parts work

We will start with some fake data, just a sequence of numbers. Make up something about what they are.

We need to download xarray using `conda` like we have before:

5.0.1 installing xarray with (Ana)conda (if you don't have it installed already or there was a problem)

to get `xarray` (you should only need to do this once): 1. go to the little `+` on the left to open a `launcher` window. 1. click the 'terminal' tile to open a terminal 1. type `conda env list` and hit enter. Make sure that there is a `*` next to the line that says swbc 1. if that's right type `conda install xarray dask netCDF4 bottleneck pandas` 1. when it asks `proceed ([y]/n)?` type `y` and hit enter

Of course, to start we need to import xarray. To save typing we usualy ask python to call xarray 'xr'

[2]:
```
# import xarray and matplotlib
import xarray as xr
import matplotlib.pyplot as plt
%matplotlib inline
```

Now let's use the `xr.DataArray()` function to make our fake dataarray

[3]:
```
da = xr.DataArray([ 1, 2, 5, 7, 10])

da
```

[3]:
```
<xarray.DataArray (dim_0: 5)>
array([ 1,  2,  5,  7, 10])
Dimensions without coordinates: dim_0
```

Ok, that is just some data, a few numbers. So far this isn't anything more special than a numpy array. But it'll get better if we can give it some dimensions and coordinates that give the data context. Maybe those are the number of birds we counted at 10 minute intervals while we were staring out the window in class.

Let's add the dimention `time` to the DataArray:

[4]:
```
da = xr.DataArray( [ 1, 2, 5, 7, 10], dims=['time'])

da
```

[4]:
```
<xarray.DataArray (time: 5)>
array([ 1,  2,  5,  7, 10])
Dimensions without coordinates: time
```

Ok, now we know that the data correspond to times, that's good. It's giving us a more complete understanding of the context of the data. But what times?

We've named the dimension, but now let's give it some coordinates, in otherwords let's specify the exact times when we measured those birds.

In order to connect the dimension to an actual set of time points (i.e. coordinates) we need to recall a Python data type we talked about a while ago.

5.1 Quick Aside: Dictionaries - remember that python data type?

Quick summary: A Dictionary is a lot like a list, but it has a label for each element in the list. Namely a Dictionary is made up of key : values pairs. You can think of it literally like a dictonary: you'd look something by it's name (key) and there would be other information in there (the values). We define a ditionary uscing curly brackets: `dict = {}` and we separate keys from their values with a colon :

```
d = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
```

For example we can make a dictionary where the key is a city, and the value is that city's baseball team:

```
MLB_team = {
     'Colorado' : 'Rockies',
     'Boston'   : 'Red Sox',
     'Minnesota': 'Twins',
     'Milwaukee': 'Brewers',
     'Seattle'  : 'Mariners'
 }
```

Dictionaries aren't too complicated, they are just another way of storing data. We will use them a lot with xarrays.

One use case we can talk about now, we have decided the dimension of our DataArray is time, and we can use a dictionary to associate that key (time) with values (t1, t2, t3, etc).

We've named the dimension, but now let's give it some coordinates, in otherwords let's specify the exact times when we measured those birds. Just to be simple we will say starting at 0 minutes we looked out every 10 minutes to count the birds.

When we specify the coordinates, we need to tell xarray that we are refering to the `time` dimention, and then tell it what times we want to include:

[5]:
```
da = xr.DataArray( [ 1, 2, 5, 7, 3],
                   dims=['time'],
```

```
                    coords = {'time': [0, 10, 20, 30, 40]})

da
```

[5]:
```
<xarray.DataArray (time: 5)>
array([1, 2, 5, 7, 3])
Coordinates:
  * time     (time) int64 0 10 20 30 40
```

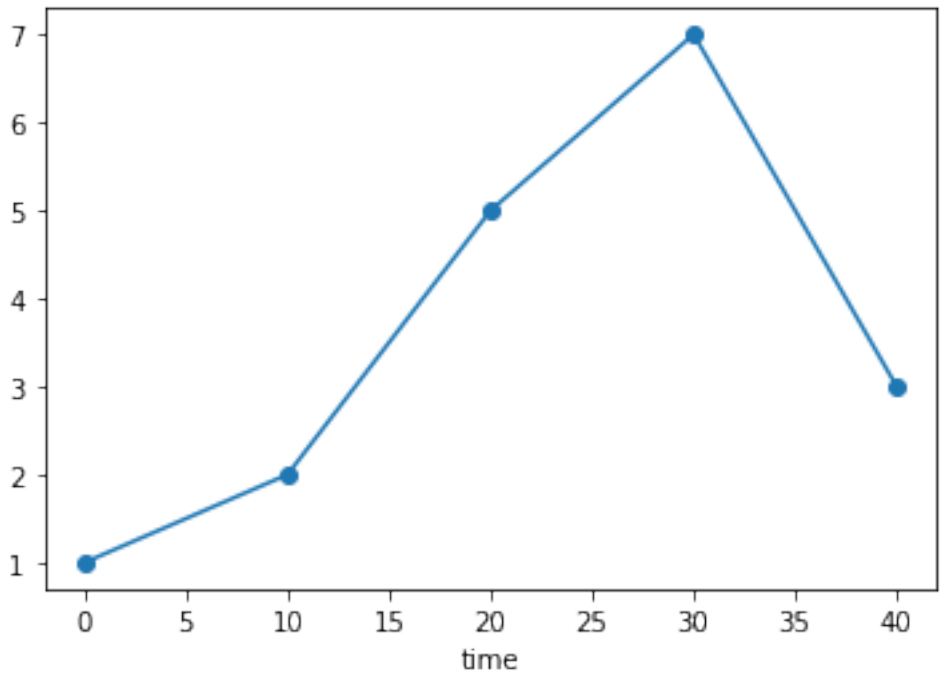
Now we have something that pretty much fully describes our experiment (almost, what might we be missing? what about attributue? can you think of some examaples?)

From here Xarray has all sorts of built in functions that allow you to interact with your data quickly. Because the DataArray has data, dimentions, and coordinates, Xarray can internally understand a lot about the data.

One thing we can do is just ask xarray to make a simple plot. This takes almost no code:

[6]:
```
# da.plot()
da.plot(marker='o')
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7fb2b0d55790>]
```



The end…

6 Breakout / exercise 01

7 Build a mulidimentional DataArray

We made up some data in the simple example. Also, did you notice it's just one dimentional? Let's go through the excercise by building some a multidimentional dataset.

This is in the next notebook