

Pandas 使用教程

一、实验介绍

1.1 实验内容

Pandas 是非常著名的开源数据处理工具，我们可以通过它对数据集进行快速读取、转换、过滤、分析等一系列操作。除此之外，Pandas 拥有强大的缺失数据处理与数据透视功能，可谓是数据预处理中的必备利器。这是 Pandas 使用教程的第 5 章节，将了解 Pandas 针对时间序列分析和处理的相关方法。

1.2 实验知识点

- 时间戳 Timestamp
- 时间索引 DatetimeIndex
- 时间转换 to_datetime
- 时间序列检索
- 时间序列计算

1.3 实验环境

- python2.7
- Xfce 终端
- ipython 终端

1.4 适合人群

本课程难度为一般，属于初级级别课程，适合具有 Python 基础，并对使用 Pandas 进行数据处理感兴趣的用户。

下面的内容均在 iPython 交互式终端 中演示，你可以通过在线环境左下角的应用程序菜单 > 附件打开。如果你在本地进行练习，推荐使用 Jupyter Notebook 环境。

二、时间序列分析介绍

2.1 简介

时间序列（英语：time series）是实证经济学的一种统计方法，它是采用时间排序的一组随机变量，国内生产毛额（GDP）、消费者物价指数（CPI）、股价指数、利率、汇率等等都是时间序列。时间序列的时间间隔可以是分秒（如高频金融数据），可以是日、周、月、季度、年、甚至更大的时间单位。[维基百科]

我们针对时间序列数据进行挖掘的过程又被成为时间序列分析，简称：时序分析。

2.2 常见问题

Pandas 经常被用于处理与时间序列相关的数据，尤其是像财务数据。在处理时间序列数据时，会遇到各类需求，包括但不限于：

1. 生成固定跨度的时期构成时间序列。
2. 将现有的时间序列，转换成需要的时间序列格式。
3. 计算序列中的相对时间，例如：每季度的第一周。

三、Pandas 处理时间序列

接下来，我们就时间序列中常遇到的一些需求类型，列举一些示例，并使用 Pandas 提供的方法进行处理。

3.1 时间戳 Timestamp

既然是时间序列类型的数据，那么就少不了时间戳这一关键元素。Pandas 中，我们有两个创建时间戳的方法，分别是：`to_datetime` 和 `Timestamp`。

`to_datetime` 后面集中详说。首先看一看 `Timestamp`，它针对于单一标量，举个例子：

```
import pandas as pd

pd.Timestamp('2017-10-01')
```

```
In [2]: pd.Timestamp('2017-10-01')
Out[2]: Timestamp('2017-10-01 00:00:00')
```

如果要包含小时：分钟：秒：

```
pd.Timestamp('2017-10-01 13:30:59')
```

```
In [3]: pd.Timestamp('2017-10-01 13:30:59')
Out[3]: Timestamp('2017-10-01 13:30:59')
```

当然，还支持其他的格式输入，比如：

```
pd.Timestamp('1/10/2017 13:30:59')
```

```
In [4]: pd.Timestamp('1/10/2017 13:30:59')
Out[4]: Timestamp('2017-01-10 13:30:59')
```

3.2 时间索引 DatetimeIndex

在实际工作中，我们很少遇到用单个时间戳的情况。而大多数时候，是使用由时间戳构成的时间索引。

首先，我们来看一下如何使用 Pandas 创建时间索引。这里用到的方法为 `date_range()`，`date_range()` 和 python 自带的 `range()` 很相似。它可以用来创建一系列等间距时间，并作为 Series 或者 DataFrame 的索引。

`date_range()` 方法带有的默认参数如下：

```
pandas.date_range(start=None, end=None, periods=None, freq='D', tz=None, normalize=False, name=None, closed=None, **kwargs)
```

常用参数的含义如下：

- start= ：设置起始时间
- end= ：设置截至时间
- periods= ：设置时间区间，若 None 则需要单独设置起止和截至时间。
- freq= ：设置间隔周期，默认为 D，也就是天。可以设置为小时、分钟、秒等。
- tz= ：设置时区。

举个例子：

```
import pandas as pd
```

```
rng1 = pd.date_range('1/10/2017', periods=24, freq='H')
```


```
In [5]: rng1
Out[5]:
DatetimeIndex(['2017-01-10 00:00:00', '2017-01-10 01:00:00',
                '2017-01-10 02:00:00', '2017-01-10 03:00:00',
                '2017-01-10 04:00:00', '2017-01-10 05:00:00',
                '2017-01-10 06:00:00', '2017-01-10 07:00:00',
                '2017-01-10 08:00:00', '2017-01-10 09:00:00',
                '2017-01-10 10:00:00', '2017-01-10 11:00:00',
                '2017-01-10 12:00:00', '2017-01-10 13:00:00',
                '2017-01-10 14:00:00', '2017-01-10 15:00:00',
                '2017-01-10 16:00:00', '2017-01-10 17:00:00',
                '2017-01-10 18:00:00', '2017-01-10 19:00:00',
                '2017-01-10 20:00:00', '2017-01-10 21:00:00',
                '2017-01-10 22:00:00', '2017-01-10 23:00:00'],
              dtype='datetime64[ns]', freq='H')
```

实验楼
shiyancelou.com

可以这样：

```
rng2 = pd.date_range('1/10/2017', periods=10, freq='D')
```

```
In [7]: rng2
Out[7]:
DatetimeIndex(['2017-01-10', '2017-01-11', '2017-01-12', '2017-01-13',
               '2017-01-14', '2017-01-15', '2017-01-16', '2017-01-17',
               '2017-01-18', '2017-01-19'],
              dtype='datetime64[ns]', freq='D')
```



我们可以发现 freq= 参数的特点：

- freq='s' : 秒
- freq='min' : 分钟
- freq='H' : 小时
- freq='D' : 天
- freq='w' : 周
- freq='m' : 月


除了上面这些参数值，还有一些特别的：

- freq='BM' : 每个月最后一天
- freq='W' : 每周的星期日

如果你想同时按天、小时更新，也是可以的。但需要像下面这样设置参数值：

```
rng3 = pd.date_range('1/10/2017', periods=20, freq='1H20min')
```

```
In [10]: rng3
Out[10]:
DatetimeIndex(['2017-01-10 00:00:00', '2017-01-10 01:20:00',
               '2017-01-10 02:40:00', '2017-01-10 04:00:00',
               '2017-01-10 05:20:00', '2017-01-10 06:40:00',
               '2017-01-10 08:00:00', '2017-01-10 09:20:00',
               '2017-01-10 10:40:00', '2017-01-10 12:00:00',
               '2017-01-10 13:20:00', '2017-01-10 14:40:00',
               '2017-01-10 16:00:00', '2017-01-10 17:20:00',
               '2017-01-10 18:40:00', '2017-01-10 20:00:00',
               '2017-01-10 21:20:00', '2017-01-10 22:40:00',
               '2017-01-11 00:00:00', '2017-01-11 01:20:00'],
              dtype='datetime64[ns]', freq='80T')
```



所以，只要适当地组合，你可以生成任意想要的时间序列索引。

3.3 时间转换 to_datetime

`to_datetime` 是 Pandas 用于处理时间序列时的一个重要方法，它可以将实参转换为时间戳。`to_datetime` 包含的默认参数如下：

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, box=True, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix')
```

- `arg`：可以接受整数、浮点数、字符串、时间、列表、元组、一维数组、Series 等。
- `errors=`：默认为 `raise`，表示遇到无法解析数据将会报错。还可以设置为 `coerce`，表示无法解析设为 `NaT`，或者设为 `ignore` 忽略错误。
- `dayfirst=`：表示首先解析日期，例如：1/10/17 被解析为 2017-10-1。
- `yearfirst=`：表示首先解析年，例如：1/10/17 被解析为 2001-10-17。
- `utc=`：返回 UTC 格式时间索引。
- `box=`：True 表示返回时间索引 `DatetimeIndex`，False 表示返回多维数组 `ndarray`。
- `format=`：时间解析格式，例如：%d /%m /%Y。

对于 `to_datetime` 的返回值而言：

- 输入列表，默认返回时间索引 `DatetimeIndex`。
- 输入 Series，默认返回 `datetime64` 的 Series。
- 输入标量，默认返回时间戳 `Timestamp`。

下面，针对输入数据类型的不同，我们来看一看 `to_datetime` 的不同用法。

3.3.1 输入标量

```
import pandas as pd


pd.to_datetime('1/10/2017 10:00', dayfirst=True)
```

```
In [5]: pd.to_datetime('1/10/2017 10:00', dayfirst=True)
Out[5]: Timestamp('2017-10-01 10:00:00')
```

3.3.2 输入列表


```
pd.to_datetime(['1/10/2017 10:00', '2/10/2017 11:00', '3/10/2017 12:00'])
```


```
In [6]: pd.to_datetime(['1/10/2017 10:00', '2/10/2017 11:00', '3/10/2017 12:00'])
Out[6]:
DatetimeIndex(['2017-01-10 10:00:00', '2017-02-10 11:00:00',
                '2017-03-10 12:00:00'],
              dtype='datetime64[ns]', freq=None)
```



3.3.2 输入 Series

```
pd.to_datetime(pd.Series(['Oct 11, 2017', '2017-10-2', '3/10/2017']),
               dayfirst=True)
```


```
In [8]: pd.to_datetime(pd.Series(['Oct 11, 2017', '2017-10-2', '3/10/2017']), day
first=True)
Out[8]:
0    2017-10-11
1    2017-10-02
2    2017-10-03
dtype: datetime64[ns]
```



3.3.2 输入 DataFrame

```
pd.to_datetime(pd.DataFrame({'year': [2017, 2018], 'month': [9, 10], '
day': [1, 2], 'hour': [11, 12]}))
```

```
In [11]: pd.to_datetime(pd.DataFrame({'year': [2017, 2018], 'month': [9, 10], 'd
ay': [1, 2], 'hour': [11, 12]}))
Out[11]:
0    2017-09-01 11:00:00
1    2018-10-02 12:00:00
dtype: datetime64[ns]
```



3.3.2 errors=

接下来，看一看 `errors=` 遇到无法解析的数据时，所对应的不同返回值。这个参数对于我们解析大量数据时非常有用。

```
pd.to_datetime(['2017/10/1', 'abc'], errors='raise')
```

```
ValueError: Unknown string format
```

```
pd.to_datetime(['2017/10/1', 'abc'], errors='ignore')
```

```
In [13]: pd.to_datetime(['2017/10/1', 'abc'], errors='ignore')
Out[13]: array(['2017/10/1', 'abc'], dtype=object)
```

```
pd.to_datetime(['2017/10/1', 'abc'], errors='coerce')
```

```
In [14]: pd.to_datetime(['2017/10/1', 'abc'], errors='coerce')
Out[14]: DatetimeIndex(['2017-10-01', 'NaT'], dtype='datetime64[ns]', freq=None)
```

3.4 时间序列检索

上面，我们介绍了时间索引 `DatetimeIndex` 的生成方法。那么，它主要是用来做什么呢？

答案当然是 Pandas 对象的索引啦。将时间变成索引的优点非常多，包括但不限于：

1. 查找和检索特定日期的字段非常快。
2. 进行数据对齐时，拥有相同时间间隔的索引的数据将会非常快。
3. 可以很方便地通过 `shift` 和 `ishift` 方法快速移动对象。

下面，针对时间序列索引的检索等操作举几个例子。首先，我们生成 10 万条数据：

```
import pandas as pd
import numpy as np

ts = pd.DataFrame(np.random.randn(100000,1), columns=['Value'], index=
pd.date_range('20170101', periods=100000, freq='T'))
```



```
In [7]: ts
```

```
Out[7]:
```

		Value
2017-01-01	00:00:00	-0.288511
2017-01-01	00:01:00	-1.772100
2017-01-01	00:02:00	-1.533241
2017-01-01	00:03:00	0.222284
2017-01-01	00:04:00	-1.921452
2017-01-01	00:05:00	1.026253
2017-01-01	00:06:00	0.822671
2017-01-01	00:07:00	-0.657731
2017-01-01	00:08:00	-0.807754
2017-01-01	00:09:00	-0.319089

2017-03-11	10:35:00	1.640321
2017-03-11	10:36:00	-0.741736
2017-03-11	10:37:00	1.876275
2017-03-11	10:38:00	0.728544
2017-03-11	10:39:00	0.979845

```
[100000 rows x 1 columns]
```



当我们对数据进行快速检索是，其实和除了 Series 和 DataFrame 数据别无二致。例如：

检索 2017 年 3 月 2 号的数据：

```
ts['2017-3-2']
```

一共 1440 行

```
In [8]: ts['2017-3-2']
```

```
Out[8]:
```

		Value
2017-03-02	00:00:00	-1.011132
2017-03-02	00:01:00	1.669641
2017-03-02	00:02:00	-0.741481



检索 2017 年 3 月 3 号下午 2 点到 5 点 23 分之间的数据：

```
ts['2017-3-2 14:00:00':'2017-3-2 17:23:00']
```

一共返回了 204 行

```
In [9]: ts['2017-3-2 14:00:00':'2017-3-2 17:23:00']
```

```
Out[9]:
```

	Value
2017-03-02 14:00:00	0.115440
2017-03-02 14:01:00	0.538701
2017-03-02 14:02:00	-1.147937
2017-03-02 14:03:00	-0.367197
2017-03-02 14:04:00	-0.724661
2017-03-02 14:05:00	-0.116990



总之，一切在 Series 和 DataFrame 上可以用的数据选择与定位的方法，像 `iloc()`，`loc()` 等均可以用于时间序列，这里就不再赘述了。

3.5 时间序列计算

在 Pandas 中，包含有很多可以被加入到时间序列计算中去的类，这些被称为 Offsets 对象。

关于这一点，我们举出几个例子就一目了然了。例如：

```
import pandas as pd
from pandas.tseries import offsets # 载入 offsets

dt = pd.Timestamp('2017-10-1 10:59:59')

dt + offsets.DateOffset(months=1, days=2, hour=3) # 增加时间
```

```
In [13]: dt + offsets.DateOffset(months=1, days=2, hour=3)
Out[13]: Timestamp('2017-11-03 03:59:59')
```

又或者我们减去 3 个周的时间：

```
dt - offsets.Week(3)
```

```
In [14]: dt - offsets.Week(3)
Out[14]: Timestamp('2017-09-10 10:59:59')
```

看明白了吧。这类的对象非常多，就不再一一演示，通过表格列举如下：

类名	描述
DateOffset	自定义设置，默认一周
BDay	工作日
CDay	自定义工作日
Week	周
WeekOfMonth	每月y周x日
LastWeekOfMonth	每月最后一周x日
MonthEnd	每月最末
MonthBegin	每月初
SemiMonthEnd	每月从尾向前数 15 天
SemiMonthBegin	每月从前向后数 15 天
QuarterEnd	1/4
QuarterBegin	1/4
YearEnd	年末
YearBegin	年初
Hour	1 小时
Minute	1 分钟
Second	1 秒
Milli	0.001 秒
Micro	1 毫秒
Nano	1 纳秒



3.6 其他方法

最后，再介绍几个与时间序列处理相关的方法。

移动 **Shifting**

shifting 可以将数据或者时间索引沿着时间轴的方向前移或后移，举例如下：

```
import pandas as pd
import numpy as np

# 生成一个时间系列数据集
ts = pd.DataFrame(np.random.randn(7,2), columns=['Value1','Value2' ],
index=pd.date_range('20170101', periods=7, freq='T'))
```

```
In [5]: ts
```

```
Out[5]:
```

	Value1	Value2
2017-01-01 00:00:00	0.192804	-0.551272
2017-01-01 00:01:00	0.613915	-0.172180
2017-01-01 00:02:00	-0.787709	0.281113
2017-01-01 00:03:00	-0.955857	0.234141
2017-01-01 00:04:00	-0.132275	0.274843
2017-01-01 00:05:00	1.766219	1.090129
2017-01-01 00:06:00	2.190012	1.168161



接下来开始移动。

```
ts.shift(3)
```

默认是数据向后移动。这里数据值向后移动了 3 行。

```
In [6]: ts.shift(3)
```

```
Out[6]:
```

	Value1	Value2
2017-01-01 00:00:00	NaN	NaN
2017-01-01 00:01:00	NaN	NaN
2017-01-01 00:02:00	NaN	NaN
2017-01-01 00:03:00	0.192804	-0.551272
2017-01-01 00:04:00	0.613915	-0.172180
2017-01-01 00:05:00	-0.787709	0.281113
2017-01-01 00:06:00	-0.955857	0.234141



```
ts.shift(-3)
```

可以通过添加负号，使得向前移动。

```
In [7]: ts.shift(-3)
```

```
Out[7]:
```

	Value1	Value2
2017-01-01 00:00:00	-0.955857	0.234141
2017-01-01 00:01:00	-0.132275	0.274843
2017-01-01 00:02:00	1.766219	1.090129
2017-01-01 00:03:00	2.190012	1.168161
2017-01-01 00:04:00	NaN	NaN
2017-01-01 00:05:00	NaN	NaN
2017-01-01 00:06:00	NaN	NaN



那么，想移动索引怎么办？这里使用 `tshift()`。

```
ts.tshift(3)
```

```
In [8]: ts.tshift(3)
Out[8]:
```

	Value1	Value2
2017-01-01 00:03:00	0.192804	-0.551272
2017-01-01 00:04:00	0.613915	-0.172180
2017-01-01 00:05:00	-0.787709	0.281113
2017-01-01 00:06:00	-0.955857	0.234141
2017-01-01 00:07:00	-0.132275	0.274843
2017-01-01 00:08:00	1.766219	1.090129
2017-01-01 00:09:00	2.190012	1.168161



向前移动索引就不再演示了，同样可以通过负号完成。除此之外，`shift()` 是可以接受一些参数的，比如 `freq=''`。而这里的 `freq=''` 参数和上文在介绍时间索引 `DatetimeIndex` 时提到的一致。

举个例子：

```
ts.shift(3, freq='D') # 日期向后移动 3 天
```

```
In [9]: ts.shift(3, freq='D')
Out[9]:
```

	Value1	Value2
2017-01-04 00:00:00	0.192804	-0.551272
2017-01-04 00:01:00	0.613915	-0.172180
2017-01-04 00:02:00	-0.787709	0.281113
2017-01-04 00:03:00	-0.955857	0.234141
2017-01-04 00:04:00	-0.132275	0.274843
2017-01-04 00:05:00	1.766219	1.090129
2017-01-04 00:06:00	2.190012	1.168161



所以说，shifting 可以让我们更加灵活地去操作时间序列数据集，完成数据对齐等目标。

重采样 Resample

重采样，即是将时间序列从一个频率转换到另一个频率的过程。实施重采样的情

形如下：

1. 有时候，我们的时间序列数据集非常大，比如百万级别甚至更高。如果将全部数据用于后序计算，其实很多情况下是没有必要的。此时，我们可以对原有的时间序列进行降频采样。
2. 除了上面的情形，重采样还可以被用于数据对齐。比如，两个数据集，但是时间索引的频率不一致，这时候，可以通过重采样使二者频率一致，方便数据合并、计算等操作。

下面，我们看一看 `resample()` 的使用。首先，还是生成一个数据集。


```
import pandas as pd
import numpy as np

# 生成一个时间序列数据集
ts = pd.DataFrame(np.random.randn(50,1), columns=['Value' ], index=pd.
date_range('2017-01', periods=50, freq='D'))
```

```
In [12]: ts = pd.DataFrame(np.random.randn(50,1), columns=['Value' ], index=pd.d
ate_range('2017-01', periods=50, freq='D'))

In [13]: ts
Out[13]:
```

	Value
2017-01-01	0.434114
2017-01-02	-1.565438
2017-01-03	-0.612463
2017-01-04	-1.755423
2017-01-05	1.471784



首先，可以升频采样，间隔变成小时。但是，由于间隔变小，我们就必须对新增的行进行填充。

```
ts.resample('H').ffill()
```



```
In [15]: ts.resample('H').ffill()
```

```
Out[15]:
```

	Value
2017-01-01 00:00:00	0.434114
2017-01-01 01:00:00	0.434114
2017-01-01 02:00:00	0.434114
2017-01-01 03:00:00	0.434114
2017-01-01 04:00:00	0.434114
2017-01-01 05:00:00	0.434114
2017-01-01 06:00:00	0.434114
2017-01-01 07:00:00	0.434114



下面，接着开始降频采样，从 1 天变成 5 天：

```
ts.resample('5D').sum()
```

```
In [16]: ts.resample('5D').sum()
```

```
Out[16]:
```

	Value
2017-01-01	-2.027426
2017-01-06	-2.668514
2017-01-11	0.488783
2017-01-16	1.419202
2017-01-21	-1.166634
2017-01-26	-1.182276
2017-01-31	-1.206285
2017-02-05	-1.531893
2017-02-10	-0.212370
2017-02-15	1.347958



四、实验总结

本章节介绍了利用 Pandas 对时间序列数据进行处理的一些手段，重点演示了时间索引的构建、时间索引转换以及移动、重采样等方法。当然，文中对这些方法的介绍依然还不够细致。如果你需要在实际工作中进行数据处理，还需要对照官方文档熟悉每一个方法的每一个参数的使用，这样才能发挥出 Pandas 的强大作用。

*本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。

上一节：Pandas 进行缺失值处理 (</courses/906/labs/3378/document>)