# Data Loading, Storage, and File Formats

Part 5

# Interacting with Web APIs

- Many websites have public APIs providing data feeds via JSON or some other format.
- There are a number of ways to access these APIs from Python; one easy-to-use method is the `requests` package.

- To find the last 30 GitHub issues for pandas on GitHub, we can make a `GET` HTTP request using the add-on `requests` library:

```
In [88]: import requests

In [89]: url = 'https://api.github.com/repos/pandas-dev/pandas/issues'

In [90]: resp = requests.get(url)

In [91]: resp
Out[91]: <Response [200]>
```

- The Response object's `json` method will return a dictionary containing JSON parsed into native Python objects:

```
In [92]: data = resp.json()

In [93]: data[0]['title']
Out[93]: 'Old date is saved as a random negative number in Excel file'
```

- Each element in `data` is a dictionary containing all of the data found on a GitHub issue page (except for the comments).
- We can pass `data` directly to DataFrame and extract fields of interest:

```
In [94]: issues = pd.DataFrame(data, columns=['number', 'title',
                                               'labels', 'state'])

In [95]: issues
```

Out[95]:

| | number | title | labels | state |
|---|---|---|---|---|
| **0** | 26291 | Old date is saved as a random negative number ... | [] | open |
| **1** | 26290 | Pandas UTF-8 handling is rubbish | [] | open |
| **2** | 26289 | CLN: remove compat.lrange, part II | [] | open |
| **3** | 26288 | Sparse DF with time column cannot concat | [] | open |
| **4** | 26287 | Remove Panel References | [{'id': 211029535, 'node_id': 'MDU6TGFiZWwyMTE... | open |
| **...** | ... | ... | ... | ... |
| **25** | 26234 | DOC: Fix validation error type `RT01` and chec... | [{'id': 134699, 'node_id': 'MDU6TGFiZWwxMzQ2OT... | open |
| **26** | 26229 | BUG: Fix #10355, std() groupby calculation | [{'id': 76811, 'node_id': 'MDU6TGFiZWw3NjgxMQ=... | open |
| **27** | 26228 | Fix .transform crash when SeriesGroupBy is em... | [{'id': 76811, 'node_id': 'MDU6TGFiZWw3NjgxMQ=... | open |
| **28** | 26227 | Sparse dataframe fails when astype() is called... | [{'id': 31404521, 'node_id': 'MDU6TGFiZWwzMTQw... | open |
| **29** | 26225 | [PERF] Get rid of MultiIndex conversion in Int... | [{'id': 150096370, 'node_id': 'MDU6TGFiZWwxNTA... | open |

30 rows × 4 columns

# Interacting with Databases

- Loading data from SQL into a DataFrame is fairly straightforward, and pandas has some functions to simplify the process.
- As an example, a SQLite database will be created using Python's built-in `sqlite3` driver:

```
In [96]: import sqlite3

In [97]: query = """
         CREATE TABLE test
         (a VARCHAR(20), b VARCHAR(20),
          c REAL,        d INTEGER
         );"""

In [98]: con = sqlite3.connect('mydata.sqlite')

In [99]: con.execute(query)
Out[99]: <sqlite3.Cursor at 0x7f22dfe42a40>

In [100]: con.commit()
```

- Then, insert a few rows of data:

```
In [101]: data = [('Atlanta', 'Georgia', 1.25, 6),
                  ('Tallahassee', 'Florida', 2.6, 3),
                  ('Sacramento', 'California', 1.7, 5)]

In [102]: stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"

In [103]: con.executemany(stmt, data)
Out[103]: <sqlite3.Cursor at 0x7f22e04a8570>

In [104]: con.commit()
```

- Most Python SQL drivers (PyODBC, psycopg2, MySQLdb, pymssql, etc.) return a list of tuples when selecting data from a table:

```
In [105]: cursor = con.execute('select * from test')

In [106]: rows = cursor.fetchall()

In [107]: rows

Out[107]: [('Atlanta', 'Georgia', 1.25, 6),
           ('Tallahassee', 'Florida', 2.6, 3),
           ('Sacramento', 'California', 1.7, 5)]
```

- You can pass the list of tuples to the DataFrame constructor, but you also need the column names, contained in the cursor's `description` attribute:

```
In [108]: cursor.description
Out[108]: (('a', None, None, None, None, None, None),
           ('b', None, None, None, None, None, None),
           ('c', None, None, None, None, None, None),
           ('d', None, None, None, None, None, None))

In [109]: pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
Out[109]:
```

| | a | b | c | d |
|---|---|---|---|---|
| **0** | Atlanta | Georgia | 1.25 | 6 |
| **1** | Tallahassee | Florida | 2.60 | 3 |
| **2** | Sacramento | California | 1.70 | 5 |

- This is quite a bit of munging that you'd rather not repeat each time you query the database.
- The SQLAlchemy project is a popular Python SQL toolkit that abstracts away many of the common differences between SQL databases.
- pandas has a `read_sql` function that enables you to read data easily from a general SQLAlchemy connection.
- Here, we'll connect to the same SQLite database with SQLAlchemy and read data from the table created before:

```
In [110]: import sqlalchemy as sqla

In [111]: db = sqla.create_engine('sqlite:///mydata.sqlite')

In [112]: pd.read_sql('select * from test', db)
```

Out[112]:

| | a | b | c | d |
|---|---|---|---|---|
| 0 | Atlanta | Georgia | 1.25 | 6 |
| 1 | Tallahassee | Florida | 2.60 | 3 |
| 2 | Sacramento | California | 1.70 | 5 |