03_xarray_netcdf

August 13, 2021

1 The Network Common Data Format: netCDF

NetCDF is one of the most common ways that geoscience data is distributed. It was developed in the early 1990s specifically to deal with the challenges associated with multidimensional arrays.

Much of the climate/earth/ocean/atmosphere data that you can access will be in the form of netCDF files. They typically have the extention`.nc`, so like `ocean_temps.nc`.

NetCDF files are machine independant, meaning that macs, PCs, linux machines, you name it, they can all read the files.

Also, the netCDF files are self contained - i.e. they carry all the information about the data they contain with them. So they are 'self-describing' like the datasets and dataArrays we have been building.

In fact, Xarray is bascially a package devoted to reading, writing, and manipulating netCDFs. This means it's a super easy and useful way to work with geophysical data from nearly anywhere.

In this lesson we are going to use Xarray to load some Sea Surface Temperature data from a netCDF file. We will see how easy it is to make calculations and plots of these big data sets using Xarray.

1.0.1 credit

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

2 Loading netCDF datasets

The primary tool in the Xarray library that we will use with netCDF files is `xr.open_dataset()`. This will read in a netCDF file and create one of our DataArrays.

In this example we are going to read in a Sea Surface Temperature dataset created by NOAA that goes back to the 1800's. You can learn more about the data here: https://www.ncdc.noaa.gov/data-access/marineocean-data/extended-reconstructed-sea-surface-temperature-ersst-v5

First, let's do our normal import statements that we need to access the libraries in this new notebook:

[1]:
```
# do our imports
import xarray as xr
import numpy as np
```

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Let's load in the data using `xr.open_dataset()` and take a look at it:

[2]:
```
# load in the data with open_dataset()

url = 'http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/noaa.ersst.v5/sst.
↪mnmean.nc'
ds = xr.open_dataset(url, drop_variables=['time_bnds'])
# ds = xr.open_dataset('../data/NOAA_ERSSTv5_monthly.nc')

ds
```

[2]:
```
<xarray.Dataset>
Dimensions:  (lat: 89, lon: 180, time: 2011)
Coordinates:
  * lat      (lat) float32 88.0 86.0 84.0 82.0 80.0 … -82.0 -84.0 -86.0 -88.0
  * lon      (lon) float32 0.0 2.0 4.0 6.0 8.0 … 350.0 352.0 354.0 356.0 358.0
  * time     (time) datetime64[ns] 1854-01-01 1854-02-01 … 2021-07-01
Data variables:
    sst      (time, lat, lon) float32 …
Attributes: (12/38)
    climatology:                    Climatology is based on 1971-2000 SST, X…
    description:                    In situ data: ICOADS2.5 before 2007 and …
    keywords_vocabulary:            NASA Global Change Master Directory (GCM…
    keywords:                       Earth Science > Oceans > Ocean Temperatu…
    instrument:                     Conventional thermometers
    source_comment:                 SSTs were observed by conventional therm…
    …                              …
    license:                        No constraints on data access or use
    comment:                        SSTs were observed by conventional therm…
    summary:                        ERSST.v5 is developed based on v4 after …
    dataset_title:                  NOAA Extended Reconstructed SST V5
    data_modified:                  2021-08-07
    DODS_EXTRA.Unlimited_Dimension:  time
```

Did that work? There is a lot of information there. Let's go through all of it to make sure we understand what our Dataset looks like.

Draw on the board and answer the following: * What are the dimensions of the data? * What is the data itself * what do the coordinate of the dimensions look like? * draw a schematic of the data and label all the 'sides' * what is the stuff in the attributes?

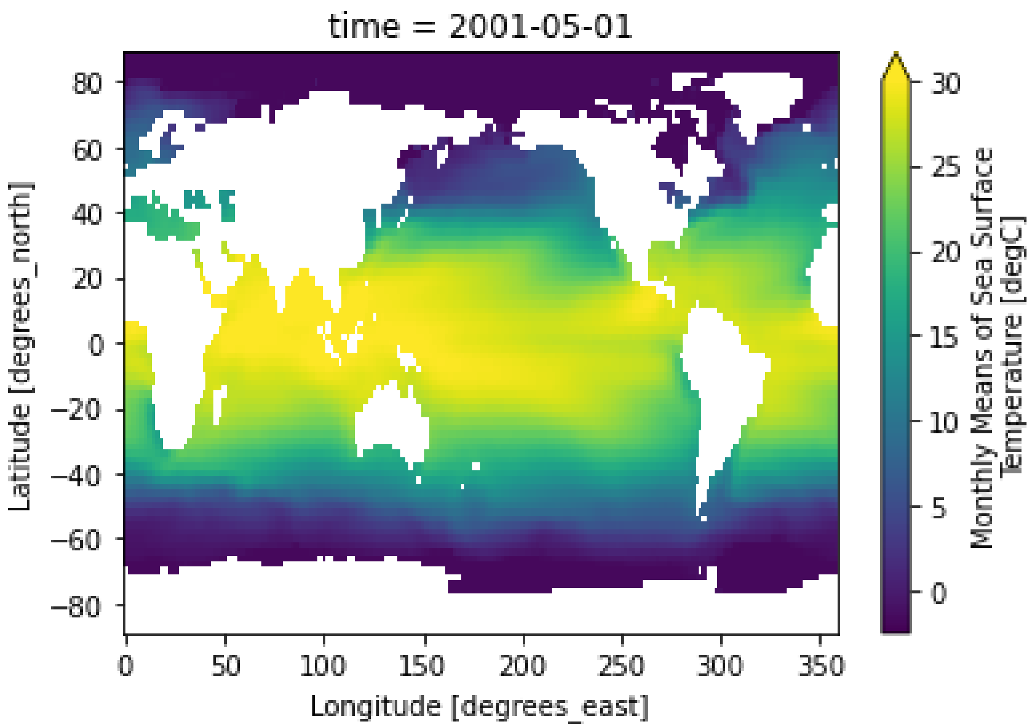
3 plotting netcdf data

Next let's make some plots to look at the data. We have lat, lon, Sea Surface Temperature data over a range of times. Maybe let's start with a simple plot of the SST all over the globe on one particular day. What is a good day?

note if you look at the time dimension, we see that the data is reported in monthly means with dates on the first of the month - let's pick the first day of a month.

[3]:
```
ds.sst.sel(time = '2001-05-01').plot( vmin=-2.5, vmax=30)
```

[3]: <matplotlib.collections.QuadMesh at 0x7fd8507372e0>



what if we pick a different day of the month?

[4]:
```
ds.sst.sel(time = '2001-05-15').plot( vmin=-2.5, vmax=30)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/pandas/
↪_libs/index.pyx in pandas._libs.index.DatetimeEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTabl.
↪get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTabl.
↪get_item()

KeyError: 989884800000000000
```

```
During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/pandas/
 ↪core/indexes/base.py in get_loc(self, key, method, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/pandas/
 ↪_libs/index.pyx in pandas._libs.index.DatetimeEngine.get_loc()

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/pandas/
 ↪_libs/index.pyx in pandas._libs.index.DatetimeEngine.get_loc()

KeyError: Timestamp('2001-05-15 00:00:00')

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/pandas/
 ↪core/indexes/datetimes.py in get_loc(self, key, method, tolerance)
    701         try:
--> 702             return Index.get_loc(self, key, method, tolerance)
    703         except KeyError as err:

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/pandas/
 ↪core/indexes/base.py in get_loc(self, key, method, tolerance)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364

KeyError: Timestamp('2001-05-15 00:00:00')

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
<ipython-input-4-8146d897c4b9> in <module>
----> 1 ds.sst.sel(time = '2001-05-15').plot( vmin=-2.5, vmax=30)

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/xarray/
 ↪core/dataarray.py in sel(self, indexers, method, tolerance, drop,␣
 ↪**indexers_kwargs)
   1313         Dimensions without coordinates: points
   1314         """
```

```
-> 1315         ds = self._to_temp_dataset().sel(

   1316             indexers=indexers,
   1317             drop=drop,

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/xarray/
↪core/dataset.py in sel(self, indexers, method, tolerance, drop,␣
↪**indexers_kwargs)
   2472         """
   2473         indexers = either_dict_or_kwargs(indexers, indexers_kwargs,␣
↪"sel")
-> 2474         pos_indexers, new_indexes = remap_label_indexers(

   2475             self, indexers=indexers, method=method, tolerance=tolerance
   2476         )

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/xarray/
↪core/coordinates.py in remap_label_indexers(obj, indexers, method, tolerance,␣
↪**indexers_kwargs)
    419     }
    420
--> 421     pos_indexers, new_indexes = indexing.remap_label_indexers(

    422         obj, v_indexers, method=method, tolerance=tolerance
    423     )

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/xarray/
↪core/indexing.py in remap_label_indexers(data_obj, indexers, method, tolerance)
    115     for dim, index in indexes.items():
    116         labels = grouped_indexers[dim]
--> 117         idxr, new_idx = index.query(labels, method=method,␣
↪tolerance=tolerance)
    118         pos_indexers[dim] = idxr
    119         if new_idx is not None:

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/xarray/
↪core/indexes.py in query(self, labels, method, tolerance)
    222                 indexer = index.get_loc(label_value)
    223             else:
--> 224                 indexer = index.get_loc(

    225                     label_value, method=method, tolerance=tolerance
    226                 )

/Users/teaching/opt/anaconda3/envs/swbc2021/lib/python3.8/site-packages/pandas/
↪core/indexes/datetimes.py in get_loc(self, key, method, tolerance)
    702             return Index.get_loc(self, key, method, tolerance)
    703         except KeyError as err:
--> 704             raise KeyError(orig_key) from err
    705
```

```
    706     def _maybe_cast_for_get_loc(self, key) -> Timestamp:

KeyError: '2001-05-15'
```

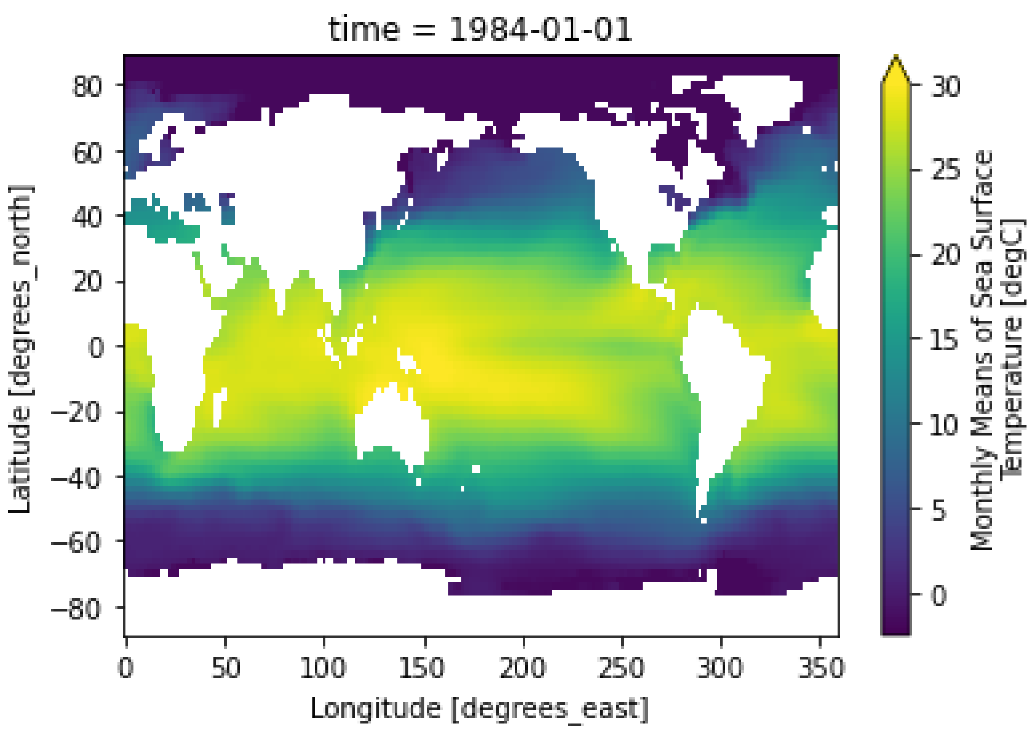
We got an error because we asked for a specific day that isn't in the dataset. We can get around this sort of thing luckily!

3.0.1 Nearest point indexing, or 'nearest neighbor lookups'

In the case above we input an exact date that is avilabile in our data. What if we didn't know all the exact dates? Try putting a random date in to the plot call. What if we want to get the time closest to some date we care about? Xarray can handle this if we give it an extra arguement using `method='nearest'`:

[5]:
```
ds.sst.sel(time='12-20-1983', method='nearest').plot(vmin=-2.5, vmax=30)
```

[5]: <matplotlib.collections.QuadMesh at 0x7fd850d2ed90>



Ok, so we can pretty easily make a plot of global SST on a single day. That is pretty cool.

We can use this dataset to see some amazing things without doing a lot of hard work thanks to the people who developed xarray (and the people who created/collected the data!!!!!).

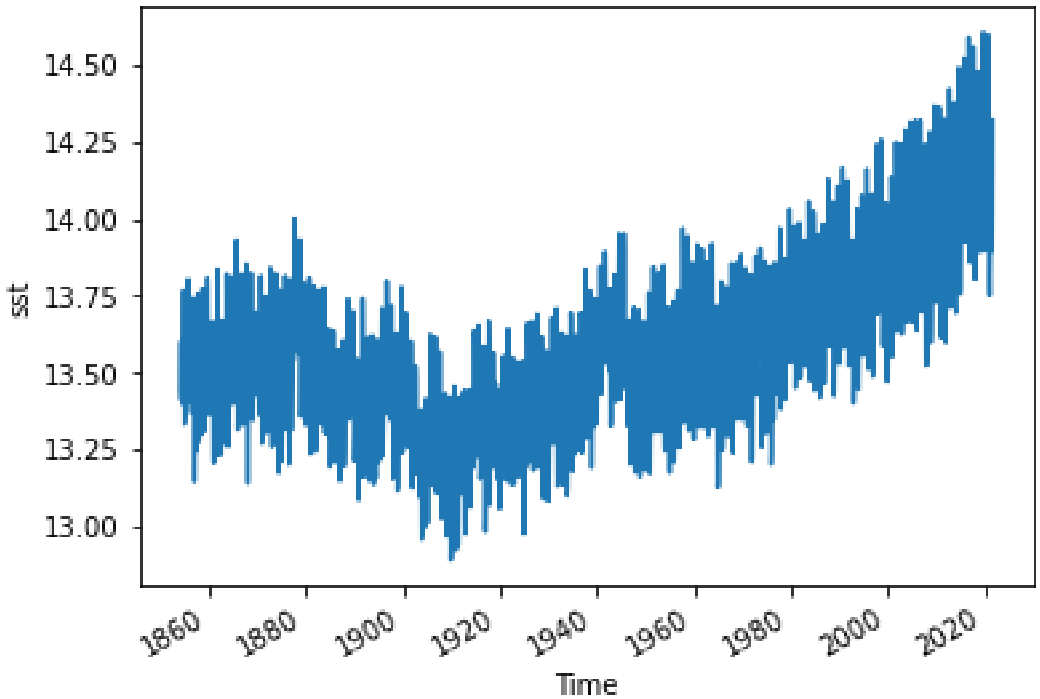
3.0.2 Let's make a simple plot to see how global average sea surface temperature has changed over time. Do you think we will be able to see a warming signal?

To do this we want to use xarray's `.mean()` function. But we need to tell it what kind of mean we want. In other words we need to define the dimensions over which to take the mean. If we are interested in makeing a plot that shows global averaged sea surface temperature over time, what are the dimentions to average over?

we are going to do something like: `ds.sst.mean( dim = ('dim1', ...) ).plot()` fill in the blanks:

[6]:
```
ds.sst.mean(dim=('lat', 'lon')).plot()
# ds.ssta.mean(dim=('lat', 'lon')).plot()
```

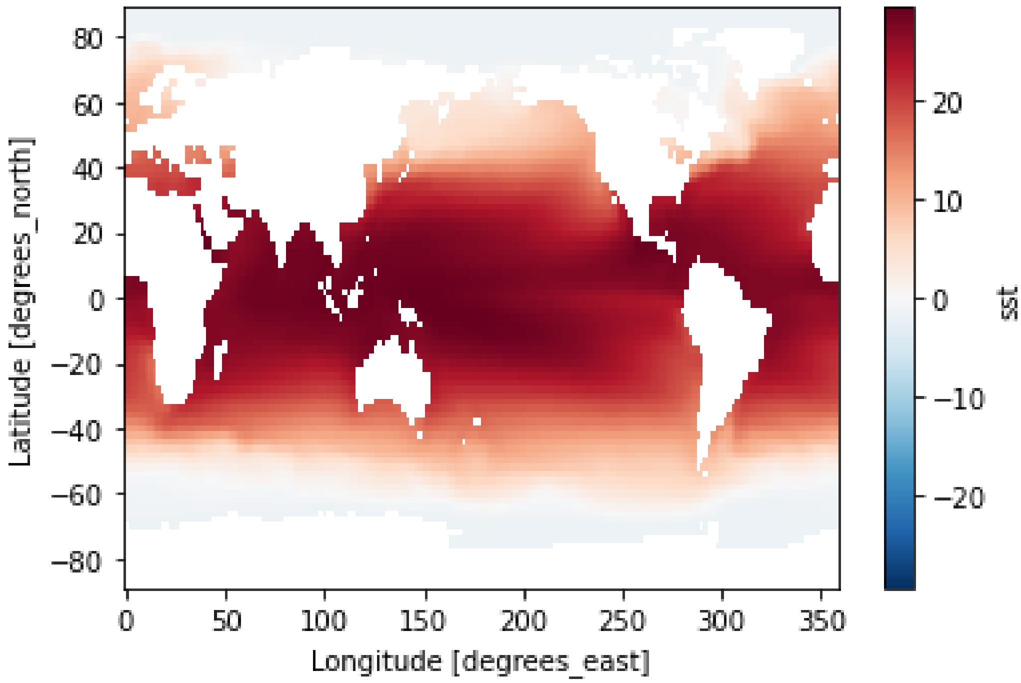
[6]:
```
[<matplotlib.lines.Line2D at 0x7fd8362bb550>]
```



What about just plotting the time average map of SST? What dimensions are we going to average over here?

[7]:
```
ds.sst.mean(dim=('time')).plot()
```

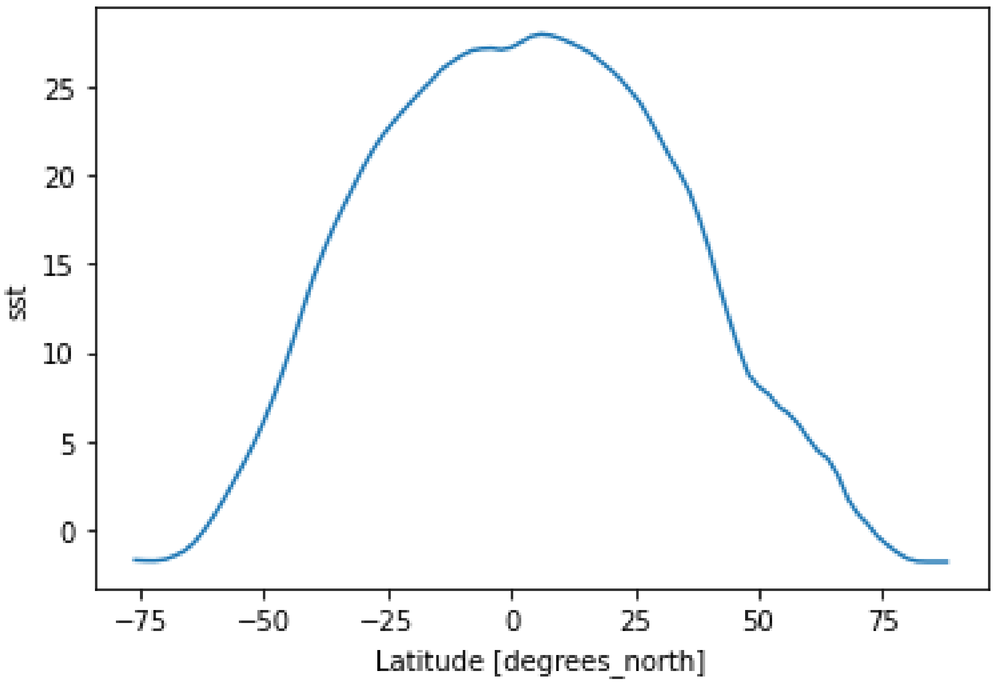
[7]:
```
<matplotlib.collections.QuadMesh at 0x7fd83648d0a0>
```



What about the average temperature as a function of latitude? We want to make a line plot that shows how temperature depends on latitude only, how would we do that?

[8]:
```
ds.sst.mean( dim=('lon', 'time')).plot()
```

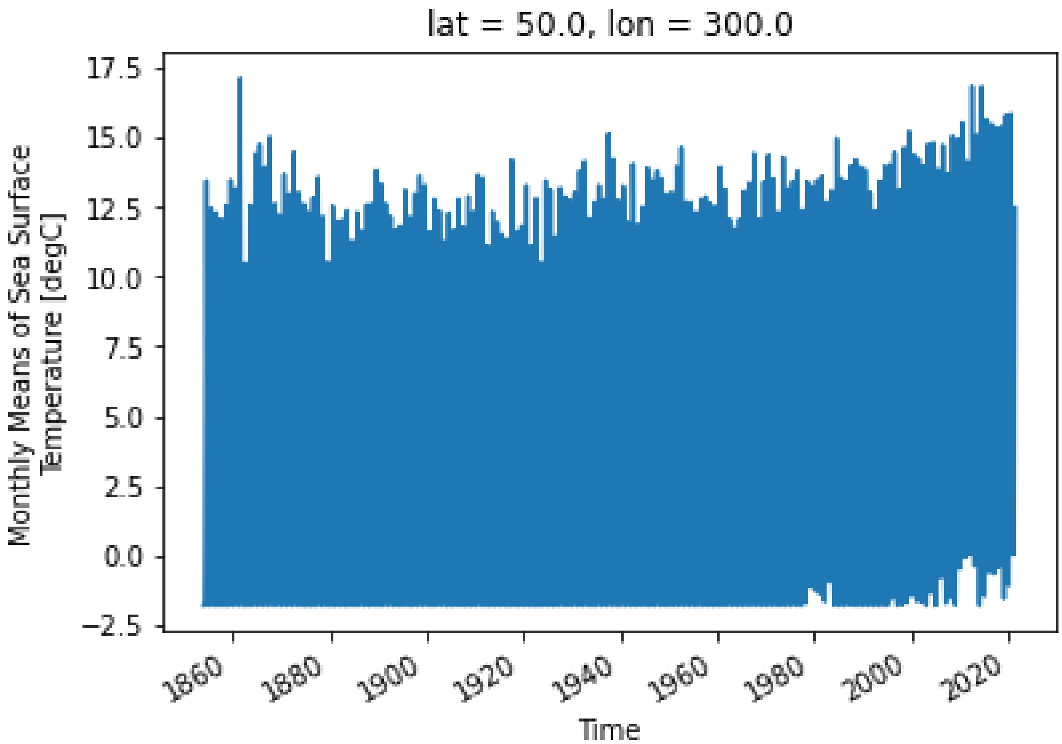
[8]: [<matplotlib.lines.Line2D at 0x7fd834421c40>]



How about a timeseries of temperature at a single point? Let's make a plot of the SST at 45 degrees north, and 230 degrees. How do we do that? Recall the `.sel()` method, and it's arguement `nearest`

[9]:
```
ds.sst.sel( lon=300, lat=50, method='nearest').plot()
```

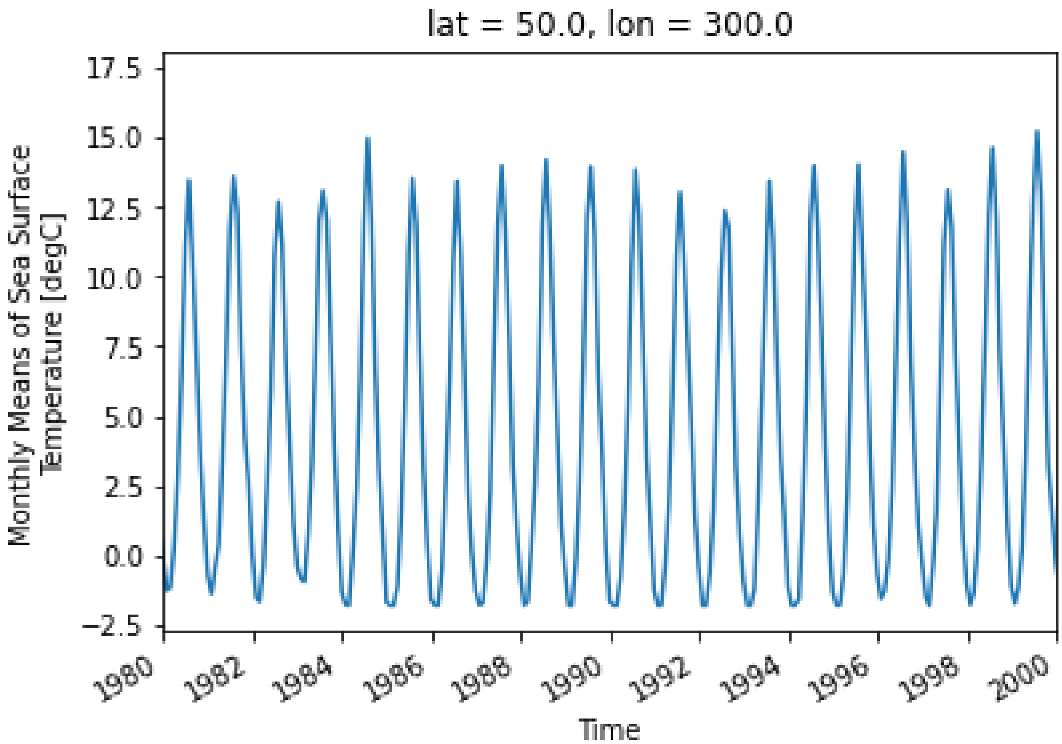
[9]: `[<matplotlib.lines.Line2D at 0x7fd83448c1f0>]`



that is a mess. Let's adjust the axis so we can see what is happening in that blue mess. Let's pick 20 years of data, from 1980 to 2000 and zoom in. We can do this by setting the range of the x axis. We are going to build up a lot of tricks to make plots look the way we want. This is one.

[10]:
```
ds.sst.sel( lon=300, lat=50, method='nearest').plot()
plt.xlim(['1-1-1980', '1-1-2000'])
```

[10]:
```
(3652.0, 10957.0)
```

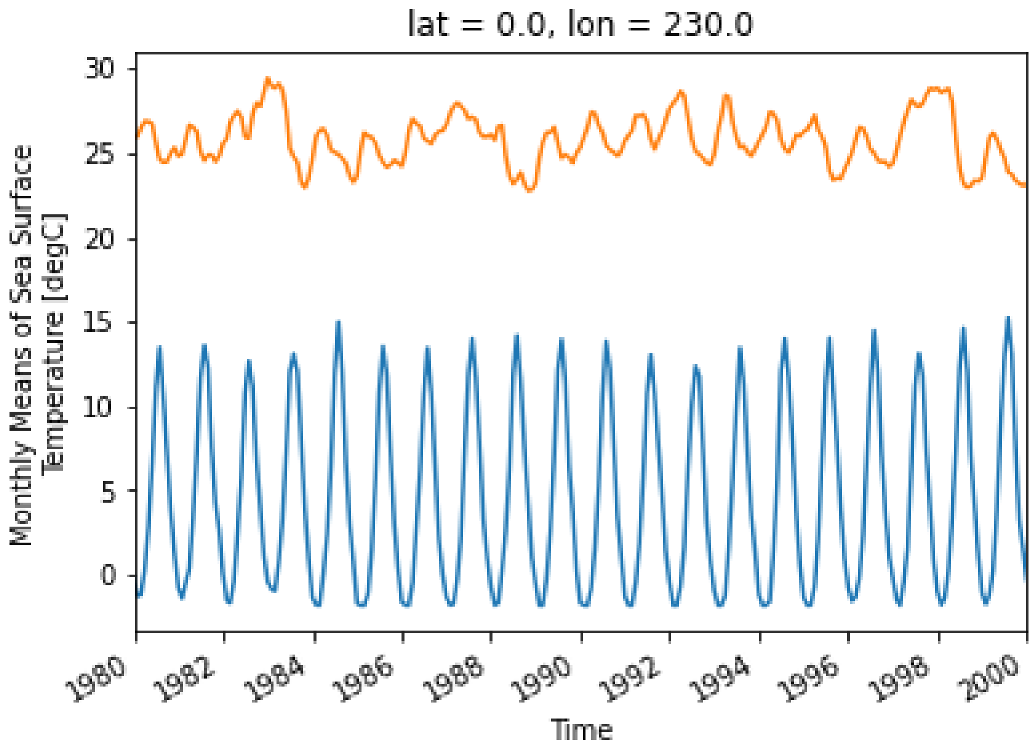


Huh. That's cool. What are we seeing here?

Let's plot two different Latitudes, one high lat and one on the equator:

[11]:
```
ds.sst.sel( lon=300, lat=50, method='nearest').plot()

ds.sst.sel( lat=0, lon=230, method='nearest').plot()

plt.xlim(['1-1-1980', '1-1-2000'])
```

[11]: (3652.0, 10957.0)



4 Groupby

Yep, we can do groupby here too.

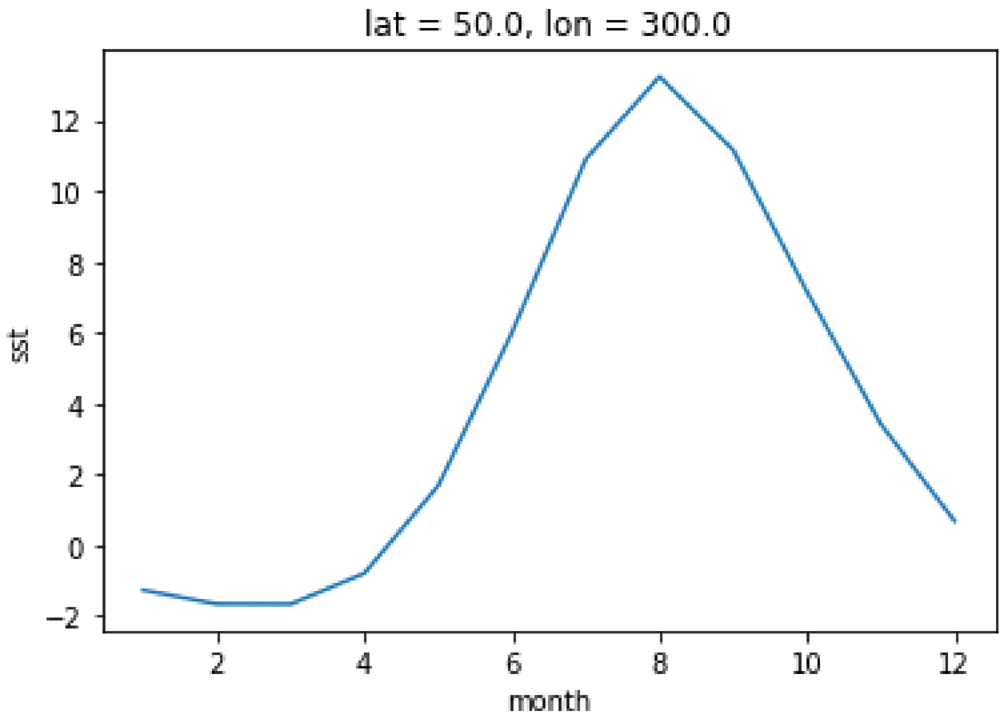
Let's groupby month and apply a mean. This will give us a climatology of SST from the past couple hundered years at every point on the globe:

[12]:
```
# group by month
gb = ds.groupby('time.month')

# apply a time mean to the groups to get a monthly mean climatology dataset
ds_mm = gb.mean(dim='time')
```

climatology at a specific point in the North Atlantic:

[13]:
```
ds_mm.sst.sel(lon=300, lat=50).plot()
```

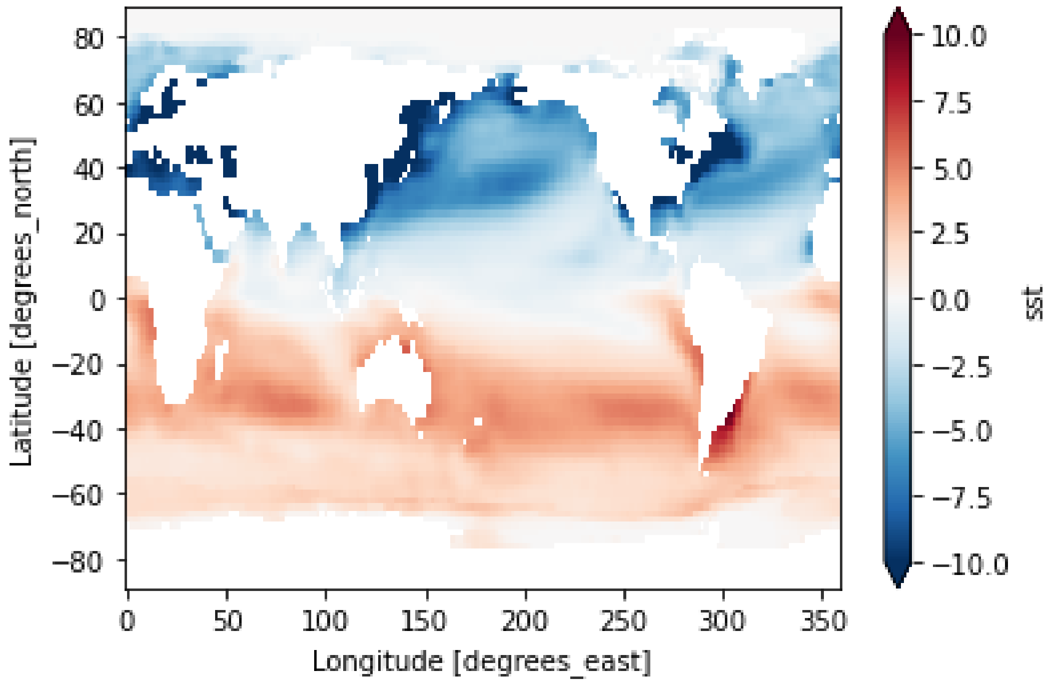
[13]: [<matplotlib.lines.Line2D at 0x7fd84c48a220>]



Plot the July minus Jan differences

[14]:
```
seasonal_diff = ds_mm.sst.sel(month=1) - ds_mm.sst.sel(month=7)

seasonal_diff.plot(vmax=10)
```

[14]: <matplotlib.collections.QuadMesh at 0x7fd829fa1070>



4.0.1 remove a time mean

Let’s look more clearly at the long term SST trend by removing the seasonal climatology

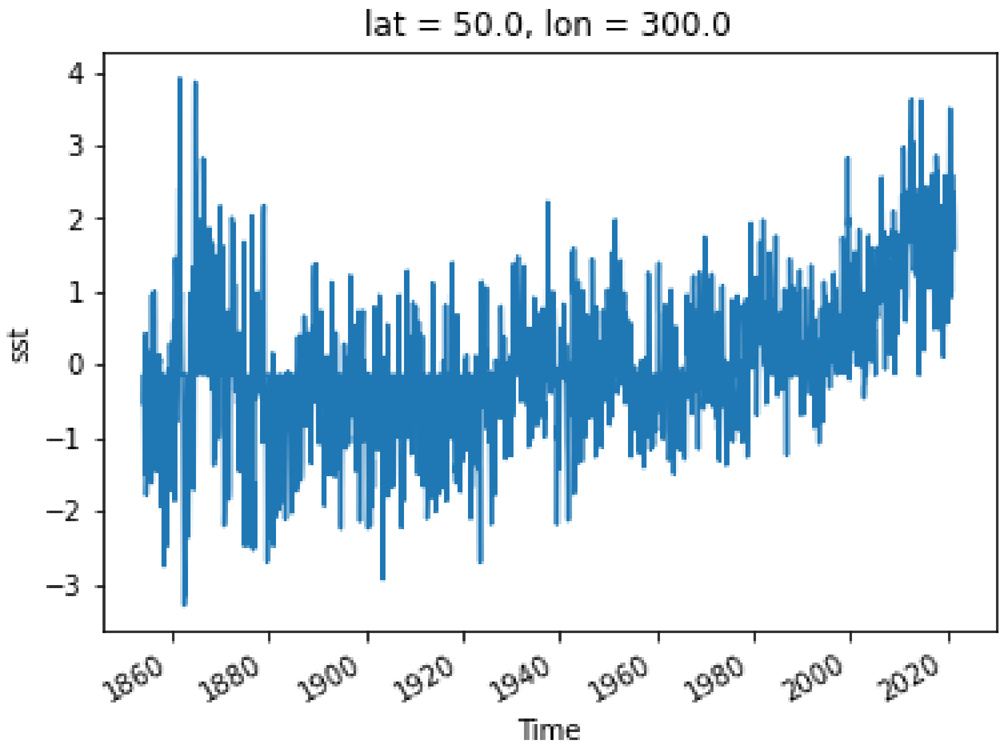
[15]:
```
gb = ds.groupby('time.month')

ds_anom = gb - gb.mean(dim='time')

ds_anom
```

[15]:
```
<xarray.Dataset>
Dimensions:  (lat: 89, lon: 180, time: 2011)
Coordinates:
  * lat      (lat) float32 88.0 86.0 84.0 82.0 80.0 … -82.0 -84.0 -86.0 -88.0
  * lon      (lon) float32 0.0 2.0 4.0 6.0 8.0 … 350.0 352.0 354.0 356.0 358.0
  * time     (time) datetime64[ns] 1854-01-01 1854-02-01 … 2021-07-01
    month    (time) int64 1 2 3 4 5 6 7 8 9 10 11 … 9 10 11 12 1 2 3 4 5 6 7
Data variables:
    sst      (time, lat, lon) float32 0.0 0.0 0.0 0.0 0.0 … nan nan nan nan
```

timeseries of SST anomaly at a certain point:

[16]:
```
ds_anom.sst.sel(lon=300, lat=50).plot()
```

[16]:
```
[<matplotlib.lines.Line2D at 0x7fd8288bed60>]
```



4.1 Saving data to netcdf

Suppose we are always working with the mean surface temperature. Here calculating the mean is fast, but suppose it were very slow… It would be useful to save the mean data so we don't have to repeat the calculation.

Xarray makes that very easy. In general it works like this:

```
name = "whatever.nc"
some_dataset.to_netcdf(name)
```

So lets try that for our data:

[17]:
```
# generate mean over latitude and longitude
sst_mean = ds.sst.mean(dim=('lat', 'lon'))

name = "sst_mean.nc"

sst_mean.to_netcdf(name)
```

The end…

5 Breakout / exercise 03