数据分析 4 数据加载、存储和文件格式

读写文本格式的数据

pandas提供了一些用于将表格型数据读取为DataFrame对象的函数，其中read_csv和read_table用得最多

函数	说明
read_csv	从文件、URL、文件型对象中加载带分隔符的数据。默认分隔符为逗号
read_table	从文件、URL、文件型对象中加载带分隔符的数据。默认分隔符为制表符('\t')
read_fwf	读取定宽列格式数据（也就是说，没有分隔符）
read_clipboard	读取剪贴板中的数据，可以看做 read_table 的剪贴板版。再将网页转换为表格时很有用
read_excel	从 Excel XLS 或 XLSX file 读取表格数据
read_hdf	读取 pandas 写的 HDF5 文件
read_html	读取 HTML 文档中的所有表格
read_json	读取 JSON (JavaScript Object Notation)字符串中的数据
read_msgpack	二进制格式编码的 pandas 数据
read_pickle	读取 Python pickle 格式中存储的任意对象
read_sas	读取存储于 SAS 系统自定义存储格式的 SAS 数据集
read_sql	（使用 SQLAlchemy）读取 SQL 查询结果为 pandas 的 DataFrame
read_stata	读取 Stata 文件格式的数据集
read_feather	读取 Feather 二进制文件格式

将文本数据转换为DataFrame时所用到的一些技术

索引：将一个或多个列当做返回的DataFrame处理，以及是否从文件、用户获取列名

类型推断和数据转换：包括用户定义值的转换、和自定义的缺失值标记列表等

日期解析：包括组合功能，比如将分散在多个列中的日期时间信息组合成结果中的单个列

迭代：支持对大文件进行逐块迭代

不规整数据问题：跳过一些行、页脚、注释或其他一些不重要的东西（比如由成千上万个逗号隔开的数值数据）

read_csv有超过50个参数

以逗号分隔的（CSV）文本文件例子

```
# Windows，你可以使用type达到同样的效果
In [8]: !cat examples/ex1.csv
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

```
In [9]: df = pd.read_csv('examples/ex1.csv')

In [10]: df
Out[10]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

使用read_table，并指定分隔符

```
In [11]: pd.read_table('examples/ex1.csv', sep=',')
Out[11]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

没有标题的文件

```
In [12]: !cat examples/ex2.csv
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

可以让pandas为其分配默认的列名，也可以自己定义列名

```
# 试下不加header

In [13]: pd.read_csv('examples/ex2.csv', header=None)
Out[13]:
   0   1   2   3      4
0  1   2   3   4  hello
1  5   6   7   8  world
2  9  10  11  12    foo
```

```
In [14]: pd.read_csv('examples/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
Out[14]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

假设你希望将message列做成DataFrame的索引。你可以明确表示要将该列放到索引4的位置上，也可以通过index_col参数指定“message”

```
In [15]: names = ['a', 'b', 'c', 'd', 'message']

In [16]: pd.read_csv('examples/ex2.csv', names=names, index_col='message')
Out[16]:
         a   b   c   d
message
hello    1   2   3   4
world    5   6   7   8
foo      9  10  11  12
```

将多个列做成一个层次化索引，只需传入由列编号或列名组成的列表即可

```
In [17]: !cat examples/csv_mindex.csv
key1,key2,value1,value2
one,a,1,2
one,b,3,4
one,c,5,6
one,d,7,8
two,a,9,10
two,b,11,12
two,c,13,14
two,d,15,16

In [18]: parsed = pd.read_csv('examples/csv_mindex.csv',
   ....:                      index_col=['key1', 'key2'])

In [19]: parsed
Out[19]:
           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
```

有些表格可能不是用固定的分隔符去分隔字段的（比如空白符或其它模式

```
In [20]: list(open('examples/ex3.txt'))
Out[20]:
['            A         B         C\n',
 'aaa -0.264438 -1.026059 -0.619500\n',
 'bbb  0.927272  0.302904 -0.032399\n',
 'ccc -0.264273 -0.386314 -0.217601\n',
 'ddd -0.871858 -0.348382  1.100491\n']
```

虽然可以手动对数据进行规整，这里的字段是被数量不同的空白字符间隔开的。这种情况下，你可以传递一个正则表达式作为read_table的分隔符。可以用正则表达式表达为\s+

```
# 由于列名比数据行的数量少，所以read_table推断第一列应该是DataFrame的索引
In [21]: result = pd.read_table('examples/ex3.txt', sep='\s+')

In [22]: result
Out[22]:
            A         B         C
aaa -0.264438 -1.026059 -0.619500
bbb  0.927272  0.302904 -0.032399
ccc -0.264273 -0.386314 -0.217601
ddd -0.871858 -0.348382  1.100491
```

异形文件格式处理，你可以用skiprows跳过文件的第一行、第三行和第四行

```
In [23]: !cat examples/ex4.csv
# hey!
a,b,c,d,message
# just wanted to make things more difficult for you
# who reads CSV files with computers, anyway?
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
In [24]: pd.read_csv('examples/ex4.csv', skiprows=[0, 2, 3])
Out[24]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

缺失值处理是文件解析任务中的一个重要组成部分。缺失数据经常是要么没有（空字符串），要么用某个标记值表示。默认情况下，pandas会用一组经常出现的标记值进行识别，比如NA及NULL

```
In [25]: !cat examples/ex5.csv
something,a,b,c,d,message
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
In [26]: result = pd.read_csv('examples/ex5.csv')

In [27]: result
Out[27]:
  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

In [28]: pd.isnull(result)
Out[28]:
   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可以用一个列表或集合的字符串表示缺失值

```
In [29]: result = pd.read_csv('examples/ex5.csv', na_values=['NULL'])

In [30]: result
Out[30]:
  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
```

字典的各列可以使用不同的NA标记值

```
In [31]: sentinels = {'message': ['foo', 'NA'], 'something': ['two']}

In [32]: pd.read_csv('examples/ex5.csv', na_values=sentinels)
```

```
Out[32]:
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
```

表6-2：read_csv/read_table函数的参数

参数	说明
path	表示文件系统位置、URL、文件型对象的字符串
sep或delimiter	用于对行中各字段进行拆分的字符序列或正则表达式
header	用作列名的行号。默认为0（第一行），如果没有header行就应该设置为None
index_col	用作行索引的列编号或列名。可以是单个名称/数字或由多个名称/数字组成的列表（层次化索引）
names	用于结果的列名列表，结合header=None
skiprows	需要忽略的行数（从文件开始处算起），或需要跳过的行号列表（从0开始）
na_values	一组用于替换NA的值
comment	用于将注释信息从行尾拆分出去的字符（一个或多个）
parse_dates	尝试将数据解析为日期，默认为False。如果为True，则尝试解析所有列。此外，还可以指定需要解析的一组列号或列名。如果列表的元素为列表或元组，就会将多个列组合到一起再进行日期解析工作（例如，日期/时间分别位于两个列中）
keep_date_col	如果连接多列解析日期，则保持参与连接的列。默认为False。
converters	由列号/列名跟函数之间的映射关系组成的字典。例如，{'foo': f}会对foo列的所有值应用函数f
dayfirst	当解析有歧义的日期时，将其看做国际格式（例如，7/6/2012 → June 7, 2012）。默认为False
date_parser	用于解析日期的函数
nrows	需要读取的行数（从文件开始处算起）
iterator	返回一个TextParser以便逐块读取文件
chunksize	文件块的大小（用于迭代）
skip_footer	需要忽略的行数（从文件末尾处算起）

表6-2：read_csv/read_table函数的参数（续）

参数	说明
verbose	打印各种解析器输出信息，比如“非数值列中缺失值的数量”等
encoding	用于unicode的文本编码格式。例如，“utf-8”表示用UTF-8编码的文本
squeeze	如果数据经解析后仅含一列，则返回Series
thousands	千分位分隔符，如“,”或“.”

逐块读取文本文件

在处理很大的文件时，或找出大文件中的参数集以便于后续处理时，可以读取文件的一小部分或逐块对文件进行迭代

设置pandas显示地更紧些

```
In [33]: pd.options.display.max_rows = 10
```

```
In [34]: result = pd.read_csv('examples/ex6.csv')

In [35]: result
Out[35]:
           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 x 5 columns]
```

如果只想读取几行（避免读取整个文件），通过nrows进行指定即可

```
In [36]: pd.read_csv('examples/ex6.csv', nrows=5)
Out[36]:
        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
```

要逐块读取文件，可以指定chunksize（行数）

```
In [874]: chunker = pd.read_csv('examples/ex6.csv', chunksize=1000)

In [875]: chunker
Out[875]: <pandas.io.parsers.TextParser at 0x8398150>
```

read_csv所返回的这个TextParser对象使你可以根据chunksize对文件进行逐块迭代。比如说，我们可以迭代处理ex6.csv，将值计数聚合到"key"列中

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

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

```
In [40]: tot[:10]
Out[40]:
E    368.0
X    364.0
L    346.0
O    343.0
Q    340.0
```

```
M    338.0
J    337.0
F    335.0
K    334.0
H    330.0
dtype: float64
```

将数据写出到文本格式

数据也可以被输出为分隔符格式的文本

```
In [41]: data = pd.read_csv('examples/ex5.csv')

In [42]: data
Out[42]:
  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
```

DataFrame的to_csv方法，我们可以将数据写到一个以逗号分隔的文件中

```
In [43]: data.to_csv('examples/out.csv')

In [44]: !cat examples/out.csv
,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
```

使用其他分隔符（由于这里直接写出到sys.stdout，所以仅仅是打印出文本结果而已）

```
In [45]: import sys

In [46]: 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
```

缺失值在输出结果中会被表示为空字符串。你可能希望将其表示为别的标记值

```
In [47]: 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
```

没有设置其他选项，则会写出行和列的标签。当然，它们也都可以被禁用

```
In [48]: 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
```

你还可以只写出一部分的列，并以你指定的顺序排列

```
In [49]: data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
a,b,c
1,2,3.0
5,6,
9,10,11.0
```

```
In [50]: dates = pd.date_range('1/1/2000', periods=7)

In [51]: ts = pd.Series(np.arange(7), index=dates)

In [52]: ts.to_csv('examples/tseries.csv')
```

```
In [53]: !cat examples/tseries.csv
2000-01-01,0
2000-01-02,1
2000-01-03,2
2000-01-04,3
2000-01-05,4
2000-01-06,5
2000-01-07,6
```

处理分隔符格式

JSON数据

pandas.read_json可以自动将特别格式的JSON数据集转换为Series或DataFrame

```
In [68]: !cat examples/example.json
[{"a": 1, "b": 2, "c": 3},
 {"a": 4, "b": 5, "c": 6},
 {"a": 7, "b": 8, "c": 9}]
```

```
In [69]: data = pd.read_json('examples/example.json')

In [70]: data
Out[70]:
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

从pandas输出到JSON，使用to_json方法

```
In [71]: 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 [72]: 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和HTML：Web信息收集

pandas有一个内置的功能，read_html，它可以使用lxml和Beautiful Soup自动将HTML文件中的表格解析为DataFrame对象。

使用例子数据：美国联邦存款保险公司一个HTML文件，它记录了银行倒闭的情况。

安装read_html用到的库

```
conda install lxml
pip install beautifulsoup4 html5lib
```

```
In [73]: tables = pd.read_html('examples/fdic_failed_bank_list.html')

In [74]: len(tables)
Out[74]: 1

In [75]: failures = tables[0]

In [76]: failures.head()
Out[76]:
                      Bank Name             City  ST   CERT  \
0                   Allied Bank         Mulberry  AR     91
1  The Woodbury Banking Company         Woodbury  GA  11297
2        First CornerStone Bank  King of Prussia  PA  35312
3            Trust Company Bank          Memphis  TN   9956
4    North Milwaukee State Bank        Milwaukee  WI  20364
```

做一些数据清洗和分析，比如计算按年份计算倒闭的银行数

```
In [77]: close_timestamps = pd.to_datetime(failures['Closing Date'])

In [78]: close_timestamps.dt.year.value_counts()
```

```
Out[78]:
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
```

二进制数据格式

pandas对象都有一个用于将数据以pickle格式保存到磁盘上的to_pickle方法

```
In [87]: frame = pd.read_csv('examples/ex1.csv')

In [88]: frame
Out[88]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo

In [89]: frame.to_pickle('examples/frame_pickle')
```

读取pickle数据

```
In [90]: pd.read_pickle('examples/frame_pickle')
Out[90]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
```

```
2  9  10  11  12    foo
```

注意： pickle仅建议用于短期存储格式。其原因是很难保证该格式永远是稳定的；今天pickle的对象可能无法被后续版本的库unpickle出来

使用HDF5格式

HDF5是一种存储大规模科学数组数据的非常好的文件格式。它可以被作为C标准库，带有许多语言的接口，如Java、Python和MATLAB等。HDF5中的HDF指的是层次型数据格式（hierarchical data format）。每个HDF5文件都含有一个文件系统式的节点结构，它使你能够存储多个数据集并支持元数据。与其他简单格式相比，HDF5支持多种压缩器的即时压缩，还能更高效地存储重复模式数据。对于那些非常大的无法直接放入内存的数据集，HDF5就是不错的选择，因为它可以高效地分块读写。

```
In [92]: frame = pd.DataFrame({'a': np.random.randn(100)})

In [93]: store = pd.HDFStore('mydata.h5')

In [94]: store['obj1'] = frame

In [95]: store['obj1_col'] = frame['a']

In [96]: store
Out[96]:
<class 'pandas.io.pytables.HDFStore'>
File path: mydata.h5
```

HDF5文件中的对象可以通过与字典一样的API进行获取

```
In [97]: store['obj1']
Out[97]:
          a
0  -0.204708
1   0.478943
2  -0.519439
3  -0.555730
```

```
4   1.965781
..       ...
95  0.795253
96  0.118110
97 -0.748532
98  0.584970
99  0.152677
[100 rows x 1 columns]
```

HDFStore支持两种存储模式，'fixed'和'table'。后者通常会更慢，但是支持使用特殊语法进行查询操作

```
In [98]: store.put('obj2', frame, format='table')

In [99]: store.select('obj2', where=['index >= 10 and index <= 15'])
Out[99]:
           a
10  1.007189
11 -1.296221
12  0.274992
13  0.228913
14  1.352917
15  0.886429

In [100]: store.close()
```

pandas.read_hdf函数可以快捷使用这些工具

```
In [101]: frame.to_hdf('mydata.h5', 'obj3', format='table')

In [102]: pd.read_hdf('mydata.h5', 'obj3', where=['index < 5'])
Out[102]:
          a
0 -0.204708
1  0.478943
2 -0.519439
3 -0.555730
```

```
4  1.965781
```

读取Microsoft Excel文件

pandas的ExcelFile类或pandas.read_excel函数支持读取存储在Excel 2003（或更高版本）中的表格型数据。这两个工具分别使用扩展包xlrd和openpyxl读取XLS和XLSX文件。你可以用pip或conda安装它们

```
In [104]: xlsx = pd.ExcelFile('examples/ex1.xlsx')
In [105]: pd.read_excel(xlsx, 'Sheet1')
Out[105]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

```
In [106]: frame = pd.read_excel('examples/ex1.xlsx', 'Sheet1')

In [107]: frame
Out[107]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

将pandas数据写入为Excel格式

```
In [108]: writer = pd.ExcelWriter('examples/ex2.xlsx')

In [109]: frame.to_excel(writer, 'Sheet1')

In [110]: writer.save()
```

也可以这样存

```
In [111]: frame.to_excel('examples/ex2.xlsx')
```

Web APIs交互

```
In [113]: import requests

In [114]: url = 'https://api.github.com/repos/pandas-dev/pandas/issues'

In [115]: resp = requests.get(url)

In [116]: resp
Out[116]: <Response [200]>

In [117]: data = resp.json()

In [118]: data[0]['title']
Out[118]: 'Period does not round down for frequencies less that 1 hour'
```

```
In [119]: issues = pd.DataFrame(data, columns=['number', 'title',
   .....:                                      'labels', 'state'])

In [120]: issues
Out[120]:
    number                                              title  \
0    17666  Period does not round down for frequencies les...
1    17665           DOC: improve docstring of function where
2    17664               COMPAT: skip 32-bit test on int repr
3    17662                           implement Delegator class
4    17654  BUG: Fix series rename called with str alterin...
..     ...                                                ...
25   17603  BUG: Correctly localize naive datetime strings...
26   17599                     core.dtypes.generic --> cython
27   17596   Merge cdate_range functionality into bdate_range
```

```
28   17587  Time Grouper bug fix when applied for list gro...
29   17583  BUG: fix tz-aware DatetimeIndex + TimedeltaInd...
[30 rows x 4 columns]
```

数据库交互

在商业场景下，大多数数据可能不是存储在文本或Excel文件中。基于SQL的关系型数据库（如SQL Server、PostgreSQL和MySQL等）使用非常广泛。

```
In [121]: import sqlite3

In [122]: query = """
   .....: CREATE TABLE test
   .....: (a VARCHAR(20), b VARCHAR(20),
   .....:  c REAL,        d INTEGER
   .....: );"""

In [123]: con = sqlite3.connect('mydata.sqlite')

In [124]: con.execute(query)
Out[124]: <sqlite3.Cursor at 0x7f6b12a50f10>

In [125]: con.commit()
```

```
In [126]: data = [('Atlanta', 'Georgia', 1.25, 6),
   .....:         ('Tallahassee', 'Florida', 2.6, 3),
   .....:         ('Sacramento', 'California', 1.7, 5)]

In [127]: stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"

In [128]: con.executemany(stmt, data)
Out[128]: <sqlite3.Cursor at 0x7f6b15c66ce0>
```

```
In [130]: cursor = con.execute('select * from test')

In [131]: rows = cursor.fetchall()

In [132]: rows
Out[132]:
[('Atlanta', 'Georgia', 1.25, 6),
 ('Tallahassee', 'Florida', 2.6, 3),
 ('Sacramento', 'California', 1.7, 5)]
```

```
In [133]: cursor.description
Out[133]:
(('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 [134]: pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
Out[134]:
             a           b     c  d
0      Atlanta     Georgia  1.25  6
1  Tallahassee     Florida  2.60  3
2   Sacramento  California  1.70  5
```

这种数据规整操作相当多，你肯定不想每查一次数据库就重写一次。SQLAlchemy项目是一个流行的Python SQL工具，它抽象出了SQL数据库中的许多常见差异。pandas有一个read_sql函数，可以让你轻松的从SQLAlchemy连接读取数据。

```
In [135]: import sqlalchemy as sqla

db= sqla.create_engine('mysql+pymysql://root:123456@127.0.0.1/taobao?
charset=utf8')

In [137]: pd.read_sql('select * from product', db)
```