> 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。

在 06-Pandas时间序列详解 (06-Pandas时间序列详解.ipynb) 介绍了使用 Pandas 中时间序列相关的处理，这节来看下如何使用 Pandas 中包含的一些计算工具吧。

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

executed in 5ms, finished 20:55:37 2018-07-09

Pandas 中包含了非常丰富的计算工具，如一些统计函数、窗口函数、聚合等计算工具。

统计函数

最常见的计算工具莫过于一些统计函数了。这里我们首先构建一个包含了用户年龄与收入的 DataFrame。

In [2]:
```
index = pd.Index(data=["Tom", "Bob", "Mary", "James", "Andy", "Alice"], name="name")

data = {
    "age": [18, 40, 28, 20, 30, 35],
    "income": [1000, 4500 , 1800, 1800, 3000, np.nan],
}

df = pd.DataFrame(data=data, index=index)
df
```

executed in 61ms, finished 20:55:37 2018-07-09

Out[2]:

name	age	income
Tom	18	1000.0
Bob	40	4500.0
Mary	28	1800.0
James	20	1800.0
Andy	30	3000.0
Alice	35	NaN

我们可以通过 cov 函数来求出年龄与收入之间的协方差，计算的时候会丢弃缺失值。

In [3]:
```
df.age.cov(df.income)
```

executed in 157ms, finished 20:55:37 2018-07-09

Out[3]: 11320.0

除了协方差之外，我们还可以通过 corr 函数来计算下它们之间的相关性，计算的时候会丢弃缺失值。

默认情况下 corr 计算相关性时用到的方法是 pearson，当然了你也可以指定 kendall 或 spearman。

In [4]:
```
df.age.corr(df.income)
```

executed in 17ms, finished 20:55:37 2018-07-09

Out[4]: 0.94416508951340194

In [5]:
```
df.age.corr(df.income, method="kendall")
```

executed in 3.44s, finished 20:55:40 2018-07-09

Out[5]: 0.94868329805051366

In [6]:
```
df.age.corr(df.income, method="spearman")
```

executed in 15ms, finished 20:55:40 2018-07-09

Out[6]: 0.97467943448089644

除了相关性的计算外，还可以通过 rank 函数求出数据的排名顺序。

In [7]:
```
df.income.rank()
```

executed in 69ms, finished 20:55:40 2018-07-09

Out[7]:
```
name
Tom      1.0
Bob      5.0
Mary     2.5
James    2.5
Andy     4.0
Alice    NaN
Name: income, dtype: float64
```

如果有相同的数，默认取其排名的平均值作为值。我们可以设置参数来得到不同的结果。可以设置的参数有：min、max、first、dense。

In [8]:
```
df.income.rank(method="first")
```

executed in 25ms, finished 20:55:40 2018-07-09

Out[8]:
```
name
Tom      1.0
Bob      5.0
Mary     2.0
James    3.0
Andy     4.0
Alice    NaN
Name: income, dtype: float64
```

窗口函数

有的时候，我们需要对不同窗口的中数据进行一个统计，常见的窗口类型为时间窗口。

例如，下面是某个餐厅 7 天的营业额，我们想要计算每两天的收入总额，如何计算呢?

In [9]:
```
data = {
    "turnover": [12000, 18000, np.nan, 12000, 9000, 16000, 18000],
    "date": pd.date_range("2018-07-01", periods=7)
}

df2 = pd.DataFrame(data=data)
df2
```

executed in 60ms, finished 20:55:40 2018-07-09

Out[9]:

	date	turnover
0	2018-07-01	12000.0
1	2018-07-02	18000.0
2	2018-07-03	NaN
3	2018-07-04	12000.0
4	2018-07-05	9000.0
5	2018-07-06	16000.0
6	2018-07-07	18000.0

通过 rolling 我们可以实现，设置 window=2 来保证窗口长度为 2，设置 on="date" 来保证根据日期这一列来滑动窗口（默认不设置，表示根据索引来滑动）

In [10]:
```
df2.rolling(window=2, on="date").sum()
```

executed in 30ms, finished 20:55:40 2018-07-09

Out[10]:

	date	turnover
0	2018-07-01	NaN
1	2018-07-02	30000.0
2	2018-07-03	NaN
3	2018-07-04	NaN
4	2018-07-05	21000.0
5	2018-07-06	25000.0
6	2018-07-07	34000.0

是不是发现，有很多结果是缺失值，导致这个结果的原因是因为在计算时，窗口中默认需要的最小数据个数与窗口长度一致，这里可以设置 `min_periods=1` 来修改下。

In [11]:
```
df2.rolling(window=2, on="date", min_periods=1).sum()
```

executed in 53ms, finished 20:55:40 2018-07-09

Out[11]:

	date	turnover
0	2018-07-01	12000.0
1	2018-07-02	30000.0
2	2018-07-03	18000.0
3	2018-07-04	12000.0
4	2018-07-05	21000.0
5	2018-07-06	25000.0
6	2018-07-07	34000.0

有时候，我想要计算每段时间的累加和，如何实现呢？先来看看第一种方式吧。

In [12]:
```
df2.rolling(window=len(df2), on="date", min_periods=1).sum()
```

executed in 59ms, finished 20:55:41 2018-07-09

Out[12]:

	date	turnover
0	2018-07-01	12000.0
1	2018-07-02	30000.0
2	2018-07-03	30000.0
3	2018-07-04	42000.0
4	2018-07-05	51000.0
5	2018-07-06	67000.0
6	2018-07-07	85000.0

还有另外一种方式，直接使用 expanding 来生成窗口。

In [13]:
```
df2.expanding(min_periods=1)["turnover"].sum()
```

executed in 32ms, finished 20:55:41 2018-07-09

Out[13]:
```
0    12000.0
1    30000.0
2    30000.0
3    42000.0
4    51000.0
5    67000.0
6    85000.0
Name: turnover, dtype: float64
```

除了可以使用 sum 函数外，还有很多其他的函数可以使用，如：count、mean、median、min、max、std、var、quantile、apply、cov、corr等等。

方法	描述
count()	非空观测值数量
sum()	值的总和
mean()	价值的平均值
median()	值的算术中值
min()	最小值
max()	最大
std()	贝塞尔修正样本标准差
var()	无偏方差
skew()	样品偏斜度（三阶矩）
kurt()	样品峰度（四阶矩）
quantile()	样本分位数（百分位上的值）
apply()	通用适用
cov()	无偏协方差（二元）

方法	描述
corr()	相关（二进制）

不过上面的方式只能生成一个结果，有时候想要同时求出多个结果（如求和和均值），如何实现呢？

借助 agg 函数可以快速实现。

In [14]:
```
df2.rolling(window=2, min_periods=1)["turnover"].agg([np.sum, np.mean])
```

executed in 60ms, finished 20:55:41 2018-07-09

Out[14]:

	sum	mean
0	12000.0	12000.0
1	30000.0	15000.0
2	18000.0	18000.0
3	12000.0	12000.0
4	21000.0	10500.0
5	25000.0	12500.0
6	34000.0	17000.0

如果传入一个字典，可以为生成的统计结果重命名。

In [15]:
```
df2.rolling(window=2, min_periods=1)["turnover"].agg({"tur_sum": np.sum, "tur_mean": np.mean})
```

executed in 22ms, finished 20:55:41 2018-07-09

Out[15]:

	tur_sum	tur_mean
0	12000.0	12000.0
1	30000.0	15000.0
2	18000.0	18000.0
3	12000.0	12000.0
4	21000.0	10500.0
5	25000.0	12500.0
6	34000.0	17000.0

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。**