> 教你学会 Pandas 不是我的目的，**教你轻松玩转 Pandas 才是我的目的。**我会通过一系列实例来带入 Pandas 的知识点，让你在学习 Pandas 的路上不再枯燥。
>
> 声明：我所写的**轻松玩转 Pandas 教程都是免费的**，如果对你有帮助，你可以持续关注我。

In [1]:
```
# 导入相关库
import numpy as np
import pandas as pd
from io import StringIO
```

executed in 6ms, finished 16:39:09 2018-08-06

数据分析过程中经常需要进行读写操作，Pandas实现了很多 IO 操作的API，这里简单做了一个列举。

格式类型	数据描述	Reader	Writer
text	CSV	read_csv	to_csv
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	clipboard	read_clipboard	to_clipboard
binary	Excel	read_excel	to_excel
binary	HDF5	read_hdf	to_hdf
binary	Feather	read_feather	to_feather
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql

格式类型	数据描述	Reader	Writer
SQL	Google Big Query	read_gbq	to_gbq

可以看到，Pandas 的 I/O API是像 `pd.read_csv()` 一样访问的一组顶级 `reader` 函数，相应的 `writer` 函数是像 `df.to_csv()` 那样访问的对象方法。

这里我们介绍几个常用的API。

read_csv

读取 csv 文件算是一种最常见的操作了。假如已经有人将一些用户的信息记录在了一个csv文件中，我们如何通过 Pandas 读取呢？

读取之前先来看下这个文件里的内容吧。

In [2]:
```
!cat ../data/user_info.csv
```

executed in 428ms, finished 16:39:10 2018-08-06

```
name,age,birth,sex
Tom,18.0,2000-02-10,
Bob,30.0,1988-10-17,male
```

可以看到，一共有 4 列，分别是 `name,age,birth,sex` 。我们可以直接使用 `pd.read_csv` 来读取。

In [3]:
```
pd.read_csv("../data/user_info.csv")
```

executed in 28ms, finished 16:39:10 2018-08-06

Out[3]:

	name	age	birth	sex
0	Tom	18.0	2000-02-10	NaN
1	Bob	30.0	1988-10-17	male

可以看到，读取出来生成了一个 DataFrame，索引是自动创建的一个数字，我们可以设置参数 `index_col` 来将某列设置为索引，可以传入索引号或者名称。

In [4]:
```
pd.read_csv("../data/user_info.csv", index_col="name")
```

executed in 36ms, finished 16:39:10 2018-08-06

Out[4]:

name	age	birth	sex
Tom	18.0	2000-02-10	NaN
Bob	30.0	1988-10-17	male

除了可以从文件中读取，我们还可以从 StringIO 对象中读取。

In [5]:
```
data="name,age,birth,sex\nTom,18.0,2000-02-10,\nBob,30.0,1988-10-17,male"
print(data)
```

executed in 35ms, finished 16:39:10 2018-08-06

```
name,age,birth,sex
Tom,18.0,2000-02-10,
Bob,30.0,1988-10-17,male
```

In [6]:
```
pd.read_csv(StringIO(data))
```

executed in 40ms, finished 16:39:10 2018-08-06

Out[6]:

	name	age	birth	sex
0	Tom	18.0	2000-02-10	NaN
1	Bob	30.0	1988-10-17	male

当然了，你还可以设置参数 sep 来自定义字段之间的分隔符，设置参数 lineterminator 来自定义每行的分隔符。

In [7]:
```
data = "name|age|birth|sex~Tom|18.0|2000-02-10|~Bob|30.0|1988-10-17|male"

pd.read_csv(StringIO(data), sep="|", lineterminator="~")
```

executed in 30ms, finished 16:39:10 2018-08-06

Out[7]:

	name	age	birth	sex
0	Tom	18.0	2000-02-10	NaN
1	Bob	30.0	1988-10-17	male

在读取时，解析器会进行类型推断，任何非数字列都会以对象dtype的形式出现。当然我们也可以自己指定数据类型。

In [8]:
```
pd.read_csv(StringIO(data), sep="|", lineterminator="~", dtype={"age": int})
```

executed in 45ms, finished 16:39:10 2018-08-06

Out[8]:

	name	age	birth	sex
0	Tom	18	2000-02-10	NaN
1	Bob	30	1988-10-17	male

Pandas 默认将第一行作为标题，但是有时候，csv文件并没有标题，我们可以设置参数 `names` 来添加标题。

In [9]:
```
data="Tom,18.0,2000-02-10,\nBob,30.0,1988-10-17,male"
print(data)
```

executed in 20ms, finished 16:39:10 2018-08-06

```
Tom,18.0,2000-02-10,
Bob,30.0,1988-10-17,male
```

In [10]:
```
pd.read_csv(StringIO(data), names=["name", "age", "birth", "sex"])
```

executed in 52ms, finished 16:39:10 2018-08-06

Out[10]:

	name	age	birth	sex
0	Tom	18.0	2000-02-10	NaN
1	Bob	30.0	1988-10-17	male

有时候可能只需要读取部分列的数据，可以指定参数 `user_cols`

In [11]:
```
data="name,age,birth,sex\nTom,18.0,2000-02-10,\nBob,30.0,1988-10-17,male"
print(data)
```

executed in 26ms, finished 16:39:10 2018-08-06

```
name,age,birth,sex
Tom,18.0,2000-02-10,
Bob,30.0,1988-10-17,male
```

In [12]:
```
pd.read_csv(StringIO(data), usecols=["name", "age"])
```

executed in 32ms, finished 16:39:10 2018-08-06

Out[12]:

	name	age
0	Tom	18.0
1	Bob	30.0

关于缺失值的处理，也是有技巧的。默认参数 `keep_default_na=False`，会将空值都填充为 NaN。

In [13]:
```
pd.read_csv(StringIO(data))
```

executed in 37ms, finished 16:39:10 2018-08-06

Out[13]:

	name	age	birth	sex
0	Tom	18.0	2000-02-10	NaN
1	Bob	30.0	1988-10-17	male

In [14]:
```
pd.read_csv(StringIO(data), keep_default_na=False)
```

executed in 40ms, finished 16:39:10 2018-08-06

Out[14]:

	name	age	birth	sex
0	Tom	18.0	2000-02-10	
1	Bob	30.0	1988-10-17	male

有时候，空值的定义比较广泛，假定我们认为 18 也是空值，那么将它加入到参数 `na_values` 中即可。

In [15]:
```
pd.read_csv(StringIO(data), na_values=[18])
```

executed in 33ms, finished 16:39:10 2018-08-06

Out[15]:

	name	age	birth	sex
0	Tom	NaN	2000-02-10	NaN
1	Bob	30.0	1988-10-17	male

了解了 `pd.read_csv` 如何使用之后，`to_csv` 就非常方便了，这里就不做介绍了。

to_json

通常在得到了 DataFrame 之后，有时候我们需要将它转为一个 json 字符串，可以使用 `to_json` 来完成。

转换时，可以通过指定参数 `orient` 来输出不同格式的格式，之后以下几个参数：

split	字典像索引 - > [索引]，列 - > [列]，数据 - > [值]}
records	列表像{[列 - >值}，...，{列 - >值}]
index	字典像{索引 - > {列 - >值}}
columns	字典像{列 - > {索引 - >值}}
values	只是值数组

DataFrame 默认情况下使用 `columns` 这种形式，Series 默认情况下使用 `index` 这种形式。

设置为 `columns` 后会将数据作为嵌套JSON对象进行序列化，并将列标签作为主索引。

In [16]:
```
df = pd.read_csv("../data/user_info.csv", index_col="name")
df
```

executed in 38ms, finished 16:39:10 2018-08-06

Out[16]:

name	age	birth	sex
Tom	18.0	2000-02-10	NaN
Bob	30.0	1988-10-17	male

In [17]:
```
print(df.to_json())
```

executed in 16ms, finished 16:39:10 2018-08-06

```
{"age":{"Tom":18.0,"Bob":30.0},"birth":{"Tom":"2000-02-10","Bob":"1988-10-17"},"sex":{"Tom":null,"Bob":"male"}}
```

设置为`index` 后会将数据作为嵌套JSON对象进行序列化，并将索引标签作为主索引。

In [18]:
```
print(df.to_json(orient="index"))
```
executed in 19ms, finished 16:39:10 2018-08-06

```
{"Tom":{"age":18.0,"birth":"2000-02-10","sex":null},"Bob":{"age":30.0,"birth":"1988-10-17","sex":"male"}}
```

设置为 records 后会将数据序列化为列 - >值记录的JSON数组，不包括索引标签。

In [19]:
```
print(df.to_json(orient="records"))
```
executed in 19ms, finished 16:39:10 2018-08-06

```
[{"age":18.0,"birth":"2000-02-10","sex":null},{"age":30.0,"birth":"1988-10-17","sex":"male"}]
```

设置为 values 后会将是一个仅用于嵌套JSON数组值，不包含列和索引标签。

In [20]:
```
print(df.to_json(orient="values"))
```
executed in 23ms, finished 16:39:10 2018-08-06

```
[[18.0,"2000-02-10",null],[30.0,"1988-10-17","male"]]
```

设置为 split 后会将序列化为包含值，索引和列的单独条目的JSON对象。

In [21]:
```
print(df.to_json(orient="split"))
```
executed in 36ms, finished 16:39:10 2018-08-06

```
{"columns":["age","birth","sex"],"index":["Tom","Bob"],"data":[[18.0,"2000-02-10",null],[30.0,"1988-10-17","male"]]}
```

对于 read_json，这些参数也是同样的道理。

想要学习更多关于人工智能的知识，请关注公众号：**AI派**

这里我将整篇文章的内容整理成了pdf，想要pdf文件的可以在公众号后台回复关键字：**pandas**。