MR2020: Coding for METOC

Module 11: Xarray

What is Xarray?

Xarray is an open-source Python library designed for working with multi-dimensional labeled datasets. Built on top of **NumPy** and **Pandas**. Ideal for **N-dimensional arrays** (like climate data, oceanography, and more). Integrates well with SciPy, Matplotlib, Dask.

We will often use Xarray to open data from a variety of formats, store the data, query the data, and grab desired parts of the data.

Xarray also integrates well with Jupyter Notebooks or interactive Python.

User documentation: https://docs.xarray.dev/en/stable/

Clear and readable data

```
ds
✓ 0.0s
```

xarray.Dataset

▶ Dimensions: (**time**: 24, **lon**: 193, **lat**: 121)

▼ Coordinates:

time	(time)	datetime64[ns]	2018-10-23T00:30:00 ... 2018-10-...
lon	(lon)	float64	60.0 60.62 61.25 ... 179.4 180.0
lat	(lat)	float64	-30.0 -29.5 -29.0 ... 29.5 30.0

▼ Data variables:

EFLUXWTR	(time, lat, lon)	float32	...
HFLUXWTR	(time, lat, lon)	float32	...
LWGNTWTR	(time, lat, lon)	float32	...
QV10M	(time, lat, lon)	float32	...
SWGNTWTR	(time, lat, lon)	float32	...
T10M	(time, lat, lon)	float32	...

▶ Indexes: (3)

▶ Attributes: (32)

Clear, readable, and self-describing data. Easily access variables, attributes, coordinates, and values for N-dimensional data!

Reading data – Supported File Formats

1. NetCDF (Network Common Data Form)
    - Commonly used format for variety of data in METOC community

2. GRIB/GRIB2 (GRIdded Binary)
    - Often used as output for various model data. Advantage is smaller file size. Disadvantage was more difficult to visualize, but this is no longer true.

3. HDF5 (Hierarchical Data Format)
    - More flexible than NetCDF; more complex data organization capabilities

4. Zarr
    - More modern data format similar to NetCDF and HDF. Works well with cloud object stores such as in Amazon S3.

5. OPeNDAP (Open-source Project for a Network Data Access Protocol)
    - Useful for reading specific subsets of remote datasets

6. CSV (Comma-Separated Values; Indirectly read via Pandas)
    - Text files used for storing tabular data. Can be opened as a spreadsheet.
    - Also useful to load as Pandas DataFrame (see Module 12).

How do I access the contents of a file?

```
ds = xr.open_dataset(filename)
```

On future slides, we will assume that the object holding the data is `ds`.

`open_dataset` will open many filetypes including NetCDF, GRIB, HDF, and OPeNDAP.

```
#GRIB
ds = xr.open_dataset('file.grib',engine='cfgrib')

#OPenDAP
ds = xr.open_dataset('https://noaa.gov/example.nc',engine='pydap')
```

Sometimes a particular engine needs to be specified, for example, with GRIB files or those downloaded via OPeNDAP.

How do I access the contents of a file?

```
#Zarr
ds = xr.open_zarr('file.zarr')
```

`open_zarr` will open Zarr files.

Some of the file types require special methods.

For GeoTIFF raster files, use rioxarray.

```
#CSV
df = pd.read_csv('file.csv')
ds = xr.Dataset.from_dataframe(df)
```

CSV files need to be opened with the Pandas `read_csv` method but can be converted to an Xarray Dataset using the `from_dataframe` method.

What are some commonly used Xarray classes?

Dataset:

- A collection of multiple DataArrays with shared coordinates. Similar to an entire NetCDF file.

```
ds
✓ 0.0s
```

xarray.Dataset

▶ Dimensions: (**time**: 24, **lon**: 193, **lat**: 121)

Note the dimensions and sizes of each. We will return to this soon.

▼ Coordinates:

time	(time)	datetime64[ns]	2018-10-23T00:30:00 ... 2018-10-...
lon	(lon)	float64	60.0 60.62 61.25 ... 179.4 180.0
lat	(lat)	float64	-30.0 -29.5 -29.0 ... 29.5 30.0

▼ Data variables:

EFLUXWTR	(time, lat, lon)	float32	...
HFLUXWTR	(time, lat, lon)	float32	...
LWGNTWTR	(time, lat, lon)	float32	...
QV10M	(time, lat, lon)	float32	...
SWGNTWTR	(time, lat, lon)	float32	...
T10M	(time, lat, lon)	float32	...

▶ Indexes: (3)

▶ Attributes: (32)

```
xarray.Dataset

▶ Dimensions:       (time: 24, lon: 193, lat: 121)
▼ Coordinates:
    time            (time)              datetime64[ns]  2018-10-23T00:30:00 ... 2018-10-...
    lon             (lon)               float64         60.0 60.62 61.25 ... 179.4 180.0
    lat             (lat)               float64         -30.0 -29.5 -29.0 ... 29.5 30.0
▼ Data variables:
    EFLUXWTR        (time, lat, lon)    float32         ...
    HFLUXWTR        (time, lat, lon)    float32         ...
    LWGNTWTR        (time, lat, lon)    float32         ...
    QV10M           (time, lat, lon)    float32         ...
    standard_name : 10-meter_specific_humidity
    long_name :     10-meter_specific_humidity
    units :         kg kg-1
    fmissing_value : 1000000000000000.0
    vmax :          1000000000000000.0
    vmin :          -1000000000000000.0
    SWGNTWTR        (time, lat, lon)    float32         ...
    T10M            (time, lat, lon)    float32         ...
▶ Indexes:    (3)
▶ Attributes: (32)
```

If I click this button for a variable, I will get its metadata. Click it again to hide the metadata.

Click the drop-down arrows to show or hide different parts of the Dataset.

▼ Attributes:

CDI :	Climate Data Interface version 1.9.8 (https://mpimet.mpg.de/cdi)
Conventions :	CF-1
History :	Original file generated: Fri Nov 2 21:55:48 2018 GMT
Comment :	GMAO filename: d5124_m2_jan10.tavg1_2d_ocn_Nx.20181023.nc4
Filename :	MERRA2_400.tavg1_2d_ocn_Nx.20181023.nc4
Institution :	NASA Global Modeling and Assimilation Office
References :	http://gmao.gsfc.nasa.gov
Format :	NetCDF-4/HDF-5
SpatialCoverage :	global
VersionID :	5.12.4
TemporalRange :	1980-01-01 -> 2016-12-31
identifier_product...	http://dx.doi.org/
ShortName :	M2T1NXOCN
GranuleID :	MERRA2_400.tavg1_2d_ocn_Nx.20181023.nc4
ProductionDateTi...	Original file generated: Fri Nov 2 21:55:48 2018 GMT
LongName :	MERRA2 tavg1_2d_ocn_Nx: 2d,1-Hourly,Time-Averaged,Single-Level,Assimilation,Ocean Surface Diagnostics
Title :	MERRA2 tavg1_2d_ocn_Nx: 2d,1-Hourly,Time-Averaged,Single-Level,Assimilation,Ocean Surface Diagnostics
SouthernmostLatit...	-90.0
NorthernmostLatit...	90.0
WesternmostLong...	-180.0
EasternmostLongi...	179.375
LatitudeResolution :	0.5
LongitudeResoluti...	0.625
DataResolution :	0.5 x 0.625
Contact :	http://gmao.gsfc.nasa.gov
identifier_product...	10.5067/Y67YQ1L3ZZ4R
RangeBeginningD...	2018-10-23
RangeBeginningTi...	00:00:00.000000
RangeEndingDate :	2018-10-23
RangeEndingTime :	23:59:59.000000
history_L34RS :	'Created by L34RS v1.4.1 @ NASA GES DISC on November 13 2020 18:36:17. Spatial: 60.0 180.0 -30.0 30.0. Variables: EFLUXWTR HFLUXWTR LWGNTWTR QV10M SWGNTWTR T10M'
CDO :	Climate Data Operators version 1.9.8 (https://mpimet.mpg.de/cdo)

Clicking the attributes drop-down arrow will display metadata for the entire file. Notice that 32 attributes are shown and compare that to the number in parentheses next to Attributes on the previous slide.

What are some commonly used Xarray classes?

DataArray:

- N-dimensional array with labeled dimensions, coordinates, and attributes. Similar to a single variable in a NetCDF file.

```
ds.T10M
```

✓ 0.0s

In Interactive Python/Jupyter notebooks, to extract a single DataArray, use `object.varname`

xarray.DataArray 'T10M' (**time**: 24, **lat**: 121, **lon**: 193)

```
[560472 values with dtype=float32]
```

▼ Coordinates:

time	(time)	datetime64[ns]	2018-10-23T00:30:00 ... 2018-10-...
lon	(lon)	float64	60.0 60.62 61.25 ... 179.4 180.0
lat	(lat)	float64	-30.0 -29.5 -29.0 ... 29.5 30.0

▶ Indexes: (3)

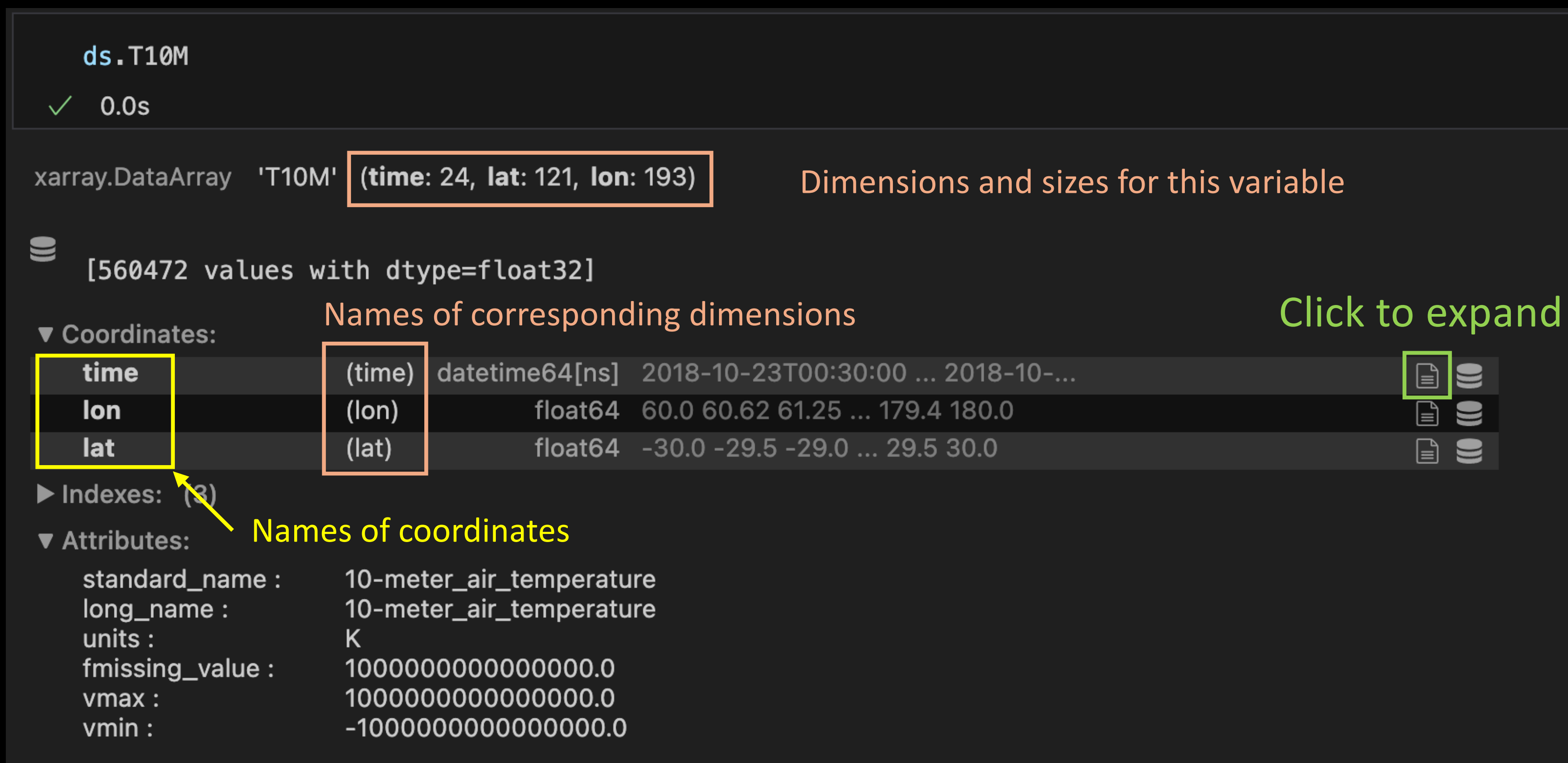
▼ Attributes:

standard_name :	10-meter_air_temperature
long_name :	10-meter_air_temperature
units :	K
fmissing_value :	1000000000000000.0
vmax :	1000000000000000.0
vmin :	-1000000000000000.0

What are some commonly used Xarray classes?

Coordinates:

- Labels for the dimensions (e.g., time, latitude, longitude). Allows for meaningful data slicing and indexing. Often the labels have the same names as the dimensions.



Exploring Datasets and DataArrays with code

```
# Get variables in Dataset
ds.variables

# Get attributes of Dataset
ds.attrs

# Get coordinates of Dataset
ds.coords

# Example: List all variables in Dataset
for var in ds.variables:
    print(var)

# Extract data for a specific DataArray as NumPy array.
ds.T10M.values
```

Indexing and slicing with Coordinates

Xarray allows us to index and slice using the coordinates of a DataArray rather than needing to know exact indices like when working with NumPy arrays.

```
# Extract data for a specific variable as NumPy array.
ds.T10M.values

# Extract data for T10M between 5N, 5S, 60E, and 80E.
ds.T10M.sel(lat=slice(-5,5), lon=slice(60,80))

# We can do the same using positional indexing but you need
# to remember which dimension corresponds to each coordinate
# in the correct order.
ds.T10M.loc[:,-5:5,60:80]

# Find time series of data at point closest to 10N, 120E.
ds.T10M.sel(lat=10, lon=120, method='nearest').values

# Returns new DataArray with NaNs were condition not met.
t10m = ds.T10M # Assign entire DataArray to new variable.
t10m.where(t10m>300) # Returns same size array but NaN where <= 300
```

Simple computations along dimensions

```
# Calculate the time mean at every location.
mean_temp = ds.T10M.mean(dim='time')

# Calculate the time-zonal mean at every latitude.
# In other words, calculate mean across two dimensions.
# Calculate the time-zonal mean
time_zonal_mean = ds.T10M.mean(dim=['time', 'lon'])

# Sum a value over time, assuming the variable 'precipitation'
exists.
total_precipitation = ds.precipitation.sum(dim='time')

# Find the max or min at each location
t10m_max = ds.T10M.max(dim='time’)
```

Applying custom functions

```
# Define a custom function, e.g., range (max – min)
def data_range(x,axis=None):
    return x.max(axis=axis) – x.min(axis=axis)

# Apply the function along the time dimension
range_temp = ds.T10M.reduce(data_range, dim='time')
```

Use the reduce method with the function name and dimension as inputs.

```
# NOTE: This example could also be accomplished with
# ds.T10M.reduce(np.ptp, dim='time')
```

Resampling and Grouping

```
# Resample to daily mean using Pandas back-end
daily_mean = ds.T10M.resample(time='D').mean()
```

Resampling can be used if you want to combine data over a regular interval longer than the existing time step between data points.

For example, suppose you had a data point for every day for 5 years, but you wanted an average for each month (e.g., Jan. 2020, Feb. 2020, Mar. 2020, etc.). You could resample to a monthly ('M') mean. You could also calculate a median, max, min, etc.

```
# Grouping using Pandas back-end
hourly_mean = ds.T10M.groupby('time.hour').mean(dim=['time’])`
```

Grouping can be used if you want to combine all data from a certain group.

For example, suppose you had daily data for 5 years, and you wanted to get the average for all Februarys during those 5 years. Then you could use `groupby`. As with resampling, you could compute a median, max, min, etc. as well.