pandas

Working with CSV data files

ESS 116 | Fall 2024

Prof. Henri Drake, Prof. Jane Baldwin, and Prof. Michael Pritchard

(Modified from Ethan Campbell and Katy Christensen's materials for UW's Ocean 215)

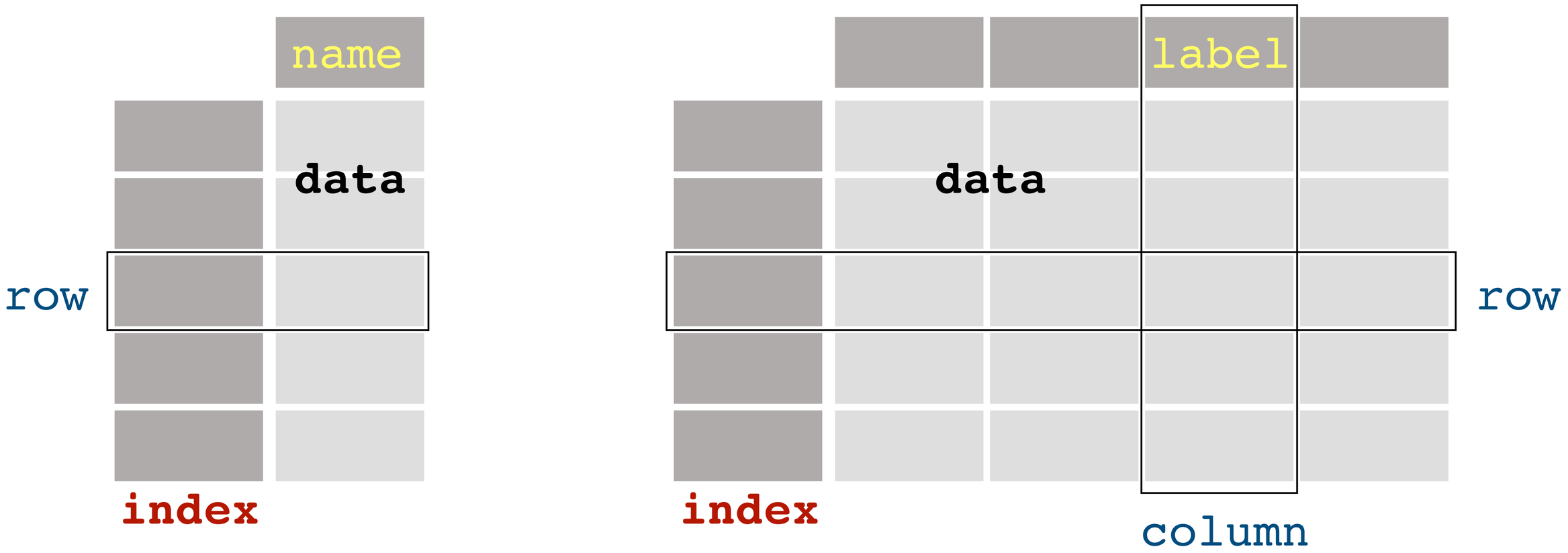
What we'll cover in this lesson

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

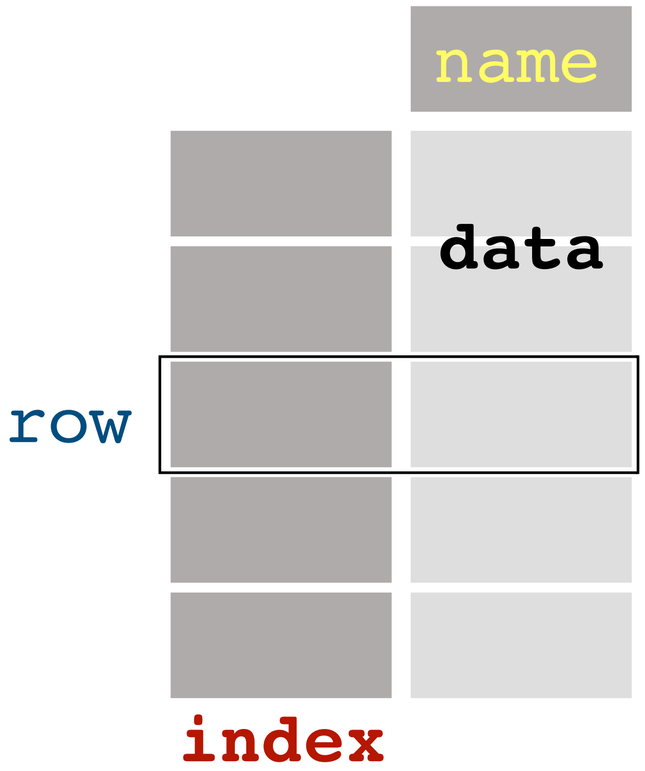
Loading `pandas`

```
import pandas as pd
```

`pandas` handles tabular data (tables or spreadsheets)



`pandas` handles tabular data (tables or spreadsheets)

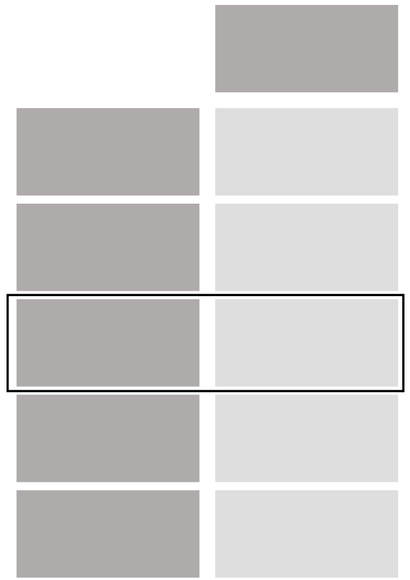


The `index` of a `Series` or `DataFrame` doesn't need to contain integers starting at 0.

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`

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

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

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

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

```
1 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>

Example:

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

Returns part of the original Series

OR <list or array of indices>

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

OR <slice of integer indices>

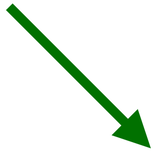
```
s1.iloc[2:5]
```

OR <Boolean array>`]`

```
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>

Returns part of the original Series

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

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[ ]`

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

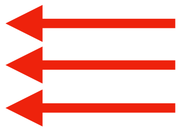
```
2016    4.1
2017    5.2
2018    5.3
2019    7.4
2020    8.5
Name: Temperature, dtype: float64
```

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

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

```
1 s1.loc[2018:2020] += 1
2 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

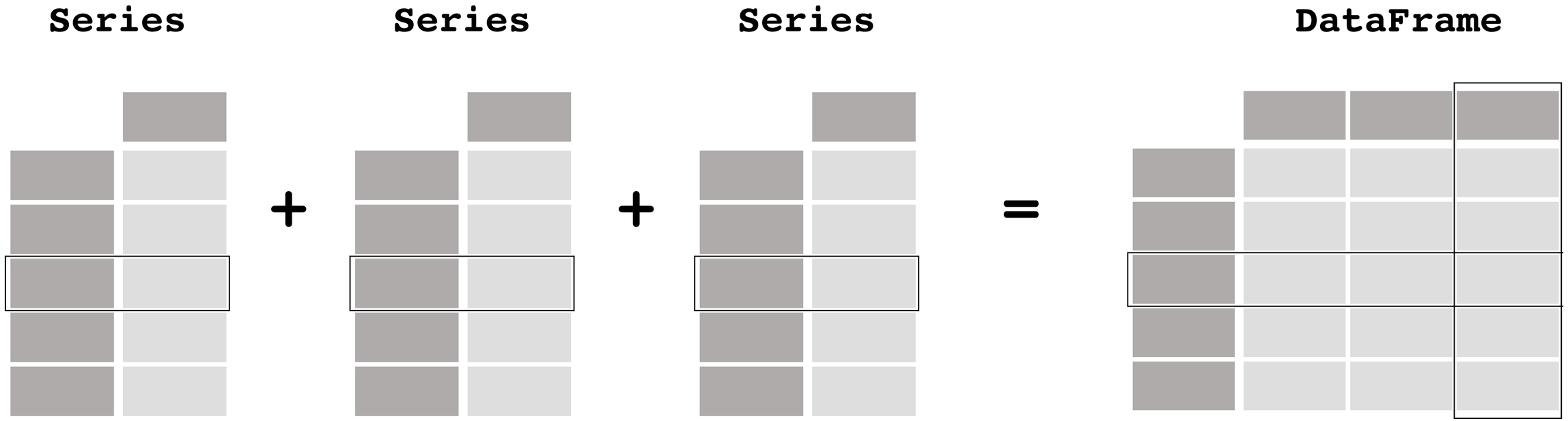
```
1 s1.loc[2021] = 9.6
2
3 print(s1)
```

```
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

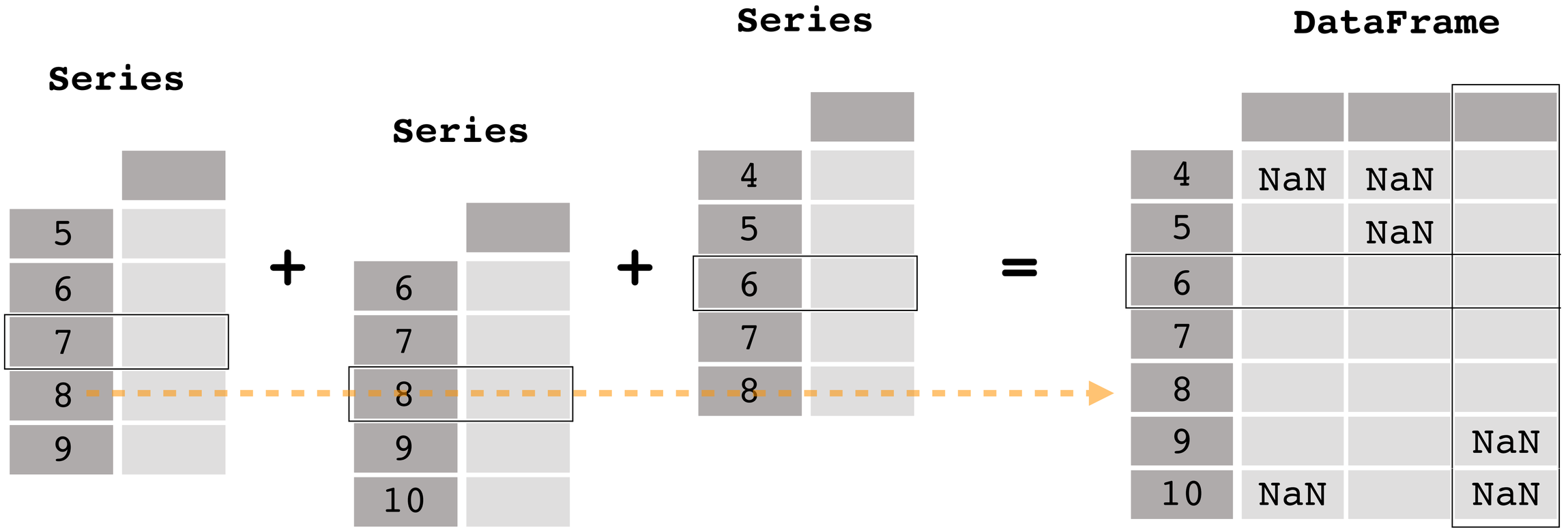
1. pandas: `Series` objects
2. **pandas: `DataFrame` objects; CSV files**
3. `xarray`: `DataArray` and `Dataset` objects; netCDF files
4. `xarray`: working with higher-dimensional data

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>**`)`**


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

Getting information about a `DataFrame`

`.shape`

```
1 df.shape
```

```
(5, 2)
```

`.size`

```
1 df.size
```

```
10
```

`print()`

```
1 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()`

```
1 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()`

```
1 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`

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

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

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

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

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

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

```
1 # Get one column as a NumPy array
2 # (think of this like dictionary indexing)
3 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:

```
1 sum = 0.0
2 for index in range(len(data)):
3   if times[index].year == 2019:
4     sum += data[index]
5 average = sum / len(data)
```

NumPy:

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

`pandas`:

```
1 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