Watch by Tuesday, November 3, 2020 | **Lesson #9**

`pandas` and `xarray`

Working with CSV and netCDF data files

OCEAN 215 | Autumn 2020

Ethan Campbell and Katy Christensen

What we'll cover in this lesson

1. **`pandas`: `Series` objects**

2. `pandas`: `DataFrame` objects; CSV files

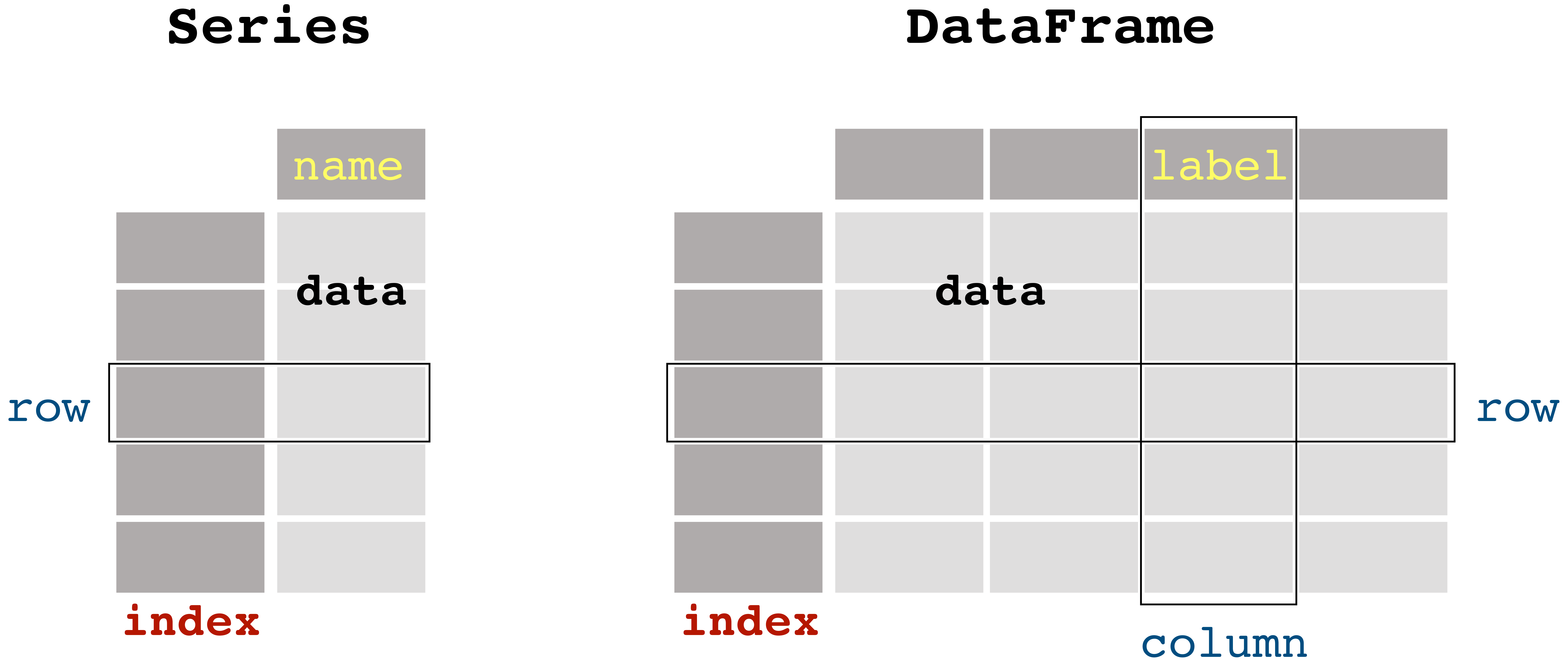
3. `xarray`: `DataArray` and `Dataset` objects; netCDF files

4. `xarray`: working with higher-dimensional data

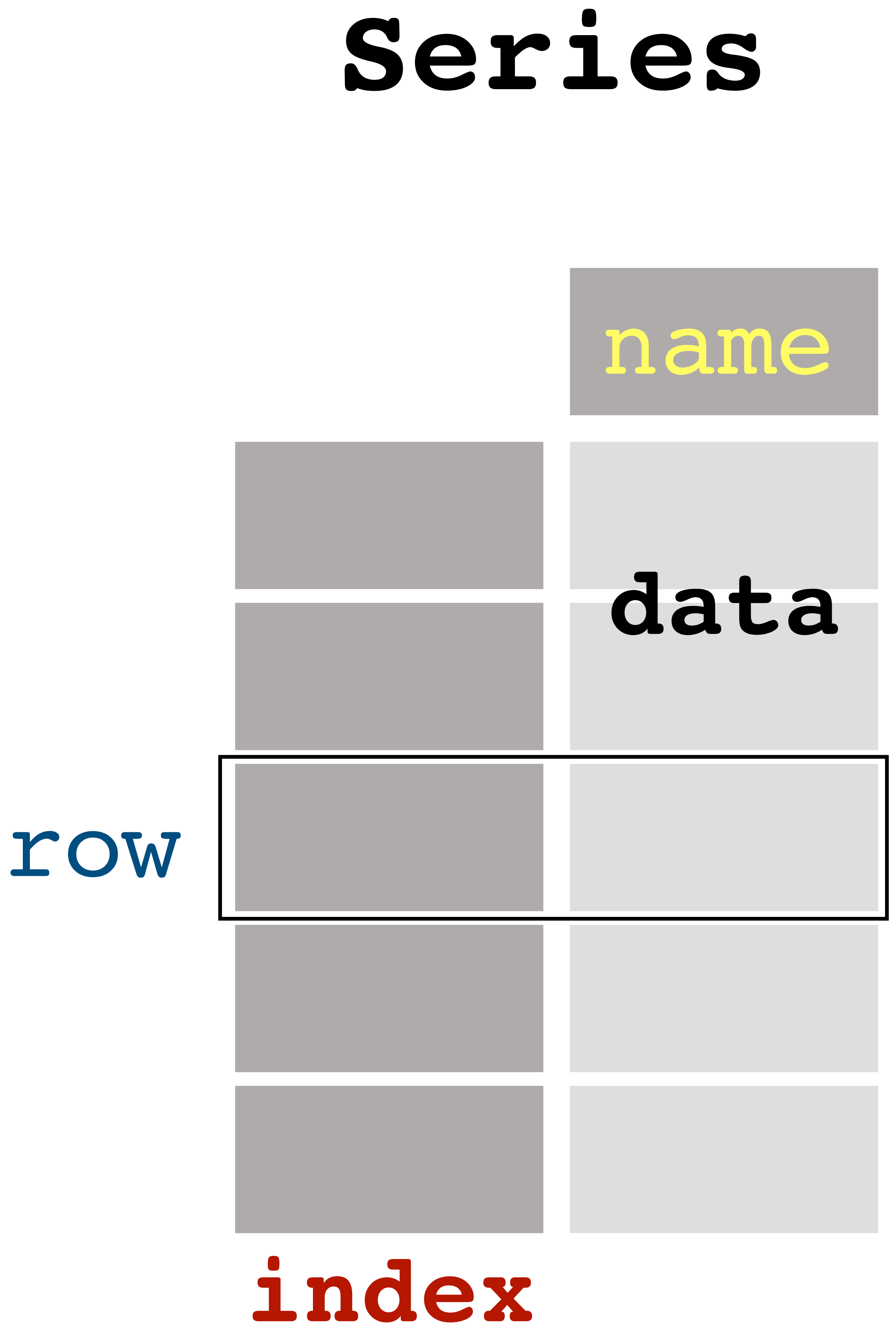
Loading `pandas`

```
import pandas as pd
```

pandas handles tabular data (tables or spreadsheets)



`pandas` handles tabular data (tables or spreadsheets)

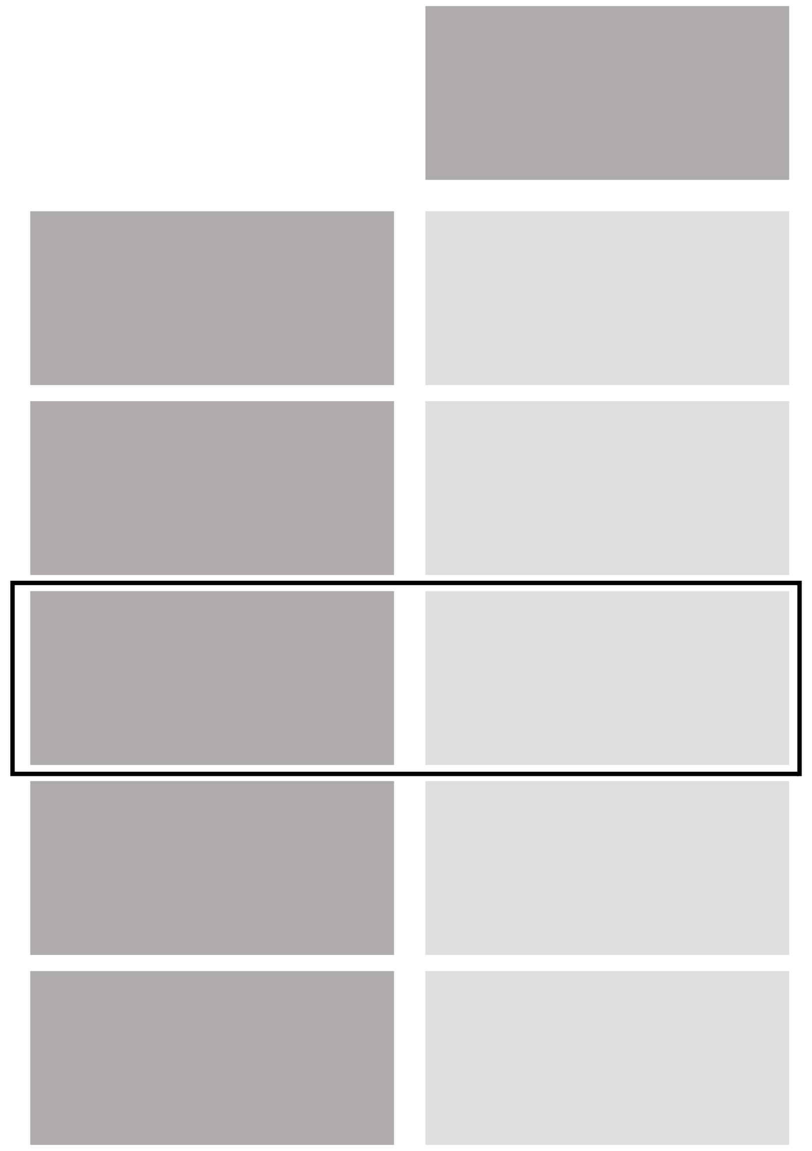


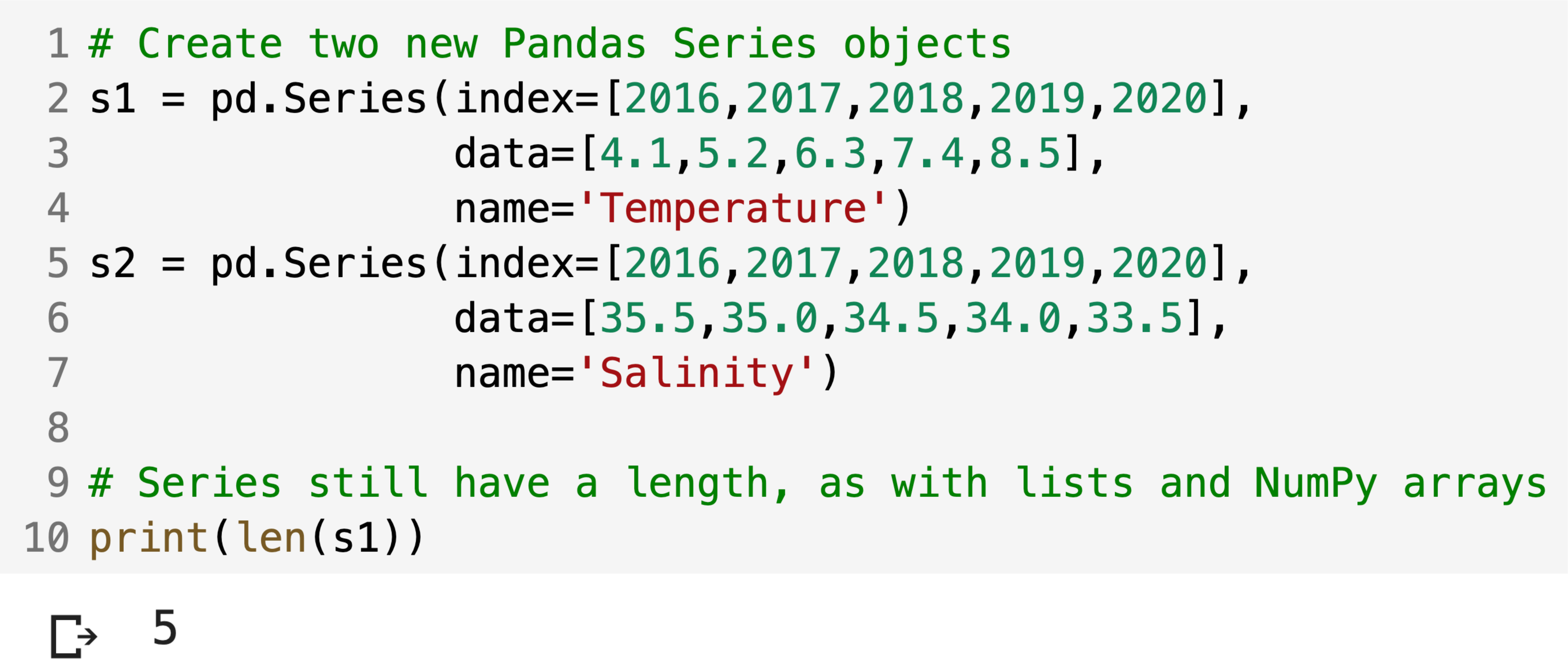
The `index` of a `Series` or `DataFrame` doesn’t need to contain integers starting at 0.

An `index` can consist of values of any type (e.g. `float`, strings, `datetime` objects).

Creating a Series object

pd.Series(index=<list or 1-D NumPy array>**,**
data=<list or 1-D NumPy array>**, name=**<string>**)**

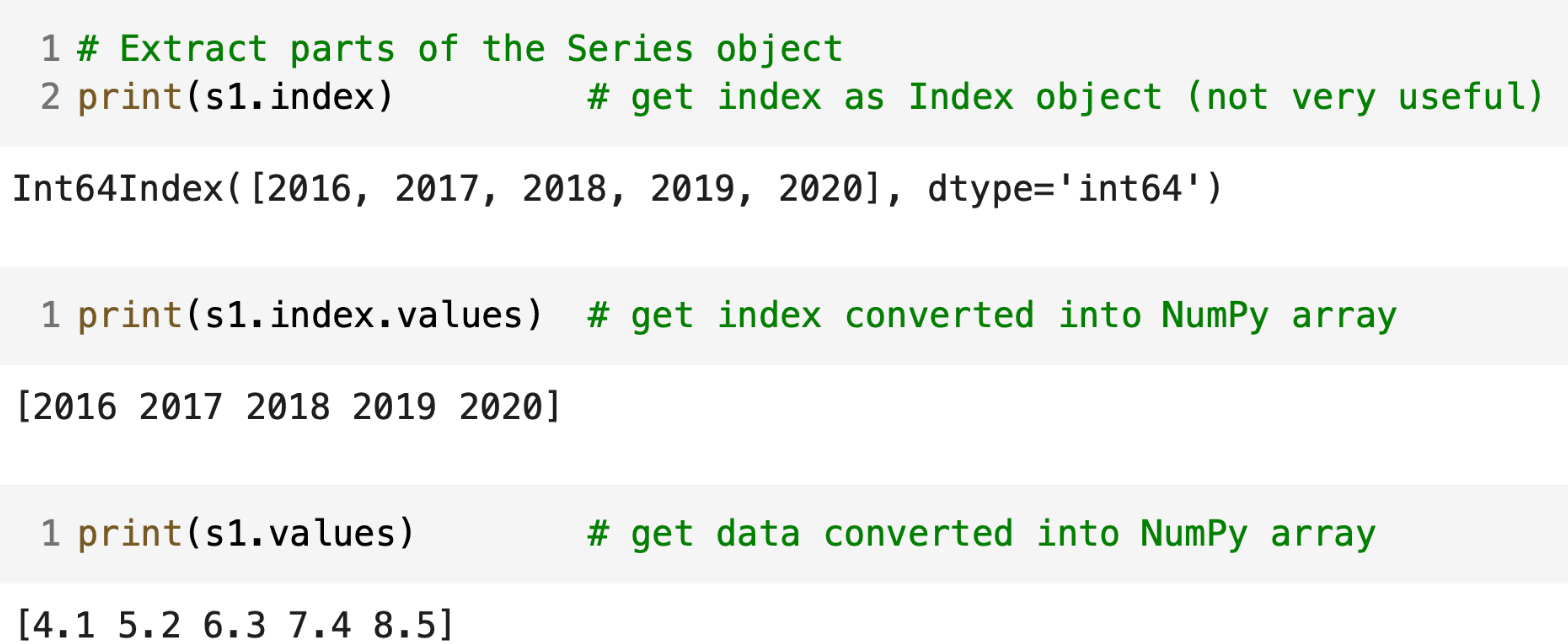




```
# Create two new Pandas Series objects
s1 = pd.Series(index=[2016,2017,2018,2019,2020],
               data=[4.1,5.2,6.3,7.4,8.5],
               name='Temperature')
s2 = pd.Series(index=[2016,2017,2018,2019,2020],
               data=[35.5,35.0,34.5,34.0,33.5],
               name='Salinity')

# Series still have a length, as with lists and NumPy arrays
print(len(s1))
```

```
5
```

Getting the index and data from a `Series`



```
# Extract parts of the Series object
print(s1.index)         # get index as Index object (not very useful)
```

```
Int64Index([2016, 2017, 2018, 2019, 2020], dtype='int64')
```

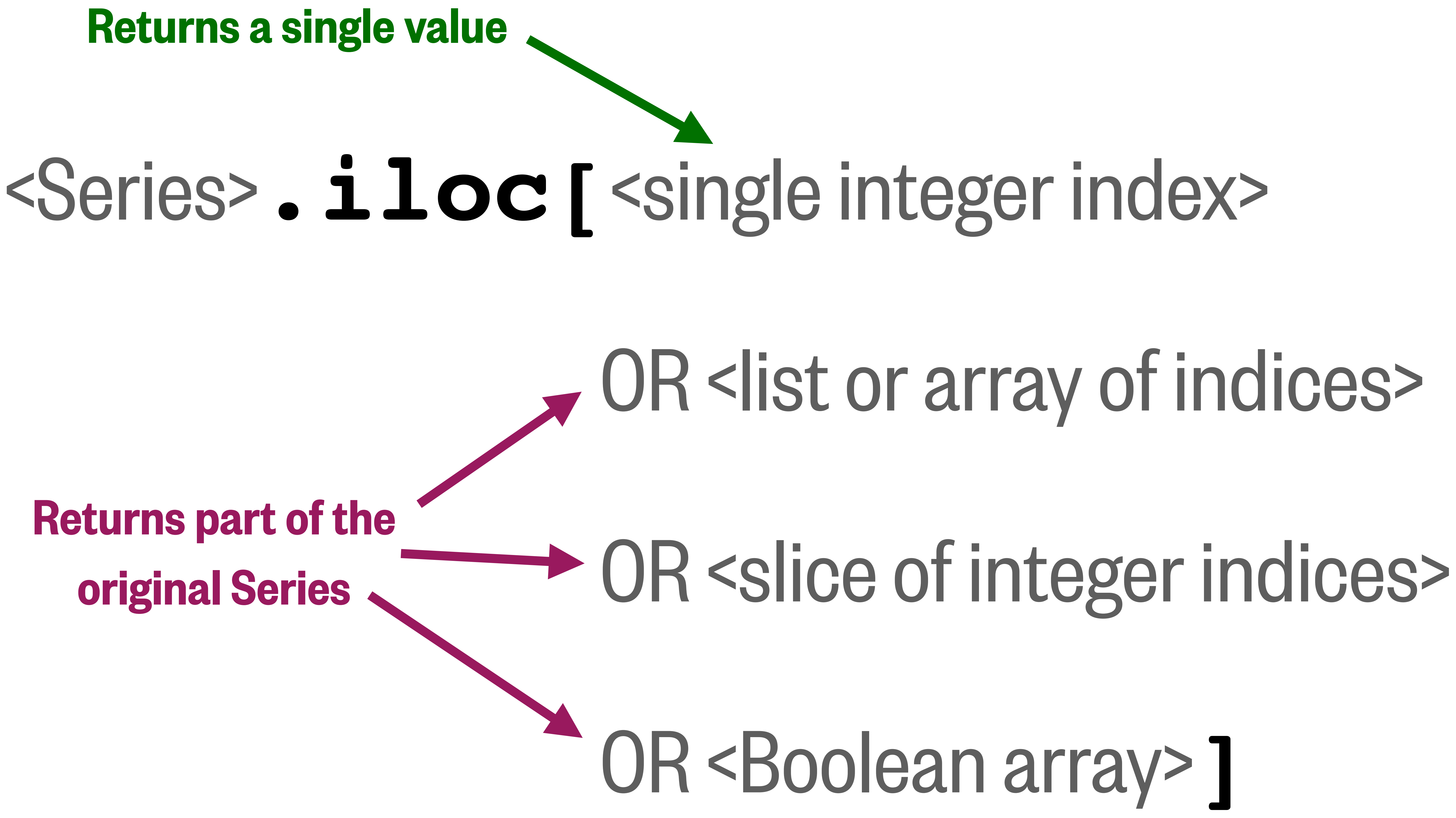
```
print(s1.index.values)  # get index converted into NumPy array
```

```
[2016 2017 2018 2019 2020]
```

```
print(s1.values)        # get data converted into NumPy array
```

```
[4.1 5.2 6.3 7.4 8.5]
```

Selecting data from a `Series` using `.iloc[ ]` (selection by integer index)



Returns a single value

<Series>`.iloc[` <single integer index>

OR <list or array of indices>

OR <slice of integer indices>

OR <Boolean array>`]`

Returns part of the original Series

Example:

```
s1.iloc[3]
```

```
s1.iloc[[2,3,4]]
```

```
s1.iloc[2:5]
```

```
s1.iloc[[False,False,
         True,True,False]]
```

Selecting data from a `Series` using `.loc[ ]` (selection by label)

Returns a single value

<Series>**`.loc[`** <single index label>

OR <list or array of labels>

OR <slice of index labels> **`]`**

Returns part of the original Series

Example:

```
s1.loc[2019]
```

```
s1.loc[[2018,2019,2020]]
```

```
s1.loc[2018:2020]
```

Unlike Python/NumPy slicing, the end value is inclusive!

Reminder: convert the resulting `Series` to a NumPy array

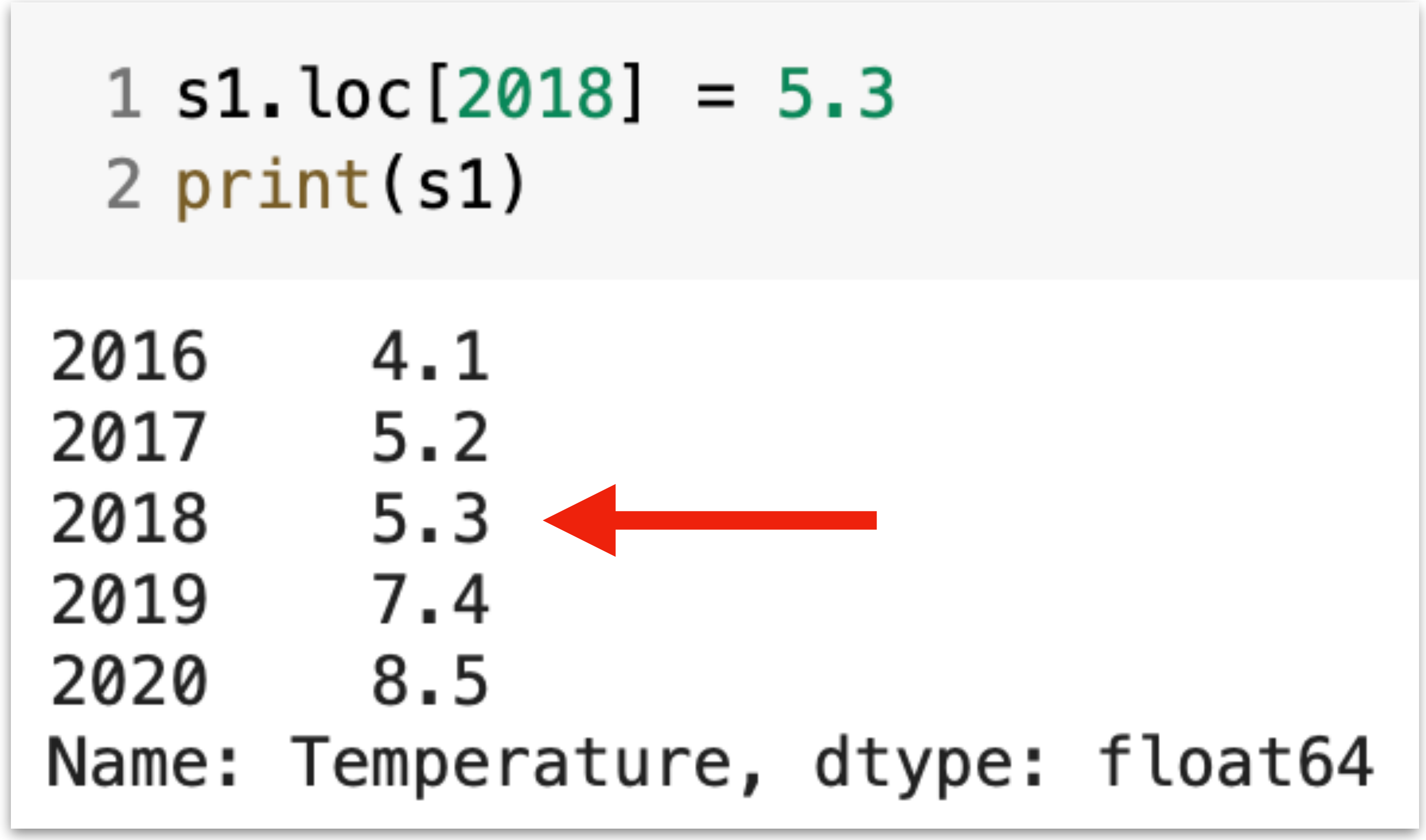
```
s1.loc[2018:2020]
```

gives a `Series` object

```
s1.loc[2018:2020].values
```

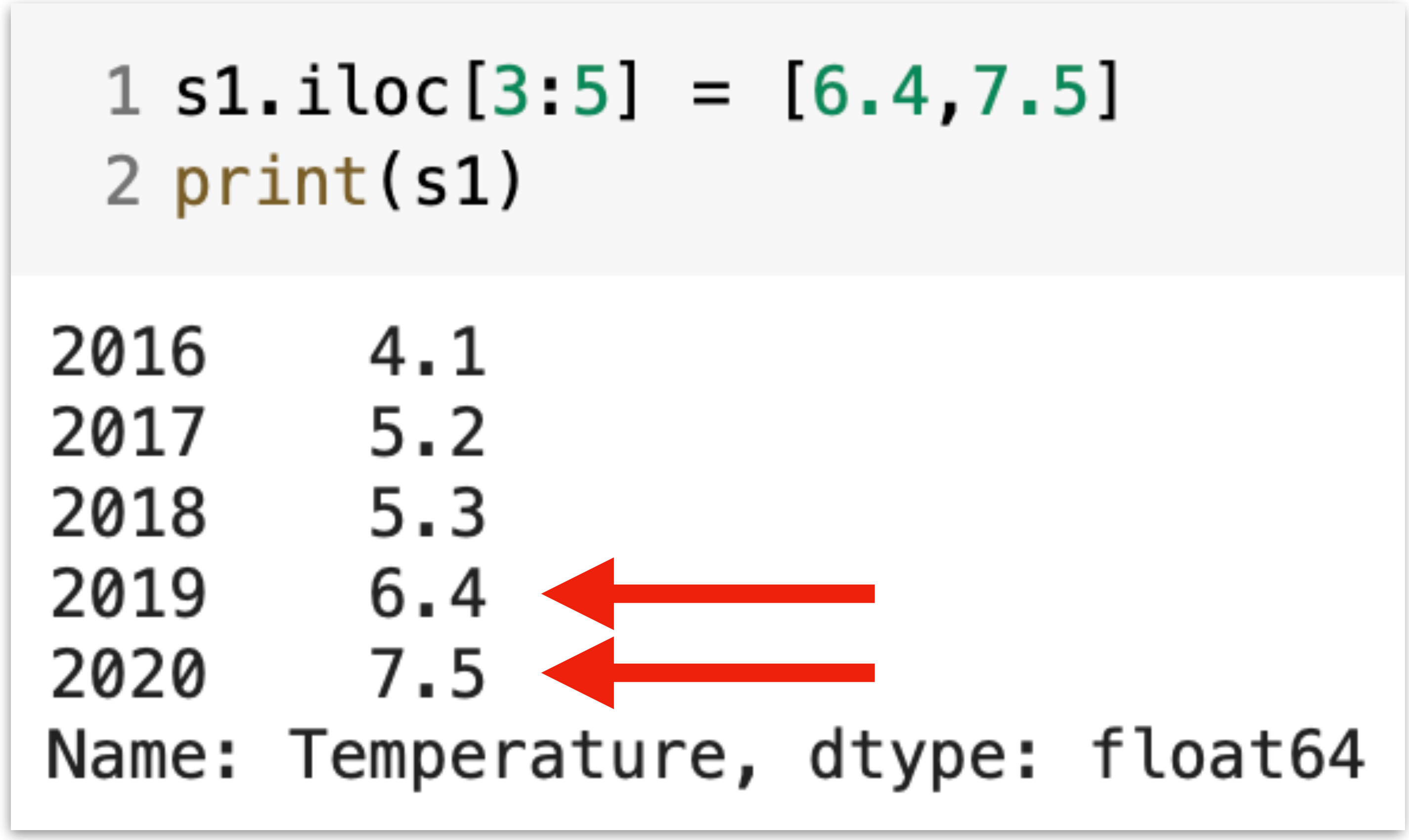
gives a NumPy array

Changing data in a `Series` using `.iloc[ ]` and `.loc[ ]`



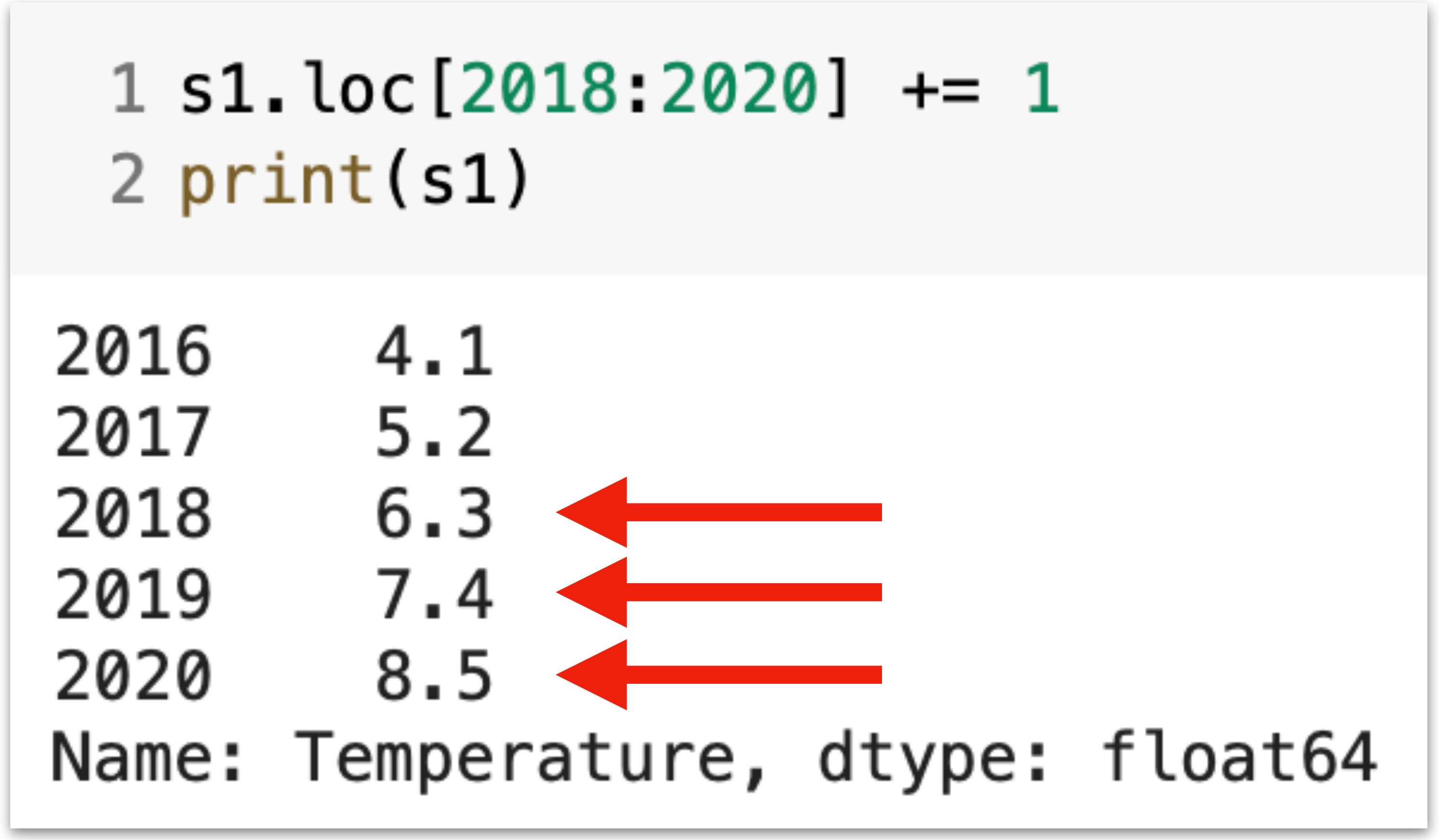
```python
s1.loc[2018] = 5.3
print(s1)
```

```
2016    4.1
2017    5.2
2018    5.3
2019    7.4
2020    8.5
Name: Temperature, dtype: float64
```



```python
s1.iloc[3:5] = [6.4,7.5]
print(s1)
```

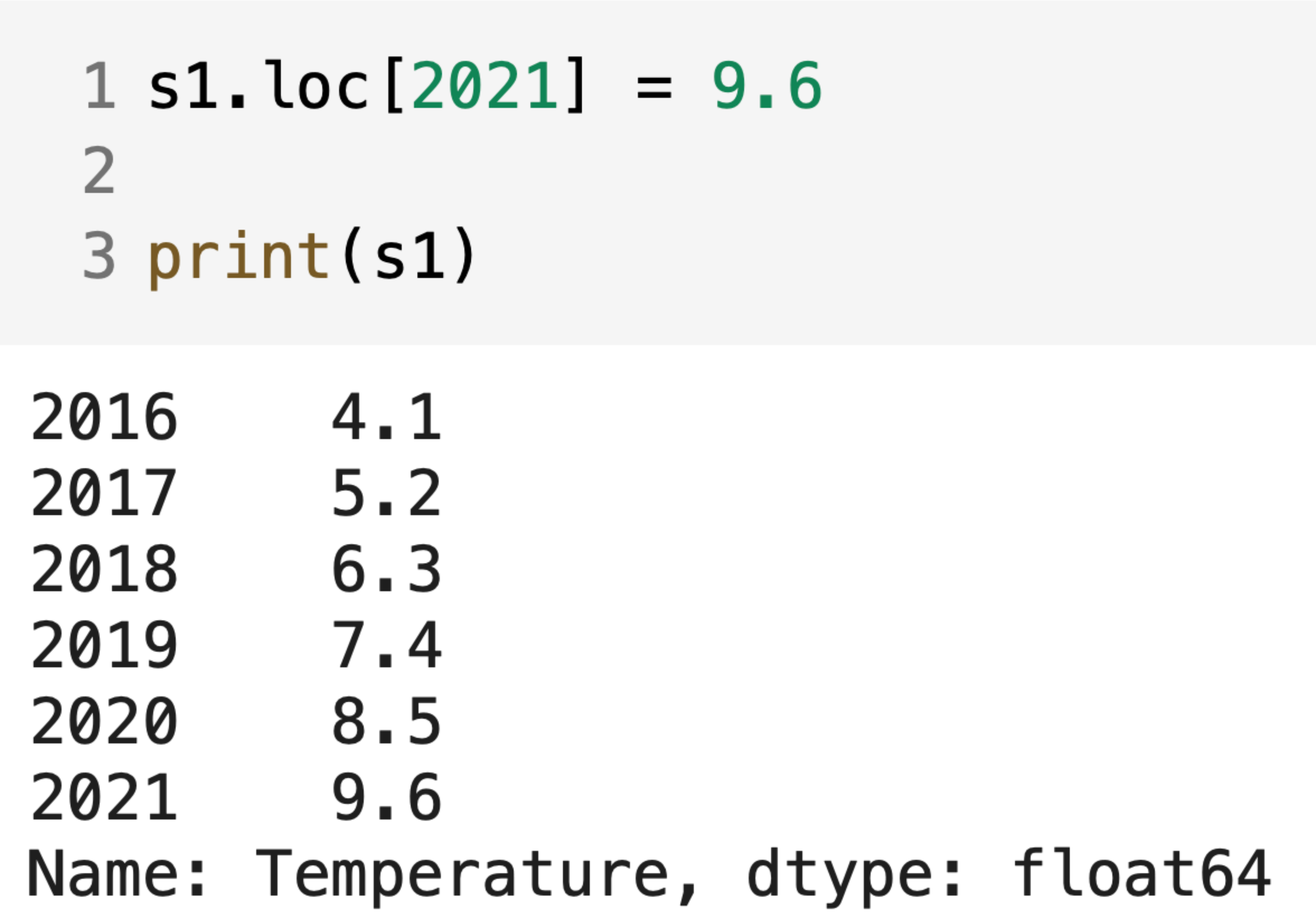
```
2016    4.1
2017    5.2
2018    5.3
2019    6.4
2020    7.5
Name: Temperature, dtype: float64
```



```python
s1.loc[2018:2020] += 1
print(s1)
```

```
2016    4.1
2017    5.2
2018    6.3
2019    7.4
2020    8.5
Name: Temperature, dtype: float64
```

Adding new data to a `Series` using `.loc[ ]` with a new label



```python
s1.loc[2021] = 9.6

print(s1)
```

```
2016    4.1
2017    5.2
2018    6.3
2019    7.4
2020    8.5
2021    9.6
Name: Temperature, dtype: float64
```

What we'll cover in this lesson

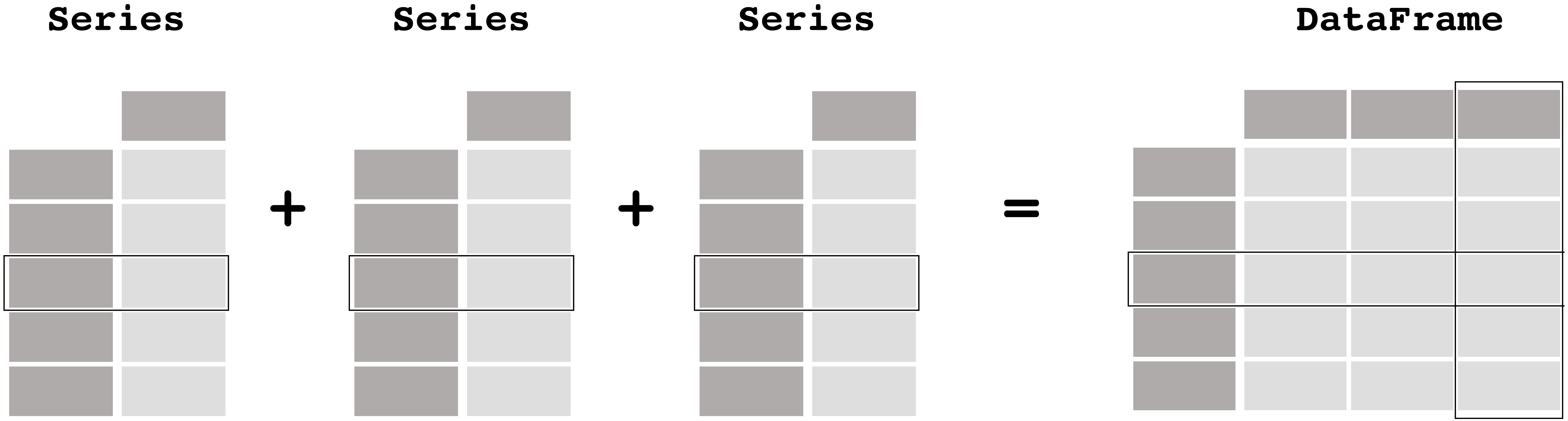
1. pandas: `Series` objects

2. **pandas: `DataFrame` objects; CSV files**

3. `xarray`: `DataArray` and `Dataset` objects; netCDF files

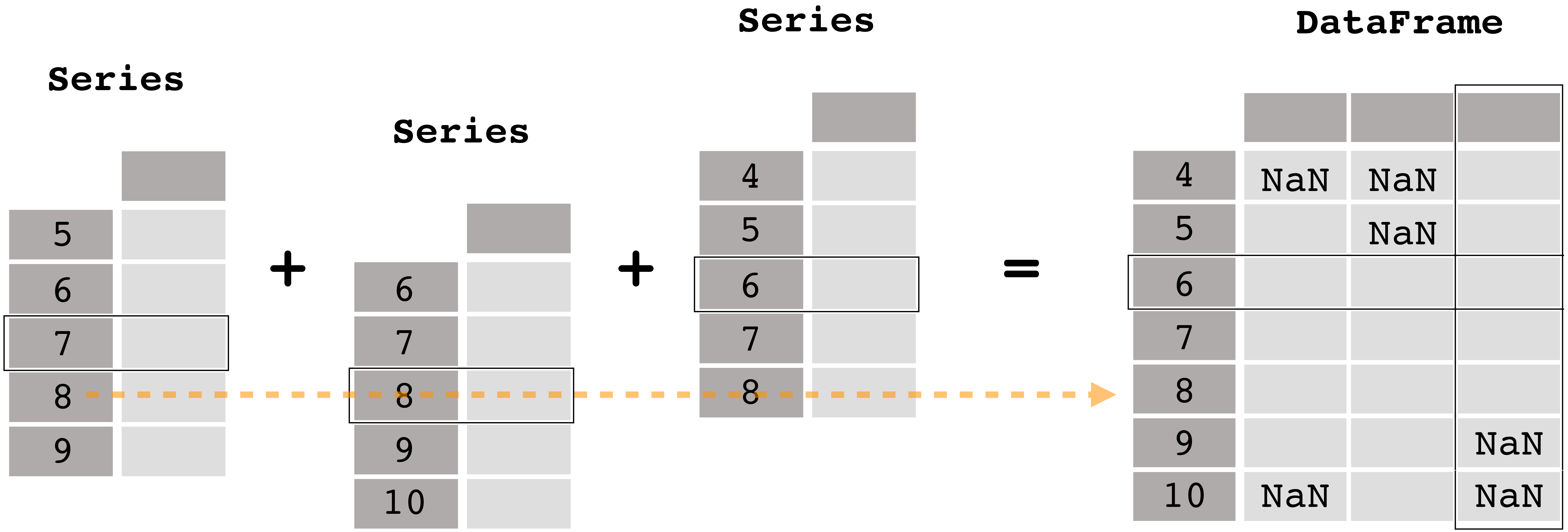
4. `xarray`: working with higher-dimensional data

Two or more `Series` can be concatenated to become a `DataFrame`



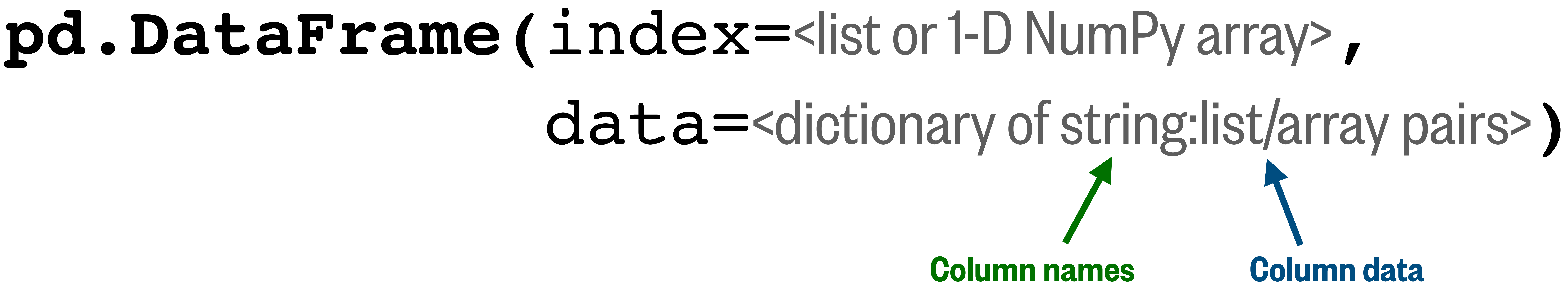
```
pd.concat([s1,s2,s3,…],axis=1)
```

Concatenation along columns respects the index values

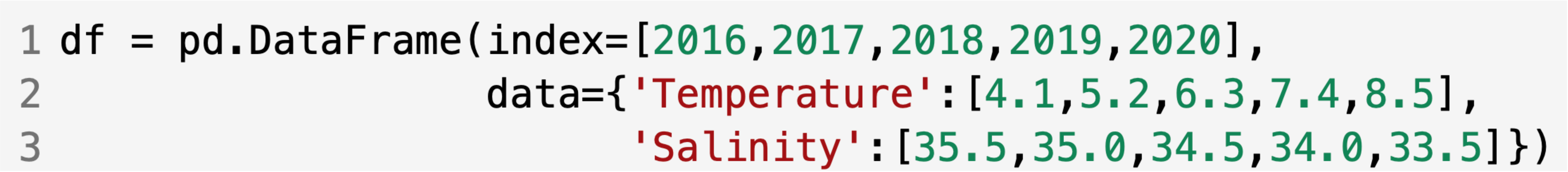


```
pd.concat([s1,s2,s3,…],axis=1,join='outer')
```

You can also create a new `DataFrame` object directly



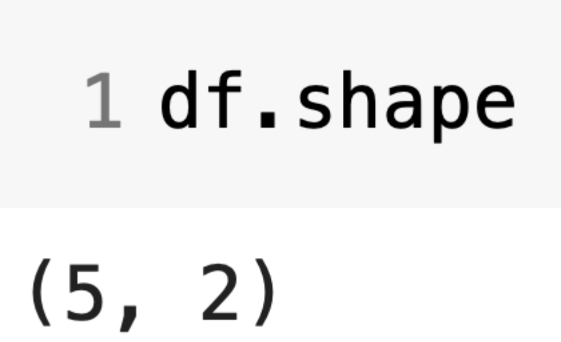
pd.DataFrame(index=<list or 1-D NumPy array>,
data=<dictionary of string:list/array pairs>**)**



```
df = pd.DataFrame(index=[2016,2017,2018,2019,2020],
                  data={'Temperature':[4.1,5.2,6.3,7.4,8.5],
                        'Salinity':[35.5,35.0,34.5,34.0,33.5]})
```

Getting information about a `DataFrame`

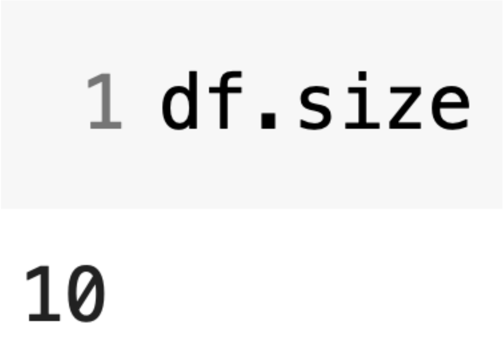
`.shape`



```
df.shape
```

```
(5, 2)
```

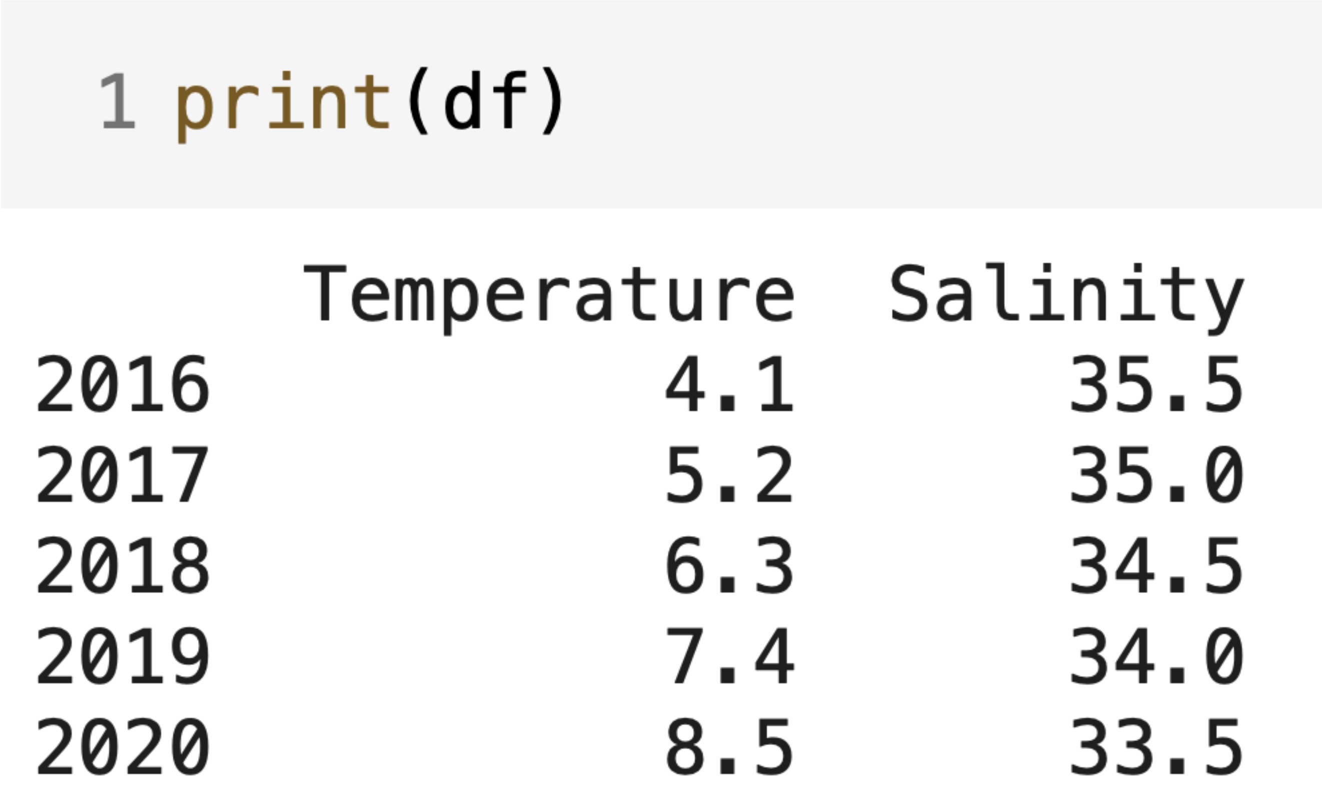
`.size`



```
df.size
```

```
10
```

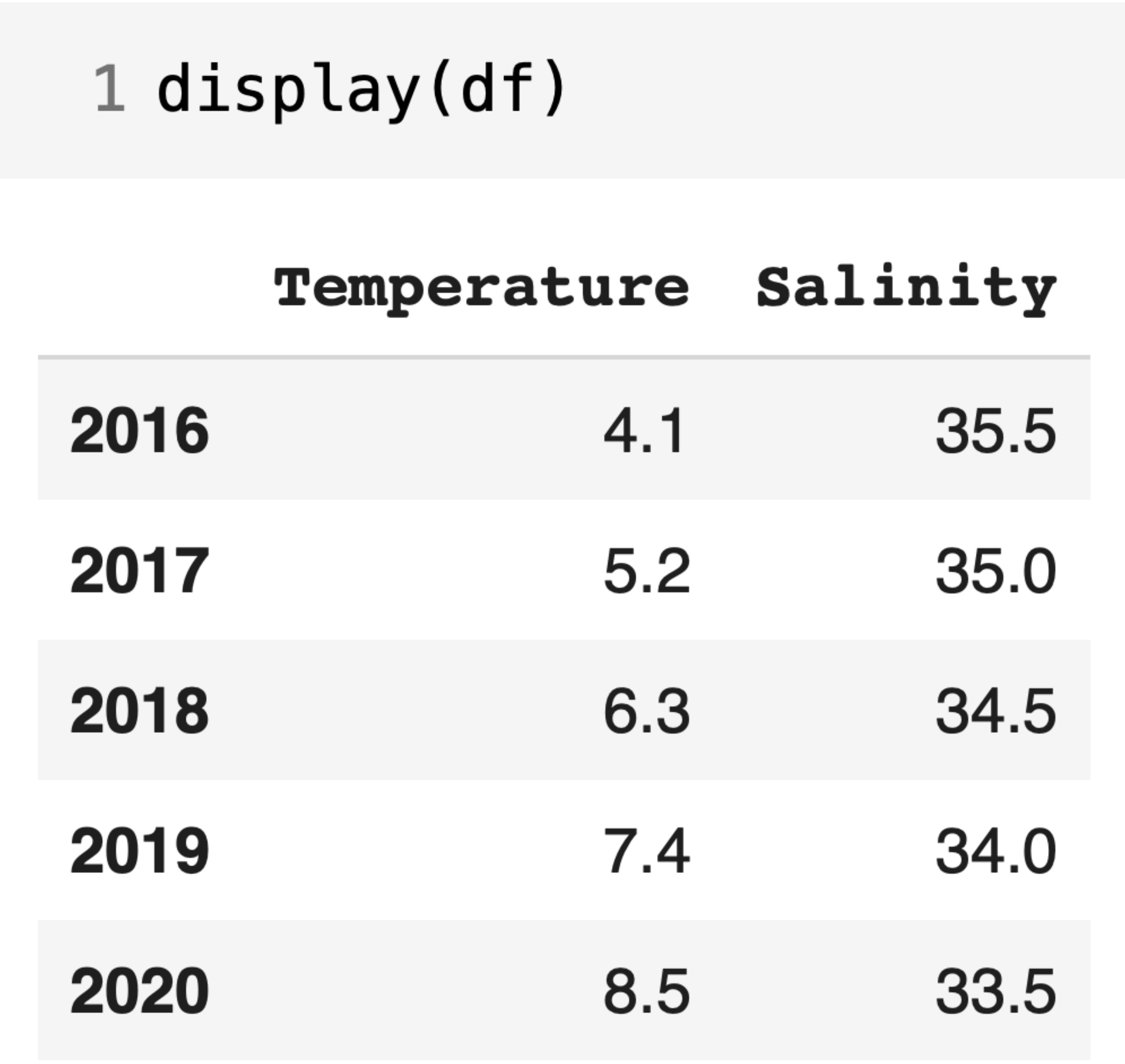
`print()`



```
print(df)
```

```
      Temperature  Salinity
2016          4.1      35.5
2017          5.2      35.0
2018          6.3      34.5
2019          7.4      34.0
2020          8.5      33.5
```

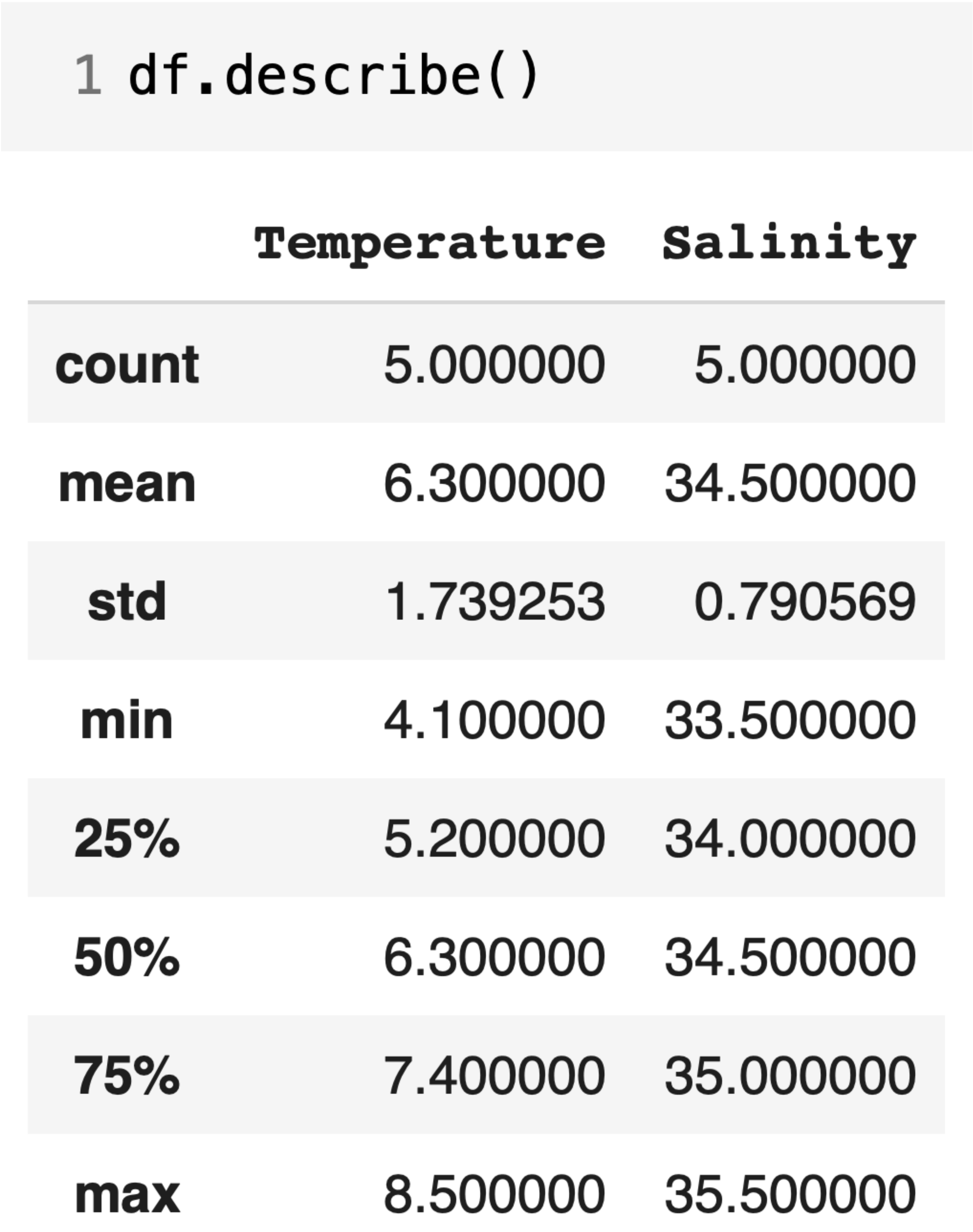
`display()`



```
display(df)
```

	Temperature	Salinity
2016	4.1	35.5
2017	5.2	35.0
2018	6.3	34.5
2019	7.4	34.0
2020	8.5	33.5

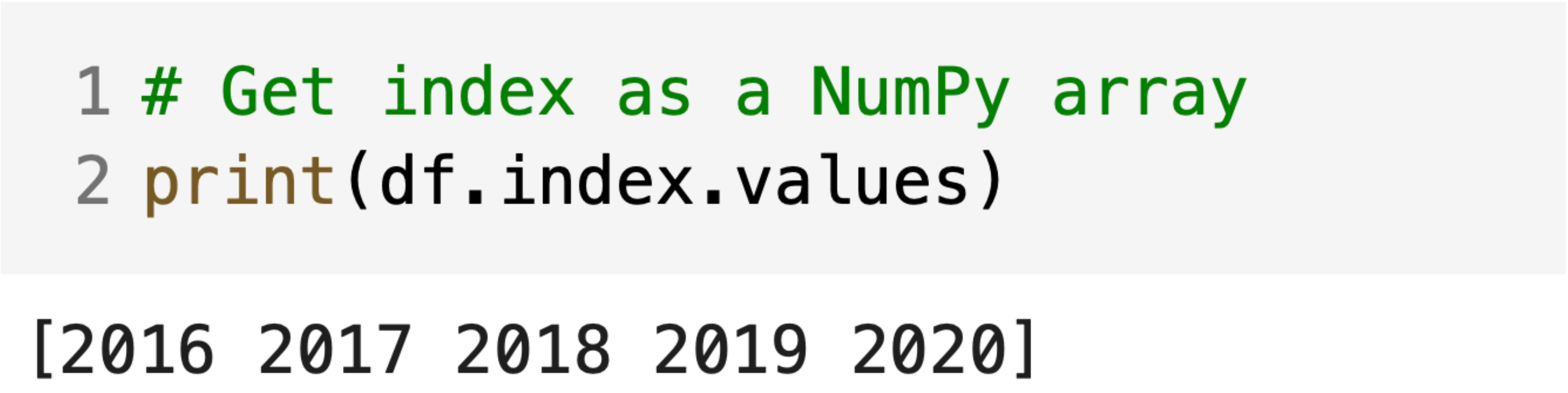
`.describe()`



```
df.describe()
```

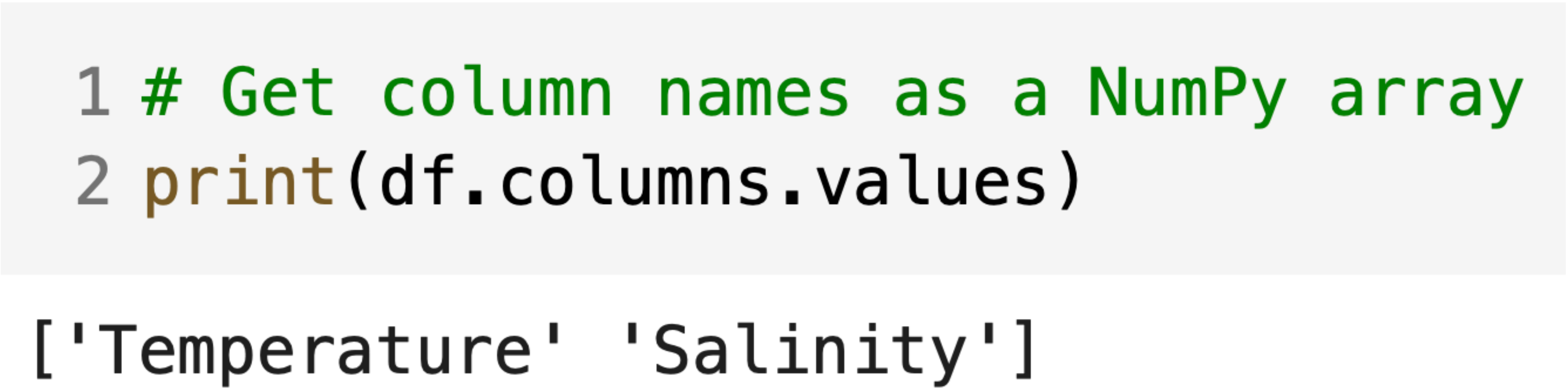
	Temperature	Salinity
count	5.000000	5.000000
mean	6.300000	34.500000
std	1.739253	0.790569
min	4.100000	33.500000
25%	5.200000	34.000000
50%	6.300000	34.500000
75%	7.400000	35.000000
max	8.500000	35.500000

Getting the columns, index, and data from a `DataFrame`



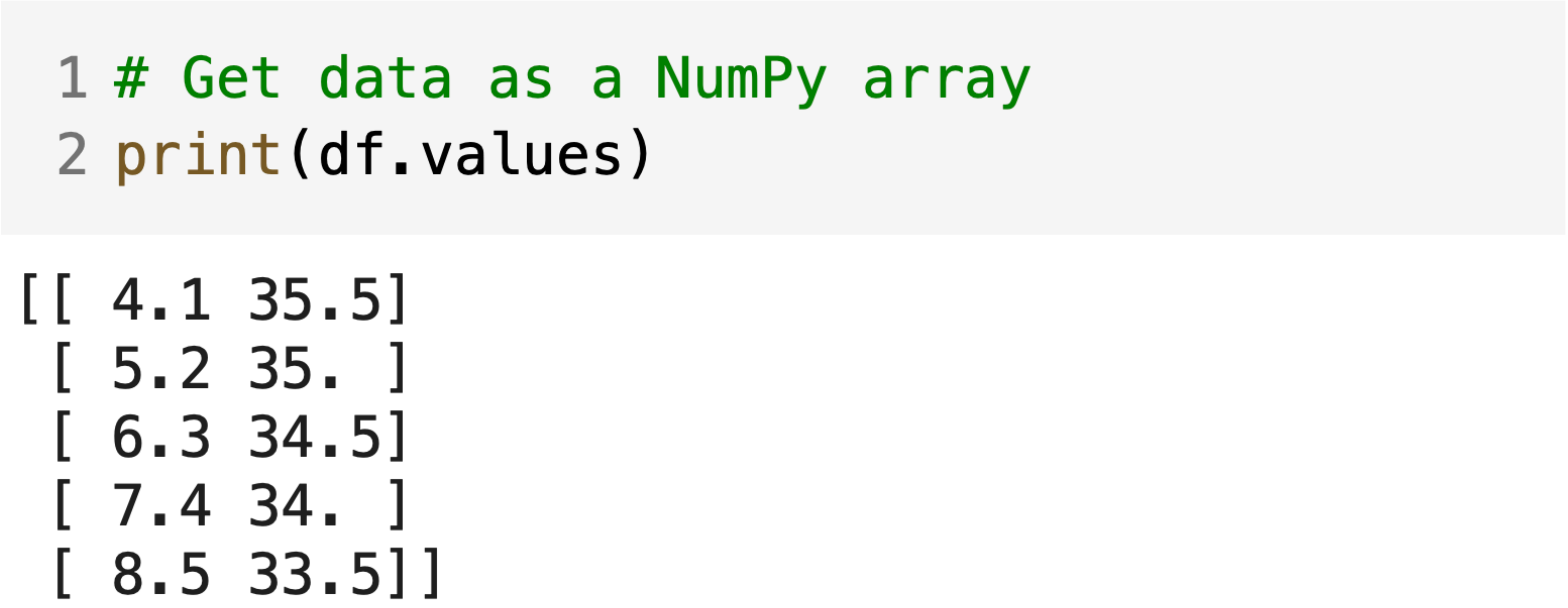
```python
# Get index as a NumPy array
print(df.index.values)
```

```
[2016 2017 2018 2019 2020]
```



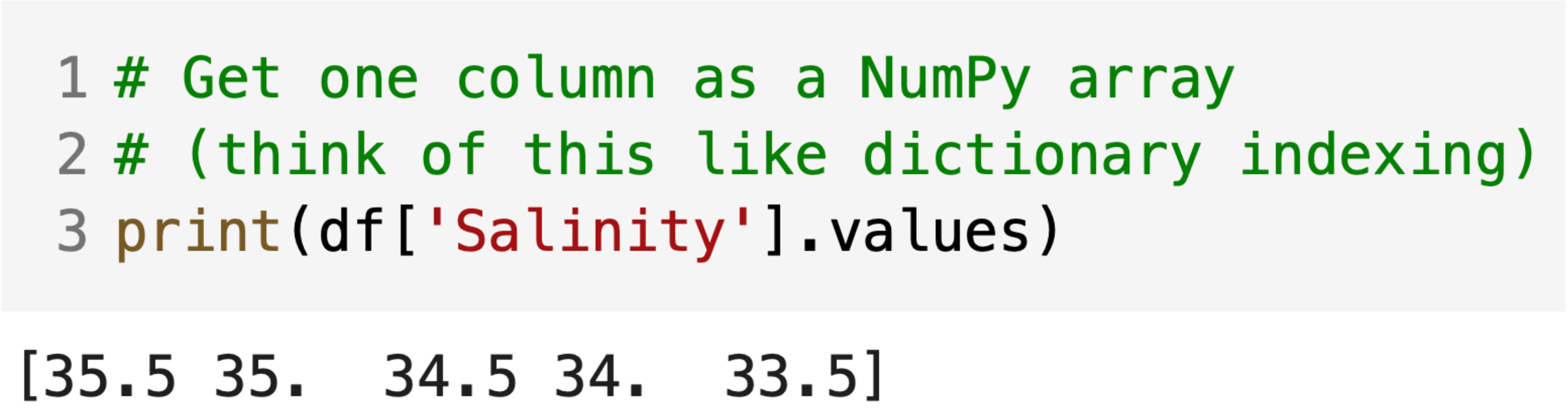
```python
# Get column names as a NumPy array
print(df.columns.values)
```

```
['Temperature' 'Salinity']
```



```python
# Get data as a NumPy array
print(df.values)
```

```
[[ 4.1 35.5]
 [ 5.2 35. ]
 [ 6.3 34.5]
 [ 7.4 34. ]
 [ 8.5 33.5]]
```



```python
# Get one column as a NumPy array
# (think of this like dictionary indexing)
print(df['Salinity'].values)
```

```
[35.5 35.  34.5 34.  33.5]
```

Selecting data from a `DataFrame` using `.iloc[ ]` and `.loc[ ]`

Selection by index: <DataFrame>**`.iloc[`** <single integer index>
OR <list or array of indices>
OR <slice of integer indices>
OR <Boolean array> **`]`**

Selection by label: <DataFrame>**`.loc[`** <single index label>
OR <list or array of labels>
OR <slice of index labels> **`]`**

Selecting data from a `DataFrame` using `.iloc[ ]` and `.loc[ ]`

Selection by index:

<DataFrame> **[** <column label(s)> **]** **`.iloc`** **[** <index or indices> **]**

Selection by label:

<DataFrame> **[** <column label(s)> **]** **`.loc`** **[** <label or labels> **]**

Example: `df['Salinity'].loc[2019]`

Applying NumPy functions to a `Series` or `DataFrame`

```
df.mean()
df.mean(axis=0)
```

both take the average along the index (axis 0)

Example:

```
Temperature      6.3
Salinity        34.5
dtype: float64
```

```
df.mean(axis=1)
```

takes the average along the columns (axis 1)

Example:

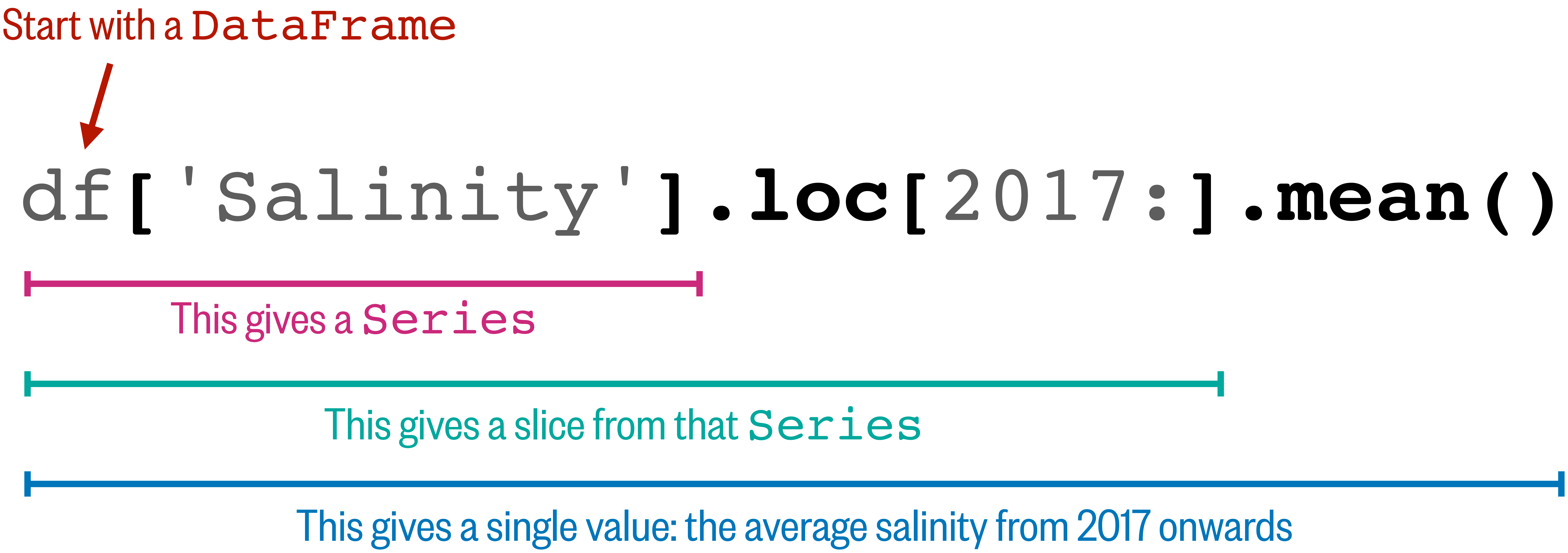
```
2016     19.8
2017     20.1
2018     20.4
2019     20.7
2020     21.0
dtype: float64
```

```
df.mean(skipna=True)
```

ignores NaN values (if present) when calculating the average

Putting it all together

Combine column extraction, selection by label, and applying a NumPy function



Start with a DataFrame

```
df['Salinity'].loc[2017:].mean()
```

This gives a Series

This gives a slice from that Series

This gives a single value: the average salinity from 2017 onwards

Philosophy of `pandas` and `xarray`

Do more with less code.

Benefit: You'll spend more time "doing science" and less time writing code.

Make your code more readable.

Benefit: You'll make fewer errors, and it will be easier to understand what you were thinking when you revisit your code a few weeks or months later.

Philosophy of `pandas` and `xarray`

Which code is easiest to understand?

`for` loop:

```python
sum = 0.0
for index in range(len(data)):
  if times[index].year == 2019:
    sum += data[index]
average = sum / len(data)
```

NumPy:

```python
data[np.logical_and(times > datetime(2019,1,1),
                    times < datetime(2019,12,31))].mean()
```

`pandas`:

```python
data.loc['2019'].mean()
```

Shortcut for indexing into a `datetime` index

Loading/saving CSV and Excel files using `pandas`

Save a `DataFrame` as a CSV file:

```
df.to_csv('filepath/including/filename.csv')
```

Read a CSV file as a `DataFrame`:

```
df = pd.read_csv('filepath/including/filename.csv',
                delimiter=',',delim_whitespace=False,
                header=0, …)
```

→ Documentation (API): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Read an Excel spreadsheet as a `DataFrame`:

```
df = pd.read_excel('filepath/including/filename.xlsx',
                   sheet_name='Sheet 1', …)
```

→ Documentation (API): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html

Resources: `pandas` documentation

“Getting started” tutorials:

https://pandas.pydata.org/docs/getting_started/intro_tutorials/

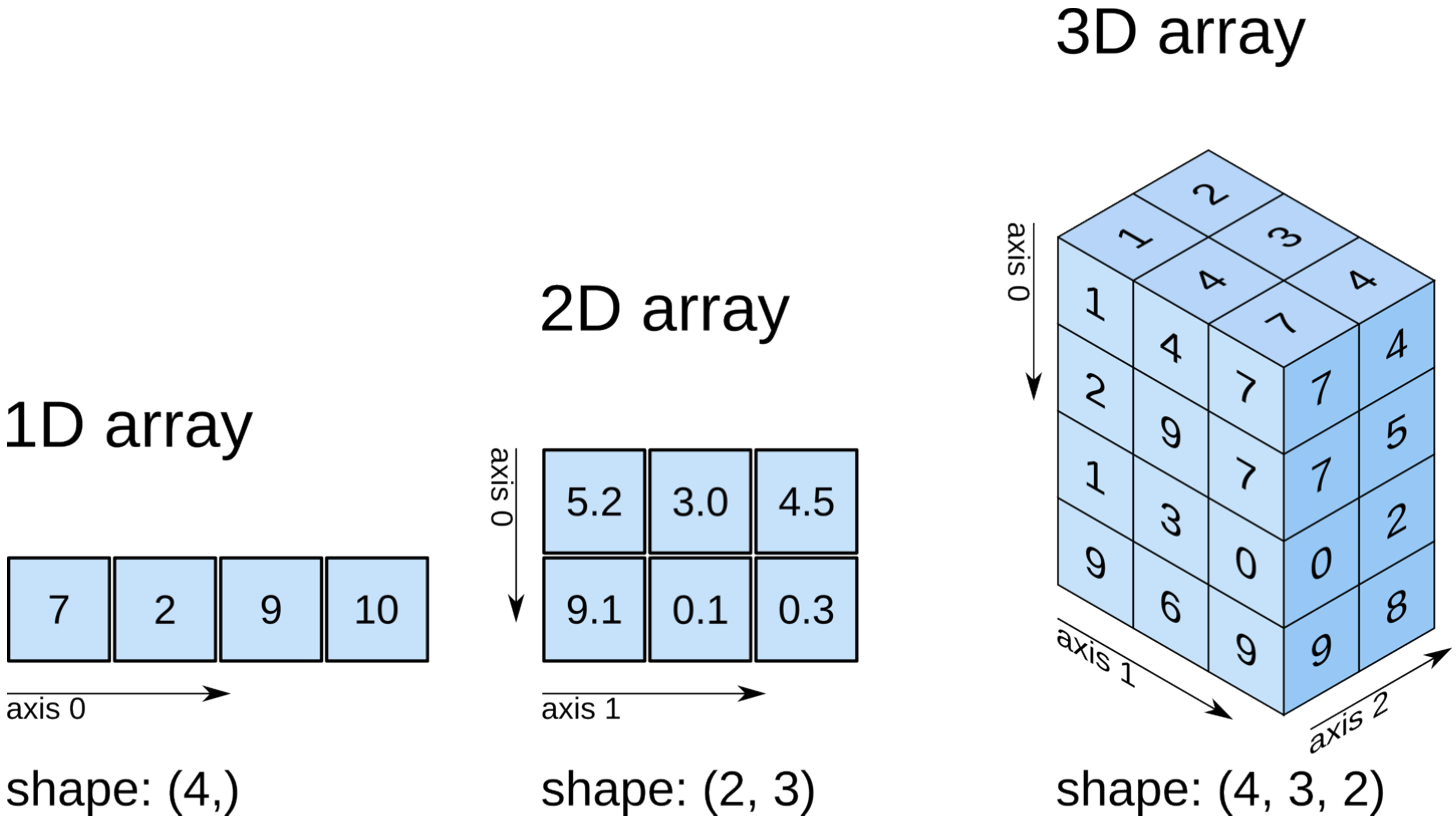
Full user guide:

https://pandas.pydata.org/docs/user_guide/index.html

What we'll cover in this lesson

1. pandas: `Series` objects
2. `pandas`: `DataFrame` objects; CSV files
3. **`xarray`: `DataArray` and `Dataset` objects; netCDF files**
4. `xarray`: working with higher-dimensional data

Array values are identified by their coordinates along axes



Source: O'Reilly

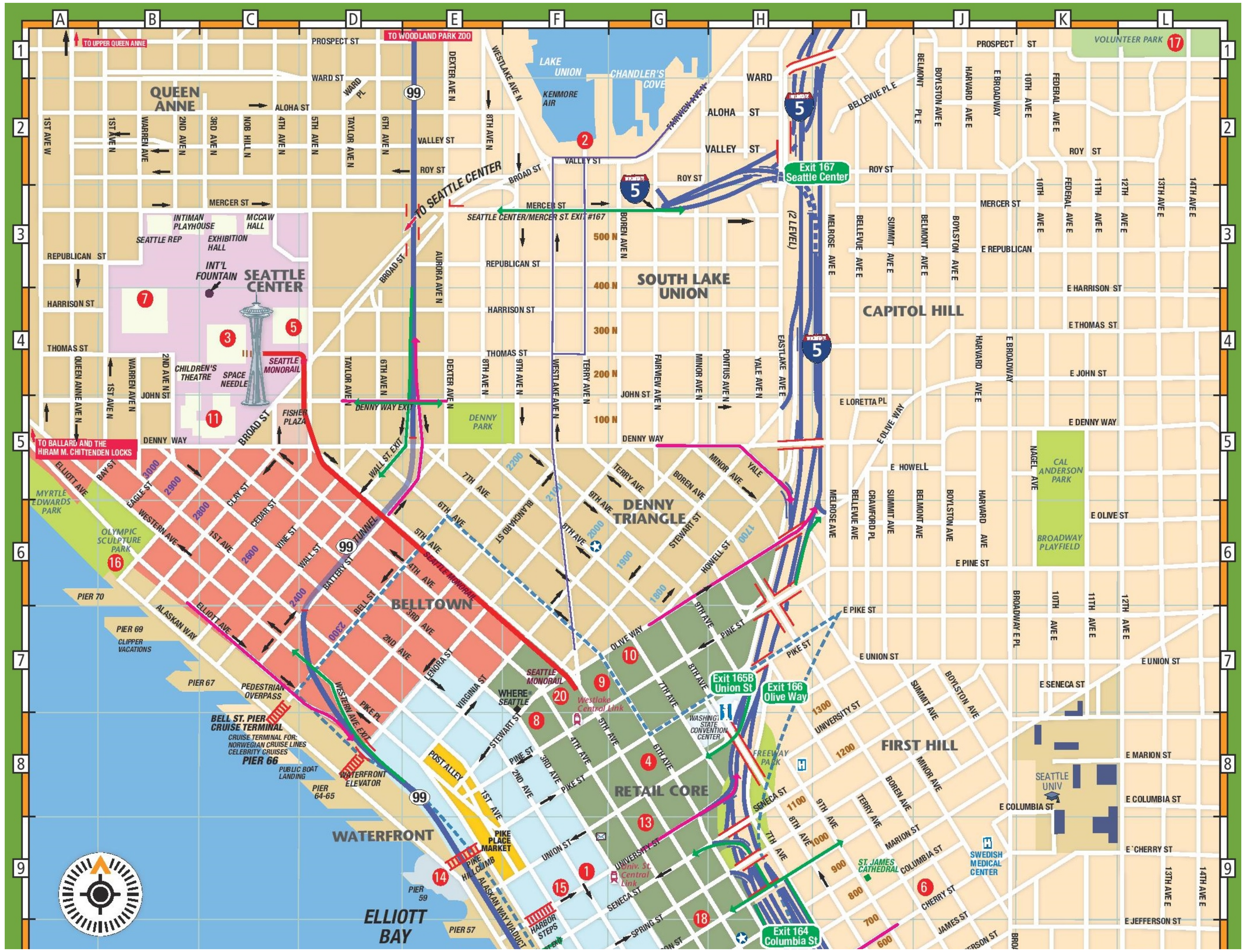
Coordinates along axes let us identify real-world locations

W ⟵⟶ E

N

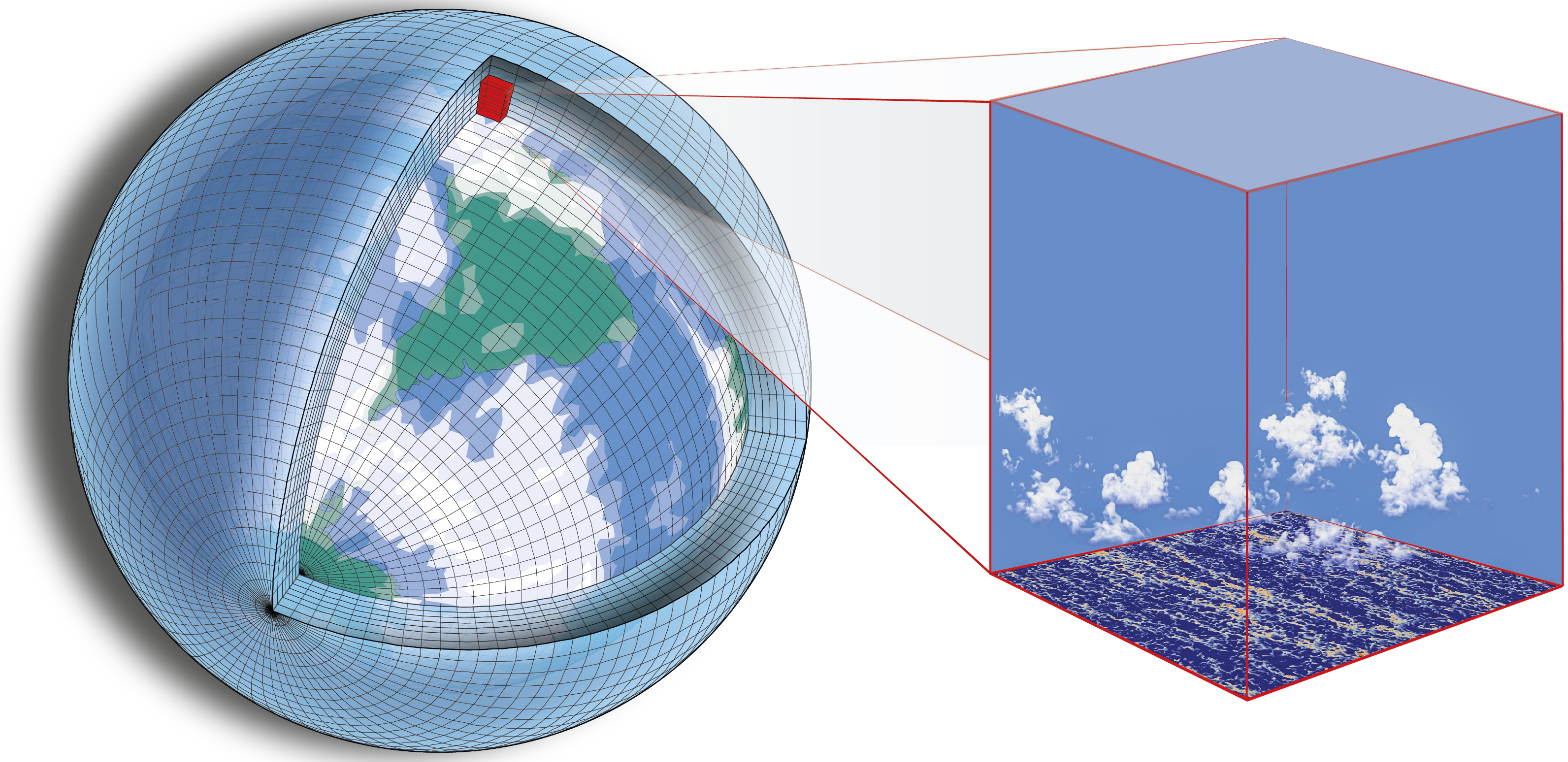
↕

S



Source: Where Maps

Models, satellites, and data providers divide the world into grid cells



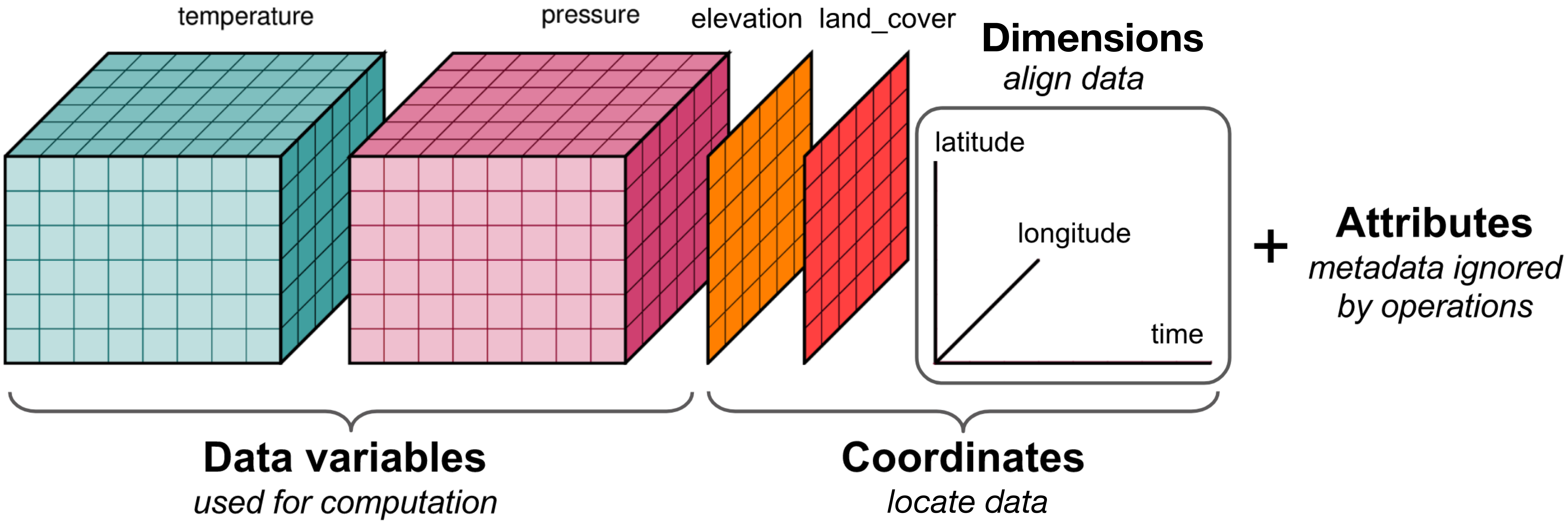
Source: Caltech

`xarray` lets us deal with gridded data...

... and gridded data is usually provided in a **netCDF file (.nc)**

`xarray` lets us deal with gridded data...

... and gridded data is usually provided in a **netCDF file (.nc)**



Dimensions
align data

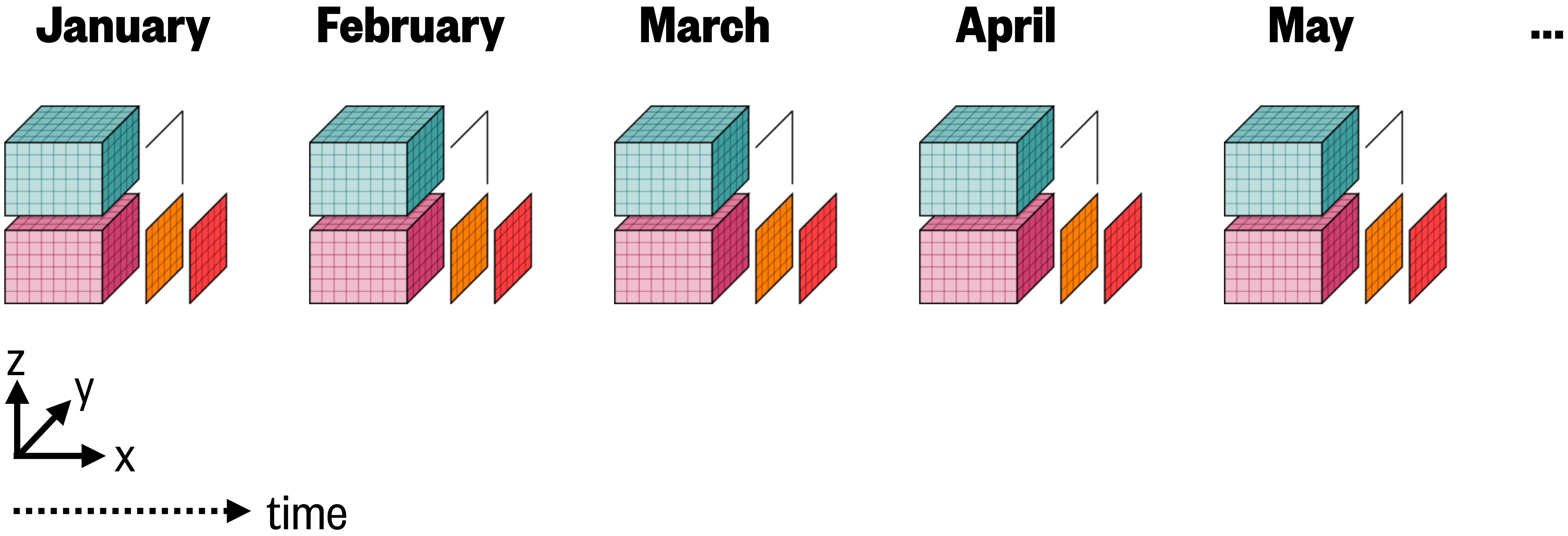
+ Attributes
metadata ignored by operations

Data variables
used for computation

Coordinates
locate data

Source: Matthew Rocklin

4-D data is usually 3-D in space (x, y, z) + time



January February March April May ...

Source: xarray

Importing `xarray` and loading the `netCDF4` library

```
import xarray as xr

!pip install netcdf4
```

You should only have to run
this line once per Colab notebook.

Importing `xarray` and loading the `netCDF4` library

```
import xarray as xr

#!pip install netcdf4
```

You should only have to run this line once per Colab notebook.

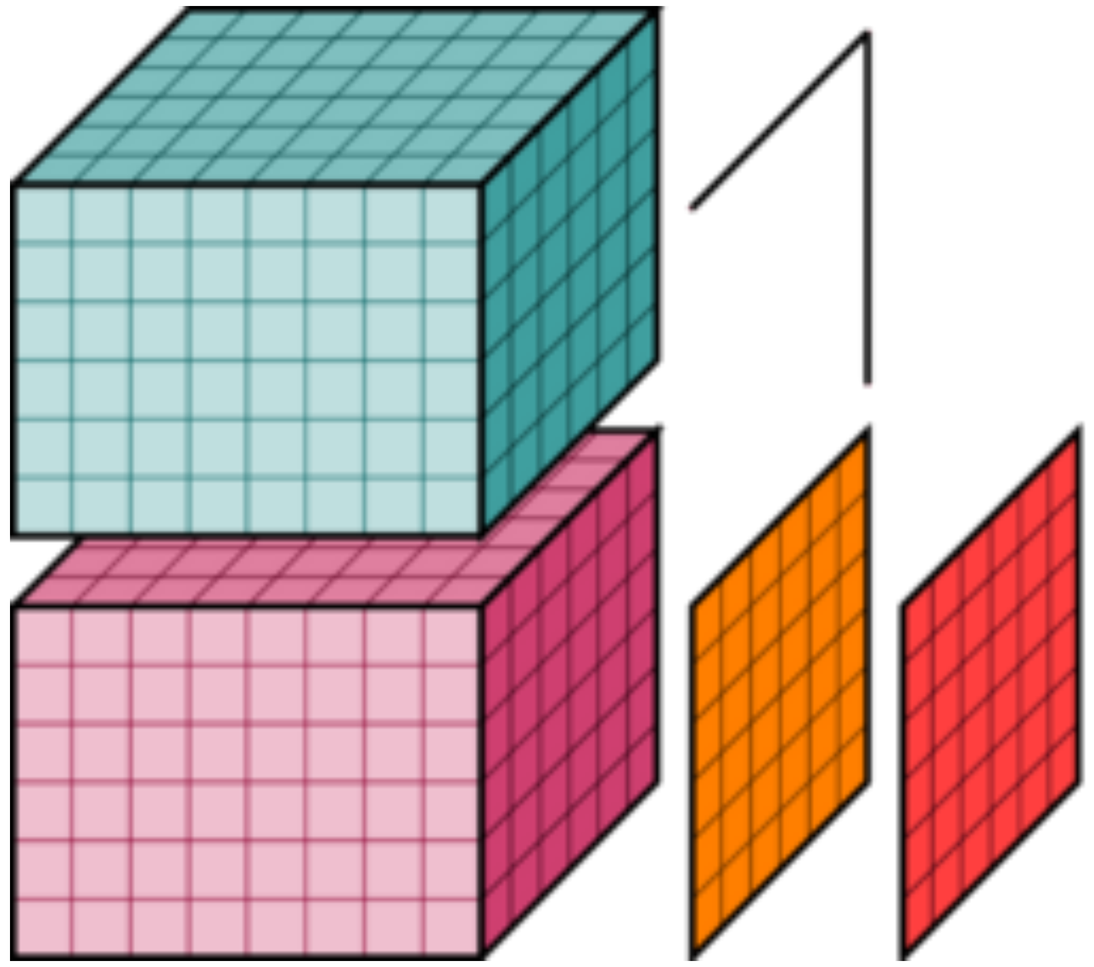
Loading a netCDF file as an `xarray Dataset`

Read a netCDF file as a `Dataset`:

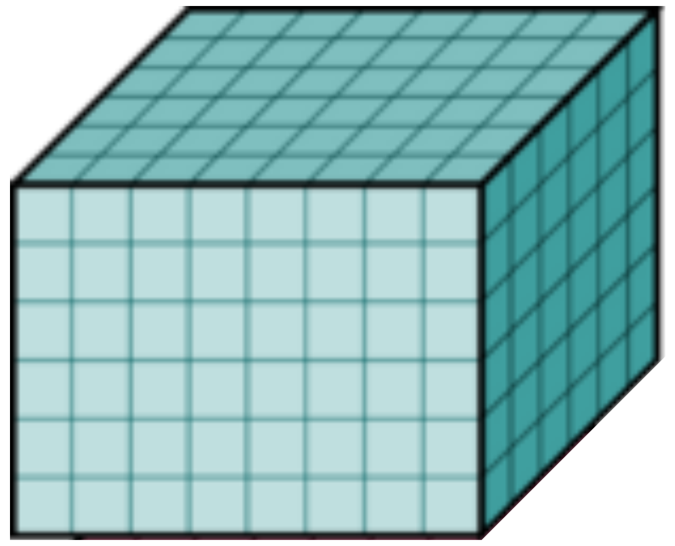
```
data = xr.open_dataset('filepath/including/filename.nc')
```

→ Documentation (API): http://xarray.pydata.org/en/stable/generated/xarray.open_dataset.html

`Dataset object:`



`DataArray object:`



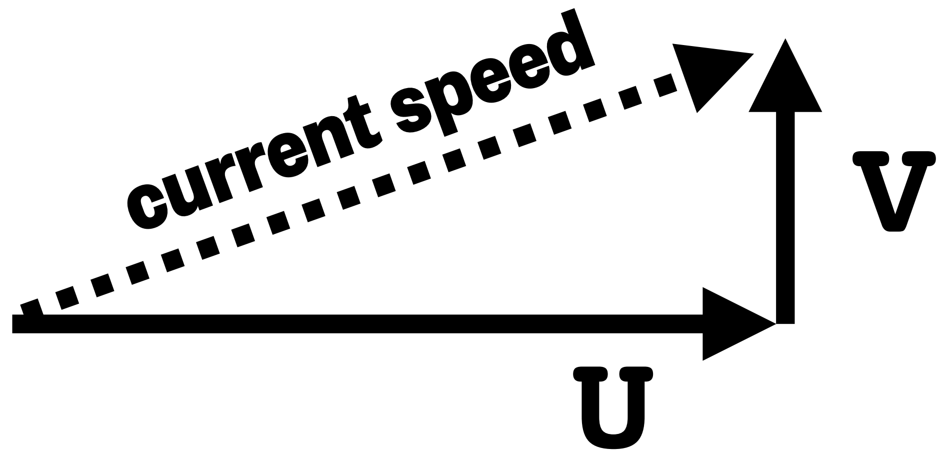
Demo: Southern Ocean current velocities from a climate model

File (~400 MB):
`bsose_monthly_velocities.nc`

Data source:
B-SOSE (Southern Ocean State Estimate) model output

Data resolution:
Time: monthly for 2012
Horizontal: 1/3° lat-lon grid
Vertical: 13 depth levels

Variables:
`U`: eastward velocity
`V`: northward velocity



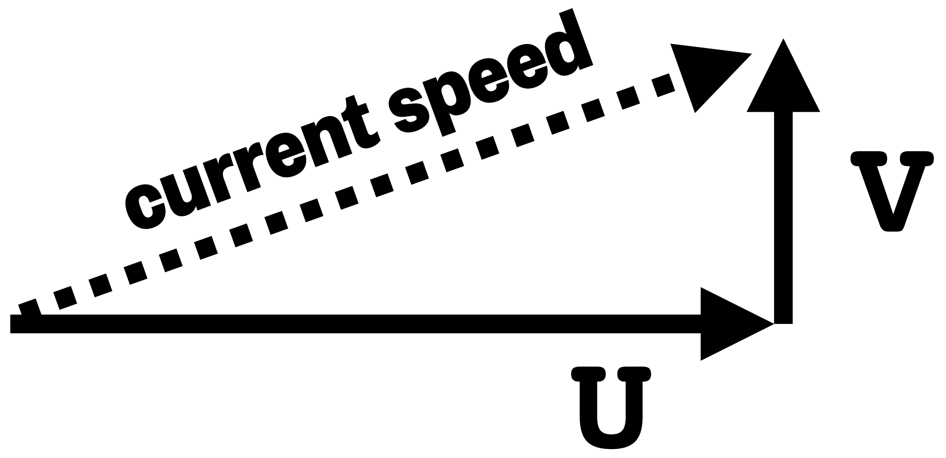
Demo: Southern Ocean current velocities from a climate model

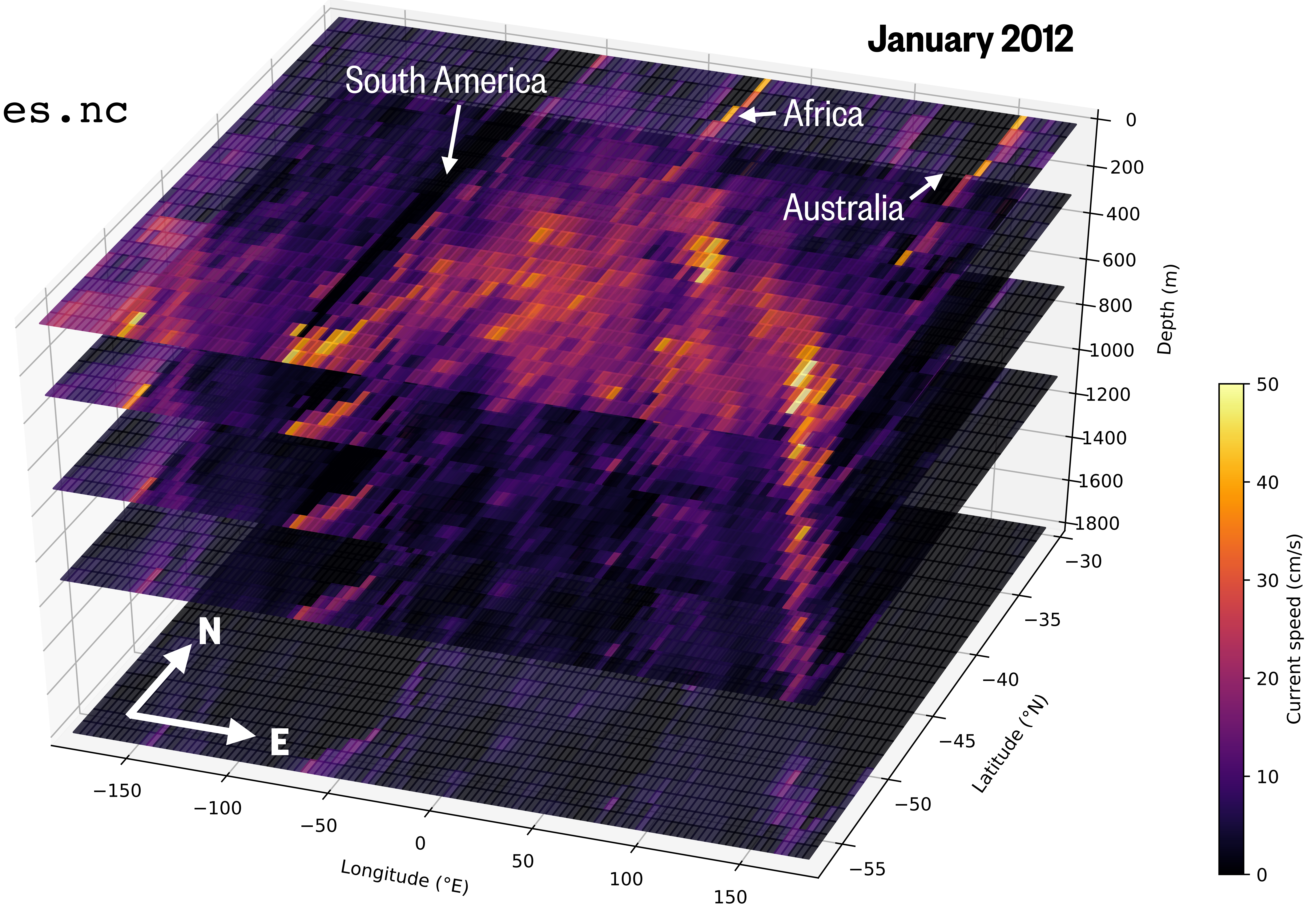
File (~400 MB):
`bsose_monthly_velocities.nc`

Data source:
B-SOSE (Southern Ocean State Estimate) model output

Data resolution:
Time: monthly for 2012
Horizontal: 1/3° lat-lon grid
Vertical: 13 depth levels

Variables:
U: eastward velocity
V: northward velocity





What we'll cover in this lesson

1. pandas: `Series` objects

2. `pandas`: `DataFrame` objects; CSV files

3. `xarray`: `DataArray` and `Dataset` objects; netCDF files

4. **`xarray`: working with higher-dimensional data**

Demo: Southern Ocean current velocities from a climate model

File (~400 MB):

`bsose_monthly_velocities.nc`

Data source:

B-SOSE (Southern Ocean State Estimate) model output

Data resolution:

Time: monthly for 2012

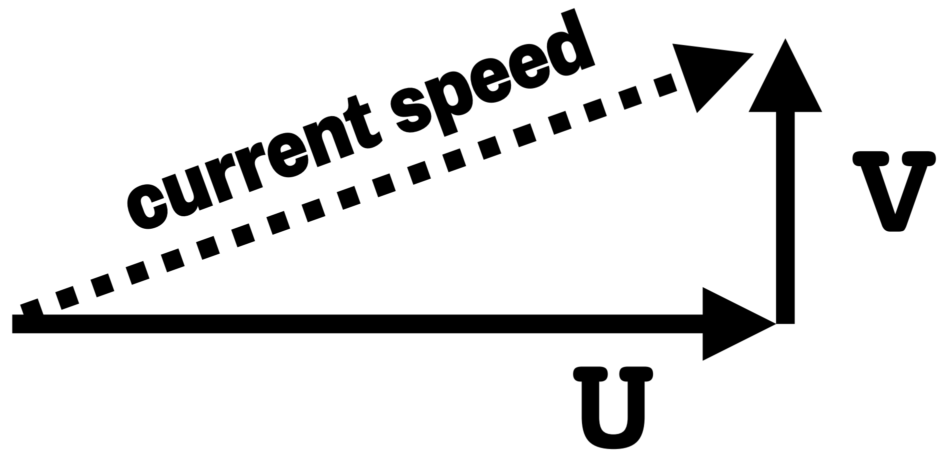
Horizontal: 1/3° lat-lon grid

Vertical: 13 depth levels

Variables:

U: eastward velocity

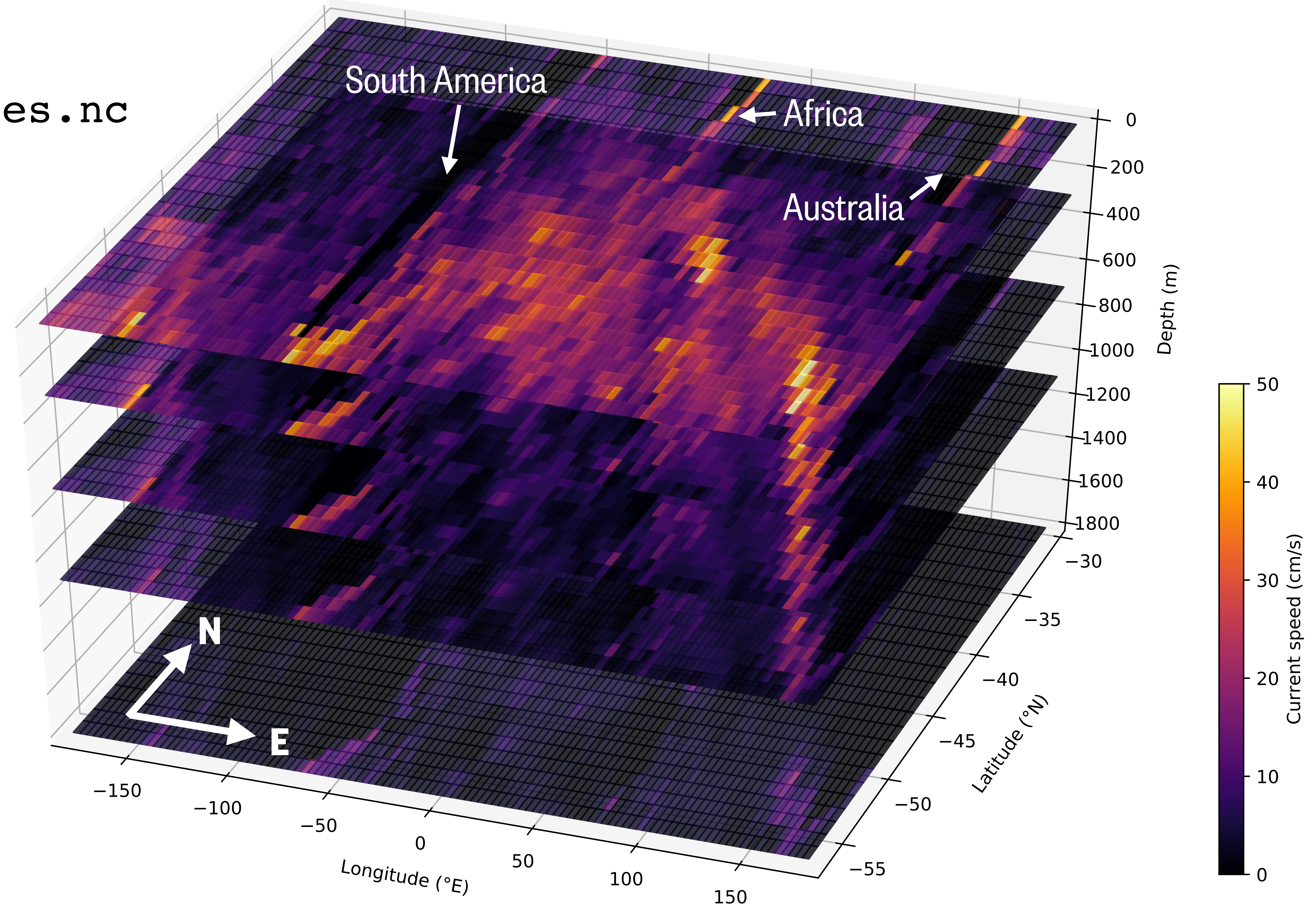
V: northward velocity



current speed

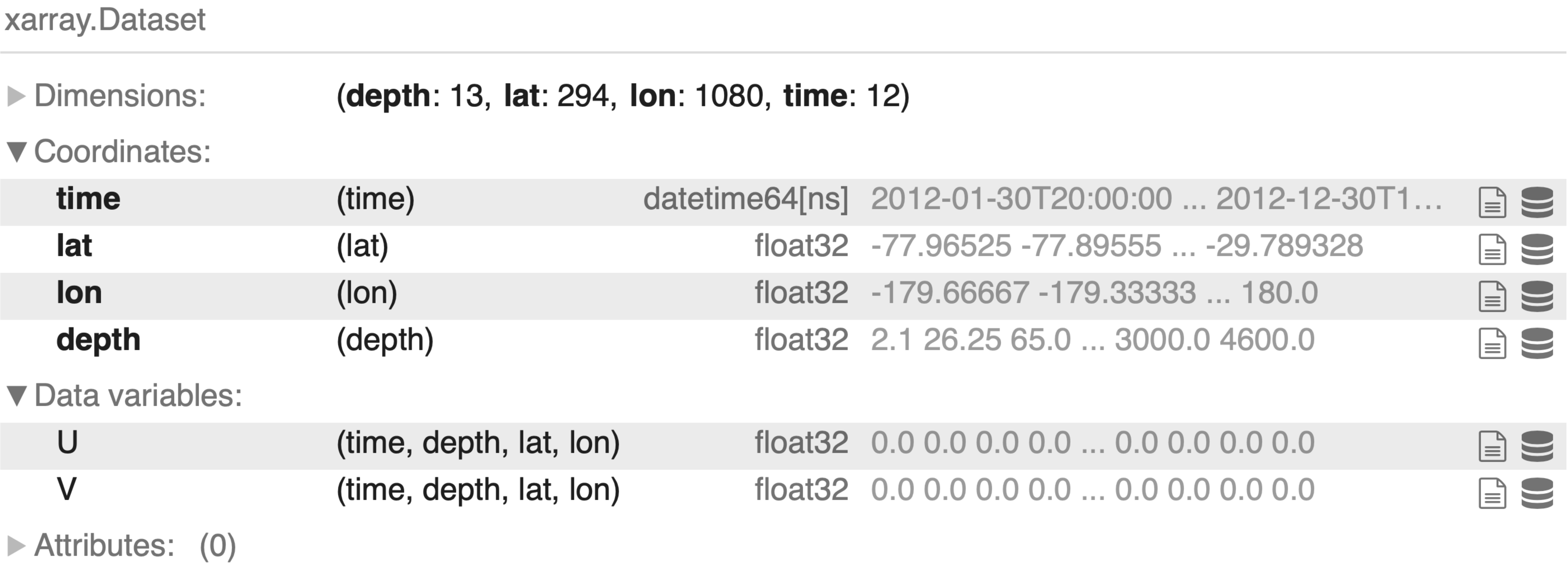
V

U



Getting information about a `Dataset`

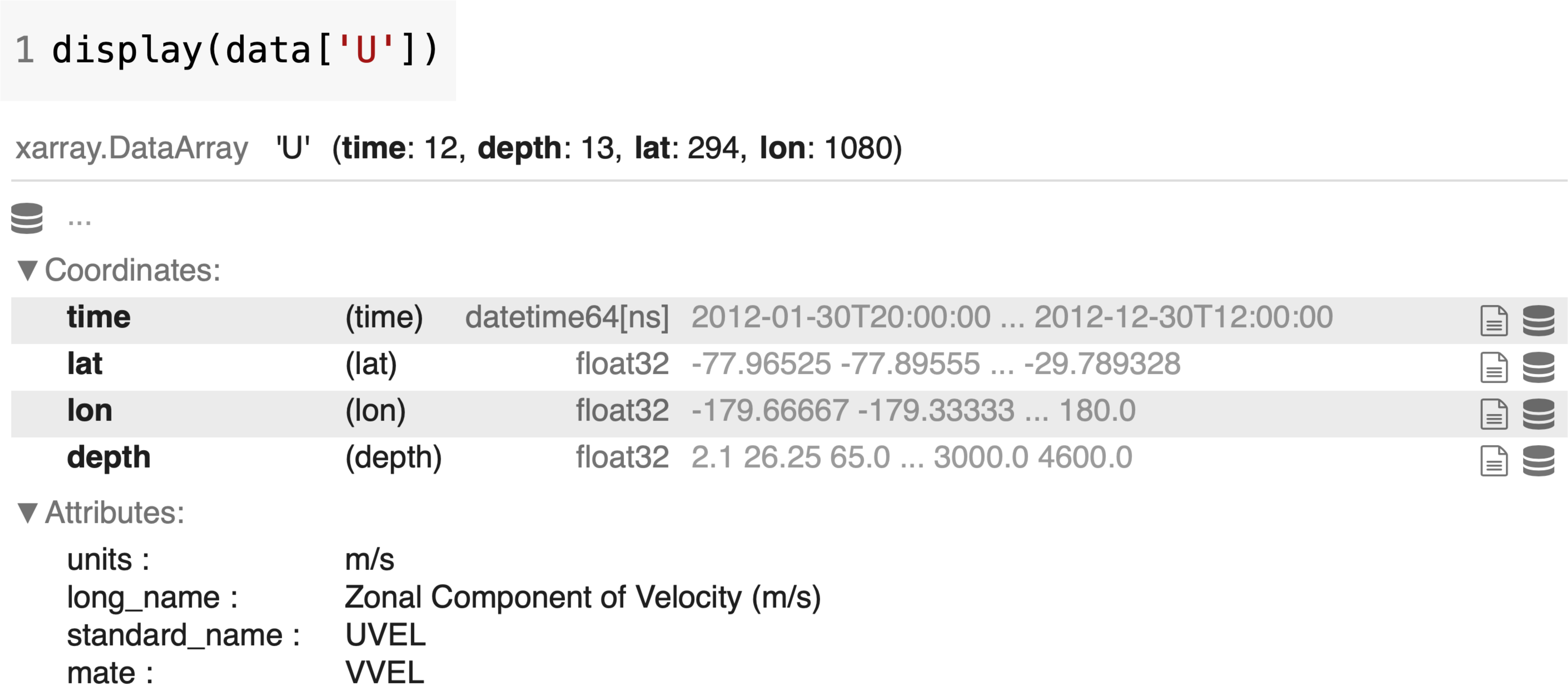
`display(`<Dataset variable>**`)`**



```
xarray.Dataset

▶ Dimensions:        (depth: 13, lat: 294, lon: 1080, time: 12)
▼ Coordinates:
    time             (time)                        datetime64[ns]  2012-01-30T20:00:00 ... 2012-12-30T1…
    lat              (lat)                                float32  -77.96525 -77.89555 ... -29.789328
    lon              (lon)                                float32  -179.66667 -179.33333 ... 180.0
    depth            (depth)                              float32  2.1 26.25 65.0 ... 3000.0 4600.0
▼ Data variables:
    U                (time, depth, lat, lon)              float32  0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0
    V                (time, depth, lat, lon)              float32  0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0
▶ Attributes:   (0)
```

Extracting a single variable's `DataArray` from a `Dataset`

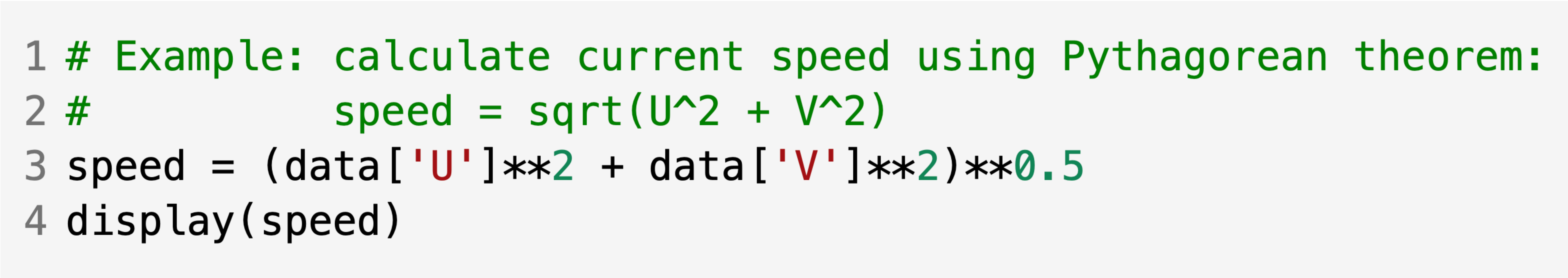
Syntax: <Dataset> **[** <variable name as a string> **]**



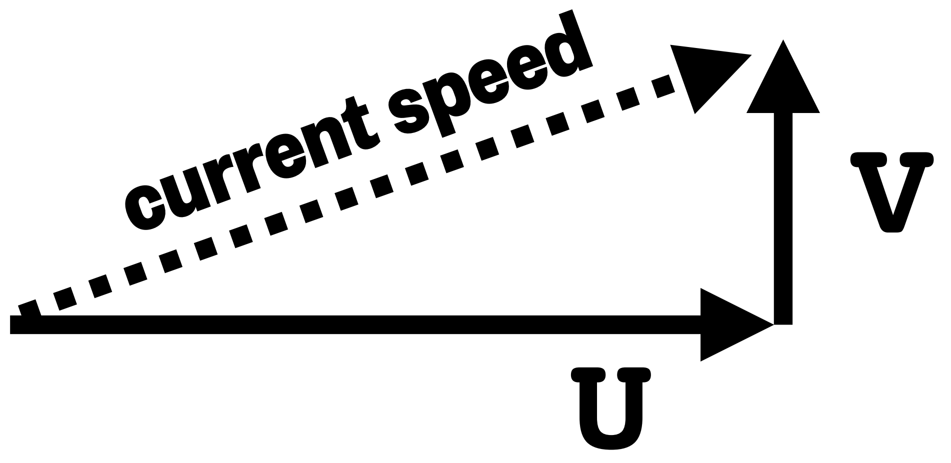
```python
display(data['U'])
```

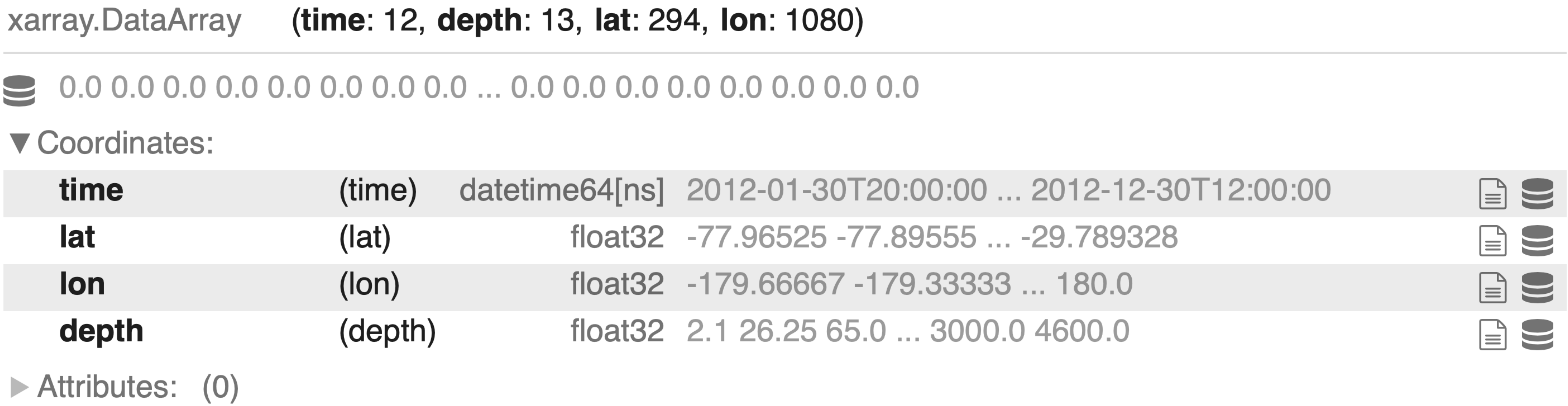
```
xarray.DataArray   'U'   (time: 12, depth: 13, lat: 294, lon: 1080)

...

▼ Coordinates:
    time        (time)    datetime64[ns]  2012-01-30T20:00:00 ... 2012-12-30T12:00:00
    lat         (lat)            float32  -77.96525 -77.89555 ... -29.789328
    lon         (lon)            float32  -179.66667 -179.33333 ... 180.0
    depth       (depth)          float32  2.1 26.25 65.0 ... 3000.0 4600.0
▼ Attributes:
    units :            m/s
    long_name :        Zonal Component of Velocity (m/s)
    standard_name :    UVEL
    mate :             VVEL
```

Mathematical calculations with `xarray` objects works like NumPy



```python
# Example: calculate current speed using Pythagorean theorem:
#             speed = sqrt(U^2 + V^2)
speed = (data['U']**2 + data['V']**2)**0.5
display(speed)
```

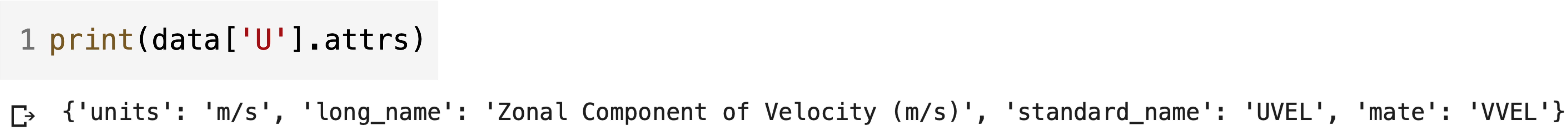




```
xarray.DataArray    (time: 12, depth: 13, lat: 294, lon: 1080)

0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

▼ Coordinates:
   time     (time)    datetime64[ns]  2012-01-30T20:00:00 ... 2012-12-30T12:00:00
   lat      (lat)            float32  -77.96525 -77.89555 ... -29.789328
   lon      (lon)            float32  -179.66667 -179.33333 ... 180.0
   depth    (depth)          float32  2.1 26.25 65.0 ... 3000.0 4600.0

▶ Attributes:   (0)
```

Accessing and changing attributes (metadata) of `xarray` objects

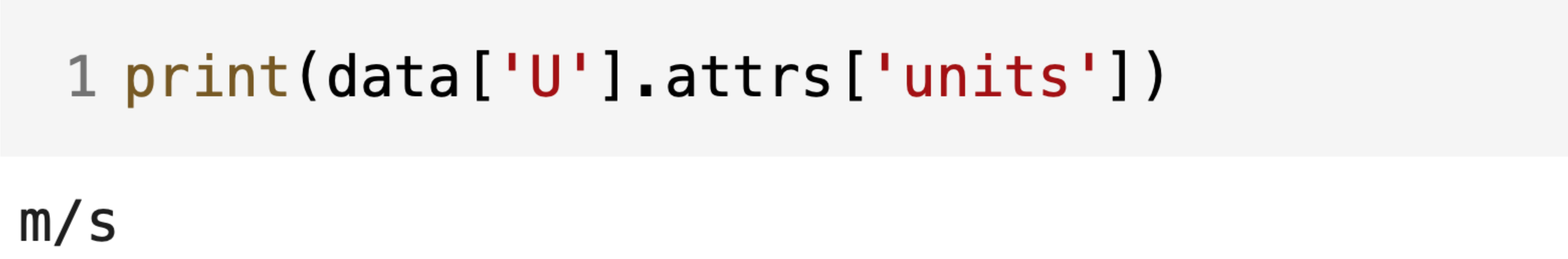
Syntax: <Dataset or DataArray>**`.attrs`** is a Python dictionary (a set of keys and values)



```
print(data['U'].attrs)
```

```
{'units': 'm/s', 'long_name': 'Zonal Component of Velocity (m/s)', 'standard_name': 'UVEL', 'mate': 'VVEL'}
```

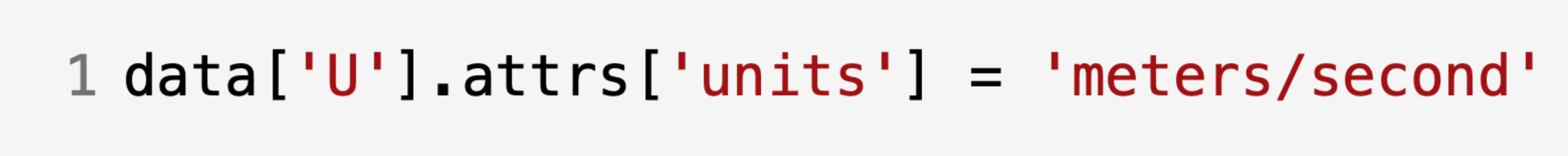
Syntax: <Dataset or DataArray>**`.attrs[`**<attribute name as string>**`]`** gets the value of an attribute



```
print(data['U'].attrs['units'])
```

```
m/s
```

Syntax: <Dataset or DataArray>**`.attrs[`**<attribute name>**`]`** **`=`** <new value> changes its value



```
data['U'].attrs['units'] = 'meters/second'
```

Selecting data from `xarray` objects using `.isel()` (selection by integer index)

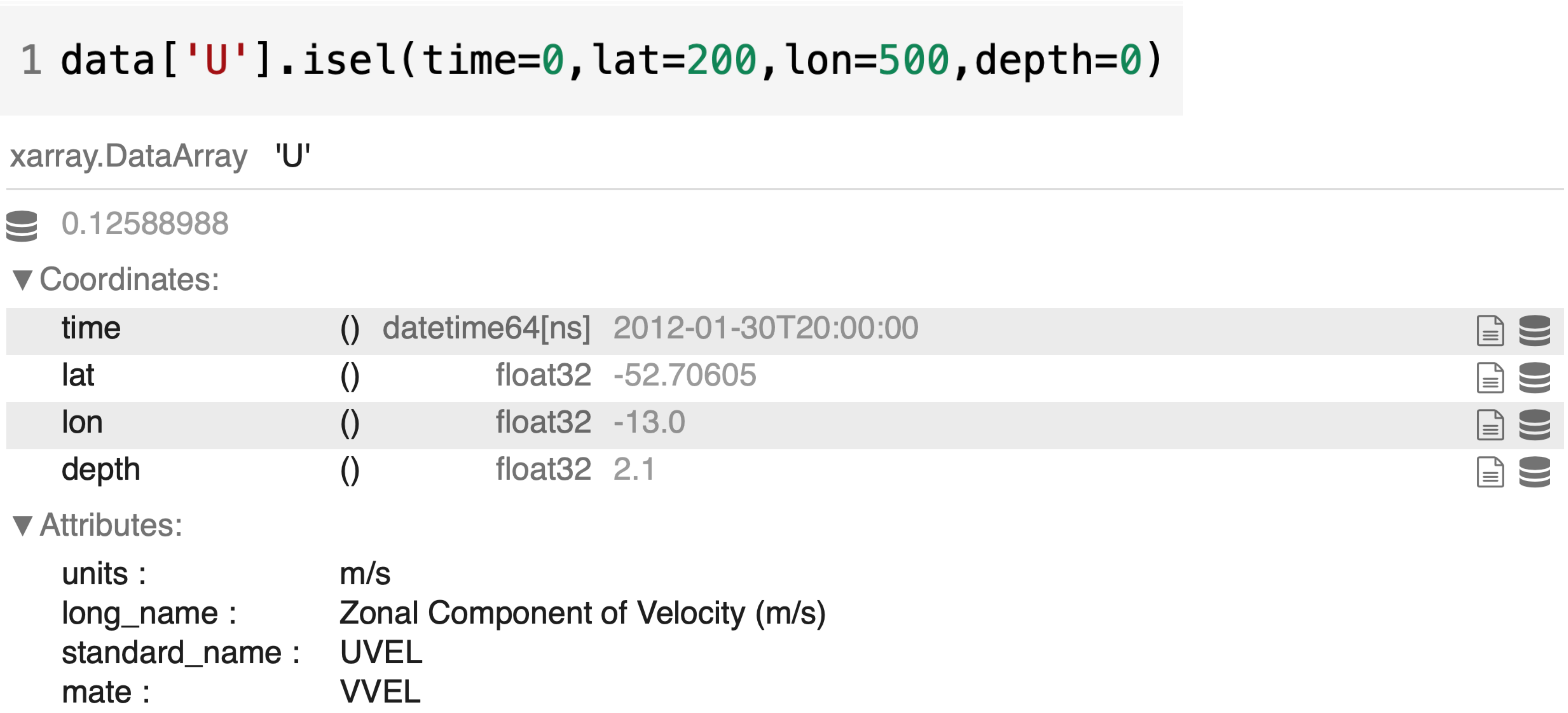
<DataArray or Dataset>**`.isel(`**<coordinate name>**`=`**<a single integer index>

OR <list or array of indices>

OR **`slice(`**<start>**`,`**<stop>**`)`**, …**`)`**

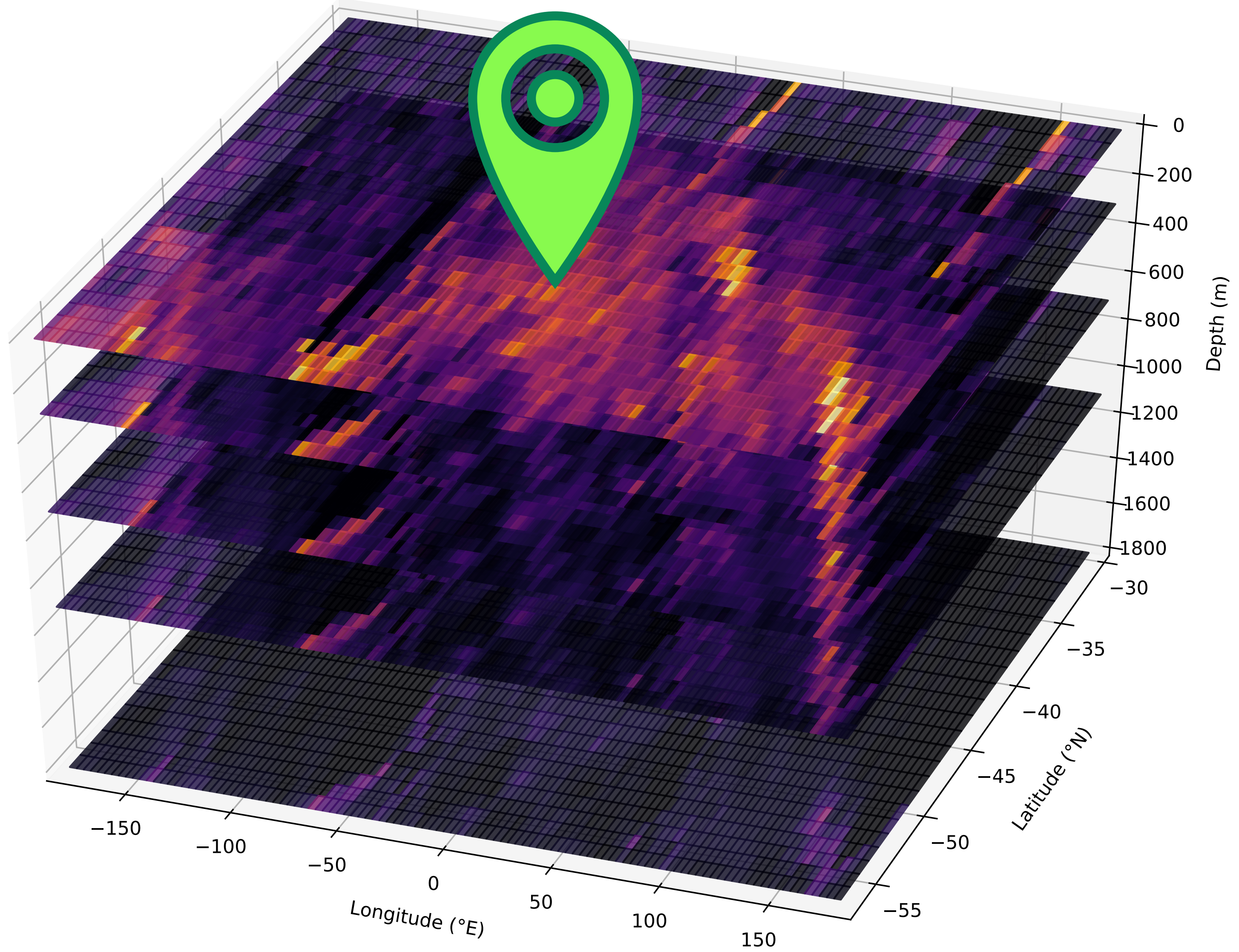
Like Python/NumPy slicing, the end value is exclusive!

Example:



```python
data['U'].isel(time=0,lat=200,lon=500,depth=0)
```

```
xarray.DataArray   'U'

0.12588988
▼ Coordinates:
   time     ()  datetime64[ns]  2012-01-30T20:00:00
   lat      ()         float32  -52.70605
   lon      ()         float32  -13.0
   depth    ()         float32  2.1
▼ Attributes:
   units :          m/s
   long_name :      Zonal Component of Velocity (m/s)
   standard_name :  UVEL
   mate :           VVEL
```



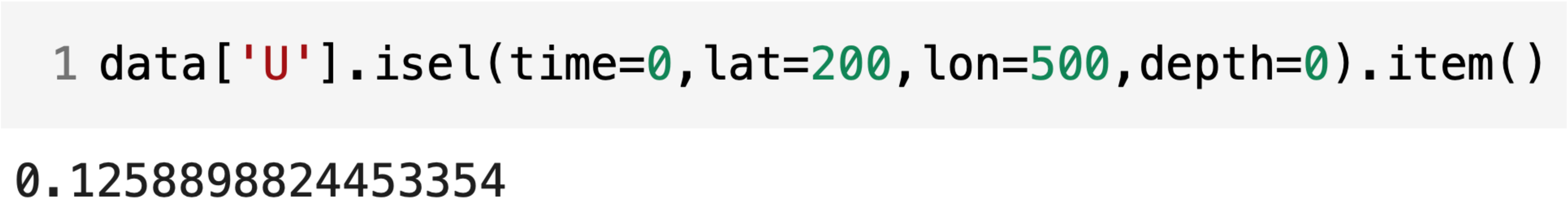
Convert a single-value `Dataset` to a number using `float( )` or `item( )`

<DataArray or Dataset>**`.isel(...).item()`**

OR...

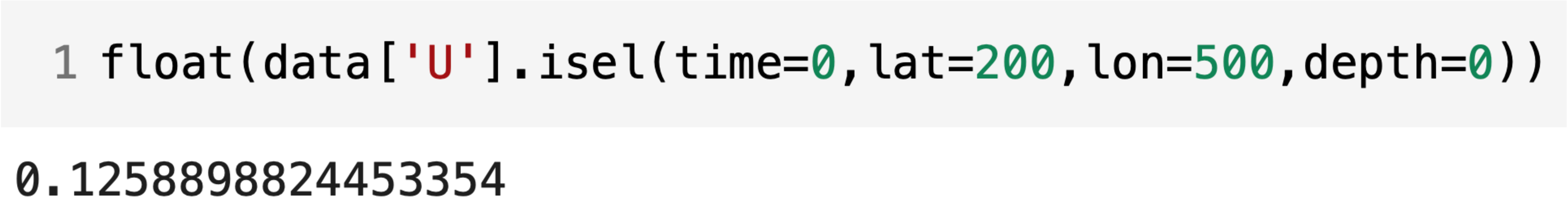
`float(`<DataArray or Dataset>**`.isel(...))`**

Example:



```
data['U'].isel(time=0,lat=200,lon=500,depth=0).item()
```

```
0.125889824453354
```



```
float(data['U'].isel(time=0,lat=200,lon=500,depth=0))
```

```
0.125889824453354
```

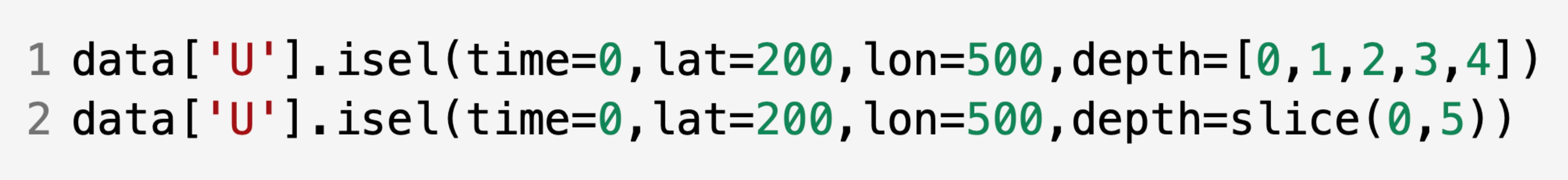
Selecting data from `xarray` objects using `.isel()` (selection by integer index)

<DataArray or Dataset>**`.isel(`**<coordinate name>=<a single integer index>

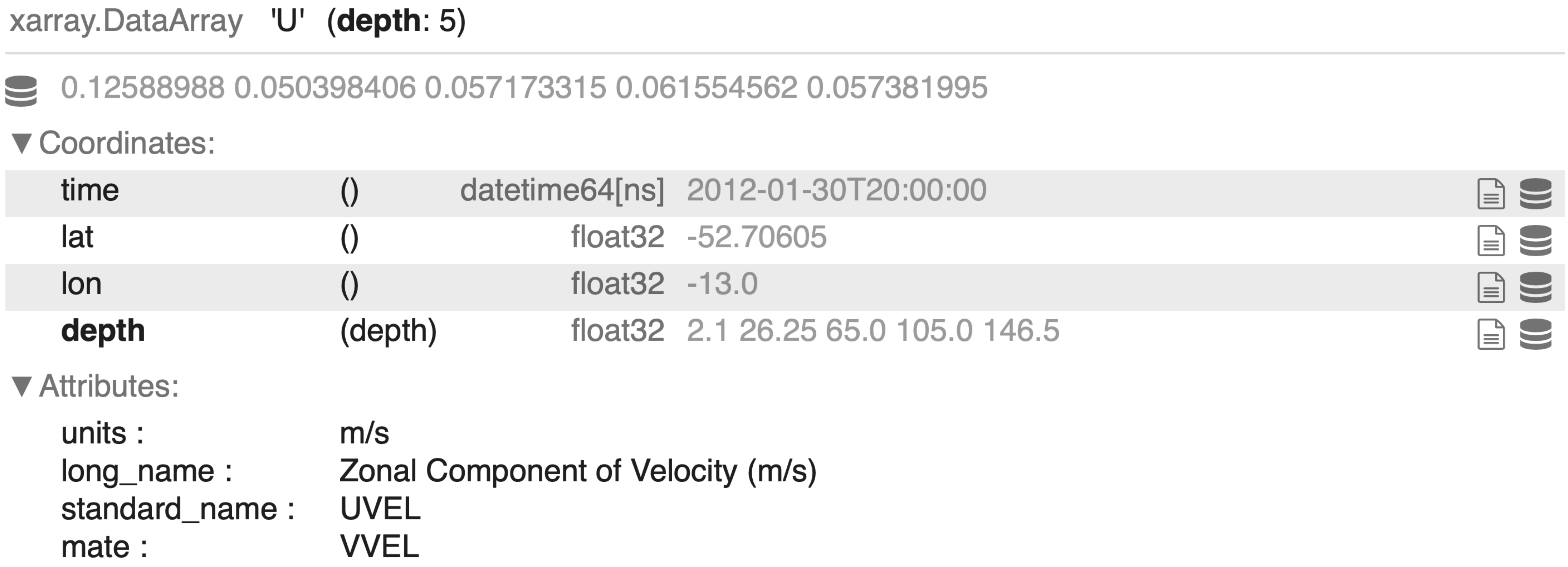
OR <list or array of indices>

OR `slice(`<start>`,`<stop>`),...)`

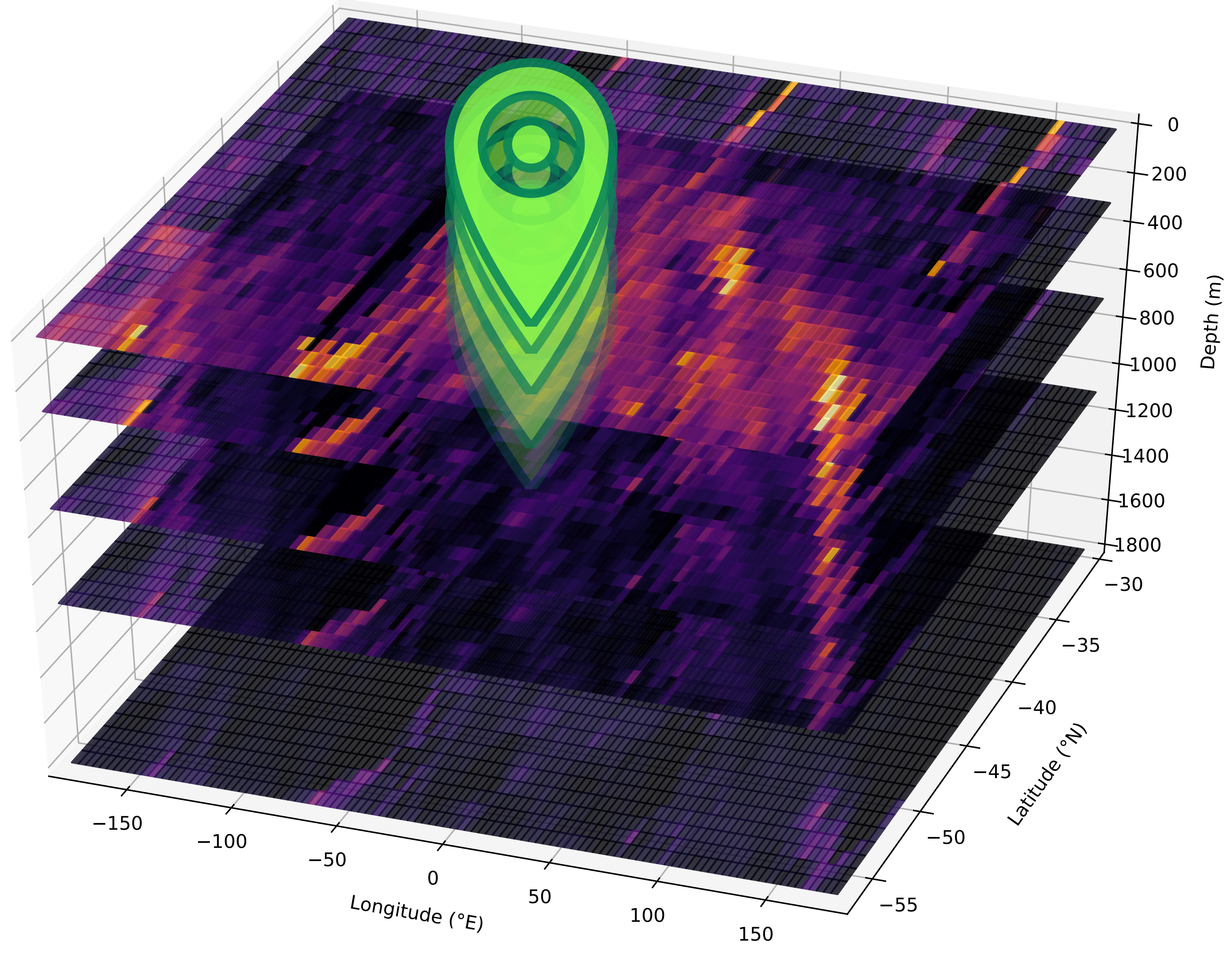
Example:



```python
data['U'].isel(time=0,lat=200,lon=500,depth=[0,1,2,3,4])
data['U'].isel(time=0,lat=200,lon=500,depth=slice(0,5))
```



```
xarray.DataArray   'U'   (depth: 5)

0.12588988 0.050398406 0.057173315 0.061554562 0.057381995

▼Coordinates:
    time            ()          datetime64[ns]  2012-01-30T20:00:00
    lat             ()                 float32  -52.70605
    lon             ()                 float32  -13.0
    depth           (depth)            float32  2.1 26.25 65.0 105.0 146.5

▼Attributes:
    units :          m/s
    long_name :      Zonal Component of Velocity (m/s)
    standard_name :  UVEL
    mate :           VVEL
```

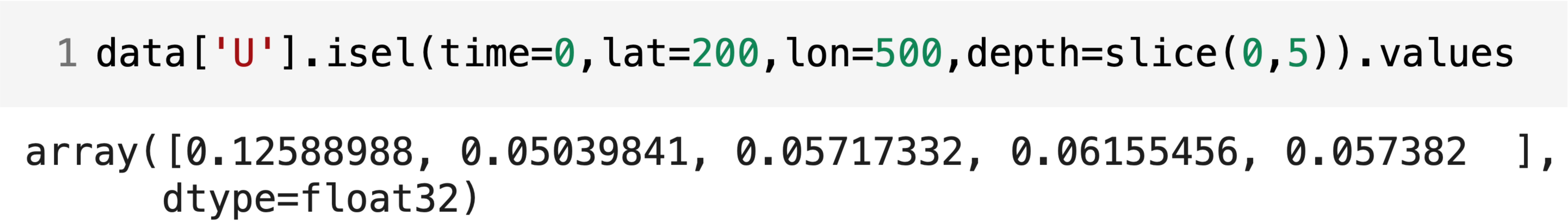


Convert a `Dataset` with multiple values to a NumPy array using `.values`

<DataArray or Dataset>**`.values`**

<DataArray or Dataset>**`.isel(…).values`**

Example:



```python
data['U'].isel(time=0,lat=200,lon=500,depth=slice(0,5)).values
```

```
array([0.12588988, 0.05039841, 0.05717332, 0.06155456, 0.057382  ],
      dtype=float32)
```

Selecting data from `xarray` objects using `.sel()` (selection by coordinate value)

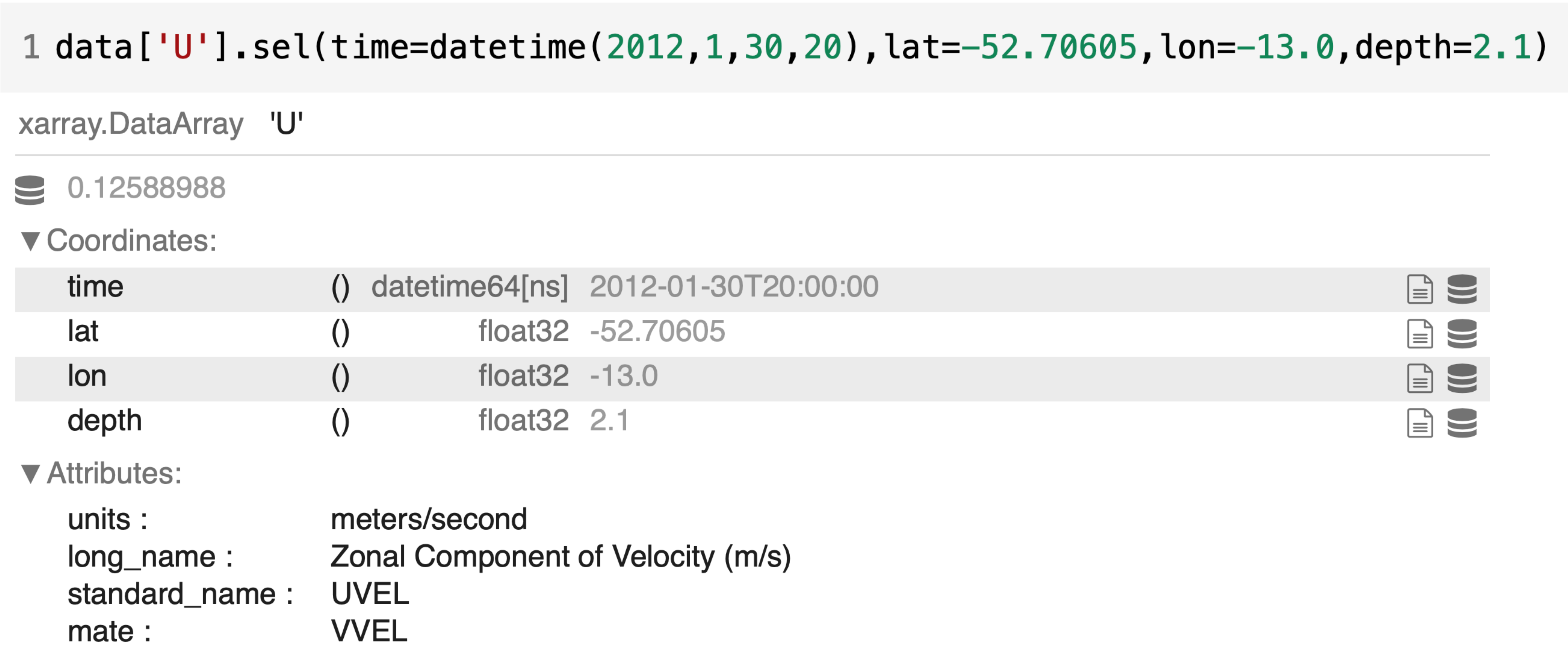
<DataArray or Dataset>`.sel(`<coordinate name>`=`<a single coordinate value>

OR <list or array of coordinate values>

OR `slice(`<start>`,`<stop>`),`…`)`

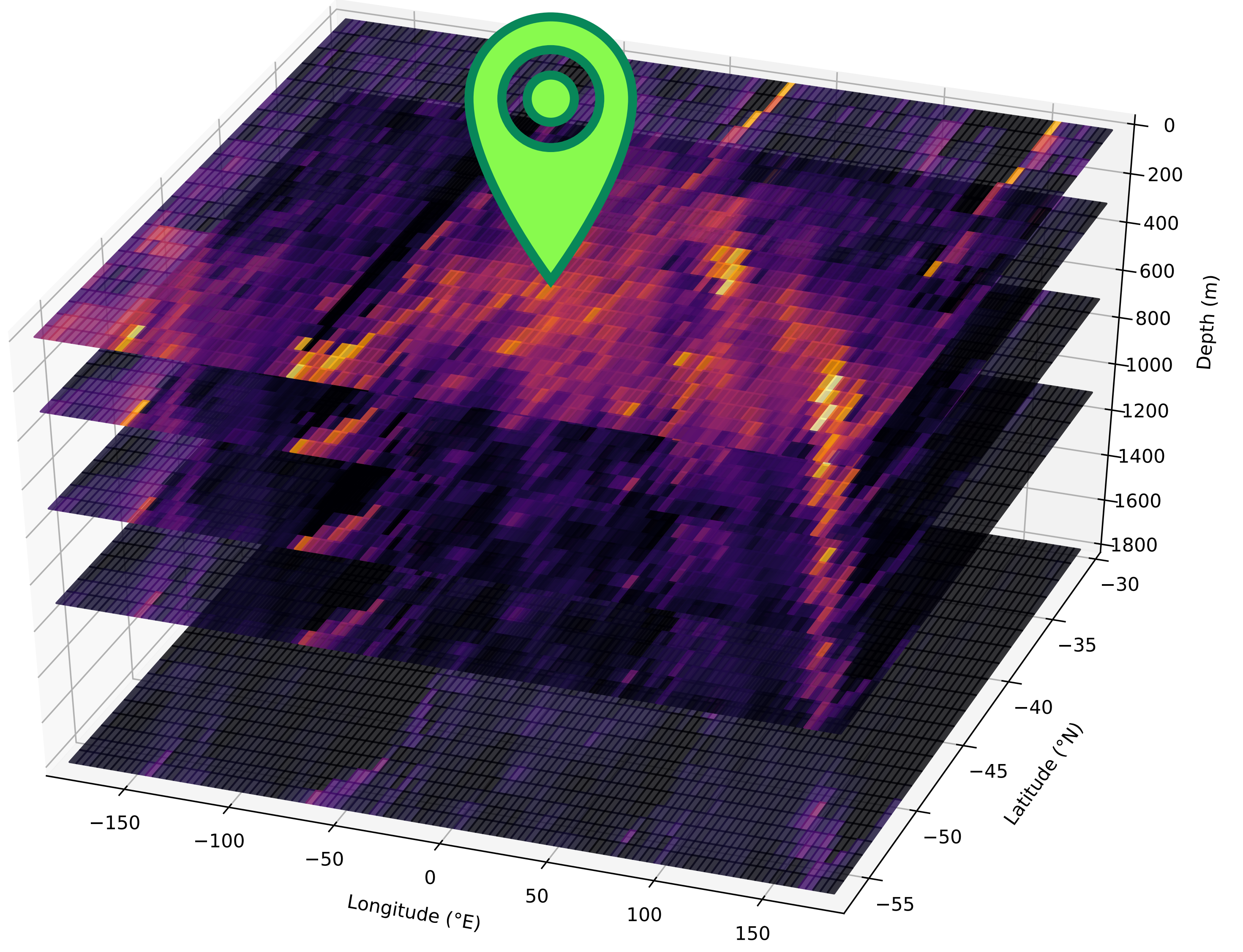
Unlike Python/NumPy slicing, the end value is **inclusive**!

Example:



```
data['U'].sel(time=datetime(2012,1,30,20),lat=-52.70605,lon=-13.0,depth=2.1)
```

```
xarray.DataArray   'U'

0.12588988
▼Coordinates:
   time        ()  datetime64[ns]  2012-01-30T20:00:00
   lat         ()         float32  -52.70605
   lon         ()         float32  -13.0
   depth       ()         float32  2.1
▼Attributes:
   units :          meters/second
   long_name :      Zonal Component of Velocity (m/s)
   standard_name :  UVEL
   mate :           VVEL
```



Selecting data from `xarray` objects using `.sel()` (selection by coordinate value)

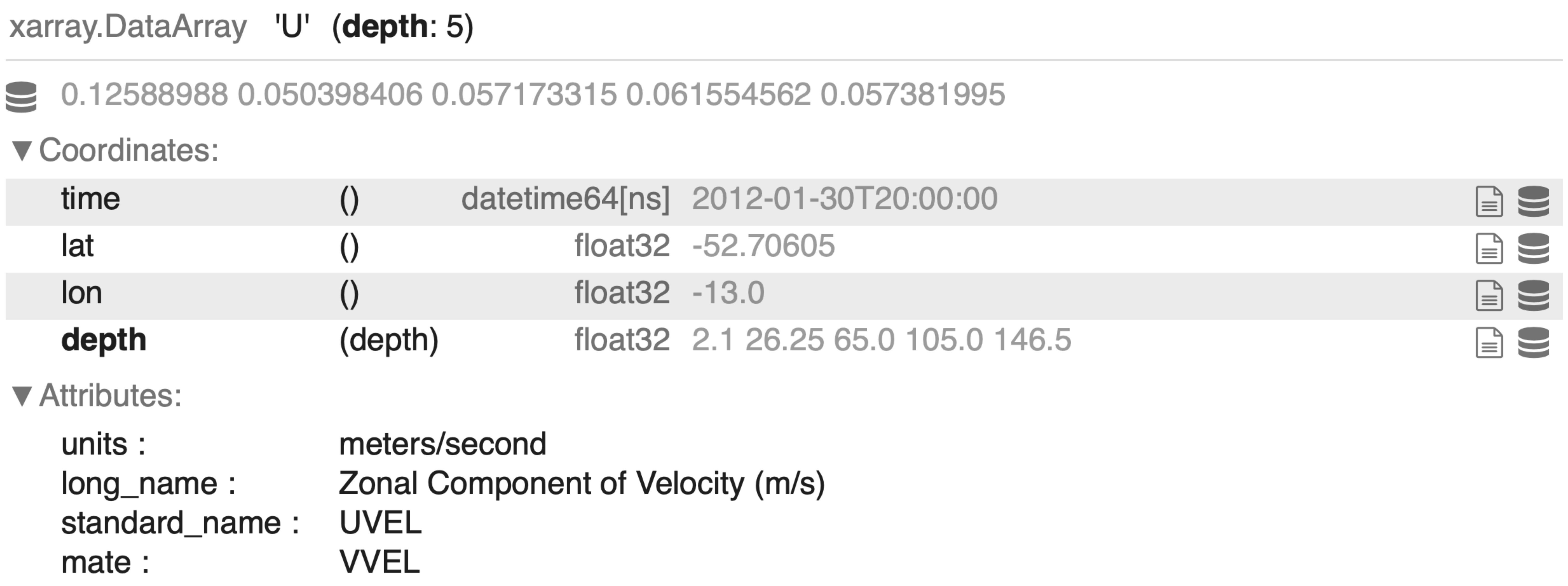
<DataArray or Dataset>**`.sel(`**<coordinate name>`=`<a single coordinate value>

OR <list or array of coordinate values>

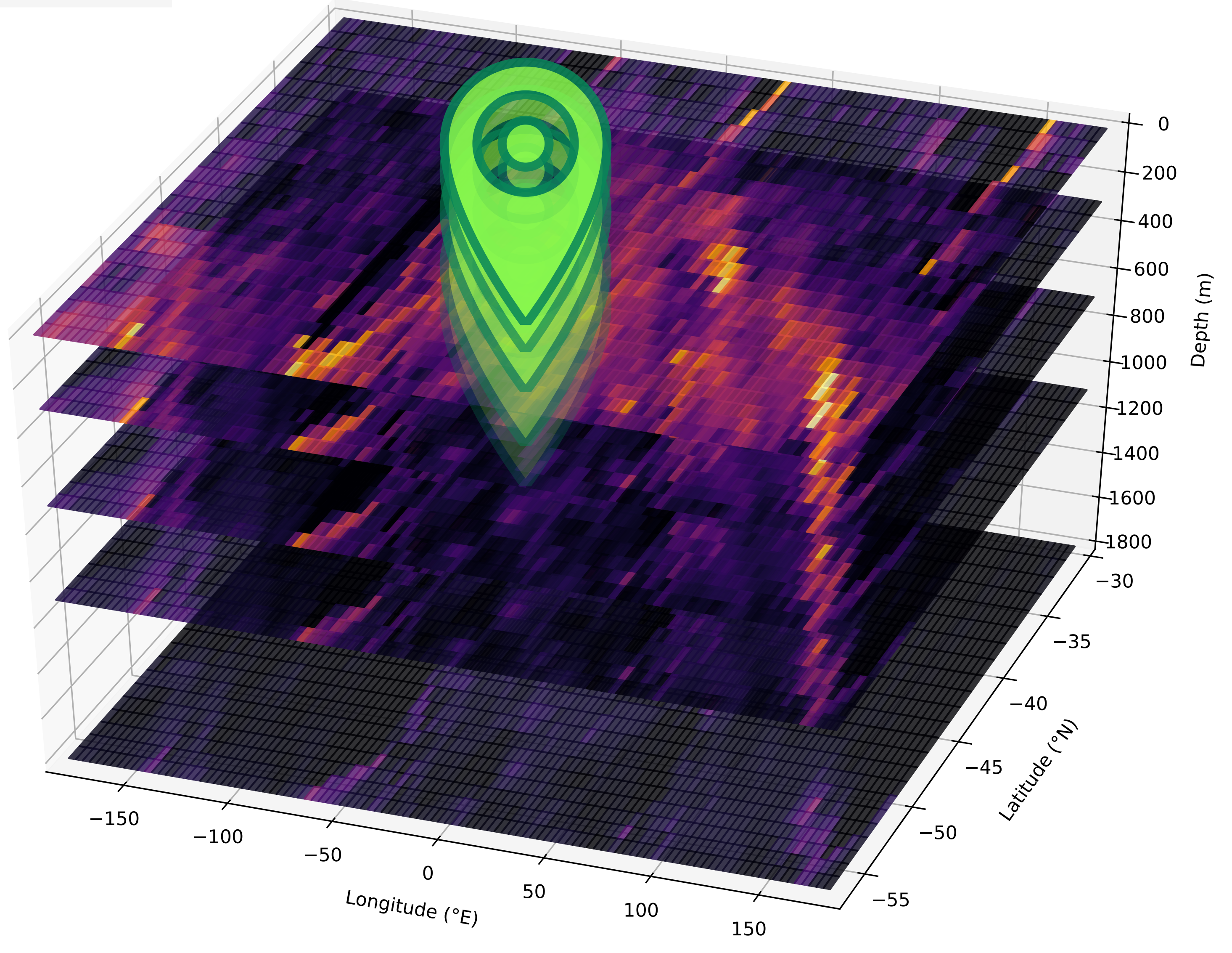
OR `slice(`<start>`,`<stop>`)`,...`)`

Example:

```
data['U'].sel(time=datetime(2012,1,30,20,0,0),lat=-52.70605,lon=-13.0,depth=slice(2,147))
```



```
xarray.DataArray   'U'   (depth: 5)

0.12588988 0.050398406 0.057173315 0.061554562 0.057381995

▼Coordinates:
   time          ()        datetime64[ns]   2012-01-30T20:00:00
   lat           ()               float32   -52.70605
   lon           ()               float32   -13.0
   depth         (depth)          float32   2.1 26.25 65.0 105.0 146.5

▼Attributes:
   units :           meters/second
   long_name :       Zonal Component of Velocity (m/s)
   standard_name :   UVEL
   mate :            VVEL
```

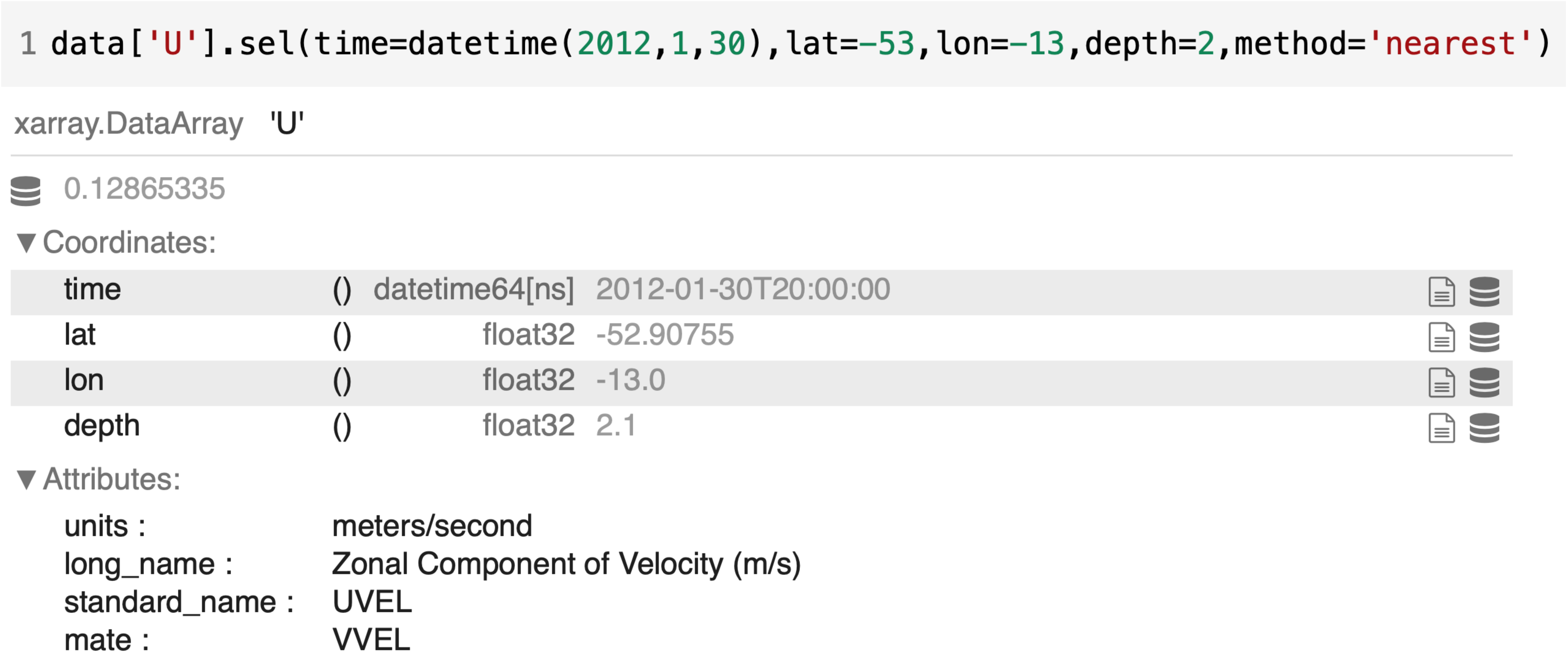


Selecting data from `xarray` objects using `.sel()` (selection by coordinate value)

Use method='nearest' when you don't know the exact coordinate values...

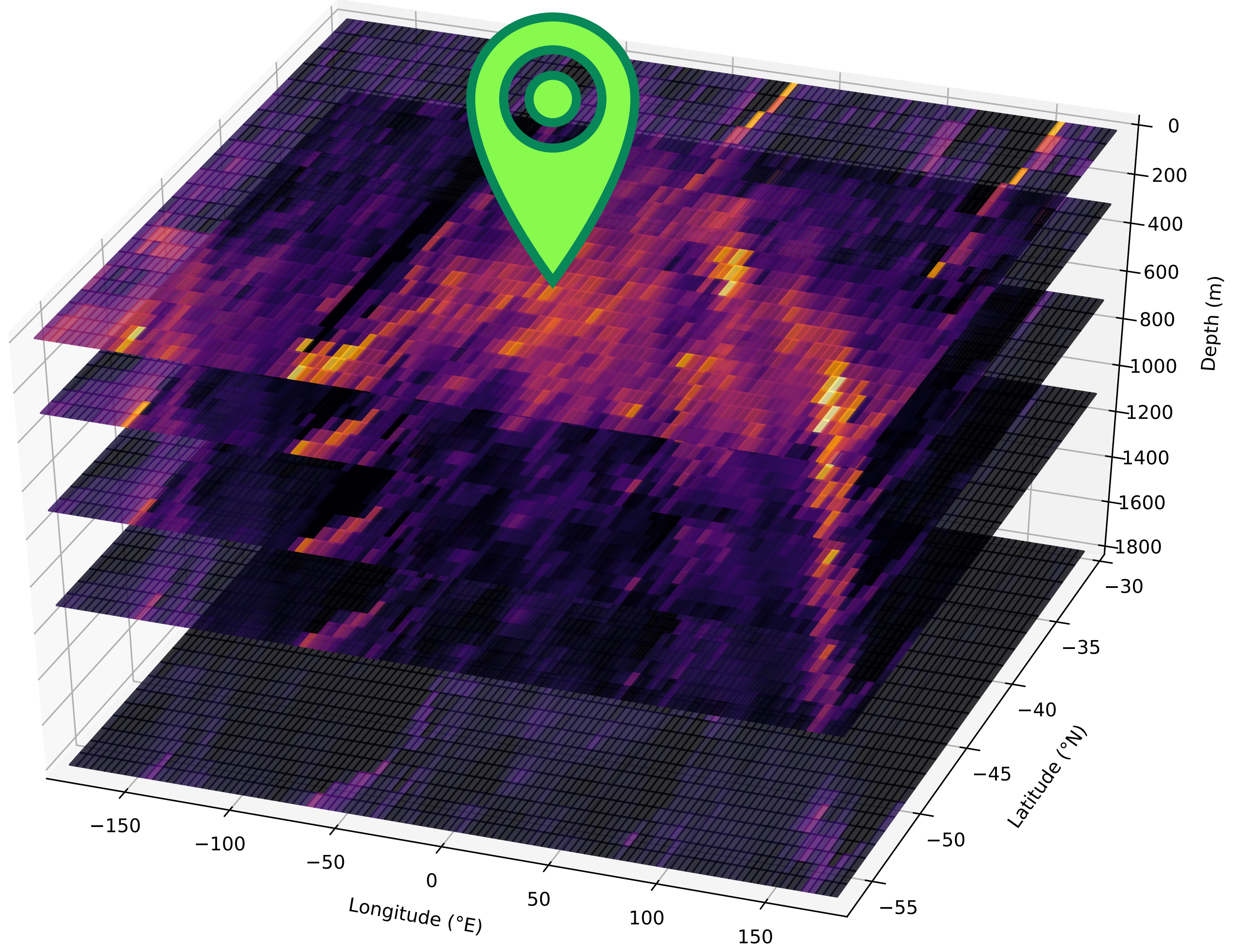
<DataArray or Dataset>**`.sel(`**<coordinate name>**`=`**<a single coordinate value>**`, ..., method='nearest')`**

Example:



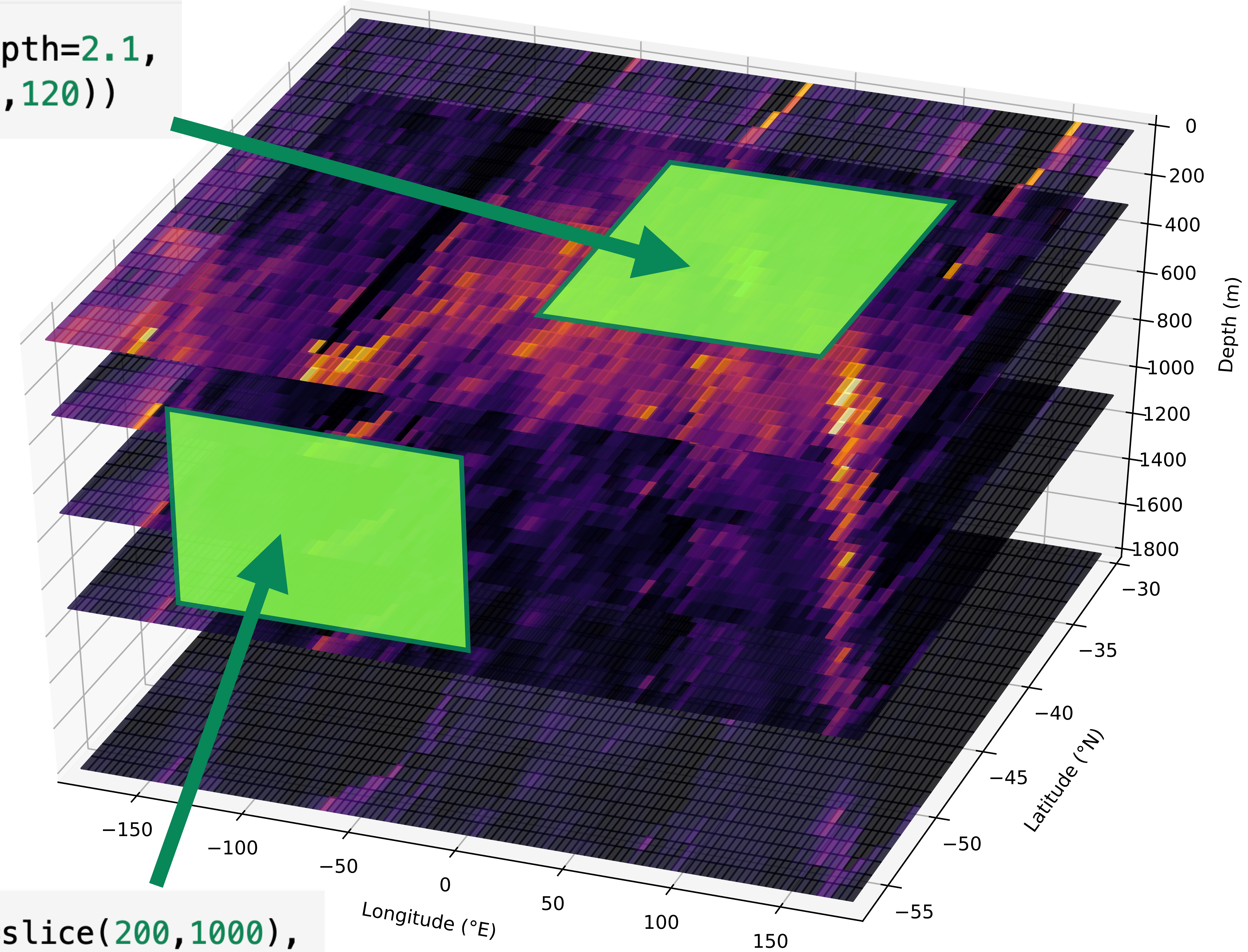
```
data['U'].sel(time=datetime(2012,1,30),lat=-53,lon=-13,depth=2,method='nearest')
```

```
xarray.DataArray   'U'

0.12865335

▼ Coordinates:
  time     ()  datetime64[ns]  2012-01-30T20:00:00
  lat      ()         float32  -52.90755
  lon      ()         float32  -13.0
  depth    ()         float32  2.1

▼ Attributes:
  units :           meters/second
  long_name :       Zonal Component of Velocity (m/s)
  standard_name :   UVEL
  mate :            VVEL
```



Selecting data from `xarray` objects using `.sel()` (selection by coordinate value)

```
data['U'].sel(time=datetime(2012,1,30,20),depth=2.1,
              lat=slice(-50,-40),lon=slice(0,120))
```

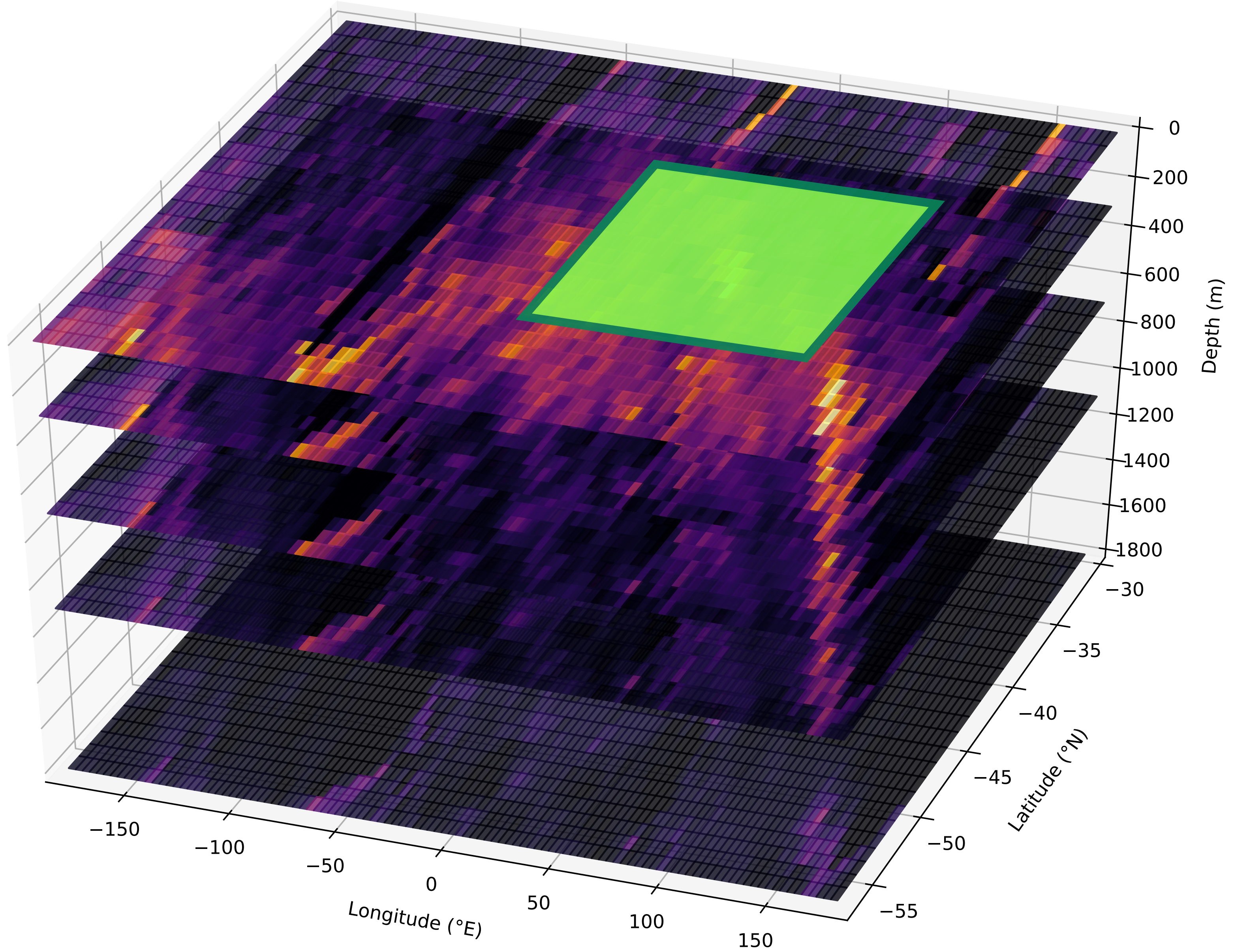


```
data['U'].sel(time=datetime(2012,1,30,20),depth=slice(200,1000),
             lon=slice(-120,0)).sel(lat=-57,method='nearest')
```

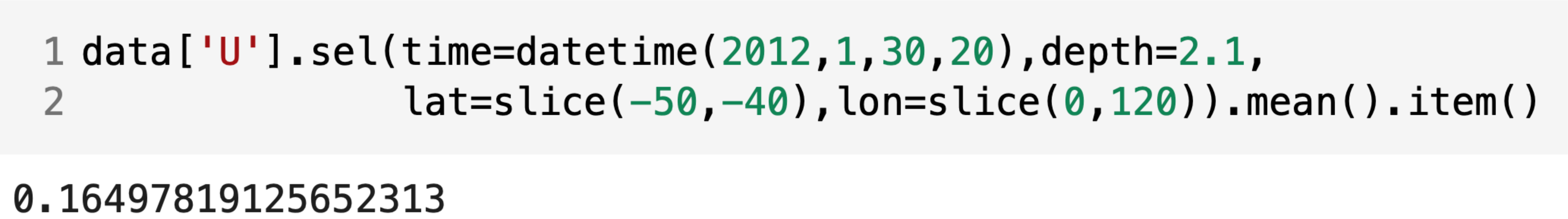
Applying NumPy functions to an `xarray` object (or selection from an object)

Take the average across ALL the dimensions:

<DataArray or Dataset>`.mean()`



Example:



```
data['U'].sel(time=datetime(2012,1,30,20),depth=2.1,
              lat=slice(-50,-40),lon=slice(0,120)).mean().item()
```

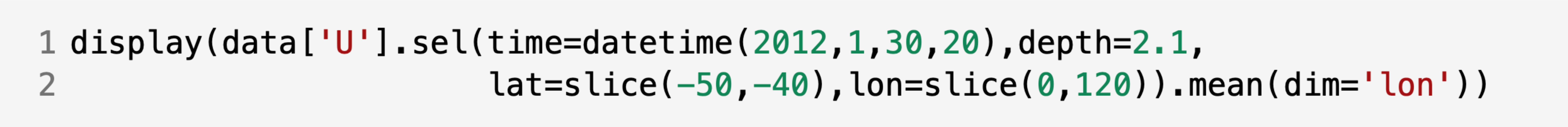
```
0.16497819125652313
```

Applying NumPy functions to an `xarray` object (or selection from an object)

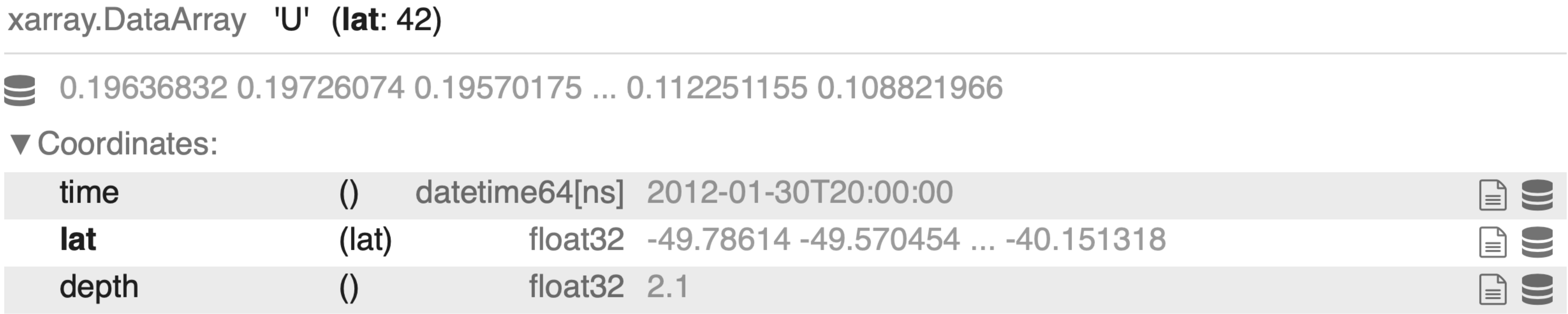
Take the average across certain dimension(s):

<DataArray or Dataset>`.mean(dim=`<dimension name(s) as string or list of strings>`)`

Example:

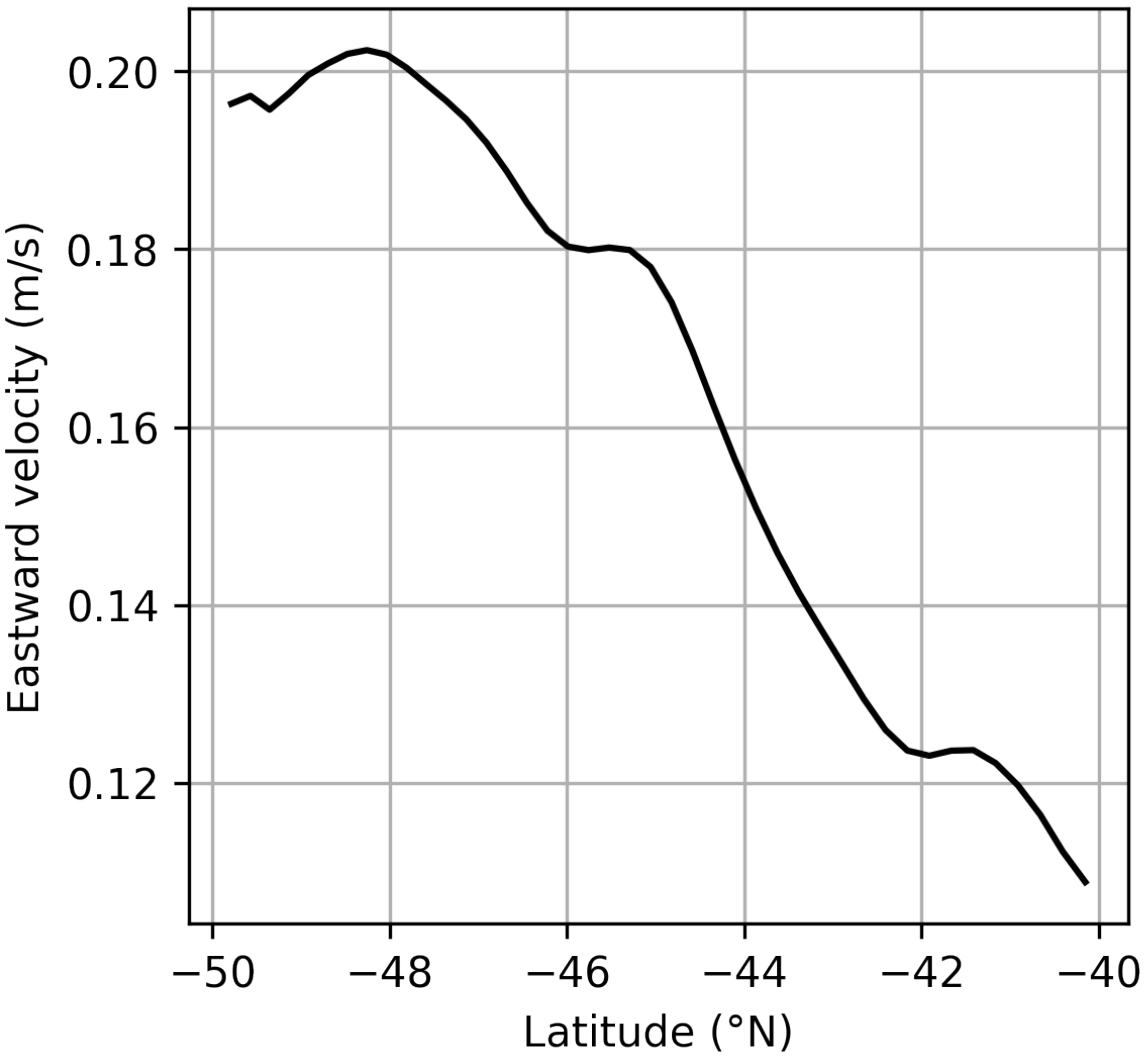


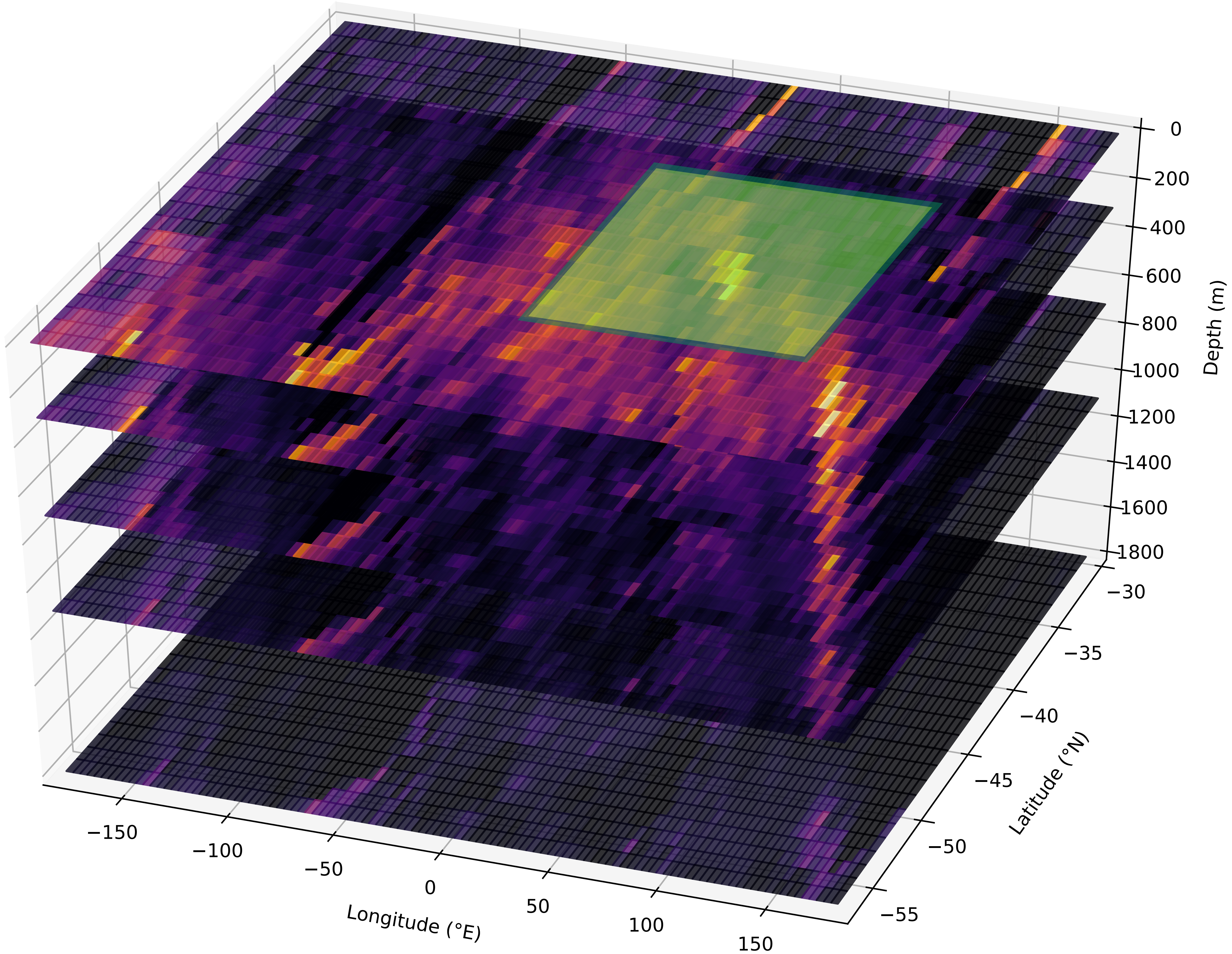
```
display(data['U'].sel(time=datetime(2012,1,30,20),depth=2.1,
                       lat=slice(-50,-40),lon=slice(0,120)).mean(dim='lon'))
```



```
xarray.DataArray   'U'   (lat: 42)

0.19636832 0.19726074 0.19570175 ... 0.112251155 0.108821966

▼Coordinates:
   time        ()      datetime64[ns]   2012-01-30T20:00:00
   lat         (lat)   float32          -49.78614 -49.570454 ... -40.151318
   depth       ()      float32          2.1
```

Applying NumPy functions to an `xarray` object (or selection from an object)


Eastward velocity (m/s)
Latitude (°N)


Depth (m)
Latitude (°N)
Longitude (°E)