UNIVERSITY OF PORTSMOUTH

Python for Data Analysis

Data Loading, Storage, and File Formats; Filtering and Merging (Week 6)

Atefeh Khazaei

atefeh.khazaei@port.ac.uk

What we will learn this week?

- Reading and Writing Data in Text Format in Pandas
- Filtering and Merging DataFrames in Pandas Library

Pandas & Files

- Accessing data is a necessary first step for using lots of data analysis tools.
- Input and output typically falls into a few main categories:
  - Reading text files and other more efficient on-disk formats,
  - Loading data from databases,
  - Interacting with network sources like web APIs
- Pandas features a number of functions for reading tabular data as a DataFrame object.

Reading and Writing Data in Text Format

read_csv and read_table are likely the ones you'll use the most.

Function	Description
`read_csv`	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
`read_table`	Load delimited data from a file, URL, or file-like object; use tab (`'\t'`) as default delimiter
`read_fwf`	Read data in fixed-width column format (i.e., no delimiters)
`read_clipboard`	Version of `read_table` that reads data from the clipboard; useful for converting tables from web pages
`read_excel`	Read tabular data from an Excel XLS or XLSX file
`read_hdf`	Read HDF5 files written by pandas
`read_html`	Read all tables found in the given HTML document
`read_json`	Read data from a JSON (JavaScript Object Notation) string representation
`read_msgpack`	Read pandas data encoded using the MessagePack binary format
`read_pickle`	Read an arbitrary object stored in Python pickle format
`read_sas`	Read a SAS dataset stored in one of the SAS system's custom storage formats
`read_sql`	Read the results of a SQL query (using SQLAlchemy) as a pandas DataFrame
`read_stata`	Read a dataset from Stata file format
`read_feather`	Read the Feather binary file format

Reading and Writing Data in Text Format (cont.)

- These functions are meant to convert text data into a DataFrame.
- The optional arguments for these functions may fall into a few categories:
  - Indexing
  - Type inference and data conversion
  - Datetime parsing
  - Iterating
  - Unclean data issues
- Because of how messy data in the real world can be, some of the data loading functions (especially read_csv) have grown very complex in their options over time.

Reading and Writing Data in Text Format (cont.)

Week06_ex1.csv

```
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

- Since this is comma-delimited, we can use **read_csv** to read it into a DataFrame.
- We could also have used **read_table** and specified the delimiter.

```
import pandas as pd
df = pd.read_csv('Week06_ex1.csv')
df
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
import pandas as pd
df2 = pd.read_table('Week06_ex1.csv', sep=',')
df2
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Reading and Writing Data in Text Format (cont.)

Week06_ex2.csv

```
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

- A file will not always have a header row.
- pandas can assign default column names, or you can specify names yourself

```
pd.read_csv('Week06_ex2.csv', header=None)
```

	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

```
pd.read_csv('Week06_ex2.csv',
            names=['a', 'b', 'c', 'd', 'message'])
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Reading and Writing Data in Text Format (cont.)

- Suppose you wanted the message column to be the index of the returned DataFrame.

```
names = ['a', 'b', 'c', 'd', 'message']
pd.read_csv('Week06_ex2.csv', names=names, index_col='message')
```

	a	b	c	d
message				
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

Reading and Writing Data in Text Format (cont.)

- Handling missing values is an important and frequently nuanced part of the file parsing process.
- Missing data is usually either not present (empty string) or marked by some sentinel value.
- By default, pandas uses a set of commonly occurring sentinels, such as NA and NULL.

Week06_ex3.csv

```
something,a,b,c,d,message
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
```

```
result = pd.read_csv('Week06_ex3.csv')
result
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

Reading and Writing Data in Text Format (cont.)
Writing Data to Text Format

- Data can also be exported to a delimited format.
- Writing the data out to a comma-separated file.

```
import pandas as pd
result = pd.read_csv('Week06_ex3.csv')
result
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
result.to_csv('out_ex3.csv')
```

Python and Other Data Formats

- Other text formats:
  - JSON Data
  - XML and HTML: Web Scraping
- Binary Data Formats
  - HDF5 Format
  - Microsoft Excel Files
- Interacting with Web APIs
- Interacting with Databases
- See more details in pandas documents and our references.

Filtering in Pandas Library

- We can apply filtering options in pandas DataFrame, filtering will help us:
  - Having specific records
  - Filtering out unrelated data

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
result.query('b > 5')
```

	something	a	b	c	d	message
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

```
result_filtered = result[result.b > 5]
result_filtered
```

	something	a	b	c	d	message
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

Filtering in Pandas Library (cont.)

```
result_filtered2 = result[(result.b > 5) & (result.d > 10)]
result_filtered2
```

	something	a	b	c	d	message
2	three	9	10	11.0	12	foo

```
result_filtered3 = result[result.a == 5]
result_filtered3
```

	something	a	b	c	d	message
1	two	5	6	NaN	8	world

Merging DataFrames in Pandas Library

- Joining and merging DataFrames is the core process to start with data analysis and machine learning tasks
- Data can be provided in different files that needed to be joined and merged.
- We can merge two data frames in pandas python by using the **merge()** function.
- The different arguments to **merge()** allow you to perform **natural inner join, left join, right join,** and **full outer join** in pandas.

Merging DataFrames in Pandas Library (cont.)

```
# data frame 1
d1 = {'id':pd.Series([1,2,3,4,5,6]),
'Product':pd.Series([ 'A', 'A', 'A', 'B', 'B', 'B'])}
df1 = pd.DataFrame(d1)
# data frame 2
d2 = {'id':pd.Series([2,4,6,7]),
'State':pd.Series([ 'Portsmouth ', 'Portsmouth', 'Southampton', 'Southampton'])}
df2 = pd.DataFrame(d2)
```

```
df1
```

	id	Product
0	1	A
1	2	A
2	3	A
3	4	B
4	5	B
5	6	B

```
df2
```

	id	State
0	2	Portsmouth
1	4	Portsmouth
2	6	Southampton
3	7	Southampton

Merging DataFrames in Pandas Library (cont.)

- **Inner join pandas:** Return only the rows in which the left table have matching keys in the right table.

```
df1
```

	id	Product
0	1	A
1	2	A
2	3	A
3	4	B
4	5	B
5	6	B

```
df2
```

	id	State
0	2	Portsmouth
1	4	Portsmouth
2	6	Southampton
3	7	Southampton

```
Newframe =pd.merge(df1, df2, on='id', how='inner')
Newframe
```

	id	Product	State
0	2	A	Portsmouth
1	4	B	Portsmouth
2	6	B	Southampton

Merging DataFrames in Pandas Library (cont.)

- **Outer join pandas:** Return all rows from both tables, join records from the left table which have matching keys in the right table.

```
df1
```

	id	Product
0	1	A
1	2	A
2	3	A
3	4	B
4	5	B
5	6	B

```
df2
```

	id	State
0	2	Portsmouth
1	4	Portsmouth
2	6	Southampton
3	7	Southampton

```
Newframe =pd.merge(df1, df2, on='id', how='outer')
Newframe
```

	id	Product	State
0	1	A	NaN
1	2	A	Portsmouth
2	3	A	NaN
3	4	B	Portsmouth
4	5	B	NaN
5	6	B	Southampton
6	7	NaN	Southampton

Merging DataFrames in Pandas Library (cont.)

- **Left outer join:** Return all rows from the left table, and any rows with matching keys from the right table.

```
Newframe =pd.merge(df1, df2, on='id', how='left')
Newframe
```

```
df1
```

	id	Product
0	1	A
1	2	A
2	3	A
3	4	B
4	5	B
5	6	B

```
df2
```

	id	State
0	2	Portsmouth
1	4	Portsmouth
2	6	Southampton
3	7	Southampton

	id	Product	State
0	1	A	NaN
1	2	A	Portsmouth
2	3	A	NaN
3	4	B	Portsmouth
4	5	B	NaN
5	6	B	Southampton

Merging DataFrames in Pandas Library (cont.)

- **Right outer join:** Return all rows from the right table and any rows with matching keys from the left table.

```
df1
```

	id	Product
0	1	A
1	2	A
2	3	A
3	4	B
4	5	B
5	6	B

```
df2
```

	id	State
0	2	Portsmouth
1	4	Portsmouth
2	6	Southampton
3	7	Southampton

```
Newframe =pd.merge(df1, df2, on='id', how='right')
Newframe
```

	id	Product	State
0	2	A	Portsmouth
1	4	B	Portsmouth
2	6	B	Southampton
3	7	NaN	Southampton

Concatenating DataFrames in Pandas Library

- When we need to combine two or more datasets.

```
result_filtered1
```

	something	a	b	c	d	message
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

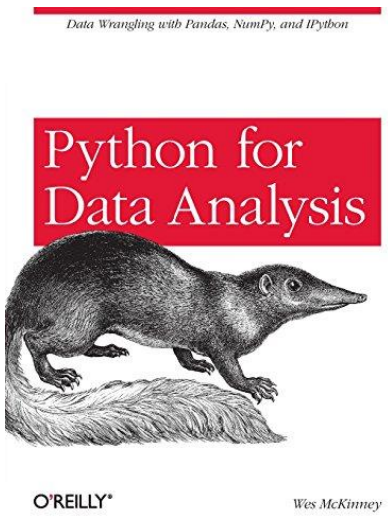
```
result_filtered2
```

	something	a	b	c	d	message
2	three	9	10	11.0	12	foo

```
pd.concat([result_filtered1, result_filtered2])
```

	something	a	b	c	d	message
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo
2	three	9	10	11.0	12	foo

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 Week06_loading.ipynb file, and run it to learn new points.
- Please read the practical sheet (Week06_Practicals.pdf) and do the exercise.