Python for Data Analysis

Get Started with Pandas (Week 5)

Atefeh Khazaei

atefeh.khazaei@port.ac.uk

What we will learn this week?

- ❑ Fundamentals of Pandas Library
- ❑ Pandas Data Structures
- ❑ Loading and viewing data

Pandas

- The pandas package is the most important tool for Data Scientists and Analysts working with Python.
- The powerful machine learning and glamorous visualization tools may get all the attention, but pandas is the backbone of most data projects .
- If you're thinking about data science as a career, then it is compulsory that one of the first things you should do is learn pandas.
- Python with pandas is in use widely in academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

Pandas (cont.)

- Pandas contains data structures and data manipulation tools designed to *make data cleaning and analysis fast and easy* in Python.
- Pandas is often used in tandem with numerical computing tools like NumPy and SciPy, analytical libraries like statsmodels and scikit-learn, and data visualization libraries like matplotlib.

Pandas (cont.)
The difference between NumPy and Pandas

- Pandas adopts significant parts of NumPy's idiomatic style of array-based computing, especially array-based functions and a preference for data processing without for loops.

- The biggest difference is that:
  - **Pandas** is designed for working with tabular or heterogeneous data.
  - **NumPy**, by contrast, is best suited for working with homogeneous numerical array data.

Introduction to pandas Data Structures

- To get started with pandas, you will need to get comfortable with its two workhorse data structures:
  - **Series**
  - **DataFrame**
- While they are not a universal solution for every problem, they provide a solid, easy-to-use basis for most applications.

Introduction to pandas Data Structures (cont.)
Series

- A Series is a one-dimensional array-like object containing a sequence of values (of similar types to NumPy types) and an associated array of data labels, called its index.
- The simplest Series is formed from only an array of data:

```
import pandas as pd
obj = pd.Series([4, 7, -5, 3])
obj
```

```
0    4
1    7
2   -5
3    3
dtype: int64
```

- The string representation of a Series displayed interactively shows the index on the left and the values on the right.
- Since we did not specify an index for the data, a default one consisting of the integers 0 through N - 1 (where N is the length of the data) is created.

Introduction to pandas Data Structures (cont.)
Series

- You can get the array representation and index object of the Series via its values and index attributes, respectively:

```
import pandas as pd
obj = pd.Series([4, 7, -5, 3])
obj
```

```
0    4
1    7
2   -5
3    3
dtype: int64
```

```
obj.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

```
obj.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

Introduction to pandas Data Structures (cont.)

Series

- Often it will be desirable to create a Series with an index identifying each data point with a label.

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
obj2
```

```
d    4
b    7
a   -5
c    3
dtype: int64
```

```
obj2.index
```

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```

```
obj2['a']
```

```
-5
```

- Here ['c', 'a', 'd'] is interpreted as a list of indices, even though it contains strings instead of integers.

```
obj2[['c', 'a', 'd']]
```

```
c    3
a   -5
d    4
dtype: int64
```

Introduction to pandas Data Structures (cont.)

Series

- Using NumPy functions or NumPy-like operations, such as filtering with a boolean array, scalar multiplication, or applying math functions, will preserve the index-value link.

```
obj2[obj2 > 0]
```

```
d    4
b    7
c    3
dtype: int64
```

```
obj2 * 2
```

```
d     8
b    14
a   -10
c     6
dtype: int64
```

```
np.exp(obj2)
```

```
d      54.598150
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

Introduction to pandas Data Structures (cont.)

Series

- Another way to think about a Series is as a fixed-length, ordered dict, as it is a mapping of index values to data values.
- It can be used in many contexts where you might use a dict.
- Should you have data contained in a Python dict, you can create a Series from it by passing the dict.

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
obj3
```

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

Introduction to pandas Data Structures (cont.)

Series

- When you are only passing a dict, the index in the resulting Series will have the dict's keys in sorted order.
- You can override this by passing the dict keys in the order you want them to appear in the resulting Series.
- Since no value for 'California' was found, it appears as NaN (not a number), which is considered in pandas to mark missing or NA values.
- Since 'Utah' was not included in states, it is excluded from the resulting object.

```python
import pandas as pd
sdata = {'Ohio': 35000, 'Texas': 71000,
         'Oregon': 16000, 'Utah': 5000}
```

```python
states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
obj4
```

```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

Introduction to pandas Data Structures (cont.)
Series

- ***isnull*** and ***notnull*** functions in pandas should be used to detect missing data.
  - Working with missing data in more detail in next weeks.

```
pd.isnull(obj4)
```

```
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

```
pd.notnull(obj4)
```

```
California    False
Ohio           True
Oregon         True
Texas          True
dtype: bool
```

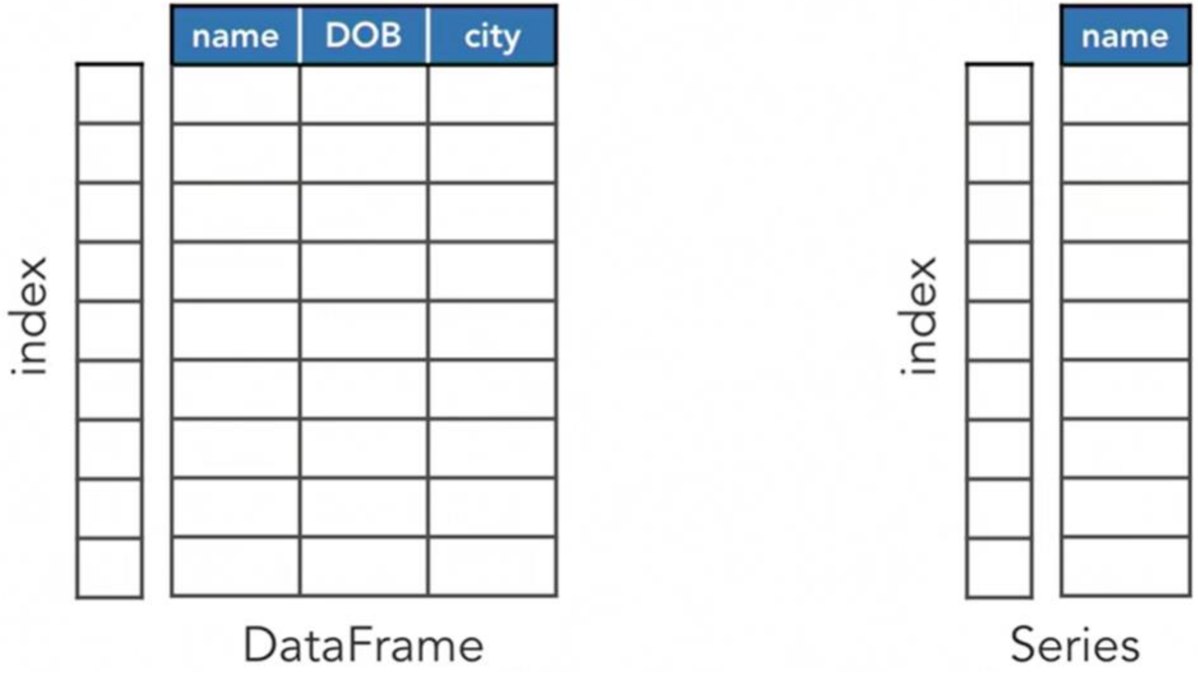
```
obj4.isnull()
```

```
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

Introduction to pandas Data Structures (cont.)

DataFrame

- A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.).
- The DataFrame has both a row and column index; it can be thought of as a dict of Series all sharing the same index.



Introduction to pandas Data Structures (cont.)
DataFrame

- There are many ways to construct a DataFrame, though one of the most common is from a dict of equal-length lists or NumPy arrays.

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
'year': [2000, 2001, 2002, 2001, 2002, 2003],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
frame
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

A column in a DataFrame can be retrieved as a Series either by dict-like notation or by attribute.

```
frame['state']

0      Ohio
1      Ohio
2      Ohio
3    Nevada
4    Nevada
5    Nevada
Name: state, dtype: object
```

```
frame.year

0    2000
1    2001
2    2002
3    2001
4    2002
5    2003
Name: year, dtype: int64
```

```
frame.loc[1]

state    Ohio
year     2001
pop       1.7
Name: 1, dtype: object
```

Introduction to pandas Data Structures (cont.)

DataFrame

- When you are assigning lists or arrays to a column, the value's length must match the length of the DataFrame.
- If you assign a Series, its labels will be realigned exactly to the DataFrame's index, inserting missing values in any holes.
- Assigning a column that doesn't exist will create a new column.

```
val = pd.Series([-1.2, -1.5, -1.7], index=[2, 4, 5])
frame['debt'] = val
frame
```

	state	year	pop	debt
0	Ohio	2000	1.5	NaN
1	Ohio	2001	1.7	NaN
2	Ohio	2002	3.6	-1.2
3	Nevada	2001	2.4	NaN
4	Nevada	2002	2.9	-1.5
5	Nevada	2003	3.2	-1.7

Introduction to pandas Data Structures (cont.)

DataFrame

- The *del* keyword will delete columns as with a dict.

```
frame['eastern'] = frame.state == 'Ohio'
frame
```

	state	year	pop	debt	eastern
0	Ohio	2000	1.5	NaN	True
1	Ohio	2001	1.7	NaN	True
2	Ohio	2002	3.6	-1.2	True
3	Nevada	2001	2.4	NaN	False
4	Nevada	2002	2.9	-1.5	False
5	Nevada	2003	3.2	-1.7	False

```
del frame['eastern']
frame.columns
```

```
Index(['state', 'year', 'pop', 'debt'], dtype='object')
```

Introduction to pandas Data Structures (cont.)
DataFrame

- Another common form of data is a nested dict of dicts.
  - The first one was a dict of equal-length lists or NumPy arrays.
  - You can see other form of data in our references.

```
pop = {'Nevada': {2001: 2.4, 2002: 2.9},
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
frame2 = pd.DataFrame(pop)
frame2
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

- If the nested dict is passed to the DataFrame, pandas will interpret the outer dict keys as the columns and the inner keys as the row indices.

DataFrame and Files

- Iris data-sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length and width.
- The rows are the samples and the columns are Sepal Length, Sepal Width, Petal Length and Petal Width.

```
import pandas as pd
data = pd.read_csv('iris.csv')
```

Iris Versicolor **Iris Setosa** **Iris Virginica**

DataFrame and Files (cont.)
Explore dataset

- ❑ How the data-set is looking like?
  - ❑ head() method

```
data.head()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

DataFrame and Files (cont.)
Explore dataset

- Having only one column from whole dataset.
- What is the difference?

```
data['sepal.length']
```

```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: sepal.length, Length: 150, dtype: float64
```

```
data[['sepal.length']]
```

	sepal.length
0	5.1
1	4.9
2	4.7
3	4.6
4	5.0
...	...
145	6.7
146	6.3
147	6.5
148	6.2
149	5.9

150 rows × 1 columns

DataFrame and Files (cont.) Explore dataset

```
data.keys()
```

```
Index(['sepal.length', 'sepal.width', 'petal.length', 'petal.width',
       'variety'],
      dtype='object')
```

- Find the different keys
  - Call columns as keys

```
data.count()
```

```
sepal.length    150
sepal.width     150
petal.length    150
petal.width     150
variety         150
dtype: int64
```

- Number of records

```
data['sepal.width'].nunique()
```

```
23
```

- Number of unique values of a column

DataFrame and Files (cont.)
Explore dataset

```
data['sepal.width'].sum()
```

```
458.6
```

```
data['sepal.width'].mean()
```

```
3.0573333333333334
```

```
data['sepal.width'].min()
```

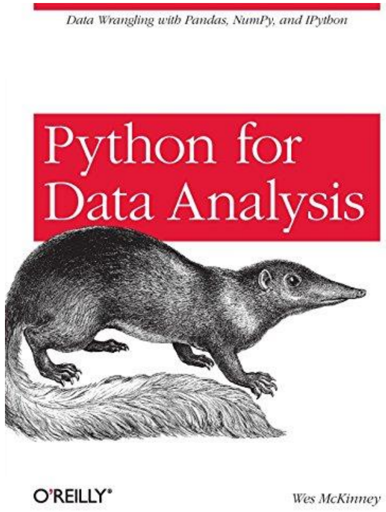
```
2.0
```

```
data['sepal.width'].max()
```

```
4.4
```

References & More Resources

- References:
  - McKinney, Wes. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc., 2012.



- More Resources:
  - Python Data Analysis on Linkedin Learning: https://www.linkedin.com/learning/python-data-analysis-2
  - Learning Python on Linkedin Learning https://www.linkedin.com/learning/learning-python
    - To use Linkedin Learning, you can log in with your university account: https://myport.port.ac.uk/study-skills/linkedin-learning

COURSE

Python Data Analysis

By: Michele Vallisneri

COURSE

Learning Python

By: Joe Marini

Practical Session

- Please download Week05_PandasBasics.ipynb file, and run it to learn new points.
- Please read the practical sheet (Week05_Practicals.pdf) and do the exercise.