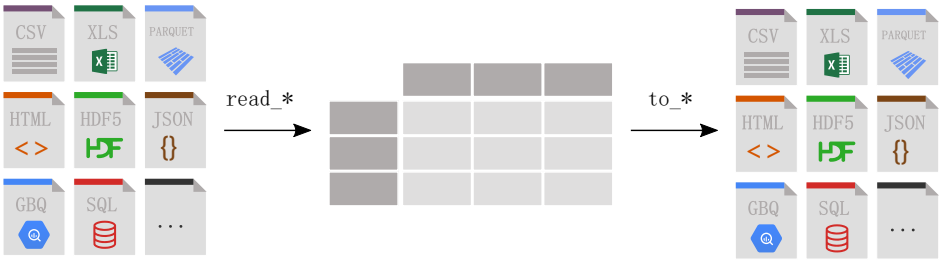
02Pandas文件读取

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处理了。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

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

```

注：csv 文件扩展名不一定是 `.csv`

2.指定分隔符号

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

3.列、索引、名称

```
 # 默认第一行是表头，可以指定，如果指定列名会被忽略
pd.read_csv(data, header=0)
pd.read_csv(data, header=None) # 没有表头
pd.read_csv(data, names=['列1', '列2']) # 指定列名列表
# 如没列名，自动指定一个：前缀加序数
pd.read_csv(data, prefix='c_', header=None)

# 读取部分列
pd.read_csv(data, usecols=[0,4,3]) # 按索引只读取指定列，顺序无关
pd.read_csv(data, usecols=['列1', '列5']) # 按索引只读取指定列

# 指定列顺序，其实是 df 的筛选功能
pd.read_csv(data, usecols=['列1', '列5'])[['列5', '列1']]
pd.read_csv(data, index_col=0) # 第几列是索引
```

4.数据类型

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

```

5.导出文件

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

```

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。

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

1.文件读取

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

多个 `Sheet` 的读取：

```
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

```
df.to_excel('path_to_file.xlsx')
# 指定 sheet 名，不要索引
df.to_excel('path_to_file.xlsx', sheet_name='Sheet1',
index=False)
# 指定索引名，不合并单元格
df.to_excel('path_to_file.xlsx', index_label='label',
merge_cells=False)
# 将多个 df 分不同 sheet 导入到一个 excel
with pd.ExcelWriter('path_to_file.xlsx') as writer:
    df1.to_excel(writer, sheet_name='Sheet1')
    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。

21世纪初，Douglas Crockford寻找一种简便的数据交换格式，能够在服务器之间交换数据。当时通用的数据交换语言是XML，但是Douglas Crockford觉得XML的生成和解析都太麻烦，所以他提出了一种简化格式，也就是Json。

Json的规格非常简单，只用一个页面几百个字就能说清楚，而且Douglas Crockford声称这个规格永远不必升级，因为该规定的都规定了。

> *1） 并列的数据之间用逗号（","）分隔。*
>
> *2） 映射用冒号（":"）表示。*
>
> *3） 并列数据的集合（数组）用方括号("[]")表示。*
>
> *4） 映射的集合（对象）用大括号（"{}"）表示。*

上面四条规则，就是Json格式的所有内容。

比如，下面这句话：

> *"北京市的面积为16800平方公里，常住人口1600万人。上海市的面积为6400平方公里，常住人口1800万。"*

写成json格式就是这样：

```
[
    {"城市":"北京","面积":16800,"人口":1600},
    {"城市":"上海","面积":6400,"人口":1800}
]
```

如果事先知道数据的结构，上面的写法还可以进一步简化：

```
[
    ["北京",16800,1600],
    ["上海",6400,1800]
]
```

由此可以看到，json非常易学易用。所以，在短短几年中，它就取代xml，成为了互联网上最受欢迎的数据交换格式。

1.读取 JSON

```
#从文件读取
pd.read_json('data.json')
#从URL读取
URL = 'https://static.runoob.com/download/sites.json'
df = pd.read_json(URL)
#自定义Json
json = '''{"columns":["col 1","col 2"],
"index":["row 1","row 2"],
"data":[["a","b"],["c","d"]]}
'''
pd.read_json(json)
pd.read_json(json, orient='split') # json 格式
'''
orient 支持：
- 'split' : dict like {index:[index], columns:[columns], data:[values]}
- 'records' : list like [{column:value}, ... , {column:value}]
- 'index' : dict like {index:{column:value}}
- '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}

```
df = pd.DataFrame([['a', 'b'], ['c', 'd']],
                  index=['row 1', 'row 2'],
                  columns=['col 1', 'col 2'])
# 输出 json 字符串
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。

```
dfs = pd.read_html('https://www.runoob.com/css/css-margin.html')
dfs[0] # 查看第一个 df
# 读取网页文件，第一行为表头
dfs = pd.read_html('data.html', header=0)
# 第一列为索引
dfs = pd.read_html(url, index_col=0)
# !!! 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：

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

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

```
#输出一个html文件
header = '''
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
'''
footer = '''
    </body>
</html>
'''
```

```
with open("data.html",'w') as f:
    f.write(header)
    f.write(df.to_html(classes='df',bold_rows=False))
    f.write(df1.to_html(classes='df1'))
    f.write(footer)
```

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：

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

保存到剪贴板：

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

六、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：

```
xml = """<?xml version='1.0' encoding='utf-8'?>
<data>
 <row>
    <shape>square</shape>
    <degrees>360</degrees>
    <sides>4.0</sides>
 </row>
 <row>
    <shape>circle</shape>
```

```
    <degrees>360</degrees>
    <sides/>
 </row>
 <row>
    <shape>triangle</shape>
    <degrees>180</degrees>
    <sides>3.0</sides>
 </row>
 </data>"""

df = pd.read_xml(xml)
df
'''
      shape  degrees  sides
0    square      360    4.0
1    circle      360    NaN
2  triangle      180    3.0
'''
```

其他常用代码：

```
# 读取 URL
pd.read_xml("https://www.w3school.com.cn/example/xdom/books.xml")
# 读取文件
with open(file_path, "r") as f:
    df = pd.read_xml(f.read())
# 将文件或者字符串加载为 StringIO / BytesIO，再读取
with open(file_path, "r") as f:
    sio = StringIO(f.read())
    # bio = BytesIO(f.read())bio = BytesIO(f.read())
df = pd.read_xml(sio)

# 仅读取元素或者属性
pd.read_xml(file_path, elems_only=True)
pd.read_xml(file_path, attrs_only=True)
```

2.生成 XML

输出 `XML` 也非常方便，以下为示例：

```python
df.to_xml() # 输出 xml 字符
# 指定根节点和各行的标签名称
df.to_xml(root_name="geometry", row_name="objects")
# 编写以属性为中心(attribute-centric)的XML
df.to_xml(attr_cols=df.columns.tolist())

# 编写元素和属性的组合
df.to_xml(
        index=False,
        attr_cols=['shape'],
        elem_cols=['degrees', 'sides'])
```

七、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：

```python
df.to_markdown()
'''
|    |   A |   B | C   |
|:---|----:|----:|:----|
| x  |   1 |   4 | p   |
| y  |   2 |   5 | q   |
| z  |   3 |   6 | r   |
'''

# 不需要索引
print(df.to_markdown(index=False))
# 填充空值
print(df.fillna('').to_markdown(index=False))
```