Python Bootcamp Day 5

Multi-dimensional data with xarray

Instructors: Ziqi Yin and Devon Dunmire

Goals for the day

- Review arrays
- Understand multi-dimensional arrays
- Learn the basics of xarray

As a reminder:

Text in black will be instruction and guidance and will usually start with a section number.

Text in blue will be tasks to do.

Text in red will be optional challenges and advanced concepts.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Reviewing two-dimensional arrays

Let's create a two-dimensional list:

In [ ]:
```
A = [[1, 2, 3, 4], ['fruit', 'fruit', 'vegetable','fruit'], ['apple', 'banana', 'carrot', 'pear']]
```

Remember, we can see the dimensions of A with `np.shape()` .

In [ ]:
```
np.shape(A)
```

Knowing we have three rows and four columns, let's practice indexing. If we want just the names of the fruits and vegetables, we can index the whole last row.

In [ ]:
```
A[-1]
# Alternatively, you can write A[2]
```

Change the above code to print out the `['fruit', 'fruit', 'vegetable','fruit']` row.

In [ ]:
```
A[1]
```

What if we want to index a specific value? Well, we'll need to add an extra index.

In [ ]:
```
A[-1][1]
```

What do you think will be printed if we call A[0][2]? Write your answer here:

3

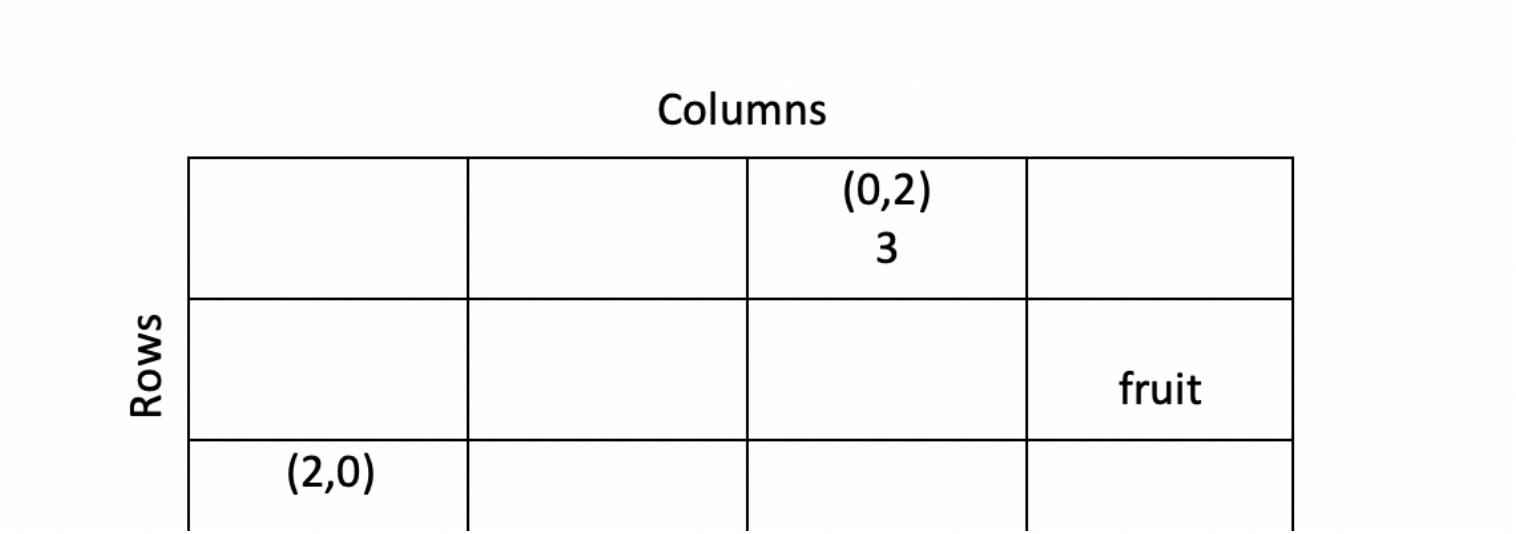
Now try it out!

In [ ]:
```
A[0][2]
```

Checkpoint: Arrays

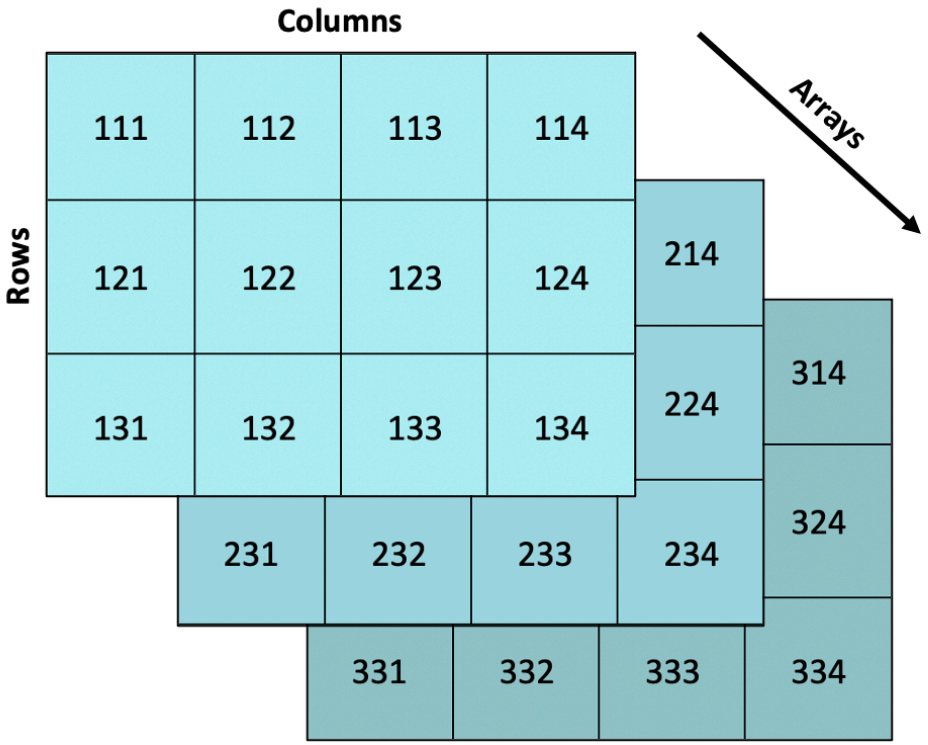
The table below represents a two-dimensional array. In each box there should be an index in the form of `(row index, col index)` and the corresponding value for that index from array `A`.

Task: On a piece of paper, fill out the remainder of the table.



Section 1: Multi-dimensional arrays

In many of our fields, we need to work with datasets that are multi-dimensional. A multi-dimensional array is essentially an array of arrays. It will therefore have more than two dimensions.

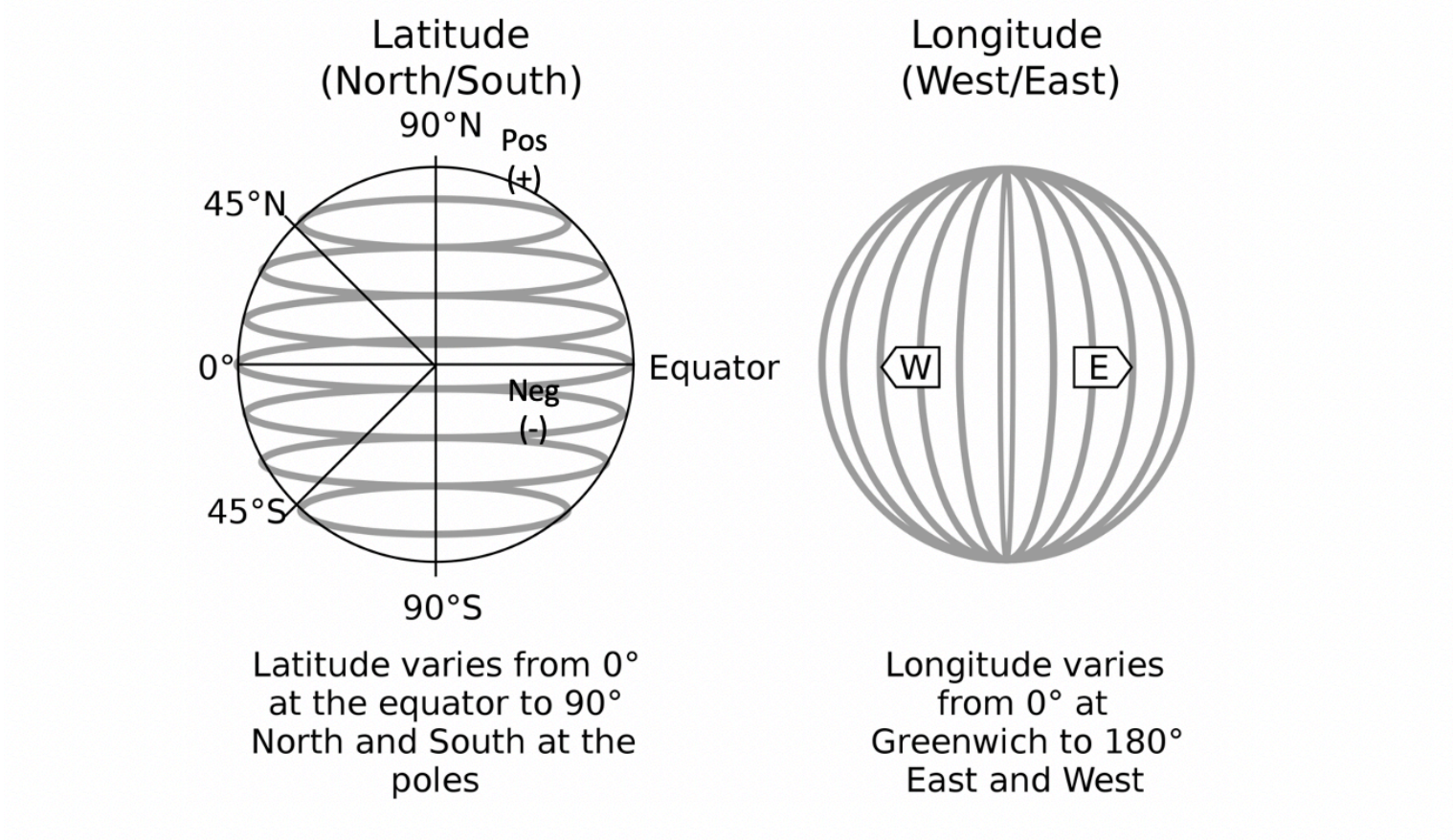


Using multi-dimensional arrays, we can look at variables over latitude, longitude, altitude/depth, and time. These can be called coordinate variables because they are used to describe the variables of interest.

Before we actually start working with multi-dimensional arrays, let's have a quick review of latitude and longitude.

1.1 Latitude and Longitude

Latitude (lat) serves as our y-axis. It runs North to South, with positive values being north of the equator and negative values south of the equator. Longitude (lon) serves as our x-axis. It runs West to East, with West of the Prime Meridian (Greenwich) being negative and East postive.



Multi-dimensional arrays can be overwhelming at first, but they give us the ability to do a lot!

Conveniently, there are powerful tools to load and manipulate multi-dimensional arrays. We've already looked at `numpy`, which handles the manipulation of arrays wonderfully. As the name of this notebook implies, the tool we'll be looking at today is `xarray`, oftentimes abbreviated as `xr`.

Section 2: Introduction to xarray

Xarray uses labels provided in datasets to separate the data into named dimensions. As we will see, this make it easier to select and manipulate the data. It's also important to note that the package distinguishes between datasets and dataarrays. Dataarrays hold a single multi-dimensional variable and its coordinates. Datasets hold multiple variables that may share the same coordinates.

Let's start by importing xarray (in the future, this should be imported at the beginning of the notebook with our other packages).

In [ ]:
```
import xarray as xr
```

2.1 Read in data

We will use precipitation data obtained from satellite estimates and gauge data. The dataset includes monthly averaged precipitation rate values (mm/day). Please download the `precip.mon.mean.nc` file from the Drive Day 6/4.

Now we can load in a multi-dimensional array, giving it an arbitrary name (fname).

In [ ]:
```
fname = xr.open_dataset('precip.mon.mean.nc')
```

Reminder: When loading in a dataset, we must either specify the file path or have the file in our current working directory.

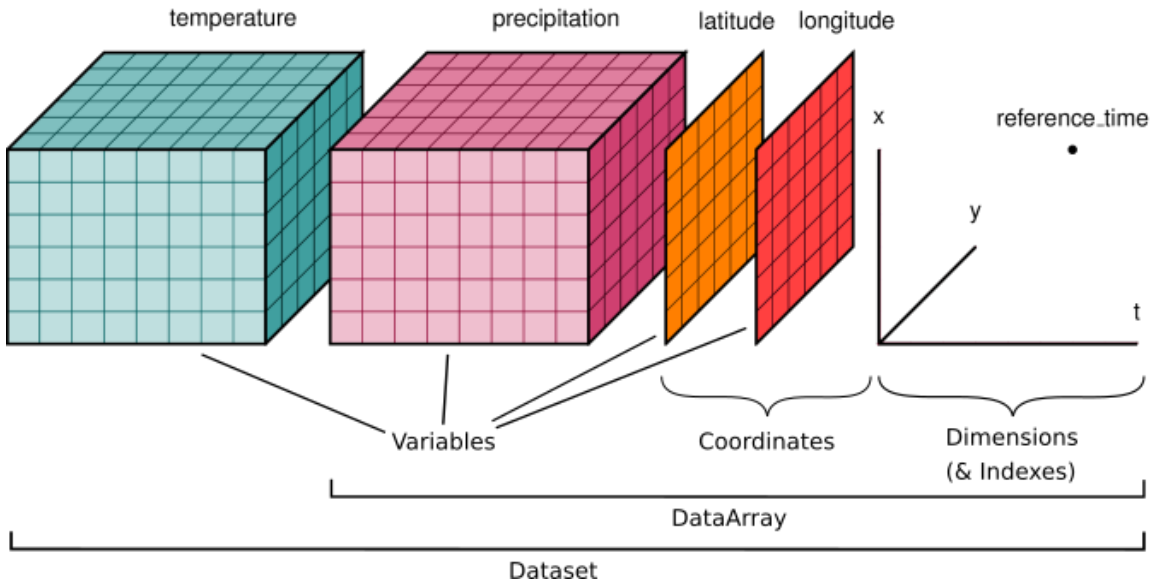
2.2 Inside the Dataset

Now that we have a file loaded in, we can look inside it.

In [ ]:
```
fname
```

The file has been broken down into four categories:

1. **Dimensions**
   Dimensions gives us information about the length of each dimension.
2. **Coordinates**
   Coordinates describe our variables.
3. **Data variables**
   Our data variables are the data that we're interested in.
4. **Attributes**
   Attributes give us more information about the file itself. Briefly take a look through the attributes provided in this file.



Task: What is the long name of the data variable, `precip`?

Average Monthly Rate of Precipitation

2.3 Browsing multi-dimensional arrays

Let's browse our datafile. It is important to recognize that these are "scratch" plots. They are great ways to look inside the data, but

Let's browse our dataset. It is important to recognize that these are "scratch" plots. They are great ways to look inside the data, but shouldn't be used as formal figures.

Two powerful xarray tools for browsing are `.isel` and `.sel`.

Using `.isel`, we can use the coordinate **indices** to quickly look inside the data. Looking at our dataset, latitude goes from pole to pole (88.75 to -88.75). Let's use `.isel` to select the south pole. We will assume a longitude of index 72.

In [ ]:
```
fname['precip'].isel(lat=-1).isel(lon=72).plot(label='Polar');
```

Using `.sel`, we can use the **coordinate value** at the index rather than the index itself. Let's use `.sel` to look at the mean monthly precipitation near the equator.

In [ ]:
```
fname['precip'].sel(lat=1.25).isel(lon=72).plot(label='Equator', color='orange');
```

We can also put them on the same figure to better compare the values.

In [ ]:
```
fname['precip'].isel(lat=-1).isel(lon=72).plot(label='Polar');
fname['precip'].sel(lat=1.25).isel(lon=72).plot(label='Equator', color='orange');
plt.legend();
```

Task: Plot a time series of mean monthly precipitation at the north pole and a longitude of 76.25°.

In [ ]:
```
fname['precip'].isel(lat=1).sel(lon=76.25).plot();
```

2.4 Manipulating multi-dimensional arrays

Tools we will use:

- `slice`
- `mean`
- `sum`
- `groupby`

For more information on xarray tools: http://xarray.pydata.org/en/stable/user-guide/index.html
For more information on xarray indexing: http://xarray.pydata.org/en/stable/user-guide/indexing.html

For convenience, we will store the precipitation data as it's own data array.

In [ ]:
```
precip = fname.precip
precip
```

We can use `slice` to constrain the data and we can use `.mean` to average the data along the given dimension. Let's look at the average precipitation in 1995.

In [ ]:
```
precip_1995 = precip.sel(time=slice('1995-01-01','1995-12-01')).mean(dim='time').plot()
```

`.groupby` is another useful tool. It allows us to split the data into multiple groups, work within these groups, and combine them back into a single dataset. Let's group time by season:

In [ ]:
```
precip.groupby('time.season')
```

The data has been divided into four seasons: 'DJF' (winter), 'JJA' (summer), 'MAM' (spring), and 'SON (fall).

Using `.sum`, we can sum over a given dimension. Combining groupby and sum, we can calculate the weighted seasonal averages. Let's just pick one year and one grid for simplicity. The below equation gives an example to calculate the weighted averaged preipitation for spring.

$$P_{spring} = P_{Mar} * \frac{31}{(31+30+31)} + P_{Apr} * \frac{30}{(31+30+31)} + P_{May} * \frac{31}{(31+30+31)}$$

In [ ]:
```
precip_example = precip.sel(time=slice('1995-01-01','1995-12-01')).sel(lat=1.25).isel(lon=72)
```

In [ ]:
```
month_length = precip_example.time.dt.days_in_month
month_length
```

In [ ]:
```
# Calculate the weights by grouping by 'time.season'
weights = month_length.groupby('time.season') / month_length.groupby('time.season').sum()

# Calculate the weighted average
precip_weighted = (precip_example * weights).groupby('time.season').sum(dim='time')
precip_weighted
```

In [ ]:
```
# Compared with unweighted seasonal averages
precip_unweighted = precip_example.groupby('time.season').mean(dim='time')
precip_unweighted
```

Task: Plot the sum of precipitation over March, April, and May of 2005.

In [ ]:
```
precip_2005 = precip.sel(time=slice('2005-03-01','2005-05-01')).sum(dim='time').plot()
```

2.5 Xarray to numpy

Once again, numpy is a powerful tool. Therefore, it is to our benefit to be able to convert xarray datasets to numpy arrays.
We can do this with `np.array()` .

In [ ]:
```
lat    = np.array(fname['lat'])
lon    = np.array(fname['lon'])
time   = np.array(fname['time'])
precip = np.array(fname['precip'])
lat
```

As seen above, Xarray introduces labels on top of raw NumPy-like multidimensional arrays, which allows for a more intuitive, more concise, and less error-prone developer experience. For example, with label names, you'll easily understand what you were thinking when you come back to look at your code weeks or months later.

2.6 Xarray to pandas

Xarray and pandas objects are also interchangeable. A shortcut to convert a dataArray directly into a pandas object is `DataArray.to_pandas()` (i.e., a 1D array is converted to a `Series` , 2D to `DataFrame` and 3D to `Panel`).

In [ ]:
```
# For example, from a 2D array with dimensions lat and lon
df = fname['precip'].sel(time='1995-01-01').to_pandas()
df
```

Xarray is heavily inspired by pandas, which also works with labeled arrays, and it uses pandas internally. The main distinguishing feature of xarray's dataarray over labeled arrays in pandas is that dimensions can have names (e.g. "time", "latitude"), which are easier to keep track of than axis numbers. While pandas is a great tool for working with tabular data, it can get a little awkward when data is of higher dimension (See more at https://xarray.pydata.org/en/v0.9.2/faq.html). Also, `Panel` (for 3D data in pandas) has been removed since version 0.25.0 in 2019. So, if the data is multi-dimensional, xarray will be a better tool.

2.7: Breakout Room Activity

We will be looking at Northern Hemisphere snowcover data.

Load in the snowcover data using xarray.

In [ ]:
```
snow_data = xr.open_dataset('snowcover.mon.mean.nc')
```

Look at the file.

In [ ]:
```
snow_data
```

Use .sel to plot a map of snowcover on September 1, 1978.

In [ ]:
```
snow_data['snowcover'].sel(time='1978-09-01').plot()
```

Use .sel, slice, and .mean to plot the average snowcover of the 1980s.

In [ ]:
```
snowco_1980s = snow_data['snowcover'].sel(time=slice('1980-01-01','1989-12-01')).mean(dim='time').plot()
```