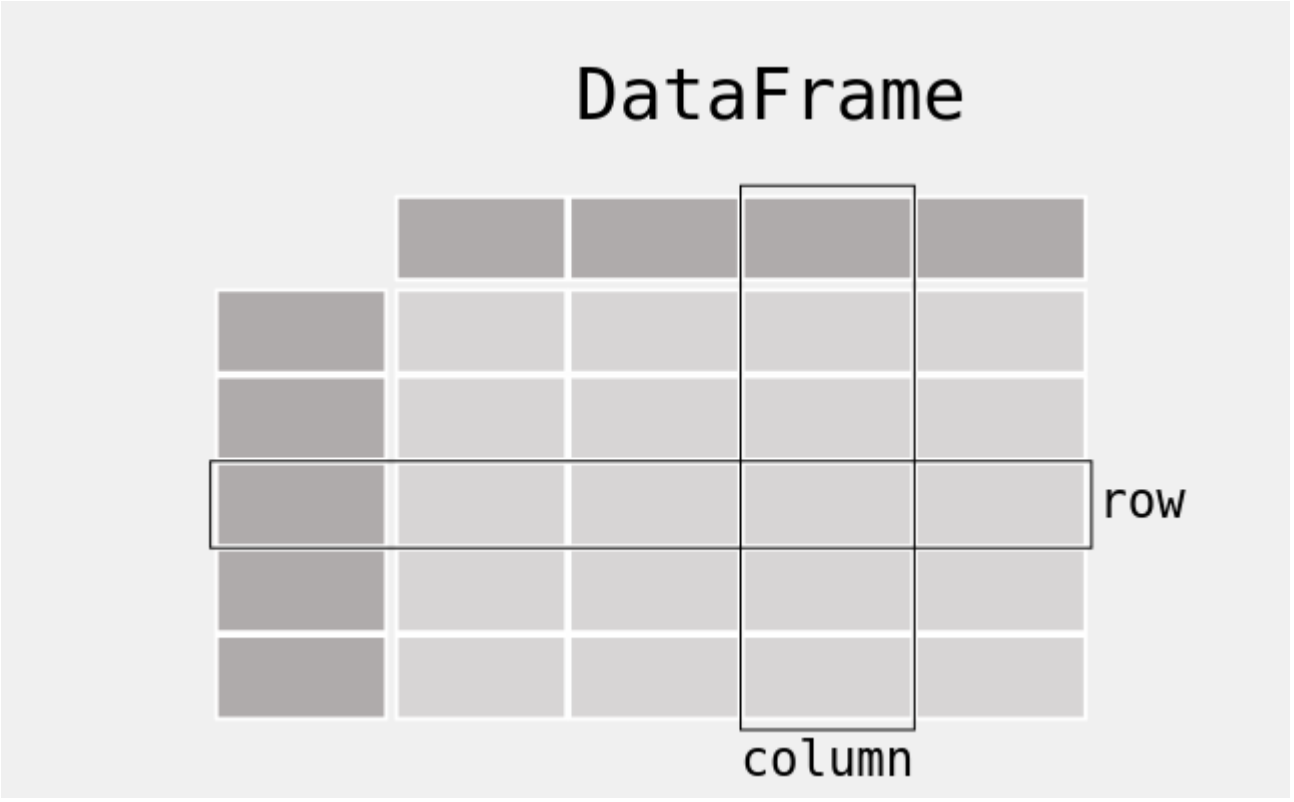
Data manipulation with Pandas

IC152 Feb 2021

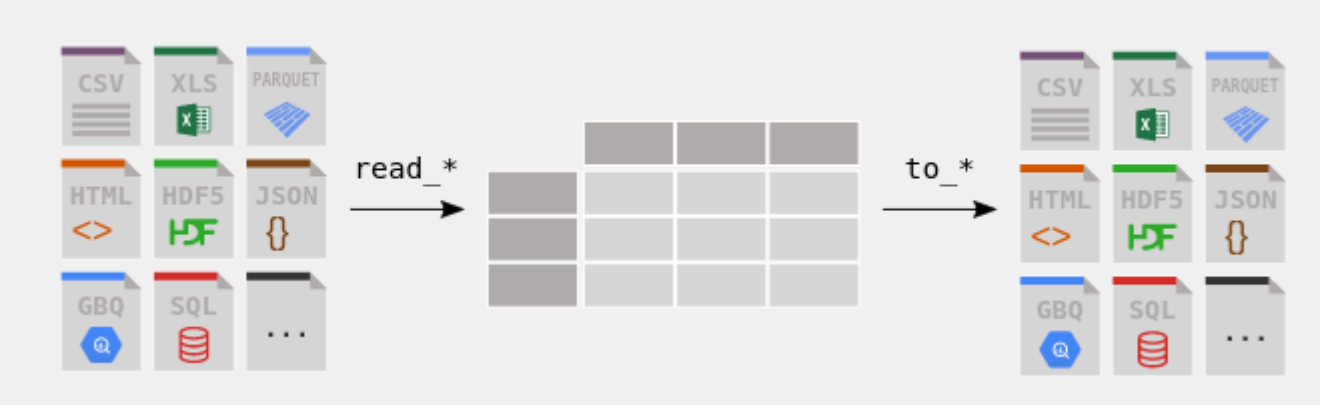
Pandas

- Pandas builds on NumPy and `ndarray`
- `DataFrame`, `Series`: basic data structure provided by Pandas
- `DataFrame` is a multidimensional array, with attached row and column labels; `Series` is a 1-D array
- Hetrogeneous types, and/or missing data
- More flexible than `ndarrays`: add labels, handle missing data, grouping data etc.



Idea of DataFrame

From https://pandas.pydata.org/



Pandas support reading/writing various formats

```
 9  import numpy as np
10  import pandas as pd
11
12  data = pd.Series([0.25,0.5,0.75,1.0])
13
```

- NumPy arrays have an implicitly defined index
- Pandas `Series` has an explicit index

```
In [2]: data
Out[2]:
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64

In [3]: data.values
Out[3]: array([0.25, 0.5 , 0.75, 1.  ])

In [4]: data.index
Out[4]: RangeIndex(start=0, stop=4, step=1)

In [5]: data[1]
Out[5]: 0.5

In [6]: data[1:3]
Out[6]:
1    0.50
2    0.75
dtype: float64
```

```
14  data2 = pd.Series([0.25, 0.5, 0.75, 1.0],index=['a', 'b', 'c', 'd'])
15
```

Explicit index

```
In [8]: data2
Out[8]:
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

`Series` is a bit like a dictionary key:value

```
16  population_dict = {'California': 38332521,'Texas': 26448193,\
17  'New York': 19651127,'Florida': 19552860,'Illinois': 12882135}
18
19  population = pd.Series(population_dict)
20
```

Can be made explictly with a dictionary

```
In [11]: population
Out[11]:
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

```
In [13]: population['California']
Out[13]: 38332521
```

Dict keys form the index

```
In [15]: population['California':'New York']
Out[15]:
California    38332521
Texas         26448193
New York      19651127
dtype: int64
```

slicing

```
21   data3 = pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2])
```

```
In [19]: data3
Out[19]:
3    c
2    a
dtype: object
```

Explictly defined incides are used

`Series` is like a 1-D array with explicit indices

- `DataFrame` is a 2-D array with flexible row indices and column names
- `DF` is a sequence of aligned `Series` objects
- Aligned = shares same index

```
23
24  area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,\
25  'Florida': 170312, 'Illinois': 149995}
26
27  area = pd.Series(area_dict)
28
29
30  states = pd.DataFrame({'population': population,\
31  'area': area})
```

From a dict of Series objects

```
In [33]: states
Out[33]:
            population    area
California    38332521  423967
Texas         26448193  695662
New York      19651127  141297
Florida       19552860  170312
Illinois      12882135  149995
```

`series 1` `series 2`

```
In [34]: states.index
Out[34]: Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'],
dtype='object')

In [35]: states.columns
Out[35]: Index(['population', 'area'], dtype='object')
```

DF has `columns` arritibute

DF maps a column name to a `Series` object

```
In [36]: states['area']
Out[36]:
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: area, dtype: int64
```

Creating a `DataFrame`

From a single `Series` object

```
In [37]: population
Out[37]:
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

Existing Series object

```
33   df_pop = pd.DataFrame(population, columns=['population'])
34
```

```
In [39]: df_pop
Out[39]:
            population
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
```

From a list of dictionaries

```
36  data = [{'a': i, 'b': 2 * i} for i in range(3)]
37
```

Dict comprehension

```
In [41]: data
Out[41]: [{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 4}]
```

Same keys here

```
38  df_2 = pd.DataFrame(data)
39
40
```

Dict keys form the column names

```
In [43]: df_2
Out[43]:
   a  b
0  0  0
1  1  2
2  2  4
```

From a 2-D NumPy array

```
41  df_3 = pd.DataFrame(np.random.rand(3, 2),columns=['foo', 'bar'],\
42  index=['a', 'b', 'c'])
43
```

Explicitly specifying index and columns

```
In [45]: df_3
Out[45]:
        foo       bar
a  0.463242  0.647092
b  0.560630  0.230305
c  0.778077  0.916524
```

Handling missing values

```
43
44  df_4 = pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}])
45
```

```
In [47]: df_4
Out[47]:
     a  b    c
0  1.0  2  NaN
1  NaN  3  4.0
```

NaN for missing values

Not a number

```
46  data = pd.Series([0.25, 0.5, 0.75, 1.0],index=['a', 'b', 'c', 'd'])
47
```

```
In [59]: data['a':'c']
Out[59]:
a    10.00
b     0.50
c     0.75
dtype: float64
```

Final index is included for explict indices. Not included for implicit indices.

Values can be changed

Can be appended

```
In [49]: data
Out[49]:
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64

In [50]: data['a']
Out[50]: 0.25

In [51]: data['a'] = 10

In [52]: data
Out[52]:
a    10.00
b     0.50
c     0.75
d     1.00
dtype: float64

In [53]: data['e'] = 10

In [54]: data
Out[54]:
a    10.00
b     0.50
c     0.75
d     1.00
e    10.00
dtype: float64
```

Indices can cause confusion

```
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
```

Explicit integer index

```
In [66]: data
Out[66]:
1    a
3    b
5    c
dtype: object
```

```
In [65]: data[1]
Out[65]: 'a'
```

Explicit index used here

```
In [67]: data[0:1]
Out[67]:
1    a
dtype: object
```

Implicit index while slicing

```
In [68]: data.loc[1]
Out[68]: 'a'

In [69]: data.iloc[1]
Out[69]: 'b'
```

Explicit index

Implicit index

Use `loc` and `iloc` indexers