02_xarray_multi_dim

August 13, 2021

1 Build a multidimentional DataArray and Dataset

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

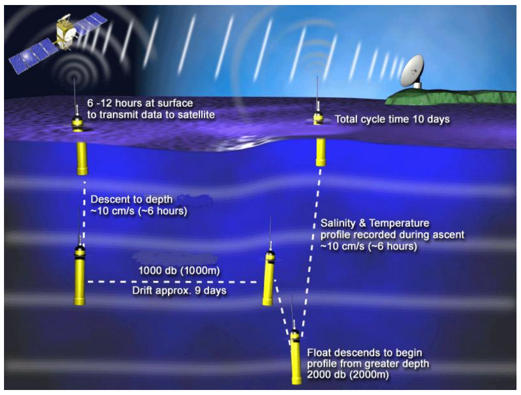
We are going to start with some data that is just a bunch of normal numpy arrays, so we need to load numpy as well as xarray

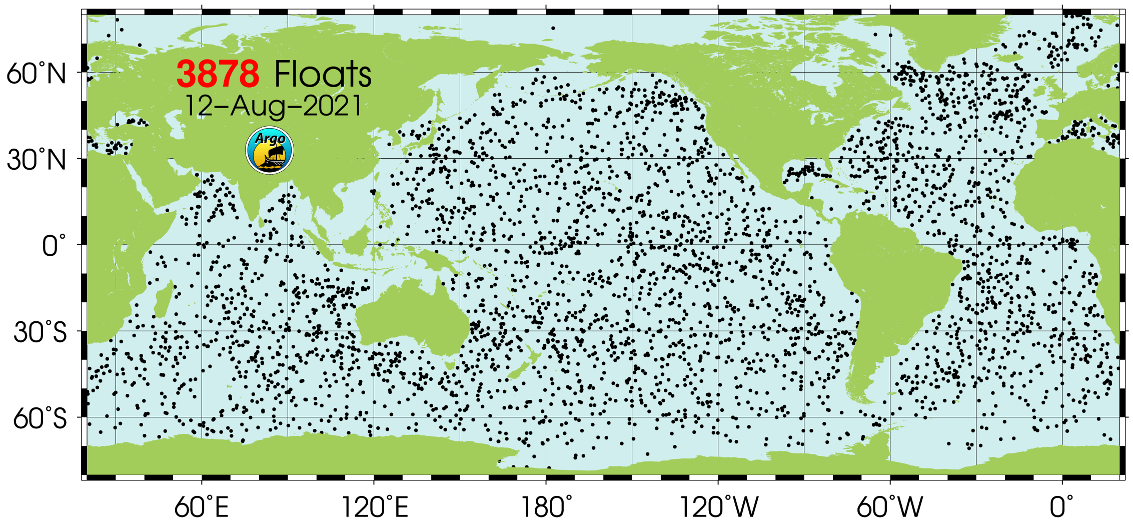
1.0.1 credit

This lesson is from Abernathy's book: (https://earth-env-data-science.github.io/lectures/xarray/xarray_intro.html).

[1]:
```
# import xarray, numpy and matplotlib
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

2 Example: ARGO float





Let's start by loading some real data. This is ARGO float data that contains temperature and salinity data {What is an Argo float? how does it take data?}. Those data are in the form of numpy arrays, or matricies. So, again, rows and columns. Let's draw on the board what the rows and columns are. They have coordinates like time, depth, latitude, longitude. Stuff you would expect to describe data collected in the ocean.

Right now, when we load the data, it's going to be a collection of numpy arrays. They are all seperate objects, and what we'd like to do is stitch them together in a sensible way. To this we are going to create a DataArray, then a Dataset.

[2]:
```python
argo_data = np.load('../data/argo_float_4901412.npz')
list(argo_data.keys())
```

[2]:
```
['S', 'T', 'levels', 'lon', 'date', 'P', 'lat']
```

They are in this container because of how they are saved. Let's break each component out into it's own numpy array

[3]:
```python
S = argo_data['S']
T = argo_data['T']
P = argo_data['P']
levels = argo_data['levels']
lon = argo_data['lon']
lat = argo_data['lat']
date = argo_data['date']


for key in argo_data:
    print( key, ' shape is: ', argo_data[key].shape)
```

```
S  shape is:  (78, 75)
T  shape is:  (78, 75)
levels  shape is:  (78,)
lon  shape is:  (75,)
```

```
date  shape is:  (75,)
P  shape is:  (78, 75)
lat  shape is:  (75,)
```

Remember from the previous notebook.

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

Let's take the salinity `S` and create a DataArray for it

[4]:
```
da_salinity = xr.DataArray(S, dims=['level', 'date'], coords={'level': levels,␣
↪'date': date})

da_salinity
```

[4]:
```
<xarray.DataArray (level: 78, date: 75)>
array([[35.6389389 , 35.51495743, 35.57297134, ..., 35.82093811,
        35.77793884, 35.66891098],
       [35.63393784, 35.5219574 , 35.57397079, ..., 35.81093216,
        35.58389664, 35.66791153],
       [35.6819458 , 35.52595901, 35.57297134, ..., 35.79592896,
        35.66290665, 35.66591263],
       ...,
       [34.91585922, 34.92390442, 34.92390442, ..., 34.93481064,
        34.94081116, 34.94680786],
       [34.91585922, 34.92390442, 34.92190552, ..., 34.93280792,
        34.93680954, 34.94380951],
       [34.91785812, 34.92390442, 34.92390442, ...,         nan,
        34.93680954,         nan]])
Coordinates:
  * level    (level) int64 0 1 2 3 4 5 6 7 8 9 ... 68 69 70 71 72 73 74 75 76 77
  * date     (date) datetime64[ns] 2012-07-13T22:33:06.019200 ... 2014-07-24T...
```

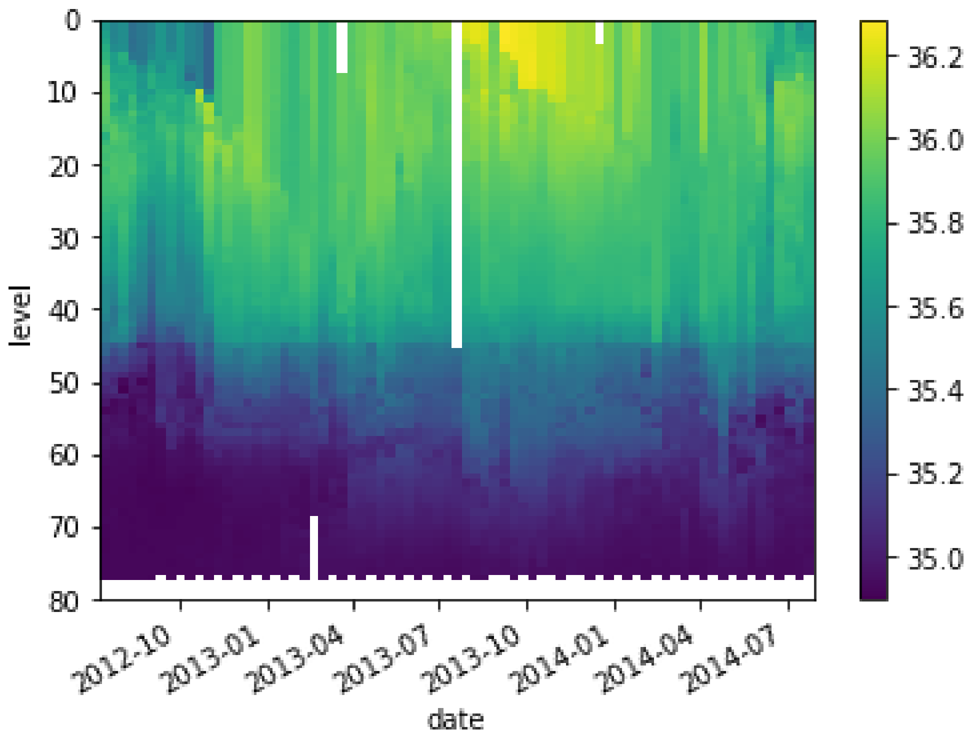
Ok, this is like the 1D fake bird data we made before, but now it's real 2D salinity data from the ocean.

Let's see what xarray does if we ask it to make a simple plot:

[5]:
```
da_salinity.plot()

# lets switch the axis order so depth goes down
plt.ylim([80, 0])
```

```
[5]: (80.0, 0.0)
```



Nice! That is sort of amazing. Xarray knew that the salinity data is 2d - so by default it smartly made a pcolor plot (not a line plot or something). It also knew that time is on the x axis, and the 'levels' (depth) are on the y axis because the dimensions match. It also labeled out axis and formatted the dates.

But we aren't done with out DataArray yet. Remember the four parts of a DataArray? `data`, `dims`, `coords`, `attrs`. We can add other important information into the `attrs` part of the DataArray. Can you think of some important info?

```
[6]: # add some attributes describing the data and units

     da_salinity.attrs['units'] = 'PSU'
     da_salinity.attrs['standard_name'] = 'sea_water_salinity'
     da_salinity
```

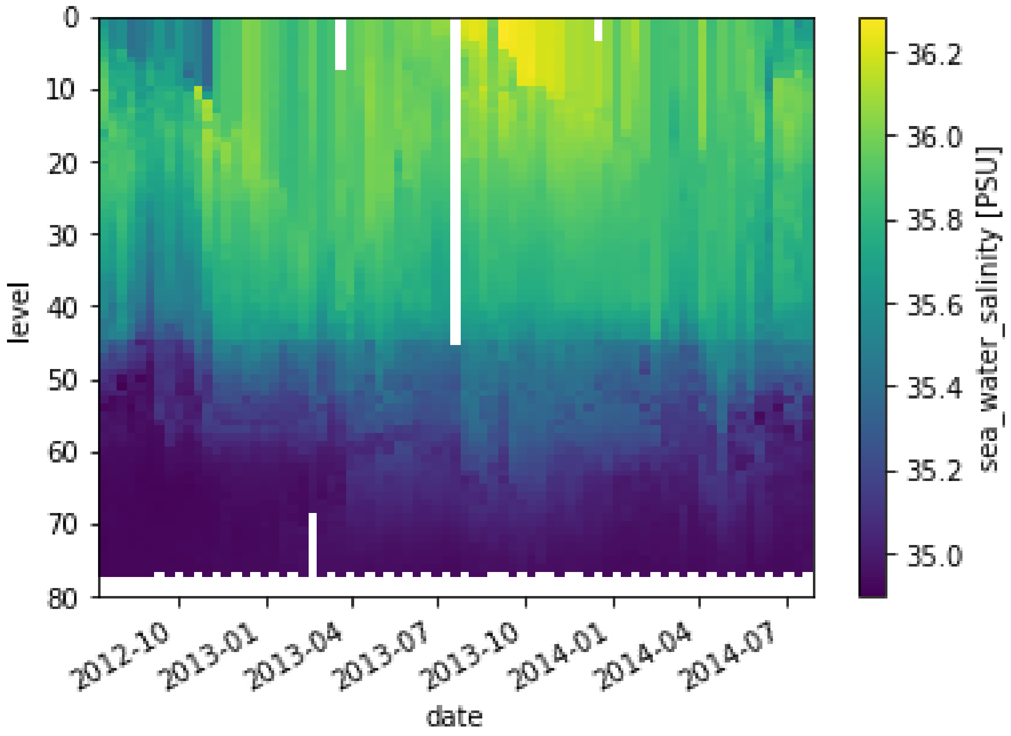
```
[6]: <xarray.DataArray (level: 78, date: 75)>
     array([[35.6389389 , 35.51495743, 35.57297134, …, 35.82093811,
             35.77793884, 35.66891098],
            [35.63393784, 35.5219574 , 35.57397079, …, 35.81093216,
             35.58389664, 35.66791153],
            [35.6819458 , 35.52595901, 35.57297134, …, 35.79592896,
             35.66290665, 35.66591263],
        …,
        [34.91585922, 34.92390442, 34.92390442, …, 34.93481064,
         34.94081116, 34.94680786],
        [34.91585922, 34.92390442, 34.92190552, …, 34.93280792,
         34.93680954, 34.94380951],
        [34.91785812, 34.92390442, 34.92390442, …,         nan,
         34.93680954,         nan]])
Coordinates:
  * level    (level) int64 0 1 2 3 4 5 6 7 8 9 … 68 69 70 71 72 73 74 75 76 77
  * date     (date) datetime64[ns] 2012-07-13T22:33:06.019200 … 2014-07-24T…
Attributes:
    units:          PSU
    standard_name:  sea_water_salinity
```

[7]:
```
# make the default plot again:

da_salinity.plot()

plt.ylim([80, 0])
```

[7]:
```
(80.0, 0.0)
```



3 Datasets

xarray datasets can hold multiple DataArrays. This makes particular sense if the data in those multiple DataArrays share dimensions and coordinates.

In our ARGO float example, both the Temperature and Salinity share the same dims and cords. So let's put them together into one dataset that holds all out float observational data.

The Dataset constructor takes three arguments:

- `data_vars` should be a dictionary with each key as the name of the variable and each can be an already constructed DataArray, or a tuple that looks like this `(dims, data[, attrs])`
- `coords` should be a dictionary of the same form as data_vars.
- `attrs` should be a dictionary.

So here is an example for our argo data:

[8]:
```
# A dictionary of data variables that specifies the dimensions and data as a␣
 ↪tuple
data_vars = {'salinity':    (('level', 'date'), S),
             'temperature': (('level', 'date'), T),
             'pressure':    (('level', 'date'), P)}

# A dictionary of coordinates
coords = coords={'level': levels, 'date': date}

argo = xr.Dataset(data_vars, coords)

# print
argo
```

[8]:
```
<xarray.Dataset>
Dimensions:      (level: 78, date: 75)
Coordinates:
  * level        (level) int64 0 1 2 3 4 5 6 7 8 … 69 70 71 72 73 74 75 76 77
  * date         (date) datetime64[ns] 2012-07-13T22:33:06.019200 … 2014-07…
Data variables:
    salinity     (level, date) float64 35.64 35.51 35.57 35.4 … nan 34.94 nan
    temperature  (level, date) float64 18.97 18.44 19.1 19.79 … nan 3.714 nan
    pressure     (level, date) float64 6.8 6.1 6.5 5.0 … 2e+03 nan 2e+03 nan
```

Let's talk through what all those parts are telling us when we print out `argo`.

What about the latitude and longitude? Those seem important and we'd like to use them for plotting and analysis later. They should be coordinates right? They should be the same size as one of the existing coordinates, either level or date. what do you think?

to add a new coordinate we can use:

```
[9]: argo.coords['lon'] = lon  # or argo_data['lon']

     argo
```

```
[9]: <xarray.Dataset>
     Dimensions:      (level: 78, date: 75, lon: 75)
     Coordinates:
       * level        (level) int64 0 1 2 3 4 5 6 7 8 … 69 70 71 72 73 74 75 76 77
       * date         (date) datetime64[ns] 2012-07-13T22:33:06.019200 … 2014-07…
       * lon          (lon) float64 -39.13 -37.28 -36.9 … -33.83 -34.11 -34.38
     Data variables:
         salinity     (level, date) float64 35.64 35.51 35.57 35.4 … nan 34.94 nan
         temperature  (level, date) float64 18.97 18.44 19.1 19.79 … nan 3.714 nan
         pressure     (level, date) float64 6.8 6.1 6.5 5.0 … 2e+03 nan 2e+03 nan
```

What we just did was add a whole new coordinate `lon`. But actually we know that each `lon` point is at a particular `date` location. So actually we can associate `lon` and `date`. To do that we set the dim of out new coord `lon` to be `date`. We can do the same for `lat`.

```
[10]: del argo['lon']
      argo.coords['lon'] = ('date', lon)
      argo.coords['lat'] = ('date', lat)
      argo
```

```
[10]: <xarray.Dataset>
      Dimensions:      (level: 78, date: 75)
      Coordinates:
        * level        (level) int64 0 1 2 3 4 5 6 7 8 … 69 70 71 72 73 74 75 76 77
        * date         (date) datetime64[ns] 2012-07-13T22:33:06.019200 … 2014-07…
          lon          (date) float64 -39.13 -37.28 -36.9 … -33.83 -34.11 -34.38
          lat          (date) float64 47.19 46.72 46.45 46.23 … 42.6 42.46 42.38
      Data variables:
          salinity     (level, date) float64 35.64 35.51 35.57 35.4 … nan 34.94 nan
          temperature  (level, date) float64 18.97 18.44 19.1 19.79 … nan 3.714 nan
          pressure     (level, date) float64 6.8 6.1 6.5 5.0 … 2e+03 nan 2e+03 nan
```

4 Working with labeled data

We've built our nice dataset for the ARGO float. It has Temperature, salinity and pressure data. Those data also have label dimensions / coordinates that include level, date, lat, and lon.

Now we are going to start to see some of the power of Xarray and how those labeled dimensions / coordinates let us make our analysis easier

4.1 Selecting data (indexing)

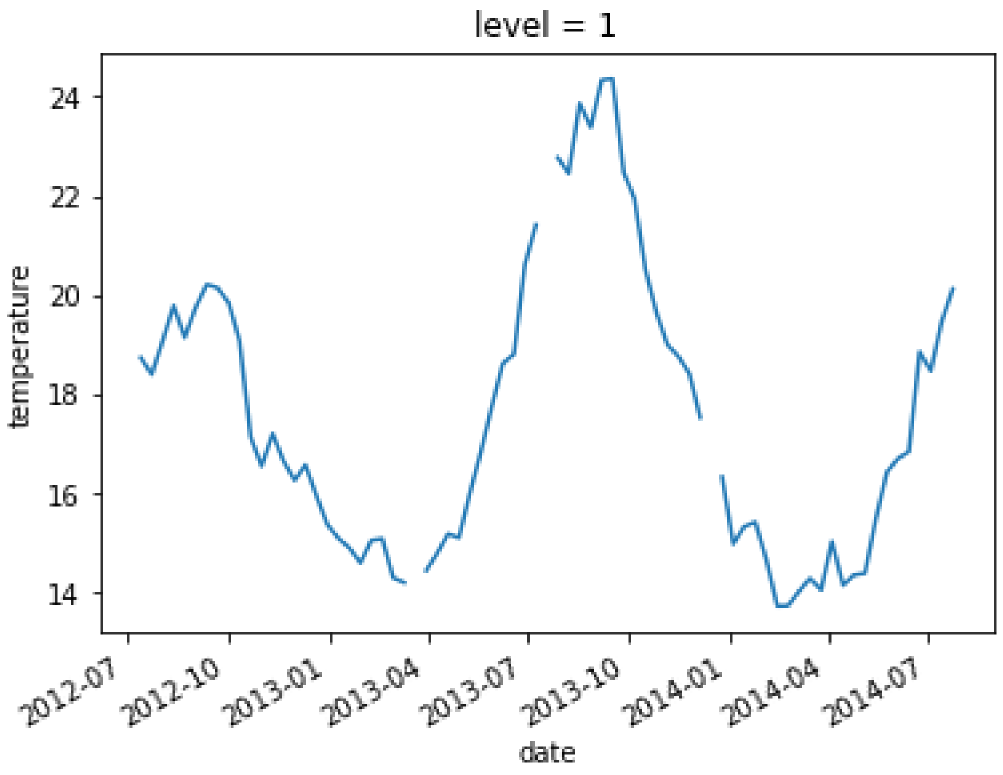
Let's say we want to look at some subset of the temperature data (just a slice). We can use standard numpy notation to do this by indicating the number of the row and column we are interested in.

Let's say we want to look at the second row and all columns (so this is like a timeseries at a particular level)

using standard numpy indexing this would look like:

```
[11]: # plot temp at level 1 over time:
      argo.temperature[1,:].plot()
      # argo.temperature.sel(levels = 5).plot()
```

```
[11]: [<matplotlib.lines.Line2D at 0x7fcc4c977610>]
```

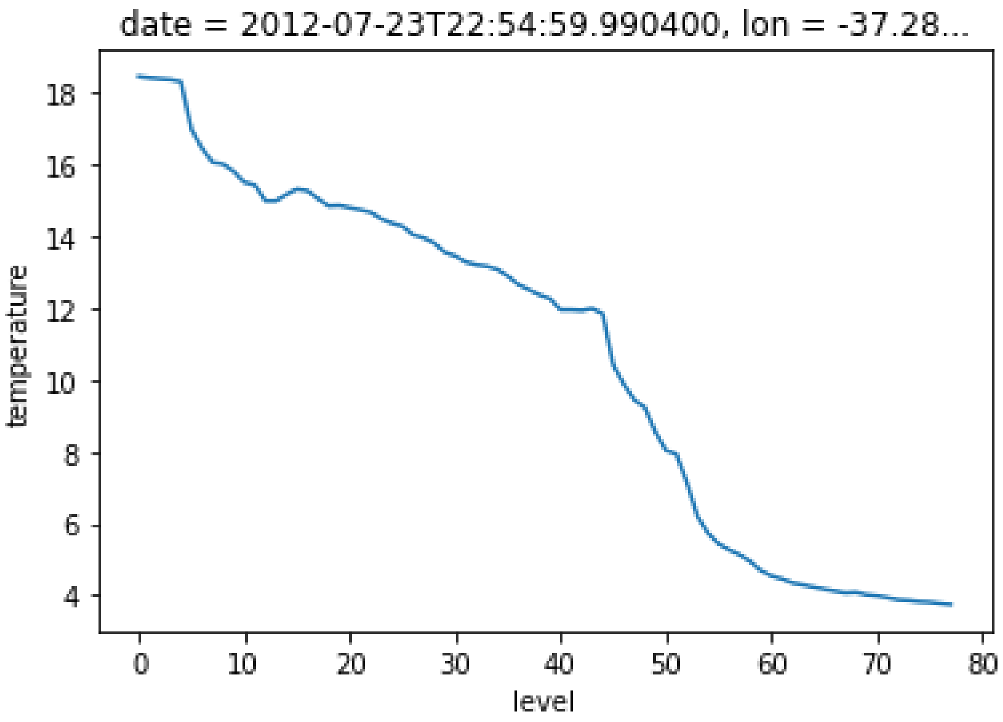


what about a particular depth profile?

in standard numpy indexing:

```
[12]: # plot temp vs level for profile 1
      argo.temperature[:,1].plot()
```

```
[12]: [<matplotlib.lines.Line2D at 0x7fcc49eb0760>]
```



That seems easy enough. But let's say you want to look at the temperature profile from a particular day. How are you going to do that? Well, you'd need to use the `date` dimension, look up the date you want, find it's index (meaning the number/position it comes in the list of dates) then put that index into the `argo.temperature[:,1].plot()` line.

This isn't impossible. This is the kind of thing you do all the time in matlab. It's annoying and takes a few lines of code.

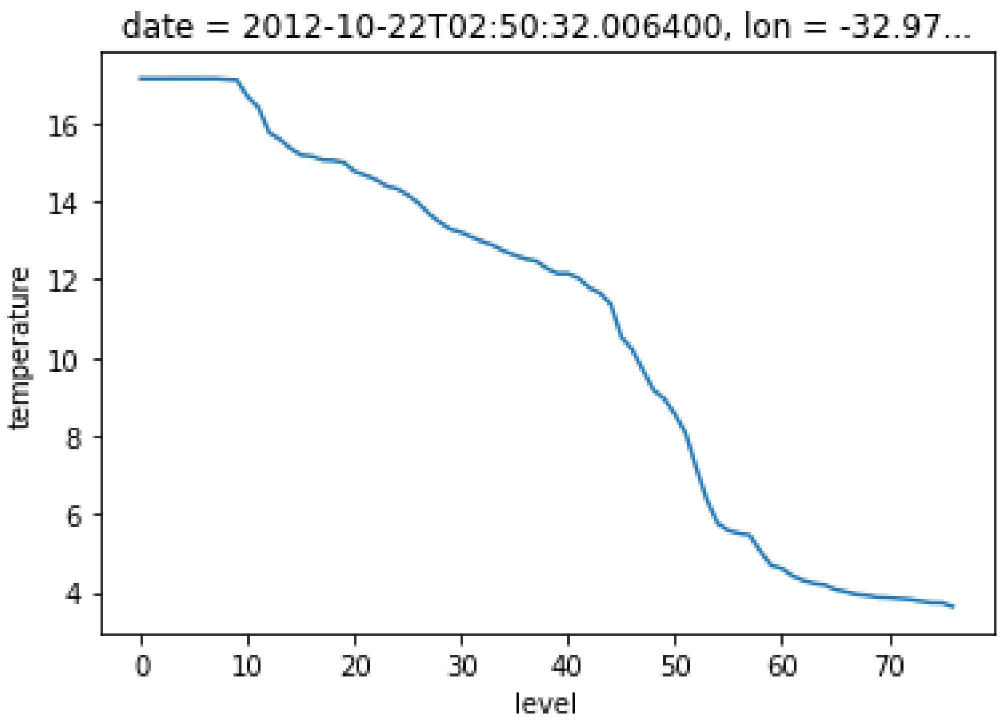
But xarray solves this problem! using the `.sel()` method you can 'select' a part of your data based on the label.

Here is how it works. let's get the profile on Oct 22 2012 by selecting based on the dimension `date`:

[13]:
```
# plot the temp profile on a particular day using .sel()

argo.temperature.sel(date='10-22-2012').plot()
# argo.temperature.sel(date='10-22-2012').plot(y='level')
```

[13]:
```
[<matplotlib.lines.Line2D at 0x7fcc49f01700>]
```



5 Slicing data

We can also grab a bunch of days. Grabbing a bunch of consecutive data is typically called 'slicing'. We have to tell xarray that we want a slice of the `date` dimension. Again, this is new syntax, so don't be worried that you don't know it. You'll learn as you go from examples and from reading the documents for different packages.

let's get a couple months around our previous profile:

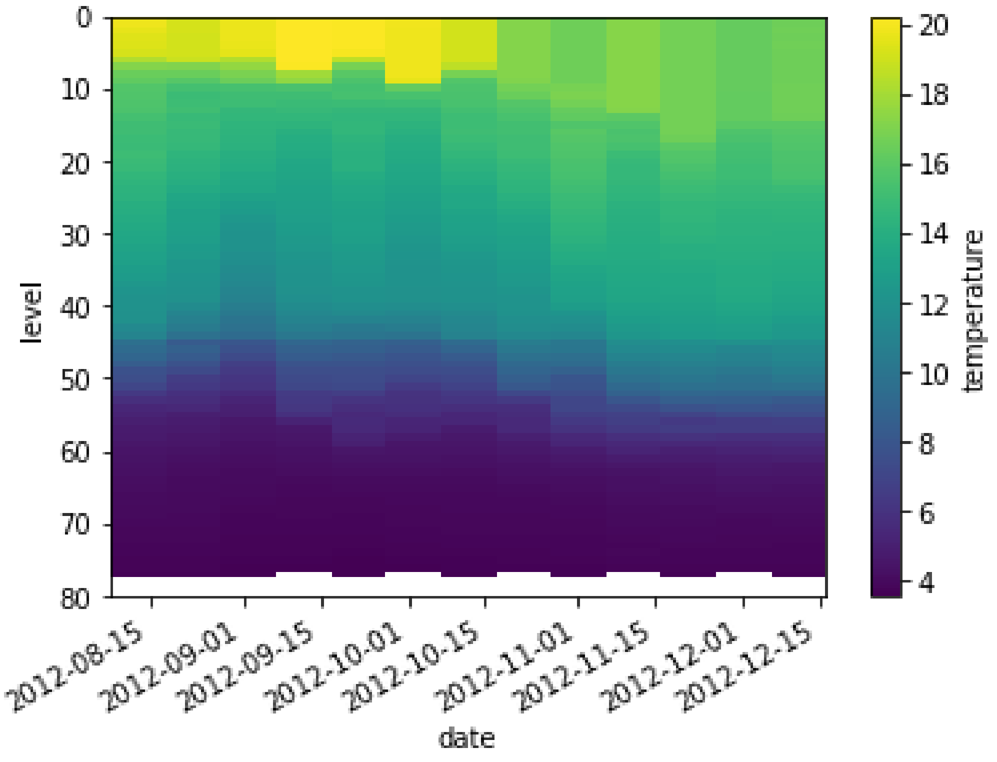
```
[14]: # select a time range using .sel() with the slice() argument

      argo.temperature.sel( date = slice('08-10-2012','12-10-2012') ).plot()

      plt.ylim([80,0])

      # plt.gcf().autofmt_xdate()
```

```
[14]: (80.0, 0.0)
```



You can also use `.sel()` on the whole dataset to, for example, grab all your data from one day:

[15]:
```
argo_one_day = argo.sel(date='10-22-2012')

argo_one_day
```

[15]:
```
<xarray.Dataset>
Dimensions:      (level: 78, date: 1)
Coordinates:
  * level        (level) int64 0 1 2 3 4 5 6 7 8 … 69 70 71 72 73 74 75 76 77
  * date         (date) datetime64[ns] 2012-10-22T02:50:32.006400
    lon          (date) float64 -32.97
    lat          (date) float64 44.13
Data variables:
    salinity     (level, date) float64 35.47 35.47 35.47 … 34.93 34.92 nan
    temperature  (level, date) float64 17.13 17.13 17.13 … 3.736 3.639 nan
    pressure     (level, date) float64 6.4 10.3 15.4 … 1.9e+03 1.951e+03 nan
```

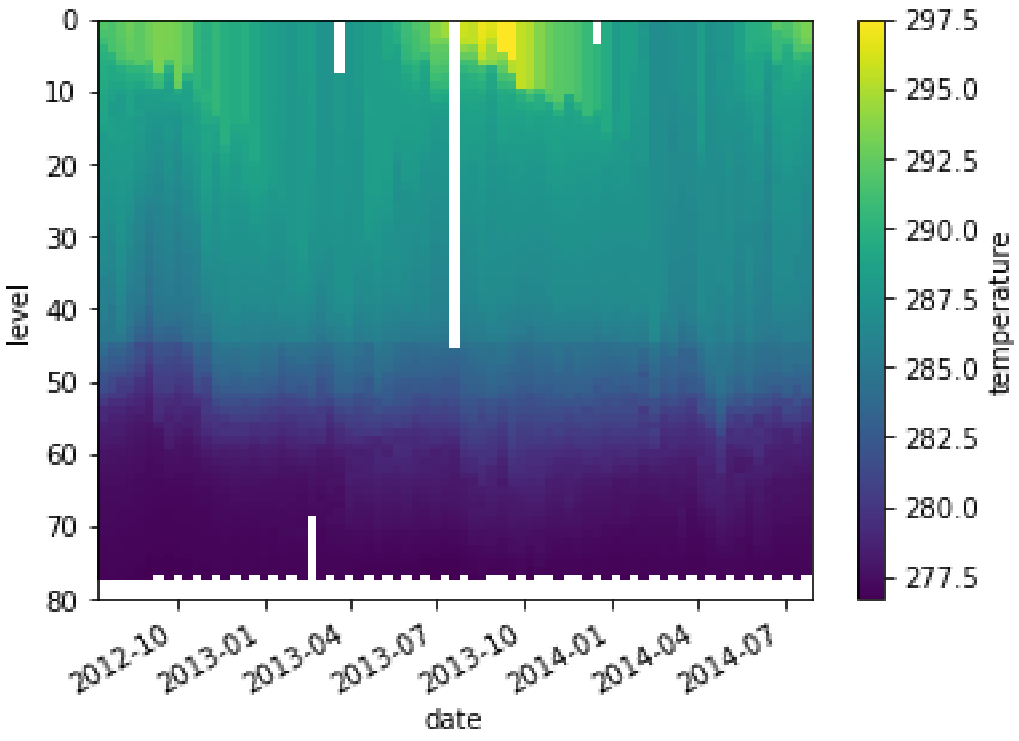
6 Math

we can do any normal math on these DataArrays and Datasets:

```
[16]: temp_kelvin = argo.temperature + 273.15

      temp_kelvin.plot()
      plt.ylim([80,0])
```

```
[16]: (80.0, 0.0)
```



you can combine DataArrays of the same size to get derived products like buoyancy:

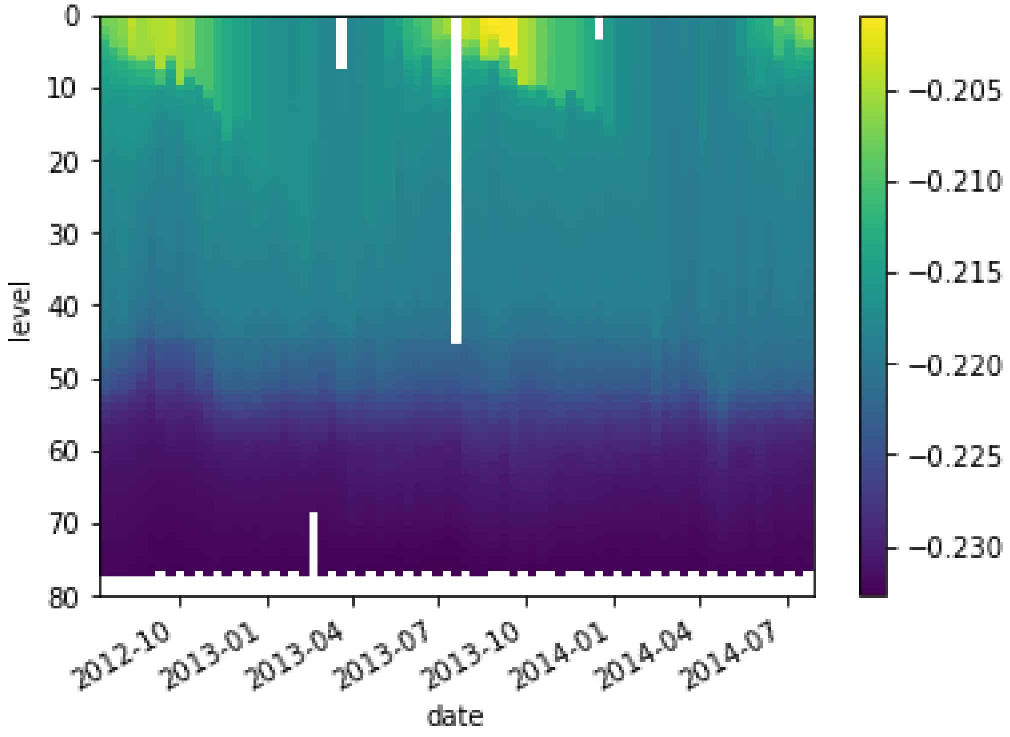
```
[17]: g = 9.8
      alpha = 2e-4
      beta = -7e-4

      buoyancy = g * (alpha * argo.temperature + beta * argo.salinity)

      buoyancy.plot()

      plt.ylim([80,0])
```

```
[17]: (80.0, 0.0)
```



we can do standard numpy math stuff like means, standard deviations, etc on dimensions.

We can average the whole dataset. xarray is smart, and it's going to average each of the data variabiles independantly:

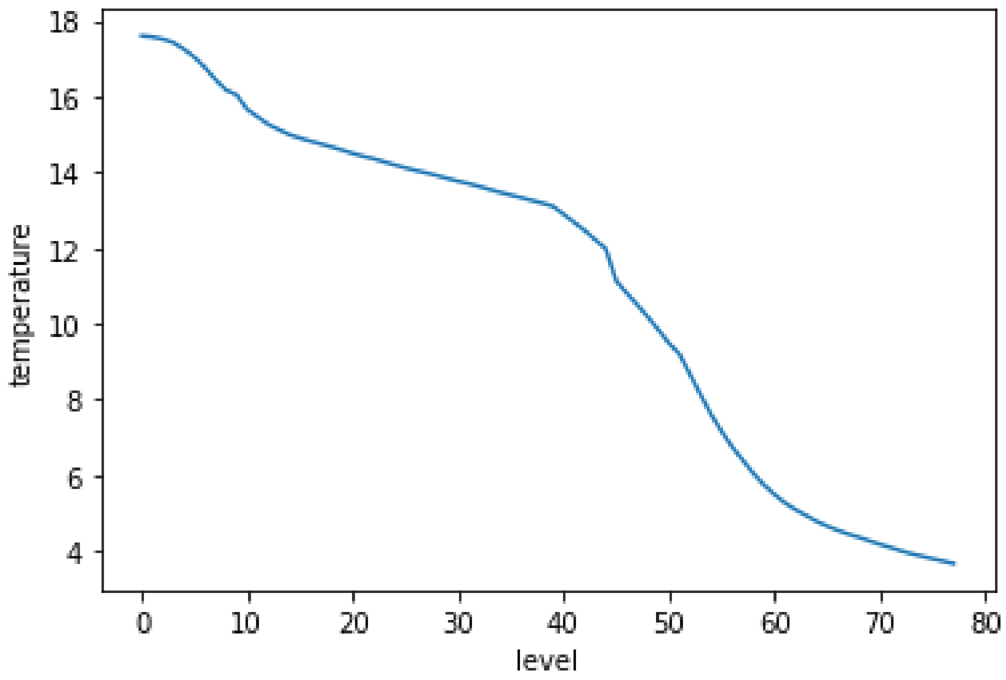
[18]:
```
argo_mean = argo.mean(dim='date')

argo_mean
```

[18]:
```
<xarray.Dataset>
Dimensions:      (level: 78)
Coordinates:
  * level        (level) int64 0 1 2 3 4 5 6 7 8 … 69 70 71 72 73 74 75 76 77
Data variables:
    salinity     (level) float64 35.91 35.9 35.9 35.9 … 34.94 34.94 34.93
    temperature  (level) float64 17.6 17.57 17.51 17.42 … 3.789 3.73 3.662
    pressure     (level) float64 6.435 10.57 15.54 … 1.95e+03 1.999e+03
```

[19]:
```
argo_mean.temperature.plot()
```

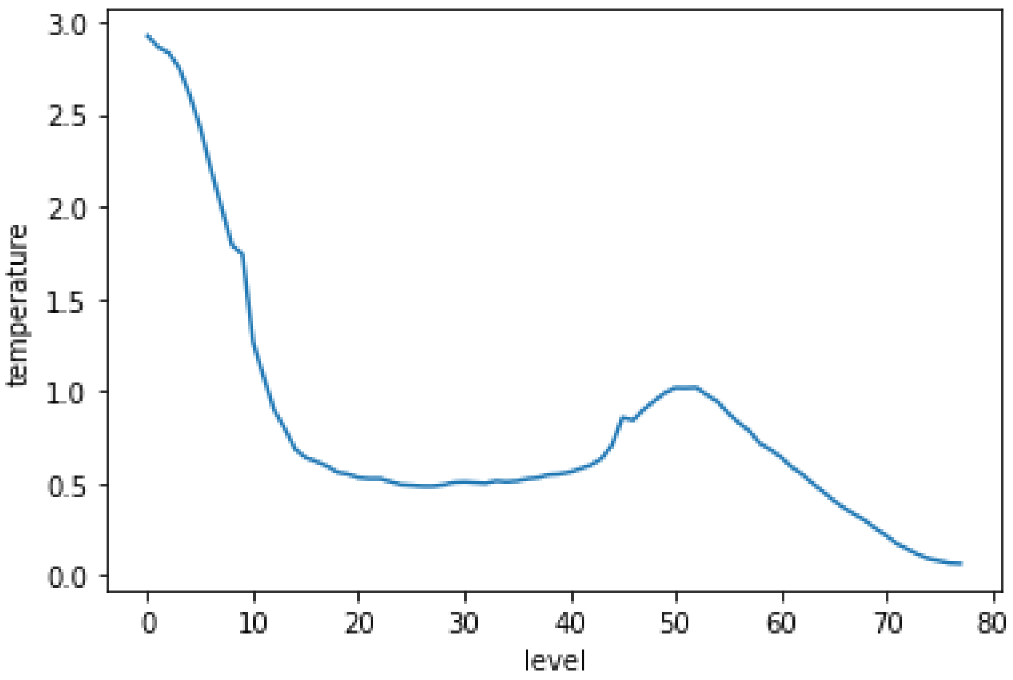
[19]:
```
[<matplotlib.lines.Line2D at 0x7fcc4ce812b0>]
```



```
[20]: # get the standard deviation:
      argo_std = argo.std(dim='date')

      argo_std.temperature.plot()
```

```
[20]: [<matplotlib.lines.Line2D at 0x7fcc4d031cd0>]
```



There are a lot more cool math/analysis functions we can do with xarray. We will see more of them later on.

The end…

7 Breakout / exercise 02