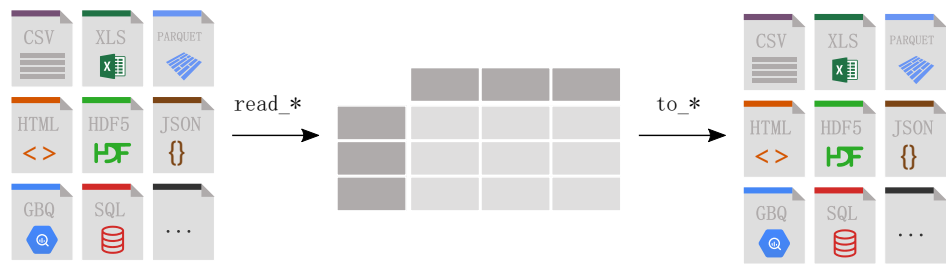


02Pandas文件读取

我们拿到的数据一般是 CSV、Excel 等格式，将文件加载到 Pandas 的 DataFrame 对象，我们就可以用它的方法进行了。在处理结束后，我们也需要将文件导出 Excel 等格式，方便查看。



本课程介绍最常用的文件格式和最基本的使用法，如果遇到更加详细的功能可以查看官方文档。

一、功能列表

下边是我们经常使用的方法：

格式	文件格式	读取函数	写入（输出）函数
binary	Excel	read_excel	to_excel
text	CSV	read_csv 、 read_table	to_csv
text	JSON	read_json	to_json
text	网页表格 HTML	read_html	to_html
text	剪贴板	read_clipboard	to_clipboard
SQL	SQL	read_sql	to_sql
XML	XML	read_xml	read_xml
text	Markdown		to_markdown

读取更多类型文件可查看[官网文档](#)。

其中：

- 读取函数一般会赋值给一个变量 `df`，`df = pd.read_<xxx>()`
- 输出函数是将变量自身进行操作并输出 `df.to_<xxx>()`

二、CSV

从 `CSV` 文件中读取数据并加载到 `DataFrame`：

1.文件

```
1 # 文件目录
2 pd.read_csv('GDP-China.csv') # 如果文件与代码文件在同目录下
3 pd.read_csv('data/my/GDP-China.csv') # 指定目录
4 # 使用网址 url
5 pd.read_csv('https://www.gairuo.com/file/data/dataset/GDP-
  China.csv')
6
```

注：csv 文件扩展名不一定是 .csv

2.指定分隔符号

```
1 # 数据分隔转化是逗号，如果是其他可以指定
2 pd.read_csv(data, sep='\t') # 制表符分隔 tab
3 pd.read_table(data) # read_table 默认是制表符分隔 tab
```

3.列、索引、名称

```
1 # 默认第一行是表头，可以指定，如果指定列名会被忽略
2 pd.read_csv(data, header=0)
3 pd.read_csv(data, header=None) # 没有表头
4 pd.read_csv(data, names=['列1', '列2']) # 指定列名列表
5 # 如没列名，自动指定一个：前缀加序数
6 pd.read_csv(data, prefix='c_', header=None)
7
8 # 读取部分列
9 pd.read_csv(data, usecols=[0,4,3]) # 按索引只读取指定列，顺序无关
10 pd.read_csv(data, usecols=['列1', '列5']) # 按索引只读取指定列
11
12 # 指定列顺序，其实是 df 的筛选功能
13 pd.read_csv(data, usecols=['列1', '列5'])[['列5', '列1']]
14 pd.read_csv(data, index_col=0) # 第几列是索引
```

4.数据类型

```
1 data = 'https://www.gairuo.com/file/data/dataset/GDP-China.csv'
2 # 指定数据类型
3 pd.read_csv(data, dtype=np.float64) # 所有数据均为此数据类型
4 pd.read_csv(data, dtype={'c1':np.float64, 'c2': str}) # 指定字段的
  类型
5 # 解析日期时间
6 pd.read_csv(data, parse_dates=True) # 自动解析日期时间格式
7
```

5.导出文件

```
1 df.to_csv('done.csv')
2 df.to_csv('data/done.csv') # 可以指定文件目录路径
3 df.to_csv('done.csv', index=False) # 不要索引
4
```

二、Excel 文件

`read_excel()` 方法可以使用 `xlrd` Python 模块（可能需要安装，下同）读取 Excel 2003（`.xls`）文件。可以使用 `xlrd` 或 `openpyxl` 读取 Excel 2007+（`.xlsx`）文件，强烈建议安装 `openpyxl`。可以使用 `pyxlsb` 读取二进制 Excel（`.xlsb`）文件。`to_excel()` 实例方法用于将 `DataFrame` 保存到 Excel。大多数用法类似于 `csv`，包括文件的读取和保存。

```
1 xlsx = pd.ExcelFile('data.xlsx')
2 df = pd.read_excel(xlsx, 'Sheet1') # 读取
3 xlsx.parse('Sheet1') # 取指定标签为 DataFrame
4 # Excel 的所有标签
5 xlsx.sheet_names
6 # ['sheet1', 'sheet2', 'sheet3', 'sheet4']
```

1.文件读取

```
1 # Returns a DataFrame
2 pd.read_excel('team.xlsx') # 默认读取第一个标签页 Sheet
3 pd.read_excel('path_to_file.xls', sheet_name='Sheet1') # 指定
  Sheet
4 # 从网址 url 读取
5 pd.read_excel('https://www.gairuo.com/file/data/dataset/team.xlsx')
6 # !!! 读取的功能基本与 read_csv 一样，可参考上文
7 # 不指定索引，不指定表头，使用自动行列索引
8 pd.read_excel('tmp.xlsx', index_col=None, header=None)
9 # 指定列的数据类型
10 pd.read_excel('tmp.xlsx', index_col=0,
11               dtype={'Name': str, 'Value': float})
```

多个 `sheet` 的读取：

```
1 pd.read_excel('path_to_file.xls', sheet_name=['Sheet1',
  'Sheet2'])
```

常用的参数使用与 `read_csv` 相同

2.导出 excel

```
1 df.to_excel('path_to_file.xlsx')
2 # 指定 sheet 名, 不要索引
3 df.to_excel('path_to_file.xlsx', sheet_name='Sheet1',
4             index=False)
5 # 指定索引名, 不合并单元格
6 df.to_excel('path_to_file.xlsx', index_label='label',
7             merge_cells=False)
8 # 将多个 df 分不同 sheet 导入到一个 excel
9 with pd.ExcelWriter('path_to_file.xlsx') as writer:
10     df1.to_excel(writer, sheet_name='Sheet1')
11     df2.to_excel(writer, sheet_name='Sheet2')
```

三、JSON 格式

Pandas 可以读取和生成 **Json** 字符串, **Series** 或 **DataFrame** 都可以被转换。**JSON** 格式在网络上非常通用, 在写爬虫时可以使用极大提高效率, 在做可视化时前端的 **JS** 库往往需要接受 **Json** 格式。

1.读取 JSON

```
1 pd.read_json('data.json')
2 json = '{"columns":["col 1","col 2"],'
3        '"index":["row 1","row 2"],'
4        '"data":[["a","b"],["c","d"]]'
5        '}'
6 pd.read_json(json)
7 pd.read_json(json, orient='split') # json 格式
8
9 orient 支持:
10 - 'split' : dict like {index:[index], columns:[columns], data:
11                  [values]}
12 - 'records' : list like [{column:value}, ... , {column:value}]
13 - 'index' : dict like {index:{column:value}}
14 - 'columns' : dict like {column:{index:value}}
```

2.输出 JSON

Series 或 **DataFrame** 转换 **JSON** 的机制如下:

- **Series** :
 - 默认为 index
 - 支持 {split, records, index}
- **DataFrame**
 - 默认为 columns
 - 支持 {split, records, index, columns, values, table}

```

1 df = pd.DataFrame([[ 'a', 'b'], [ 'c', 'd']],
2                     index=[ 'row 1', 'row 2'],
3                     columns=[ 'col 1', 'col 2'])
4 # 输出 json 字符串
5 df.to_json(orient='split')

```

四、HTML

`read_html()` 函数可以接受 HTML字符串 /html文件 /URL，并将HTML表解析为 `DataFrame`。返回的是一个 `df` 列表，可以通知索引取第几个。

1.读取html

仅解析网页内 `<table>` 标签里的数据。

```

1 dfs = pd.read_html('https://www.gairuo.com/p/pandas-io')
2 dfs[0] # 查看第一个 df
3 # 读取网页文件，第一行为表头
4 dfs = pd.read_html('data.html', header=0)
5 # 第一列为索引
6 dfs = pd.read_html(url, index_col=0)
7 # !!! 常用的功能与 read_csv 相同，可参考上文

```

如果一个网页表格很多，可以指定元素来取得：

```

1 # id='table' 的表格，注意这儿仍然可能返回多个
2 dfs1 = pd.read_html(url, attrs={'id': 'table'})
3 # dfs1[0]
4 # class='sortable'
5 dfs2 = pd.read_html(url, attrs={'class': 'sortable'})

```

常用的参数使用与 `read_csv` 相同。

2.输出 html

会输出 html 表格代码字符串。

```

1 print(df.to_html())
2 print(df.to_html(columns=[0])) # 输出指定列
3 print(df.to_html(bold_rows=False)) # 表头不加粗体
4 # 表格指定样式，支持多个
5 print(df.to_html(classes=['class1', 'class2']))

```

五、剪贴板 Clipboard

剪贴板（**Clipboard**）是操作系统级的一个暂存数据的地方，它存在内存中，可以在不同软件之间传递，非常方便。**Pandas** 支持读取剪贴板中的结构化数据，这就意味着我们不用将数据保存成文件，直接从网页、文件中复制，然后中直接读取，非常方便。

读取剪贴板，它的参数使用与 `read_csv` 完全一样：

```
1 '''
2   A B C
3 x 1 4 p
4 y 2 5 q
5 z 3 6 r
6 '''
7 # 复制上边的数据，然后直接赋值
8 cdf = pd.read_clipboard()
```

保存到剪贴板：

```
1 # 执行完找个地方粘贴一下看看效果
2 df = pd.DataFrame({'A': [1, 2, 3],
3                    'B': [4, 5, 6],
4                    'C': ['p', 'q', 'r']},
5                  index=['x', 'y', 'z'])
6 df.to_clipboard()
```

六、SQL

Pandas 支持连接数据库进行查询，有以下几个方法：

- `read_sql_table(table_name, con[, schema, ...])`, 把数据表里的数据转成 **DataFrame**
- `read_sql_query(sql, con[, index_col, ...])`, 用 sql 查询数据到 **DataFrame**
- `read_sql(sql, con[, index_col, ...])`, 同时支持上边两个功能
- `DataFrame.to_sql(self, name, con[, schema, ...])`, 把记录数据写到数据库里

```
1 # 需要安装 sqlalchemy 库
2 from sqlalchemy import create_engine
3 # 创建数据库对象, sqlite 内存模式
4 engine = create_engine('sqlite:///memory:')
5 # 把表名为 data 的表数据拿出来
6 with engine.connect() as conn, conn.begin():
7     data = pd.read_sql_table('data', conn)
8
9 # data
10 # 将数据写入
11 data.to_sql('data', engine)
12 # 大量写入
13 data.to_sql('data_chunked', engine, chunksize=1000)
14 # 使用 sql 查询
```

```

15 pd.read_sql_query('SELECT * FROM data', engine)
16 # 使用 sql 查询后直接指定数据类型, 1.3.0+
17 pd.read_sql_query('SELECT * FROM data', dtype={'a': np.float64,
        'b': 'str', 'c': int})

```

七、XML

Pandas 1.3.0 的 I/O 模块添加了 `read_xml()` 和 `DataFrame.to_xml()` 支持来读取和导出 XML 文档。它使用 `lxml` 作为解析器, `xPath1.0` 和 `XSLT1.0` 都可用。

1.读取 XML

XML 文件读取的一个简单示例:

```

1  xml = """<?xml version='1.0' encoding='utf-8'?>
2  <data>
3    <row>
4      <shape>square</shape>
5      <degrees>360</degrees>
6      <sides>4.0</sides>
7    </row>
8    <row>
9      <shape>circle</shape>
10     <degrees>360</degrees>
11     <sides/>
12   </row>
13   <row>
14     <shape>triangle</shape>
15     <degrees>180</degrees>
16     <sides>3.0</sides>
17   </row>
18 </data>"""
19
20 df = pd.read_xml(xml)
21 df
22 '''
23      shape  degrees  sides
24 0   square      360    4.0
25 1   circle      360    NaN
26 2 triangle      180    3.0
27 '''

```

其他常用代码:

```

1  # 读取 URL
2  pd.read_xml("https://www.w3schools.com/xml/books.xml")
3  # 读取文件
4  with open(file_path, "r") as f:
5      df = pd.read_xml(f.read())
6  # 将文件或者字符串加载为 StringIO / BytesIO, 再读取

```

```

7 with open(file_path, "r") as f:
8     sio = StringIO(f.read())
9     # bio = BytesIO(f.read())bio = BytesIO(f.read())
10 df = pd.read_xml(sio)
11 # 从 AWS S3
12 df = pd.read_xml(
13     "s3://irs-form-990/201923199349319487_public.xml",
14     xpath="//irs:Form990PartVIISectionAGrp",
15     namespaces={"irs": "http://www.irs.gov/efile"}
16 )
17 # 使用 lxml 作为默认解析器, XPath 选择查询节点
18 pd.read_xml(file_path, xpath="//book[year=2005]")
19 # 仅读取元素或者属性
20 pd.read_xml(file_path, elems_only=True)
21 pd.read_xml(file_path, attrs_only=True)

```

2.生成 XML

输出 XML 也非常方便，以下为例：

```

1 df.to_xml() # 输出 xml 字符
2 # 指定根节点和各行的标签名称
3 df.to_xml(root_name="geometry", row_name="objects")
4 # 编写以属性为中心(attribute-centric)的XML
5 df.to_xml(attr_cols=df.columns.tolist())
6 # <row index="0" name="Liver" team="E" Q1="89" Q2="21" Q3="24"
   Q4="64"/>
7 # 编写元素和属性的组合
8 (df.to_xml(
9     index=False,
10     attr_cols=['shape'],
11     elem_cols=['degrees', 'sides'])
12 )
13 # 具有默认命名空间的 XML
14 df.to_xml(namespaces={"": "https://example.com"})
15 # 具有命名空间前缀的 XML
16 df.to_xml(namespaces={"doc": "https://example.com"},
17           prefix="doc")
18 # 编写不带声明或漂亮打印的XML
19 df.to_xml(xml_declaration=False,
20           pretty_print=False)
21 # XML和样式表转换, xsl 为样式字符串
22 df.to_xml(stylesheets=xsl)

```

八、输出 Markdown

Markdown 是一种常用的技术文档编写语言，Pandas 支持输出 Markdown 格式字符串：


```
1 print(df.to_markdown())
2 '''
3 |   | A | B | C |
4 |:---|---:|---:|:---|
5 | x | 1 | 4 | p |
6 | y | 2 | 5 | q |
7 | z | 3 | 6 | r |
8 '''
9
10 # 不需要索引
11 print(df.to_markdown(index=False))
12 # 填充空值
13 print(df.fillna('').to_markdown(index=False))
```