Ch6 Data Loading, Storage, and File Formats 数据载入、存储及文件格式 ¶

In [ ]:
```
#https://wesmckinney.com/book/accessing-data.html
#dataset: https://github.com/wesm/pydata-book/tree/3rd-edition/examples
```

In [103]:
```
import pandas as pd
import numpy as np
```

In [33]:
```
#3种pd.read读取csv文件方式:
```

In [27]:
```
file_path = r'C:\Users\miran\lpthw\ex1.csv'
df = pd.read_csv(file_path)
df
#   a   b   c   d   message
# 0 1   2   3   4   hello
# 1 5   6   7   8   world
# 2 9   10  11  12  foo
```

Out[27]:

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

In [34]:
```
df = pd.read_csv('C:/Users/miran/lpthw/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
```

Out[34]:

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

In [36]:
```
#使用read table, 指定分隔符:
pd.read_table('C:/Users/miran/lpthw/ex1.csv', sep = ',')
#   a   b   c   d   message
# 0 1   2   3   4   hello
# 1 5   6   7   8   world
# 2 9   10  11  12  foo
```

Out[36]:

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

In [43]:
```
#当文件不包含表头行。可以允许pandas分配默认列名0123

pd.read_csv('C:/Users/miran/lpthw/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

#也可以自己指定列名:
pd.read_csv('C:/Users/miran/lpthw/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

#如想要某列成为返回data frame的索引, 可以指定位置4的列为索引, 或将某列传给参数index_
names = ['a','b','c','d','message']
pd.read_csv('C:/Users/miran/lpthw/ex2.csv', names = names, index_col = 'messag
# a b   c   d
# message
# hello 1   2   3   4
# world 5   6   7   8
# foo   9   10  11  12
```

Out[43]:

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

In [52]:
```
parsed = pd.read_csv('C:/Users/miran/lpthw/csv_mindex.csv',
                    index_col = ['key1','key2'])
parsed
#       value1  value2
# key1  key2
# one   a   1   2
# b 3   4
# c 5   6
# d 7   8
# two   a   9   10
# b 11  12
# c 13  14
# d 15  16
```

Out[52]:

		value1	value2
key1	**key2**		
one	**a**	1	2
	b	3	4
	c	5	6
	d	7	8
two	**a**	9	10
	b	11	12
	c	13	14
	d	15	16

In [60]:
```
#sep = '\s+', 可以用来分隔空格回车等

#有时一张表或txt的分隔符并不是固定的，使用空白或其他方式来分割字段。

list(open('C:/Users/miran/lpthw/ex3.txt'))
# ['       A      B       C\n',
#  'aaa  -0.26   -1.02    -0.61\n',
#  'bbb  0.92    -0.30    -0.03\n',
#  'ccc  -0.26   -0.38    -0.21\n',
#  'ddd  -0.87   -0.34    1.10']

#当字段是以不同数量的空格分开时，可以想read_table传入一个正则表达式作为分隔符，如sep
result = pd.read_table('C:/Users/miran/lpthw/ex3.txt', sep = '\s+')
result
# A B   C
# aaa   -0.26   -1.02   -0.61
# bbb   0.92    -0.30   -0.03
# ccc   -0.26   -0.38   -0.21
# ddd   -0.87   -0.34   1.10
```

Out[60]:

	A	B	C
aaa	-0.26	-1.02	-0.61
bbb	0.92	-0.30	-0.03
ccc	-0.26	-0.38	-0.21
ddd	-0.87	-0.34	1.10

In [63]:
```
#read_csv('file_pat', skip rows = [a,b])可以用来跳过a,b行

pd.read_csv('C:/Users/miran/lpthw/ex4.csv')
#   XXX Unnamed: 1  Unnamed: 2  Unnamed: 3  Unnamed: 4
# 0 a   b   c   d   message
# 1 YYYY    NaN NaN NaN NaN
# 2 ZZZZ    NaN NaN NaN NaN
# 3 1   2   3   4   hello
# 4 5   6   7   8   world
# 5 9   10  11  12  foo
pd.read_csv('C:/Users/miran/lpthw/ex4.csv', skiprows = [0, 2, 3])
# a b   c   d   message
# 0 1   2   3   4   hello
# 1 5   6   7   8   world
# 2 9   10  11  12  foo
```

Out[63]:

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

In [83]:
```
#缺失值或不显示，或用标识值。
#pandas会使用标识：NA或Null

result = pd.read_csv('C:/Users/miran/lpthw/ex5.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

pd.isnull(result)
# something a   b   c   d   message
# 0 False   False   False   False   False   True
# 1 False   False   False   True    False   False
# 2 False   False   False   False   False   False

#na_values可以传入一个列表或一组字符串来处理缺失值：
result = pd.read_csv('C:/Users/miran/lpthw/ex5.csv', na_values= ['NULL'])     #
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

#字典中，每列可以指定不同的缺失值标识：
sentinels = {'message': ['foo', 'NA'], 'something': ['two']}                  #
pd.read_csv('C:/Users/miran/lpthw/ex5.csv', na_values = sentinels)
#   something   a   b   c   d   message
# 0 one 1   2   3.0 4   NaN
# 1 NaN 5   6   NaN 8   world
# 2 three   9   10  11.0    12  NaN
```

Out[83]:

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

In [84]:
```
#pandas.read_csv和pandas.read_table常用选项：

# path：表名文件系统位置的字符串，URL或文件型对象
# sep或delimiter：分隔每行字段的字符序列或正则表达式
# header：用作列名的行号，默认是0（第一行），如果没有列名则为None
# index_col：行索引的列号或列名。
# names：结果列名列表，和header = None 一起用
# skiprows：文件开头处起，需要跳过的行数或行号列表
# na_values：需要用NA替换的值序列
# comment：在行结尾处分隔注释的字符
# ...
```

6.1 文本格式的读写：Reading and Writing Data in Text Format

6.1.1 分块读入文本文件：Reading Text Files in Pieces

In [90]:
```
#处理大型文件或找出正确的参数集来正确处理大文件时，需要读入文件的一个小片段，或者按小

pd.options.display.max_rows = 10
result = pd.read_csv('C:/Users/miran/lpthw/ex6.csv')
result
# one    two three    four     key
# 0 0.467976    -0.038649    -0.295344    -1.824726    L
# 1 -0.358893    1.404453    0.704965    -0.200638    B
# 2 -0.501840    0.659254    -0.421691    -0.057688    G
# 3 0.204886    1.074134    1.388361    -0.982404    R
# 4 0.354628    -0.133116    0.283763    -0.837063    Q
# ...    ... ... ... ... ...
# 9995  2.311896    -0.417070    -1.409599    -0.515821    L
# 9996  -0.479893   -0.650419    0.745152     -0.646038    E
# 9997  0.523331    0.787112     0.486066     1.093156     K
# 9998  -0.362559   0.598894     -1.843201    0.887292     G
# 9999  -0.096376   -1.012999    -0.657431    -0.573315    0
# 10000 rows × 5 columns

#如果只想读取一小部分行，可以指明nrows:
#pd.read_csv('file_path', nrows = n)
pd.read_csv('C:/Users/miran/lpthw/ex6.csv', nrows = 5)
# one    two three    four     key
# 0 0.467976    -0.038649    -0.295344    -1.824726    L
# 1 -0.358893    1.404453    0.704965    -0.200638    B
# 2 -0.501840    0.659254    -0.421691    -0.057688    G
# 3 0.204886    1.074134    1.388361    -0.982404    R
# 4 0.354628    -0.133116    0.283763    -0.837063    Q

#分块读入文件，可以指定chunksize座位每一块的行数:
chunker = pd.read_csv('C:/Users/miran/lpthw/ex6.csv', chunksize = 1000)

tot = pd.Series([])
for piece in chunker:
    tot = tot.add(piece['key'].value_counts(), fill_value = 0)
tot = tot.sort_values(ascending = False)
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_21400\3113304847.py:33: FutureWar
ning: The default dtype for empty Series will be 'object' instead of 'float6
4' in a future version. Specify a dtype explicitly to silence this warning.
  tot = pd.Series([])
```

6.1.2 将数据写入文本格式： Writing Data to Text Format

In [107]:
```
data = pd.read_csv('C:/Users/miran/lpthw/ex5.csv')
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

#使用data frame的to_csv方法，可以将数据导出为逗号分隔的文件：
data.to_csv('C:/Users/miran/lpthw/out.csv')

#其他的分隔符也是可以的：
import sys
data.to_csv(sys.stdout, sep = '|')
# |something|a|b| c |d|message
# 0|one|1|2|3.0|4|
# 1|two|5|6||8|world
# 2|three|9|10|11.0|12|foo

#缺失值在输出时，以空字符串出现。可以用其他标示值，对缺失值进行标注：
data.to_csv(sys.stdout, na_rep = 'NULL')
# ,something,a,b, c ,d,message
# 0,one,1,2,3.0,4,NULL
# 1,two,5,6,NULL,8,world
# 2,three,9,10,11.0,12,foo

#如果没有其他选项被指定，行和列都会被写入。不过二者也可以禁止写入：
data.to_csv(sys.stdout, index = False, header = False)
# one,1,2,3.0,4,
# two,5,6,,8,world
# three,9,10,11.0,12,foo

#可以仅写入列的子集，按照选择的顺序写入：
#data.to_csv(sys.stdout, index = False, columns = ['a', 'b', 'c'])

#series也有to_csv方法：
dates = pd.date_range('1/1/2000', periods = 7)
ts = pd.Series(np.arange(7), index = dates)
ts.to_csv('C:/Users/miran/lpthw/tseries.csv')
#   0
# 1/1/2000  0
# 1/2/2000  1
# 1/3/2000  2
# 1/4/2000  3
# 1/5/2000  4
# 1/6/2000  5
# 1/7/2000  6
```

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

```
The syntax of the command is incorrect.
```

6.1.3 使用分隔格式： Working with Delimited Formats

In [119]:
```
#函数pandas.read_table。对于任何带有单字符分隔符的文件，可以使用python内建csv模块，
import csv
f = open('C:/Users/miran/lpthw/ex7.csv')
reader = csv.reader(f)

#遍历文件，遍历reader，产生元组，元组的值，为了删除引号的字符:
for line in reader:
    print(line)
# ['a', 'b', 'c']
# ['1', '2', '3']
# ['1', '2', '3']

#将文件读取为行的列表:
with open('C:/Users/miran/lpthw/ex7.csv') as f:
    lines = list(csv.reader(f))

#将数据拆分为列名行和数据行:
header, values = lines[0], lines[1:]

#使用字典推导式和表达式zip(*values)生成一个包含数据列的字典，字典中行转置成列。
data_dict = {h: v for h, v in zip(header, zip(*values))}
data_dict
#{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}

#csv有多种不同风格，如需不同分隔符，字符串引用约定或行终止符定义一种新的格式，可以使
# class my_dialect(csv.Dialect):
#     lineterminator = '\n'
#     delimiter = ';'
#     quotechar = '"'
#     quoting = csv.QUOTE_MINIMAL

# reader = csv.reader(f, dialect = my_dialect)
# reader = csv.reader(f, delimiter = '|')
```

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

Out[119]:
```
{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

In [ ]:
```
#csv方言选项:
# delimiter: 分隔字符
# lineterminator: 行终止符。
# quotechar: 含有特殊字符的引号
# quoting: 引用惯例
# skipinitialspace: 忽略分隔符后的空白，默认false
# doublequote: 处理字段内部的引号。
```

In [120]:
```
#对于更复杂或固定的多字符分隔符文件，无法使用csv模块，将要使用字符串split方法或正则表
with open('mydata.csv', 'w') as f:
    writer = csv.writer(f, dialect = my_dialect)
    writer.writerow(('one','two','three'))
    writer.writerow(('1','2','3'))
    writer.writerow(('4','5','6'))
    writer.writerow(('7','8','9'))
```

6.1.4 JSON数据： JSON Data

In [129]:
```
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"]}]
}
"""
import json
result = json.loads(obj)
result
# {'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可以将python对象，转换为Json:
asjson = json.dumps(result)
siblings = pd.DataFrame(result['siblings'], columns = ['name','age'])
siblings
# name  age
# 0 Scott   30
# 1 Katie   38

#pandas.read_json可以将json转换为series或data frame.
data = pd.read_json('C:/Users/miran/lpthw/example.json')
data
#   a   b   c
# 0 1   2   3
# 1 4   5   6
# 2 7   8   9

#从pandas将数据导出为JSON格式:

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}}

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

```
{"a":{"0":1,"1":4,"2":7},"b":{"0":2,"1":5,"2":8},"c":{"0":3,"1":6,"2":9}}
[{"a":1,"b":2,"c":3},{"a":4,"b":5,"c":6},{"a":7,"b":8,"c":9}]
```

6.1.5 XML和HTML：网络抓取： XML and HTML: Web Scraping

In [130]:
```
#lxml, beautiful soup, html5lib的库，可以读取写入数据的库
```

In [137]:
```
#conda install lxml
#pip install beautifulsoup4 html5lib
#In command prompt:
#C:\Users\miran>pip install beautifulsoup4
#C:\Users\miran>pip install html5lib
```

In [148]:
```
#pandas.read_html函数，默认会搜索解析所有包含在table中的表格型数据，发挥的事data fr

tables = pd.read_html('C:/Users/miran/lpthw/fdic_failed_bank_list.html')
len(tables)
failures = tables[0]
failures.head()            #前5行

# Bank Name City    ST  CERT    Acquiring Institution   Closing Date    Update
# 0 Allied Bank Mulberry    AR  91  Today's Bank    September 23, 2016  Novemb
# 1 The Woodbury Banking Company    Woodbury    GA  11297   United Bank August
# 2 First CornerStone Bank  King of Prussia PA  35312   First-Citizens Bank &
# 3 Trust Company Bank  Memphis TN  9956    The Bank of Fayette County  April
# 4 North Milwaukee State Bank  Milwaukee   WI  20364   First-Citizens Bank &

#因为failures有很多列，pandas在行内插入了换行符。

#现在开始数据清理分析，计算每年银行倒闭的数量：在'Closing Date'列中的不同年份在数据集

#将failures数据框中的'Closing Date'列转换为日期时间类型，并将结果存储在close_times
#通过pd.to_datetime()函数，将日期字符串转换为Timestamp对象，便于后续的日期操作和分析
close_timestamps = pd.to_datetime(failures['Closing Date'])

#计算close_timestamps中每个日期时间对象的年份，并使用value_counts()函数统计每个年份
#这将返回一个包含年份计数的Series对象，其中索引是年份，值是对应年份出现的次数。
close_timestamps.dt.year.value_counts()

# 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
```

Out[148]:
```
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
```

6.2 二进制格式：Binary Data Formats

In [160]:
```
#使用python内建的pickle序列化模块进行二进制操作是存储数据最高效方便的方式。

#读取csv文件
frame = pd.read_csv('C:/Users/miran/lpthw/ex1.csv')
frame
# a b   c   d   message
# 0 1   2   3   4   hello
# 1 5   6   7   8   world
# 2 9   10  11  12  foo

#写入一个名为frame_pickle的文件
frame.to_pickle('C:/Users/miran/lpthw/frame_pickle')

#将其读取
pd.read_pickle('C:/Users/miran/lpthw/frame_pickle')

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

Out[160]:

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

6.2.1 使用HDF5格式： Using HDF5 Format

In [181]:
```
#pandas提供高阶接口，简化series和data frame的存储。HDFStore类像字典一样：
frame = pd.DataFrame({'a': np.random.randn(100)})
store = pd.HDFStore('mydata.h5')
store['obj1'] = frame
store['obj1_col'] = frame['a']
store
# <class 'pandas.io.pytables.HDFStore'>
# File path: mydata.h5

store['obj1']
# a
# 0 0.985822
# 1 1.518504
# 2 -0.092633
# 3 -2.556538
# 4 1.333706
# ...   ...
# 95    -0.202865
# 96    0.428260
# 97    2.223956
# 98    0.983035
# 99    -0.362057
# 100 rows × 1 columns

#HDFStore支持2中存储模式，fixed和table。
store.put('obj2', frame, format = 'table')

store.select('obj2', where = ['index >= 10 and index <= 15'])
# a
# 10    0.678830
# 11    -1.783086
# 12    -1.539850
# 13    1.939564
# 14    -2.005488
# 15    1.499467

#put是store['obj2'] = frame的显式版本，但允许其他选项，如存储格式。
store.close()

#pandas.read_hdf是快捷方法。
# frame.to_hdf('mydata.h5', 'obj3', format = 'table')
# pd.read_hdf('mydata.h5', 'obj3', where = ['index < 5'])
```

6.2.2 读取Excel文件： Reading Microsoft Excel Files

In [189]:
```
#pandas支持通过excelfile或pandas.read_excel读取存储
xlsx = pd.ExcelFile('C:/Users/miran/lpthw/ex1.xlsx')
#存储在表中的数据，可以通过pandas.read_excel读取到data frame中
pd.read_excel(xlsx, 'Sheet1')
#   Unnamed: 0  a   b   c   d   message
# 0 0   1   2   3   4   hello
# 1 1   5   6   7   8   world
# 2 2   9   10  11  12  foo

#如果读取多个表的文件，可以更简洁地将文件名传入pandas.read_excel：
frame = pd.read_excel('C:/Users/miran/lpthw/ex1.xlsx','Sheet1')
frame
# Unnamed: 0    a   b   c   d   message
# 0 0   1   2   3   4   hello
# 1 1   5   6   7   8   world
# 2 2   9   10  11  12  foo

#如需将pandas数据写入到excel格式中，可以先生成一个excelwriter，然后使用pandas对象的
writer = pd.ExcelWriter('C:/Users/miran/lpthw/ex2.xlsx')
frame.to_excel(writer, 'Sheet1')
writer.save()

#也可以将文件路径传给to_excel，避免直接调用ExcelWriter
frame.to_excel('C:/Users/miran/lpthw/ex2.xlsx')
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_21400\807895415.py:21: FutureWarn
ing: save is not part of the public API, usage can give unexpected results an
d will be removed in a future version
  writer.save()
```

6.3 与web API交互：Interacting with Web APIs

In [197]:

```
#很多网站有公开API，通过JSON或其他格式提供数据服务。有多种方式可以利用Python来访问AP
#简单易用方式是使用requests包。

import requests
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
resp = requests.get(url)
resp
#<Response [200]>

#response对象的json方法，返回一个包含解析为本地python对象的JSON的字典：
data = resp.json()
data[0]['title']
#'Split Multiple Header into CSV file'

#data中的每个元素都是一个包含github问题页面上的所有数据的字典。
#可以将data直接传给data frame，并提取感兴趣的字段：只要以下4个列
issues = pd.DataFrame(data, columns = ['number','title',
                                       'labels','state'])
issues
# number    title   labels  state
# 0 53433   Split Multiple Header into CSV file [{'id': 34444536, 'node_id': '
# 1 53432   DOC: Add release notes for pandas 2.0.3 [{'id': 134699, 'node_id':
# 2 53431   TST: Add test for series str decode GH#22613    []  open
# 3 53430   RLS: 2.0.3  [{'id': 131473665, 'node_id': 'MDU6TGFiZWwxMzE...   op
# 4 53429   REF: Remove side effects from importing Styler 2    []  open
# ...   ... ... ... ...
# 25    53390   BUG: Reading fails when `dtype` is defined wit...   [{'id': 76
# 26    53387   PERF: RangeIndex cache is written when calling...   [{'id': 28
# 27    53385   TST: Add test for pandas on sys.getsizeof GH#2...   [{'id': 12
# 28    53384   BLD: remove `pkg_resources` usage from `setup.py`   [{'id': 12
# 29    53379   MNT: Mark all `nogil` functions as `noexcept`   [{'id': 490944
```

Out[197]:

	number	title	labels	state
0	53434	Bump pypa/cibuildwheel from 2.12.3 to 2.13.0	[{'id': 48070600, 'node_id': 'MDU6TGFiZWw0ODA3...	open
1	53433	Split Multiple Header into CSV file	[{'id': 34444536, 'node_id': 'MDU6TGFiZWwzNDQ0...	open
2	53432	DOC: Add release notes for pandas 2.0.3	[{'id': 134699, 'node_id': 'MDU6TGFiZWwxMzQ2OT...	open
3	53431	TST: Add test for series str decode GH#22613	[]	open
4	53430	RLS: 2.0.3	[{'id': 131473665, 'node_id': 'MDU6TGFiZWwxMzE...	open
...	...	...	...	...
25	53391	BUG: read_csv with dtype=bool[pyarrow]	[{'id': 47229171, 'node_id': 'MDU6TGFiZWw0NzIy...	open
26	53390	BUG: Reading fails when `dtype` is defined wit...	[{'id': 76811, 'node_id': 'MDU6TGFiZWw3NjgxMQ=...	open
27	53387	PERF: RangeIndex cache is written when calling...	[{'id': 2822098, 'node_id': 'MDU6TGFiZWwyODIyM...	open
28	53385	TST: Add test for pandas on sys.getsizeof GH#2...	[{'id': 127685, 'node_id': 'MDU6TGFiZWwxMjc2OD...	open
29	53384	BLD: remove `pkg_resources` usage from `setup.py`	[{'id': 129350, 'node_id': 'MDU6TGFiZWwxMjkzNT...	open

30 rows × 4 columns

6.4 与数据库交互：Interacting with Databases

In [233]:
```
import sqlite3

# SQL中将数据读取为data frame容易，pandas有多个函数可以简化过程。

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

con = sqlite3.connect('mydata.sqlite')
con.execute(query)
#<sqlite3.Cursor at 0x1eaff9a22c0>

con.commit()

#插入几行数据

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)
#<sqlite3.Cursor at 0x1eaffa0ce40>
con.commit()

#大部分python的sql驱动，返回的是元组的列表：

cursor = con.execute('select * from test')
rows = cursor.fetchall()
rows
# [('Atlanta', 'Georgia', 1.25, 6),
#  ('Tallahassee', 'Florida', 2.6, 3),
#  ('Sacramento', 'California', 1.7, 5)]

#可以将元组的列表传给data frame构造函数，但还需要包含在游标的description属性中的列名
cursor.description
# (('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))

pd.DataFrame(rows, columns = [x[0] for x in cursor.description])
# a b    c    d
# 0 Atlanta Georgia 1.25    6
# 1 Tallahassee Florida 2.60    3
# 2 Sacramento  California  1.70    5
```

Out[233]:

	a	b	c	d
0	Atlanta	Georgia	1.25	6
1	Tallahassee	Florida	2.60	3
2	Sacramento	California	1.70	5
3	Atlanta	Georgia	1.25	6
4	Tallahassee	Florida	2.60	3
...	...	...	...	...
22	Tallahassee	Florida	2.60	3
23	Sacramento	California	1.70	5
24	Atlanta	Georgia	1.25	6
25	Tallahassee	Florida	2.60	3
26	Sacramento	California	1.70	5

27 rows × 4 columns

In [236]:

```
#SQLAlchemy项目是个流行的python sql包。pandas有一个read_sql允许从sqlalchemy链接中
#sqlalchemy连接到相同的sqlite数据库，从之前的创建的表中读取数据。

import sqlalchemy as sqla
db = sqla.create_engine('sqlite:///mydata.sqlite')
pd.read_sql('select * from test', db)
# a b   c   d
# 0 Atlanta Georgia 1.25    6
# 1 Tallahassee Florida 2.60    3
# 2 Sacramento  California  1.70    5
```

Out[236]:

	a	b	c	d
0	Atlanta	Georgia	1.25	6
1	Tallahassee	Florida	2.60	3
2	Sacramento	California	1.70	5
3	Atlanta	Georgia	1.25	6
4	Tallahassee	Florida	2.60	3
...	...	...	...	...
22	Tallahassee	Florida	2.60	3
23	Sacramento	California	1.70	5
24	Atlanta	Georgia	1.25	6
25	Tallahassee	Florida	2.60	3
26	Sacramento	California	1.70	5

27 rows × 4 columns

6.5 小结：Conclusion