Data Loading, Storage, and File Formats

Part 3

Reading and Writing Data in Text Format

Part 3

JSON Data

- JSON (short for JavaScript Object Notation) has become one of the standard formats for sending data by HTTP request between web browsers and other applications.
- It is a much more free-form data format than a tabular text form like CSV.
- Here is an example:

In [38]:
```
obj = """
{"name": "Wes",
 "places_lived": ["United States", "Spain", "Germany"],
 "pet": null,
 "siblings": [{"name": "Scott", "age": 30, "pets": ["Zeus", "Zuko"]},
              {"name": "Katie", "age": 38,
               "pets": ["Sixes", "Stache", "Cisco"]}]
}
"""
```

- There are several Python libraries for reading and writing JSON data.
- `json` will be used here, as it is built into the Python standard library.
- To convert a JSON string to Python form, use `json.loads`:

```
In [39]: import json

In [40]: result = json.loads(obj)

In [41]: result

Out[41]: {'name': 'Wes',
          'places_lived': ['United States', 'Spain', 'Germany'],
          'pet': None,
          'siblings': [{'name': 'Scott', 'age': 30, 'pets': ['Zeus', 'Zuko']},
           {'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Stache', 'Cisco']}]}
```

- `json.dumps`, on the other hand, converts a Python object back to JSON:

```
In [42]: asjson = json.dumps(result)

In [43]: asjson

Out[43]: '{"name": "Wes", "places_lived": ["United States", "Spain", "Germany"], "pet": null, "siblings": [{"name": "Scott", "age
": 30, "pets": ["Zeus", "Zuko"]}, {"name": "Katie", "age": 38, "pets": ["Sixes", "Stache", "Cisco"]}]}'
```

- How you convert a JSON object or list of objects to a DataFrame or some other data structure for analysis will be up to you.
- Conveniently, you can pass a list of dicts (which were previously JSON objects) to the DataFrame constructor and select a subset of the data fields:

```
In [44]: siblings = pd.DataFrame(result['siblings'], columns=['name', 'age'])

In [45]: siblings
```

Out[45]:

	name	age
0	Scott	30
1	Katie	38

- The `pandas.read_json` can automatically convert JSON datasets in specific arrangements into a Series or DataFrame.

```
In [46]: !cat examples/example.json
```

```
[{"a": 1, "b": 2, "c": 3},
 {"a": 4, "b": 5, "c": 6},
 {"a": 7, "b": 8, "c": 9}]
```

- The default options for `pandas.read_json` assume that each object in the JSON array is a row in the table:

```
In [47]: data = pd.read_json('examples/example.json')
```

```
In [48]: data
```

```
Out[48]:
```

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

- If you need to export data from pandas to JSON, one way is to use the `to_json` methods on Series and DataFrame:

```
In [49]: print(data.to_json())
{"a":{"0":1,"1":4,"2":7},"b":{"0":2,"1":5,"2":8},"c":{"0":3,"1":6,"2":9}}

In [50]: print(data.to_json(orient='records'))
[{"a":1,"b":2,"c":3},{"a":4,"b":5,"c":6},{"a":7,"b":8,"c":9}]
```

XML and HTML: Web Scraping

- Python has many libraries for reading and writing data in the ubiquitous HTML and XML formats.
- Examples include lxml, Beautiful Soup, and html5lib.
- While lxml is comparatively much faster in general, the other libraries can better handle malformed HTML or XML files.

- pandas has a built-in function, `read_html`, which uses libraries like lxml and Beautiful Soup to automatically parse tables out of HTML files as DataFrame objects.
- To show how this works, I downloaded an HTML file (used in the pandas documentation) from the United States FDIC government agency showing bank failures.
- First, you must install some additional libraries used by `read_html`:

```
(base) joshua@joshua-Virtual-Machine:~$ conda list | grep lxml
lxml                      4.3.2            py37hefd8a0e_0
(base) joshua@joshua-Virtual-Machine:~$ conda list | grep beautifulsoup4
beautifulsoup4            4.7.1                    py37_1
(base) joshua@joshua-Virtual-Machine:~$ conda list | grep html5lib
html5lib                  1.0.1                    py37_0
(base) joshua@joshua-Virtual-Machine:~$
```

- The `pandas.read_html` function has a number of options, but by default it searches for and attempts to parse all tabular data contained within `<table>` tags.
- The result is a list of DataFrame objects:

```
In [51]: tables = pd.read_html('examples/fdic_failed_bank_list.html')

In [52]: len(tables)
Out[52]: 1

In [53]: failures = tables[0]

In [54]: failures.head()
```

Out[54]:

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
0	Allied Bank	Mulberry	AR	91	Today's Bank	September 23, 2016	November 17, 2016
1	The Woodbury Banking Company	Woodbury	GA	11297	United Bank	August 19, 2016	November 17, 2016
2	First CornerStone Bank	King of Prussia	PA	35312	First-Citizens Bank & Trust Company	May 6, 2016	September 6, 2016
3	Trust Company Bank	Memphis	TN	9956	The Bank of Fayette County	April 29, 2016	September 6, 2016
4	North Milwaukee State Bank	Milwaukee	WI	20364	First-Citizens Bank & Trust Company	March 11, 2016	June 16, 2016

- From here we could proceed to do some data cleaning and analysis, like computing the number of bank failures by year:

```
In [55]: close_timestamps = pd.to_datetime(failures['Closing Date'])

In [56]: close_timestamps.dt.year.value_counts()

Out[56]: 2010    157
         2009    140
         2011     92
         2012     51
         2008     25
                ...
         2004      4
         2001      4
         2007      3
         2003      3
         2000      2
         Name: Closing Date, Length: 15, dtype: int64
```

Parsing XML with lxml.objectify

- The New York Metropolitan Transportation Authority (MTA) publishes a number of data series about its bus and train services.
- Here we'll look at the performance data, which is contained in a set of XML files.

- Each train or bus service has a different file (like *Performance_MNR.xml* for the Metro-North Railroad) containing monthly data as a series of XML records that look like this:

```
<INDICATOR>
  <INDICATOR_SEQ>373889</INDICATOR_SEQ>
  <PARENT_SEQ></PARENT_SEQ>
  <AGENCY_NAME>Metro-North Railroad</AGENCY_NAME>
  <INDICATOR_NAME>Escalator Availability</INDICATOR_NAME>
  <DESCRIPTION>Percent of the time that escalators are operational
  systemwide. The availability rate is based on physical observations performed
  the morning of regular business days only. This is a new indicator the agency
  began reporting in 2009.</DESCRIPTION>
  <PERIOD_YEAR>2011</PERIOD_YEAR>
  <PERIOD_MONTH>12</PERIOD_MONTH>
  <CATEGORY>Service Indicators</CATEGORY>
  <FREQUENCY>M</FREQUENCY>
  <DESIRED_CHANGE>U</DESIRED_CHANGE>
  <INDICATOR_UNIT>%</INDICATOR_UNIT>
  <DECIMAL_PLACES>1</DECIMAL_PLACES>
  <YTD_TARGET>97.00</YTD_TARGET>
  <YTD_ACTUAL></YTD_ACTUAL>
  <MONTHLY_TARGET>97.00</MONTHLY_TARGET>
  <MONTHLY_ACTUAL></MONTHLY_ACTUAL>
</INDICATOR>
```

- Using `lxml.objectify`, we parse the file and get a reference to the root node of the XML file with `getroot`:

```
In [57]: from lxml import objectify

         path = 'datasets/mta_perf/Performance_MNR.xml'
         parsed = objectify.parse(open(path))
         root = parsed.getroot()
```

- `root.INDICATOR` returns a generator yielding each `<INDICATOR>` XML element.
- For each record, we can populate a dict of tag names (like `YTD_ACTUAL`) to data values (excluding a few tags):

```
In [58]: data = []

         skip_fields = ['PARENT_SEQ', 'INDICATOR_SEQ',
                        'DESIRED_CHANGE', 'DECIMAL_PLACES']

         for elt in root.INDICATOR:
             el_data = {}
             for child in elt.getchildren():
                 if child.tag in skip_fields:
                     continue
                 el_data[child.tag] = child.pyval
             data.append(el_data)
```

- Lastly, convert this list of dicts into a DataFrame:

```
In [59]: perf = pd.DataFrame(data)

In [60]: perf.head()
```

Out[60]:

	AGENCY_NAME	CATEGORY	DESCRIPTION	FREQUENCY	INDICATOR_NAME	INDICATOR_UNIT	MONTHLY_ACTUAL	MONTHLY_TARGET	PERIOD_MONTH
0	Metro-North Railroad	Service Indicators	Percent of commuter trains that arrive at thei...	M	On-Time Performance (West of Hudson)	%	96.9	95	1
1	Metro-North Railroad	Service Indicators	Percent of commuter trains that arrive at thei...	M	On-Time Performance (West of Hudson)	%	95	95	2
2	Metro-North Railroad	Service Indicators	Percent of commuter trains that arrive at thei...	M	On-Time Performance (West of Hudson)	%	96.9	95	3
3	Metro-North Railroad	Service Indicators	Percent of commuter trains that arrive at thei...	M	On-Time Performance (West of Hudson)	%	98.3	95	4
4	Metro-North Railroad	Service Indicators	Percent of commuter trains that arrive at thei...	M	On-Time Performance (West of Hudson)	%	95.8	95	5

- XML data can get much more complicated than this example.
- Each tag can have metadata, too.
- Consider an HTML link tag, which is also valid XML:

```
In [61]: from io import StringIO
         tag = '<a href="http://www.google.com">Google</a>'
         root = objectify.parse(StringIO(tag)).getroot()
```

- You can now access any of the fields (like `href`) in the tag or the link text:

```
In [62]: root
Out[62]: <Element a at 0x7f22e3c75ec8>

In [63]: root.get('href')
Out[63]: 'http://www.google.com'

In [64]: root.text
Out[64]: 'Google'
```