Data Loading, Storage, and File Formats

functions for reading tabular data into a dataframe .
.
read_csv -- Load data from a file, URL, or file-like object; use comma as default delimiter
read_fwf -- Read data in fixed-width column format
read_clipboard -- Version of read_csv that reads data from the clipboard; useful for converiting tables from web pages
read_excel -- read tabular data from an XLS or XLSX file
read_hdf -- Read HDF5 files written by pandas
read_html -- Read all tables found in the given HTML documnet
read_json -- Read data from a JSON 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 as a pandas DataFrame
read_stata -- Read a dataset from stata file format
read_feather -- Read the Feather binary file format

In [2]:

```
import pandas as pd
df = pd.read_csv('exampledata/ex1.csv')
df
```

Out[2]:

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

A file will not always have a header row. You can load a dataframe without a header

In [3]:

```
df = pd.read_csv('exampledata/ex1.csv', header=None)
df
```

Out[3]:

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

to specify the header yourself you can use the following code

In [4]:

```
df2 = pd.read_csv('exampledata/ex1.csv', names=['a', 'b', 'c', 'd', 'message'])
df2
```

Out[4]:

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

Suppose you wanted a column to be the index.

In [5]:

```
df = pd.read_csv('exampledata/ex1.csv', index_col='message')
df
```

Out[5]:

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

In [32]:

```

```

Out[32]:

```
message
hello     2
world     6
foo      10
Name: b, dtype: int64
```

should your data file represent missing cells in a certain way e.g. Na, missing, Nan you can specify this to pandas and it
will replace it with its own missing format.

In [6]:

```
result = pd.read_csv('exampledata/ex1.csv', na_values=['Null']) #pandas will replace th
e missing value with its own sentinel NaN
```

Some Read CSV functions

path -- String indicating filesystem location, URL, or file-like object
sep or delimiter -- Character sequence or regular expression to use to split fields in each row
header -- Row number to use as column names; defaults to 0 (frist row), but should be None if there is no header row index_col -- Column number or names to use as the row index in the result; can be a single name/nmber or a list of them for a hierachical index
names -- List of column names for result, combine with header=None skiprows -- Number of rows at beginning of file to ignore or list of row numbers to skip. na_values -- Sequence of values to replace with na_valuescomment -- Characters to split comments of the end of lines. parse_dates -- Attempt to parse data to datetime; False by default. If True, will attempt to parse all columns, Otherwise can specify a list of column numbers or name to parse. If
element of list is tuple or list, will combine multiple columns together and parse to date dayfirst -- When parsing potentially ambiguous dates, treat as international format, False by default
nrows -- Number of rows to read from beginning of file iterator -- Return a TextParser object for reading file piecemeal
chinksize -- for iteration, size of file chunks
skip_footer -- number of lines to ignore at end of file
.
.
.
.

Reading Text Files in Pieces

In [7]:

```
pd.read_csv('exampledata/homes.csv', nrows=5) #reads just 5 rows
```

Out[7]:

	Sell	"List"	"Living"	"Rooms"	"Beds"	"Baths"	"Age"	"Acres"	"Taxes"
0	142	160	28	10	5	3	60	0.28	3167
1	175	180	18	8	4	1	12	0.43	4033
2	129	132	13	6	3	1	41	0.33	1471
3	138	140	17	7	3	1	22	0.46	3204
4	232	240	25	8	4	3	5	2.05	3613

In [8]:

```
# to read in a number of chunks
chunker = pd.read_csv('exampledata/homes.csv', chunksize=10) # Specifies the number of
 rows per chunk
chunker
```

Out[8]:

```
<pandas.io.parsers.TextFileReader at 0x2707a6f2700>
```

In [9]:

```
tot = pd.Series([])
for piece in chunker:
    #print('Piece value counts: \n', piece['Sell'].value_counts())
    tot = tot.add(piece['Sell'].value_counts(), fill_value=0)#.value counts returns a c
ount of unique values. fill_value specifies what value to give missing data.

tot = tot.sort_values(ascending=False)
tot
```

Out[9]:

```
152    2.0
129    2.0
157    2.0
175    2.0
180    2.0
135    2.0
110    2.0
148    2.0
123    1.0
127    1.0
128    1.0
111    1.0
133    1.0
150    1.0
136    1.0
106    1.0
138    1.0
142    1.0
143    1.0
145    1.0
89     1.0
146    1.0
567    1.0
151    1.0
190    1.0
271    1.0
265    1.0
247    1.0
234    1.0
232    1.0
212    1.0
207    1.0
185    1.0
293    1.0
184    1.0
183    1.0
170    1.0
167    1.0
166    1.0
165    1.0
153    1.0
87     1.0
dtype: float64
```

Writing data to Text Format

In [10]:

```
data = pd.read_csv('exampledata/homes.csv')
data.to_csv('exampledata/ouput.csv')

#To use other delimeter formats
import sys
data.to_csv(sys.stdout,  sep='|')
```

```
|Sell|" ""List"""|" ""Living"""|" ""Rooms"""|" ""Beds"""|" ""Baths"""|"
""Age"""|" ""Acres"""|" ""Taxes"""
0|142|160|28|10|5|3|60|0.28|3167
1|175|180|18|8|4|1|12|0.43|4033
2|129|132|13|6|3|1|41|0.33|1471
3|138|140|17|7|3|1|22|0.46|3204
4|232|240|25|8|4|3|5|2.05|3613
5|135|140|18|7|4|3|9|0.57|3028
6|150|160|20|8|4|3|18|4.0|3131
7|207|225|22|8|4|2|16|2.22|5158
8|271|285|30|10|5|2|30|0.53|5702
9|89|90|10|5|3|1|43|0.3|2054
10|153|157|22|8|3|3|18|0.38|4127
11|87|90|16|7|3|1|50|0.65|1445
12|234|238|25|8|4|2|2|1.61|2087
13|106|116|20|8|4|1|13|0.22|2818
14|175|180|22|8|4|2|15|2.06|3917
15|165|170|17|8|4|2|33|0.46|2220
16|166|170|23|9|4|2|37|0.27|3498
17|136|140|19|7|3|1|22|0.63|3607
18|148|160|17|7|3|2|13|0.36|3648
19|151|153|19|8|4|2|24|0.34|3561
20|180|190|24|9|4|2|10|1.55|4681
21|293|305|26|8|4|3|6|0.46|7088
22|167|170|20|9|4|2|46|0.46|3482
23|190|193|22|9|5|2|37|0.48|3920
24|184|190|21|9|5|2|27|1.3|4162
25|157|165|20|8|4|2|7|0.3|3785
26|110|115|16|8|4|1|26|0.29|3103
27|135|145|18|7|4|1|35|0.43|3363
28|567|625|64|11|4|4|4|0.85|12192
29|180|185|20|8|4|2|11|1.0|3831
30|183|188|17|7|3|2|16|3.0|3564
31|185|193|20|9|3|2|56|6.49|3765
32|152|155|17|8|4|1|33|0.7|3361
33|148|153|13|6|3|2|22|0.39|3950
34|152|159|15|7|3|1|25|0.59|3055
35|146|150|16|7|3|1|31|0.36|2950
36|170|190|24|10|3|2|33|0.57|3346
37|127|130|20|8|4|1|65|0.4|3334
38|265|270|36|10|6|3|33|1.2|5853
39|157|163|18|8|4|2|12|1.13|3982
40|128|135|17|9|4|1|25|0.52|3374
41|110|120|15|8|4|2|11|0.59|3119
42|123|130|18|8|4|2|43|0.39|3268
43|212|230|39|12|5|3|202|4.29|3648
44|145|145|18|8|4|2|44|0.22|2783
45|129|135|10|6|3|1|15|1.0|2438
46|143|145|21|7|4|2|10|1.2|3529
47|247|252|29|9|4|2|4|1.25|4626
48|111|120|15|8|3|1|97|1.11|3205
49|133|145|26|7|3|1|42|0.36|3059
```

In [11]:

```
data.to_csv(sys.stdout, na_rep='NULL') # will replace all empty values with 'NULL'
# with no other options specified the row and column labels will be written to the file
# however you can disable them
data.to_csv('exampledata/output.csv', index=False, header=False) #this file has been sa
ved with no index or header
```

```
,Sell," ""List""," ""Living""," ""Rooms""," ""Beds""," ""Baths""," 
""Age""," ""Acres""," ""Taxes""
0,142,160,28,10,5,3,60,0.28,3167
1,175,180,18,8,4,1,12,0.43,4033
2,129,132,13,6,3,1,41,0.33,1471
3,138,140,17,7,3,1,22,0.46,3204
4,232,240,25,8,4,3,5,2.05,3613
5,135,140,18,7,4,3,9,0.57,3028
6,150,160,20,8,4,3,18,4.0,3131
7,207,225,22,8,4,2,16,2.22,5158
8,271,285,30,10,5,2,30,0.53,5702
9,89,90,10,5,3,1,43,0.3,2054
10,153,157,22,8,3,3,18,0.38,4127
11,87,90,16,7,3,1,50,0.65,1445
12,234,238,25,8,4,2,2,1.61,2087
13,106,116,20,8,4,1,13,0.22,2818
14,175,180,22,8,4,2,15,2.06,3917
15,165,170,17,8,4,2,33,0.46,2220
16,166,170,23,9,4,2,37,0.27,3498
17,136,140,19,7,3,1,22,0.63,3607
18,148,160,17,7,3,2,13,0.36,3648
19,151,153,19,8,4,2,24,0.34,3561
20,180,190,24,9,4,2,10,1.55,4681
21,293,305,26,8,4,3,6,0.46,7088
22,167,170,20,9,4,2,46,0.46,3482
23,190,193,22,9,5,2,37,0.48,3920
24,184,190,21,9,5,2,27,1.3,4162
25,157,165,20,8,4,2,7,0.3,3785
26,110,115,16,8,4,1,26,0.29,3103
27,135,145,18,7,4,1,35,0.43,3363
28,567,625,64,11,4,4,4,0.85,12192
29,180,185,20,8,4,2,11,1.0,3831
30,183,188,17,7,3,2,16,3.0,3564
31,185,193,20,9,3,2,56,6.49,3765
32,152,155,17,8,4,1,33,0.7,3361
33,148,153,13,6,3,2,22,0.39,3950
34,152,159,15,7,3,1,25,0.59,3055
35,146,150,16,7,3,1,31,0.36,2950
36,170,190,24,10,3,2,33,0.57,3346
37,127,130,20,8,4,1,65,0.4,3334
38,265,270,36,10,6,3,33,1.2,5853
39,157,163,18,8,4,2,12,1.13,3982
40,128,135,17,9,4,1,25,0.52,3374
41,110,120,15,8,4,2,11,0.59,3119
42,123,130,18,8,4,2,43,0.39,3268
43,212,230,39,12,5,3,202,4.29,3648
44,145,145,18,8,4,2,44,0.22,2783
45,129,135,10,6,3,1,15,1.0,2438
46,143,145,21,7,4,2,10,1.2,3529
47,247,252,29,9,4,2,4,1.25,4626
48,111,120,15,8,3,1,97,1.11,3205
49,133,145,26,7,3,1,42,0.36,3059
```

Working With Delimited Formats

In [12]:

```
bad_data = pd.read_csv('exampledata/bad_csv.csv')
bad_data
```

Out[12]:

	a	b	c
0	1	2	3
1	1	2	3

As you can see the elements are represented in double quote marks. This is more common than usual. in this case it may be necessary to manually read in the file using pythons built-in csv module.

In [13]:

```
!cat exampledata/bad_csv.csv
```

```
"a","b","c"
"1","2","3"
"1","2","3"
```

In [14]:

```
import csv
f = open('exampledata/bad_csv.csv')
reader = csv.reader(f)

for line in reader:
    print(line)
```

```
['a', 'b', 'c']
['1', '2', '3']
['1', '2', '3']
```

from here, it's up to you to do the wrangling.

In [15]:

```
with open('exampledata/bad_csv.csv') as f:
    lines = list(csv.reader(f))

header, values = lines[0], lines[1:] #lines 0 is the header. lines 1 and onwards are th
e values.

# Now create a dictionary of data columns
# *values collects all the positions in the tuple
# zip joins a list all positional elements together in a tuple

data_dict = {h: v for h, v in zip(header, zip(*values))}
data_dict
```

Out[15]:

```
{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

csv files can come in many different forms you can specify your own by creating a class like the one below

In [16]:

```
class my_dialect(csv.Dialect):
    lineterminator = '\n'
    delimiter = ';'
    quotechar = '"'
    quoting = csv.QUOTE_MINIMAL
f = open('exampledata/homes.csv')
reader = csv.reader(f, delimiter='|')

with open('exampledata/mydata.csv', 'w') as f:
    writer = csv.writer(f, dialect=my_dialect)
    writer.writerow(('hello', 'this', 'is', 'a', 'string'))
    writer.writerow(('1', '2', '3', '4'))
    writer.writerow(('%', '5', '6', '7'))

!cat 'exampledata/mydata.csv'
```

```
hello;this;is;a;string
1;2;3;4
%;5;6;7
```

JSON Data

JSON JavaScript Object Notations has become the standard format for sending HTTPS requests. (asking for and receiving data from servers through the web) below is an example

In [17]:

```
obj = """
  {"name": ["Ollie", "NaN"],
  "places_lived": ["battle", "robertsbridge"],
  "pet": ["Mac", "philip"],
  "siblings":["thomas", "Giles"]
  }"""
```

JSON is very similar to python code apart from some nuances. All keys must be strings. Null is represented by null.

In [18]:

```
import json
result = json.loads(obj) # loads a JSON object into python
result
```

Out[18]:

```
{'name': ['Ollie', 'NaN'],
 'places_lived': ['battle', 'robertsbridge'],
 'pet': ['Mac', 'philip'],
 'siblings': ['thomas', 'Giles']}
```

In [19]:

```
asjson = json.dumps(result) #converts to JSON
asjson
```

Out[19]:

```
'{"name": ["Ollie", "NaN"], "places_lived": ["battle", "robertsbridge"],
"pet": ["Mac", "philip"], "siblings": ["thomas", "Giles"]}'
```

once you have received the JSON into data you can load it into a dataframe like any python dicts

In [20]:

```
df = pd.DataFrame(result, index=result['pet'])
df[['name', 'places_lived']]
```

Out[20]:

	name	places_lived
Mac	Ollie	battle
philip	NaN	robertsbridge

XML and HTML: Web Scraping

pandas has a built-in function, read_html which uses a few different methods to read tables out of HTML files as a DataFrame. you must have 'lxml' and 'beautifulsoup4 html5lib' installed. You can use pip to install them

In [24]:

```
tables = pd.read_html('exampledata/test_webpage.html')
tables
```

Out[24]:

```
[  First name    Last name  Age
 0       Tinu     Elejogun   14
 1  Blaszczyk  Kostrzewski   25
 2       Lily    McGarrett   18
 3  Olatunkbo     Chijiaku   22
 4   Adrienne     Anthoula   22
 5     Axelia   Athanasios   22
 6  Jon-Kabat         Zinn   22
 7    Thabang        Mosoa   15
 8   Kgaogelo        Mosoa   11,
    ×  1  2  3
 0  1  1  2  3
 1  2  2  4  6
 2  3  3  6  9,
                          0             1  \
 0  Standard Representation           NaN
 1                   2 3 1           NaN
 2              Health Risk  Flammability
 3                  Level 3       Level 2

                                                 2        3
 0                            Tabular Representation      NaN
 1  Risk levels of hazardous materials in this fac...      NaN
 2                                       Reactivity  Special
 3                                          Level 1      NaN  ,
   Health Risk Flammability Reactivity  Special
 0     Level 3      Level 2    Level 1      NaN,
     0                                          1
 0 NaN  Look up table in Wiktionary, the free dictionary.,
   vteVisualization of technical information  \
 0                                    Fields
 1                               Image types
 2                                    People
 3                           Related topics

         vteVisualization of technical information.1
 0  Biological data visualization Chemical imaging...
 1  Chart Diagram Engineering drawing Graph of a f...
 2  Jacques Bertin Cynthia Brewer Stuart Card Shee...
 3  Cartography Chartjunk Computer graphics in com...  ]
```

above shows all of the tabular data from the html file as a DataFrame object. the read_html method searches the code to find the
tags to find tabular data and then loads it.

In [27]:

Out[27]:

	First name	Last name	Age
0	Tinu	Elejogun	14
1	Blaszczyk	Kostrzewski	25
2	Lily	McGarrett	18
3	Olatunkbo	Chijiaku	22
4	Adrienne	Anthoula	22
5	Axelia	Athanasios	22
6	Jon-Kabat	Zinn	22
7	Thabang	Mosoa	15
8	Kgaogelo	Mosoa	11

Interacting with Web APIs

a number of Websites provide APIs for data collection via JSON. There are a number of ways to accsess these APIs from python. the following is a simple way.

to find the last 30 GitHub issues for pandas we can make a get HTTP request using the add-on requests library:

In [52]:

```
import requests
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
resp = requests.get(url)
resp
```

Out[52]:

```
<Response [200]>
```

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

In [60]:

```
data = resp.json()
data[1]
```

Out[60]:


```
{'url': 'https://api.github.com/repos/pandas-dev/pandas/issues/35388',
 'repository_url': 'https://api.github.com/repos/pandas-dev/pandas',
 'labels_url': 'https://api.github.com/repos/pandas-dev/pandas/issues/35388/label
s{/name}',
 'comments_url': 'https://api.github.com/repos/pandas-dev/pandas/issues/35388/com
ments',
 'events_url': 'https://api.github.com/repos/pandas-dev/pandas/issues/35388/event
s',
 'html_url': 'https://github.com/pandas-dev/pandas/issues/35388',
 'id': 664156158,
 'node_id': 'MDU6SXNzdWU2NjQxNTYxNTg=',
 'number': 35388,
 'title': "BUG: timezone-aware DatetimeIndex doesn't handle offsets very well",
 'user': {'login': 'rwijtvliet',
  'id': 4106013,
  'node_id': 'MDQ6VXNlcjQxMDYwMTM=',
  'avatar_url': 'https://avatars1.githubusercontent.com/u/4106013?v=4',
  'gravatar_id': '',
  'url': 'https://api.github.com/users/rwijtvliet',
  'html_url': 'https://github.com/rwijtvliet',
  'followers_url': 'https://api.github.com/users/rwijtvliet/followers',
  'following_url': 'https://api.github.com/users/rwijtvliet/following{/other_use
r}',
  'gists_url': 'https://api.github.com/users/rwijtvliet/gists{/gist_id}',
  'starred_url': 'https://api.github.com/users/rwijtvliet/starred{/owner}{/rep
o}',
  'subscriptions_url': 'https://api.github.com/users/rwijtvliet/subscriptions',
  'organizations_url': 'https://api.github.com/users/rwijtvliet/orgs',
  'repos_url': 'https://api.github.com/users/rwijtvliet/repos',
  'events_url': 'https://api.github.com/users/rwijtvliet/events{/privacy}',
  'received_events_url': 'https://api.github.com/users/rwijtvliet/received_event
s',
  'type': 'User',
  'site_admin': False},
 'labels': [{'id': 76811,
   'node_id': 'MDU6TGFiZWw3NjgxMQ==',
   'url': 'https://api.github.com/repos/pandas-dev/pandas/labels/Bug',
   'name': 'Bug',
   'color': 'e10c02',
   'default': False,
   'description': None},
  {'id': 1954720290,
   'node_id': 'MDU6TGFiZWwxOTU0NzIwMjkw',
   'url': 'https://api.github.com/repos/pandas-dev/pandas/labels/Needs%20Triage',
   'name': 'Needs Triage',
   'color': '0052cc',
   'default': False,
   'description': 'Issue that has not been reviewed by a pandas team member'}],
 'state': 'open',
 'locked': False,
 'assignee': None,
 'assignees': [],
 'milestone': None,
 'comments': 0,
 'created_at': '2020-07-23T02:07:01Z',
 'updated_at': '2020-07-23T02:09:34Z',
 'closed_at': None,
 'author_association': 'NONE',
 'active_lock_reason': None,
 'body': "- [x] I have checked that this issue has not already been reported.\r\n
\r\n- [x] I have confirmed this bug exists on the latest version of pandas.\r\n\r
\n- [ ] (optional) I have confirmed this bug exists on the master branch of panda
s.\r\n\r\n---\r\n\r\n#### Code Sample, a copy-pastable example\r\n\r\n```python\r
\nimport pandas as pd\r\ni = pd.date_range('2020-03-28', periods=4, freq='D', tz
='Europe/Berlin')\r\ni # DatetimeIndex(['2020-03-28 00:00:00+01:00',  '2020-03-29
00:00:00+01:00', '2020-03-30 00:00:00+02:00', '2020-03-31 00:00:00+02:00'], dtype
='datetime64[ns, Europe/Berlin]', freq='D')\r\ni[0] + i.freq == i[1] #True\r\ni
```

```
[2] + i.freq == i[3] #True\r\ni[1] + i.freq == i[2] #False (!)\r\n```\r\n\r\n#### Problem description\r\n\r\nVariably-spaced timestamps are not handled well, if the variation is caused by DST. The result of `i[1] + i.freq` is \r\n`Timestamp('2020-03-30 01:00:00+0200', tz='Europe/Berlin', freq='D')`, whereas \r\n`Timestamp('2020-03-30 00:00:00+0200', tz='Europe/Berlin', freq='D')` was expected.\r\n\r\n This is in contrast to timestamps where the variation is caused by e.g. months being different lengths, which *are* handled correctly.\r\n\r\n#### Output of ``pd.show_versions()``\r\n\r\n<details>\r\n\r\nINSTALLED VERSIONS\r\n------------------\ncommit           : None\r\npython           : 3.8.3.final.0\r\npython-bits      : 64\r\nOS               : Windows\r\nOS-release       : 10\r\nmachine          : AMD64\r\nprocessor        : Intel64 Family 6 Model 158 Stepping 10, GenuineIntel\r\nbyteorder        : little\r\nLC_ALL           : None\r\nLANG             : en\r\nLOCALE           : de_DE.cp1252\r\n\r\npandas           : 1.0.5\r\nnumpy            : 1.18.5\r\npytz             : 2020.1\r\ndateutil         : 2.8.1\r\npip              : 20.1.1\r\nsetuptools       : 49.2.0.post20200714\r\nCython           : None\r\npytest           : 5.4.3\r\nhypothesis       : None\r\nsphinx           : None\r\nblosc            : None\r\nfeather          : None\r\nxlsxwriter       : None\r\nlxml.etree       : None\r\nhtml5lib         : 1.0.1\r\npymysql          : None\r\npsycopg2         : None\r\njinja2           : 2.11.2\r\nIPython          : 7.16.1\r\npandas_datareader: None\r\nbs4              : 4.9.1\r\nbottleneck       : None\r\nfastparquet      : None\r\ngcsfs            : None\r\nlxml.etree       : None\r\nmatplotlib       : 3.2.2\r\nnumexpr          : None\r\nodfpy            : None\r\nopenpyxl         : 3.0.4\r\npandas_gbq       : None\r\npyarrow          : None\r\npytables         : None\r\npytest           : 5.4.3\r\npyxlsb           : None\r\ns3fs             : None\r\nscipy            : 1.5.0\r\nsqlalchemy       : 1.3.18\r\ntables           : None\r\ntabulate         : None\r\nxarray           : None\r\nxlrd             : 1.2.0\r\nxlwt             : None\r\nxlsxwriter       : None\r\nnumba            : None\r\n\r\n</details>\r\n",
 'performed_via_github_app': None}
```

In [61]:

```
data[1]['title']
```

Out[61]:

```
"BUG: timezone-aware DatetimeIndex doesn't handle offsets very well"
```

In [64]:

```
issues = pd.DataFrame(data, columns=['number', 'title', 'labels', 'state'])
issues.head(10)
```

Out[64]:

	number	title	labels	state
0	35389	ENH: Request for `Index.to_string` & `MultiInd...	[{'id': 76812, 'node_id': 'MDU6TGFiZWw3NjgxMg=...	open
1	35388	BUG: timezone-aware DatetimeIndex doesn't hand...	[{'id': 76811, 'node_id': 'MDU6TGFiZWw3NjgxMQ=...	open
2	35387	Display dataframe name or title when using dis...	[{'id': 76812, 'node_id': 'MDU6TGFiZWw3NjgxMg=...	open
3	35386	CLN: resolve isort mypy import confilict test_...	[]	open
4	35385	fix #35227	[]	open
5	35384	ENH: The parameter merge_cells parameter in fu...	[{'id': 76812, 'node_id': 'MDU6TGFiZWw3NjgxMg=...	open
6	35383	Drop similar rows of dataframe except for one ...	[{'id': 1954720290, 'node_id': 'MDU6TGFiZWwxOT...	open
7	35382	BUG: Inconsistent ordering of rows when mergin...	[{'id': 76811, 'node_id': 'MDU6TGFiZWw3NjgxMQ=...	open
8	35381	Storage options	[]	open
9	35380	CLN: resolve isort mypy import confilict test_...	[{'id': 211029535, 'node_id': 'MDU6TGFiZWwyMTE...	open

Interacting with Databases

in the business settings many companies may store their data in SQL based relational databases such as SQL Server, PostgreSQ, and MySQL. loading data from SQL into a DataFrame is fairly straightfoward, and pandas has some functions to simplify the process. As an Exmaple, lets create a SQLite database using Python's built-in sqlite3 driver

In [66]:

```
import sqlite3
query = """
CREATE TABLE test
(a VARCHAR(20), b VARCHAR(20),
 c REAL,        d INTEGER
); """

con = sqlite3.connect('mydata.sqlite')
con.execute(query)
```

Out[66]:

```
<sqlite3.Cursor at 0x2707bf59880>
```

In [69]:

```
# insert a few rows of data
data = [('Atlanta', 'Georgia', 1.25, 6),
        ('Tallahassee', 'Florida', 2.6, 3),
        ('Sacramento', 'California', 1.7, 5)]
stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"
con.executemany(stmt, data)
```

Out[69]:

```
<sqlite3.Cursor at 0x2707d703ab0>
```

In [73]:

```
con.commit()
# Most Python SQL drivers return a list of tuples whn selecting data from a talbe
cursor = con.execute('select * from test')
rows = cursor.fetchall()
rows
```

Out[73]:

```
[('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 need the column names, contained in the cursor's description attribute:

In [74]:

```
cursor.description
```

Out[74]:

```
(('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 [84]:

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

Out[84]:

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

In [ ]:

```

```