Table of Contents

In [1]:
```
#全部行都能输出
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# 导入相关库
import numpy as np
import pandas as pd
from datetime import datetime
from datetime import timedelta
```

在做金融领域方面的分析时，经常会对时间进行一系列的处理。

Pandas 内部自带了很多关于时间序列相关的工具，所以它非常适合处理时间序列。在处理时间序列的的过程中，我们经常会去做以下一些任务：

1. 生成固定频率日期和时间跨度的序列
2. 将时间序列整合或转换为特定频率
3. 基于各种非标准时间增量（例如，在一年的最后一个工作日之前的5个工作日）计算“相对”日期，或向前或向后“滚动”日期

使用 Pandas 可以轻松完成以上任务。

1 基础概述

下面列出了 Pandas中 和时间日期相关常用的类以及创建方法。

类	备注	创建方法
Timestamp	时刻数据	to_datetime，Timestamp
DatetimeIndex	Timestamp的索引	to_datetime，date_range，DatetimeIndex
Period	时期数据	Period
PeriodIndex	Period	period_range，PeriodIndex

创建Timestamp的方法有两种: 使用 to_datetime , Timestamp

1.1 生成时间戳

In [2]:
```
pd.to_datetime('2018-12-12')

pd.to_datetime('2018.12.12')

pd.to_datetime('2018/12/12')

pd.to_datetime('2018, 12, 12')

pd.to_datetime("dec 12, 2018")
```

Out[2]: Timestamp('2018-12-12 00:00:00')

Out[2]: Timestamp('2018-12-12 00:00:00')

Out[2]: Timestamp('2018-12-12 00:00:00')

Out[2]: Timestamp('2018-12-12 00:00:00')

Out[2]: Timestamp('2018-12-12 00:00:00')

In [3]:
```
pd.Timestamp('2018-12-12')

pd.Timestamp('2018.12.12')

pd.Timestamp('2018/12/12')

pd.Timestamp('2018, 12, 12')

pd.Timestamp("dec 12, 2018")
```

Out[3]: Timestamp('2018-12-12 00:00:00')

Out[3]: Timestamp('2018-12-12 00:00:00')

Out[3]: Timestamp('2018-12-12 00:00:00')

Out[3]: Timestamp('2018-12-12 00:00:00')

Out[3]: Timestamp('2018-12-12 00:00:00')

1.2 转换时间戳

可以把各种各样的字符串格式转换为时间戳（缺失时间标记为NaT）

In [4]:
```
a=["Jul 31, 2018", "2018-05-10", None, "2005/11/23", "2010.12.31"]
b = pd.to_datetime(a) #空的时间是NaT
b
```

Out[4]: DatetimeIndex(['2018-07-31', '2018-05-10', 'NaT', '2005-11-23', '2010-12-31'], dtype='datetime64[ns]', freq=None)

In [9]:
```
pd.Series(b)
```

Out[9]:
```
0   2018-07-31
1   2018-05-10
2          NaT
3   2005-11-23
4   2010-12-31
dtype: datetime64[ns]
```

```
In [6]: type(pd.NaT)

Out[6]: pandas._libs.tslibs.nattype.NaTType
```

如果不希望转换时间戳后生成DatetimeIndex，也可以先把列表转化为Series格式：

1.3 把unix时间戳转换为时间戳

unix时间戳是从1970年1月1日（UTC/GMT的午夜）开始所经过的秒数，不考虑闰秒。

```
In [10]: pd.to_datetime([1349720105, 1349806505, 1349892905], unit="s")

Out[10]: DatetimeIndex(['2012-10-08 18:15:05', '2012-10-09 18:15:05',
                        '2012-10-10 18:15:05'],
                       dtype='datetime64[ns]', freq=None)
```

```
In [11]: pd.to_datetime([1, 2, 20], unit="s")

Out[11]: DatetimeIndex(['1970-01-01 00:00:01', '1970-01-01 00:00:02',
                        '1970-01-01 00:00:20'],
                       dtype='datetime64[ns]', freq=None)
```

```
In [12]: pd.to_datetime([1, 2, 3], unit="D")

Out[12]: DatetimeIndex(['1970-01-02', '1970-01-03', '1970-01-04'], dtype='datetime64[ns]', freq=None)
```

2 生成时间戳范围

有时候，我们可能想要生成某个范围内的时间戳。例如，我想要生成 "2018-6-26" 这一天之后的8天时间戳，如何完成呢？我们可以使用 date_range 和bdate_range 来完成时间戳范围的生成。

2.1 pd.date_range()

pd.date_range（['start = None'，'end = None'，'periods = None'，'freq = None'，'tz = None'，'normalize = False'，'name = None'，'closed = None'，' ** kwargs']，）

- 作用：返回固定频率DatetimeIndex。
  - start：str或datetime-like，可选
    - 生成日期的左边界。
  - end：str或datetime-like，可选
    - 生成日期的权利。
  - periods：整数，可选
    - 要生成的周期数。
  - freq：str或DateOffset，默认为'D'（每日日历）
    - 频率字符串可以具有倍数，例如'5H'。
  - tz：str或tzinfo，可选
    - 例如，返回本地化DatetimeIndex的时区名称。“亚洲/香港”。
  - normalize：bool，默认为False
    - 在生成日期范围之前将开始/结束日期标准化为午夜。
  - name：str，默认无
    - 生成的DatetimeIndex的名称。

- closed：{None, 'left', 'right'}，可选
  - 使间隔相对于给定频率关闭。“左”，“右”或双方（无，默认）。

```
In [13]: pd.date_range('2018-6-26',  periods=8, freq='D')
```

```
Out[13]: DatetimeIndex(['2018-06-26', '2018-06-27', '2018-06-28', '2018-06-29',
                        '2018-06-30', '2018-07-01', '2018-07-02', '2018-07-03'],
                       dtype='datetime64[ns]', freq='D')
```

```
In [2]: pd.date_range("2018-6-26", periods=8,freq='W')
```

```
Out[2]: DatetimeIndex(['2018-07-01', '2018-07-08', '2018-07-15', '2018-07-22',
                       '2018-07-29', '2018-08-05', '2018-08-12', '2018-08-19'],
                      dtype='datetime64[ns]', freq='W-SUN')
```

2.2 pd.bdate_range()

返回固定频率DatetimeIndex，默认为工作日频率。

```
In [27]: pd.bdate_range("2019-6-26", '2019-9-01')
```

```
Out[27]: DatetimeIndex(['2019-06-26', '2019-06-27', '2019-06-28', '2019-07-01',
                        '2019-07-02', '2019-07-03', '2019-07-04', '2019-07-05',
                        '2019-07-08', '2019-07-09', '2019-07-10', '2019-07-11',
                        '2019-07-12', '2019-07-15', '2019-07-16', '2019-07-17',
                        '2019-07-18', '2019-07-19', '2019-07-22', '2019-07-23',
                        '2019-07-24', '2019-07-25', '2019-07-26', '2019-07-29',
                        '2019-07-30', '2019-07-31', '2019-08-01', '2019-08-02',
                        '2019-08-05', '2019-08-06', '2019-08-07', '2019-08-08',
                        '2019-08-09', '2019-08-12', '2019-08-13', '2019-08-14',
                        '2019-08-15', '2019-08-16', '2019-08-19', '2019-08-20',
                        '2019-08-21', '2019-08-22', '2019-08-23', '2019-08-26',
                        '2019-08-27', '2019-08-28', '2019-08-29', '2019-08-30'],
                       dtype='datetime64[ns]', freq='B')
```

注意：pd.bdate_range()不设置freq='D'会默认设置freq='B'，日期间隔会跳过周六日。
但是日期频率如果不是按天算的话，不会跳过工作日：

```
In [29]: pd.bdate_range("2019-6-28", '2019-7-1',freq="4h")  #四个小时为时间间隔
```

```
Out[29]: DatetimeIndex(['2019-06-28 00:00:00', '2019-06-28 04:00:00',
                        '2019-06-28 08:00:00', '2019-06-28 12:00:00',
                        '2019-06-28 16:00:00', '2019-06-28 20:00:00',
                        '2019-06-29 00:00:00', '2019-06-29 04:00:00',
                        '2019-06-29 08:00:00', '2019-06-29 12:00:00',
                        '2019-06-29 16:00:00', '2019-06-29 20:00:00',
                        '2019-06-30 00:00:00', '2019-06-30 04:00:00',
                        '2019-06-30 08:00:00', '2019-06-30 12:00:00',
                        '2019-06-30 16:00:00', '2019-06-30 20:00:00',
                        '2019-07-01 00:00:00'],
                       dtype='datetime64[ns]', freq='4H')
```

In [30]:
```
pd.bdate_range("2019-6-28", periods=9,freq="3h")  #三个小时为时间间隔
```

Out[30]:
```
DatetimeIndex(['2019-06-28 00:00:00', '2019-06-28 03:00:00',
               '2019-06-28 06:00:00', '2019-06-28 09:00:00',
               '2019-06-28 12:00:00', '2019-06-28 15:00:00',
               '2019-06-28 18:00:00', '2019-06-28 21:00:00',
               '2019-06-29 00:00:00'],
              dtype='datetime64[ns]', freq='3H')
```

可以看出，date_range 默认使用的频率是日历日，而 bdate_range 默认使用的频率是营业日。当然了，我们可以自己指定频率，比如，我们可以按周来生成时间戳范围。

In [31]:
```
pd.date_range("2018-6-26", periods=8, freq="W")
```

Out[31]:
```
DatetimeIndex(['2018-07-01', '2018-07-08', '2018-07-15', '2018-07-22',
               '2018-07-29', '2018-08-05', '2018-08-12', '2018-08-19'],
              dtype='datetime64[ns]', freq='W-SUN')
```

In [32]:
```
pd.date_range("2018-6-26", periods=8, freq="Y")
```

Out[32]:
```
DatetimeIndex(['2018-12-31', '2019-12-31', '2020-12-31', '2021-12-31',
               '2022-12-31', '2023-12-31', '2024-12-31', '2025-12-31'],
              dtype='datetime64[ns]', freq='A-DEC')
```

3 DatetimeIndex

DatetimeIndex 的主要作用是之一是用作 Pandas 对象的索引，使用它作为索引除了拥有普通索引对象的所有基本功能外，还拥有简化频率处理的高级时间序列方法。

比如，我们想要给一共有100条数据的DataFrame设置index，index以"2018-6-24"为开始端，频数100：

In [25]:
```
np.random.seed(100)

time_index = pd.date_range("2018-6-24", periods=100, freq="D")   #生成时间索引
profit=np.random.randint(800,5000,100)                           #生成利润

profit_table=pd.DataFrame(profit,index=time_index,columns=["利润"])
profit_table.head(10)
```

Out[25]:

	利润
2018-06-24	4727
2018-06-25	879
2018-06-26	1150
2018-06-27	2690
2018-06-28	4949
2018-06-29	1602
2018-06-30	3600
2018-07-01	2479
2018-07-02	4100
2018-07-03	2699

In [31]:
```
type(time_index)
```

Out[31]:
```
pandas.core.indexes.datetimes.DatetimeIndex
```

如果要查看上面表中2018年6月的数据:

In [105]:
```
profit_table["2018-6"]
```

Out[105]:

	利润
2018-06-24	4727
2018-06-25	879
2018-06-26	1150
2018-06-27	2690
2018-06-28	4949
2018-06-29	1602
2018-06-30	3600

In [32]:
```
profit_table["2018-6-24":"2018-7-1"]
```

Out[32]:

	利润
2018-06-24	4727
2018-06-25	879
2018-06-26	1150
2018-06-27	2690
2018-06-28	4949
2018-06-29	1602
2018-06-30	3600
2018-07-01	2479

In [109]:
```
profit_table["2018-6-30":"2018-8-11":7]
```

Out[109]:

	利润
2018-06-30	3600
2018-07-07	3234
2018-07-14	3683
2018-07-21	802
2018-07-28	4798
2018-08-04	3040
2018-08-11	3541

In [55]:
```
profit_table['星期几'] = profit_table.index.dayofweek+1
profit_table['当年第几天'] = profit_table.index.dayofyear
profit_table['当年第几周'] = profit_table.index.weekofyear
profit_table.head(10)
```

Out[55]:

	利润	星期几	当年第几天	当年第几周
2018-06-24	4727	7	175	25
2018-06-25	879	1	176	26
2018-06-26	1150	2	177	26
2018-06-27	2690	3	178	26
2018-06-28	4949	4	179	26
2018-06-29	1602	5	180	26
2018-06-30	3600	6	181	26
2018-07-01	2479	7	182	26
2018-07-02	4100	1	183	27
2018-07-03	2699	2	184	27

我们可以通过 Timestamp 或 DateTimeIndex 访问一些时间/日期的属性。

这里列举一些常见的，想要查看所有的属性见官方链接：

Time/Date Components (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#time-date-components (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#time-date-components))

In [33]:
```
profit_table.index.is_month_start  #判断是否在月初
profit_table.index.is_month_end    #判断是否在月末
```

Out[33]:
```
array([False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False,  True, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
        True])
```

Out[33]:
```
array([False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False,  True,
       False])
```

4 DateOffset对象

DateOffset 从名称中就可以看出来是要做日期偏移的，它的参数与 dateutil.relativedelta 基本相同，工作方式如下

In [42]:
```
from pandas.tseries.offsets import *
d = pd.Timestamp("2018-06-25")
d
```

Out[42]:
```
Timestamp('2018-06-25 00:00:00')
```

In [43]:
```
d + pd.DateOffset(months=3, days=1)  #当然可以再里面再添加years、hours等参数

d + pd.DateOffset(weeks=2, days=2)
```

Out[43]:
```
Timestamp('2018-09-26 00:00:00')
```

Out[43]:
```
Timestamp('2018-07-11 00:00:00')
```

除了可以使用 DateOffset 完成上面的功能外，还可以使用偏移量实例来完成。

In [44]:
```
d + pd.tseries.offsets.Week(2) + pd.tseries.offsets.Day(2)
```

Out[44]:
```
Timestamp('2018-07-11 00:00:00')
```

In [ ]:
```

```

5 练习

In [46]:
```
import datetime
now=datetime.datetime.now().date()
now
```

Out[46]:
```
datetime.date(2019, 5, 28)
```

生成一个时间序列，从今天开始，往后一年时间，时间间隔为7天，然后把所有时间统一偏移到一年零一天之后，

In [47]:
```

```

Out[47]:
```
DatetimeIndex(['2020-05-29', '2020-06-05', '2020-06-12', '2020-06-19',
               '2020-06-26', '2020-07-03', '2020-07-10', '2020-07-17',
               '2020-07-24', '2020-07-31',
               ...
               '2027-03-18', '2027-03-25', '2027-04-01', '2027-04-08',
               '2027-04-15', '2027-04-22', '2027-04-29', '2027-05-06',
               '2027-05-13', '2027-05-20'],
              dtype='datetime64[ns]', length=365, freq=None)
```

6 与时间序列相关的方法

在做时间序列相关的工作时，经常要对时间做一些移动/滞后、频率转换、采样等相关操作，我们来看下这些操作如何使用吧。

6.1 移动df.shift()

如果你想移动或滞后时间序列，你可以使用 shift 方法。

In [48]:
```
index = pd.date_range("2018-6-24", periods=4, freq="W")
ts = pd.Series(['吃饭', '睡觉', '上厕所', '打豆豆'], index=index)
ts
```

Out[48]:
```
2018-06-24     吃饭
2018-07-01     睡觉
2018-07-08    上厕所
2018-07-15    打豆豆
Freq: W-SUN, dtype: object
```

In [50]:
```
ts.shift(2)        #将数据往后移两位
```

Out[50]:
```
2018-06-24    NaN
2018-07-01    NaN
2018-07-08     吃饭
2018-07-15     睡觉
Freq: W-SUN, dtype: object
```

In [51]:
```
ts.shift(-2)      #将数据往前移两位
```

Out[51]:
```
2018-06-24    上厕所
2018-07-01    打豆豆
2018-07-08     NaN
2018-07-15     NaN
Freq: W-SUN, dtype: object
```

可以看到，Series 所有的值都移动了 2 个距离。

如果不想移动值，而是移动日期索引，可以使用 freq 参数，它可以接受一个 DateOffset 类或其他timedelta 类对象或一个 offset 别名。

所有别名详细介绍见：Offset Aliases (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases))

In [48]:
```
ts.shift(2, freq='D')
```

Out[48]:
```
2018-06-26     吃饭
2018-07-03     上班
2018-07-10    上厕所
2018-07-17     出差
Freq: W-TUE, dtype: object
```

In [52]:
```
ts.shift(2, freq='D')
```

Out[52]:
```
2018-06-26     吃饭
2018-07-03     睡觉
2018-07-10    上厕所
2018-07-17    打豆豆
Freq: W-TUE, dtype: object
```

6.2 频率转换df.asfreq()

频率转换可以使用 asfreq 函数来实现。下面演示了将频率由周转为了天。

In [56]:
```
profit_table.head(10)
```

Out[56]:

	利润	星期几	当年第几天	当年第几周
2018-06-24	4727	7	175	25
2018-06-25	879	1	176	26
2018-06-26	1150	2	177	26
2018-06-27	2690	3	178	26
2018-06-28	4949	4	179	26
2018-06-29	1602	5	180	26
2018-06-30	3600	6	181	26
2018-07-01	2479	7	182	26
2018-07-02	4100	1	183	27
2018-07-03	2699	2	184	27

In [58]:
```
#上面的时间频率为一个月，现在改为10天
profit_table.asfreq(freq='10D').head(10)  #并不会改变原表
```

Out[58]:

	利润	星期几	当年第几天	当年第几周
2018-06-24	4727	7	175	25
2018-07-04	2140	3	185	27
2018-07-14	3683	6	195	28
2018-07-24	2227	2	205	30
2018-08-03	1074	5	215	31
2018-08-13	3210	1	225	33
2018-08-23	2597	4	235	34
2018-09-02	2276	7	245	35
2018-09-12	3134	3	255	37
2018-09-22	4095	6	265	38

如果时间频率过小，有“空隙”，会自动填充缺失值：

In [61]:
```
profit_table.asfreq(freq='0.5D').head(10)
```

Out[61]:

	利润	星期几	当年第几天	当年第几周
2018-06-24 00:00:00	4727.0	7.0	175.0	25.0
2018-06-24 12:00:00	NaN	NaN	NaN	NaN
2018-06-25 00:00:00	879.0	1.0	176.0	26.0
2018-06-25 12:00:00	NaN	NaN	NaN	NaN
2018-06-26 00:00:00	1150.0	2.0	177.0	26.0
2018-06-26 12:00:00	NaN	NaN	NaN	NaN
2018-06-27 00:00:00	2690.0	3.0	178.0	26.0
2018-06-27 12:00:00	NaN	NaN	NaN	NaN
2018-06-28 00:00:00	4949.0	4.0	179.0	26.0
2018-06-28 12:00:00	NaN	NaN	NaN	NaN

In [65]:
```
profit_table.asfreq(freq='0.5D',fill_value="没记录").head(10)  #指定填充
```

Out[65]:

	利润	星期几	当年第几天	当年第几周
2018-06-24 00:00:00	4727	7	175	25
2018-06-24 12:00:00	没记录	没记录	没记录	没记录
2018-06-25 00:00:00	879	1	176	26
2018-06-25 12:00:00	没记录	没记录	没记录	没记录
2018-06-26 00:00:00	1150	2	177	26
2018-06-26 12:00:00	没记录	没记录	没记录	没记录
2018-06-27 00:00:00	2690	3	178	26
2018-06-27 12:00:00	没记录	没记录	没记录	没记录
2018-06-28 00:00:00	4949	4	179	26
2018-06-28 12:00:00	没记录	没记录	没记录	没记录

In [66]:
```
profit_table.asfreq(freq='0.5D',method="bfill").head(10)  #并缺失值用后面一个值来填充
```

Out[66]:

	利润	星期几	当年第几天	当年第几周
2018-06-24 00:00:00	4727	7	175	25
2018-06-24 12:00:00	879	1	176	26
2018-06-25 00:00:00	879	1	176	26
2018-06-25 12:00:00	1150	2	177	26
2018-06-26 00:00:00	1150	2	177	26
2018-06-26 12:00:00	2690	3	178	26
2018-06-27 00:00:00	2690	3	178	26
2018-06-27 12:00:00	4949	4	179	26
2018-06-28 00:00:00	4949	4	179	26
2018-06-28 12:00:00	1602	5	180	26

7 重采样

resample 表示根据日期维度进行数据聚合，可以按照分钟、小时、工作日、周、月、年等来作为日期维度，更多的日期维度见 Offset Aliases(http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases))。

这里我们先以月来作为时间维度来进行聚合。

In [73]:
```
profit_table.rename(columns={"利润":'营业额'}, inplace=True)
profit_table.head(5)
```

Out[73]:

	营业额	星期几	当年第几天	当年第几周
2018-06-24	4727	7	175	25
2018-06-25	879	1	176	26
2018-06-26	1150	2	177	26
2018-06-27	2690	3	178	26
2018-06-28	4949	4	179	26

In [74]:
```
months = profit_table.resample('M')    #按月进行数据聚合
months.sum()
```

Out[74]:

	营业额	星期几	当年第几天	当年第几周
2018-06-30	19597	28	1246	181
2018-07-31	106361	122	6107	886
2018-08-31	84025	124	7068	1023
2018-09-30	84137	125	7755	1120
2018-10-31	1568	1	274	40

In [75]:
```
months.max()
months.mean()
```

Out[75]:

	营业额	星期几	当年第几天	当年第几周
2018-06-30	4949	7	181	26
2018-07-31	4827	7	212	31
2018-08-31	4909	7	243	35
2018-09-30	4491	7	273	39
2018-10-31	1568	1	274	40

Out[75]:

	营业额	星期几	当年第几天	当年第几周
2018-06-30	2799.571429	4.000000	178.0	25.857143
2018-07-31	3431.000000	3.935484	197.0	28.580645
2018-08-31	2710.483871	4.000000	228.0	33.000000
2018-09-30	2804.566667	4.166667	258.5	37.333333
2018-10-31	1568.000000	1.000000	274.0	40.000000

8 练习

- 求出每月的营业额，并找出哪月是赚钱最多？
- 找出哪个月是赚钱最少？

In [76]:
```
profit_table.head(5)
```

Out[76]:

	营业额	星期几	当年第几天	当年第几周
2018-06-24	4727	7	175	25
2018-06-25	879	1	176	26
2018-06-26	1150	2	177	26
2018-06-27	2690	3	178	26
2018-06-28	4949	4	179	26

In [146]:
```

```

Out[146]:
```
2018-06-30     19597
2018-07-31    106361
2018-08-31     84025
2018-09-30     84137
2018-10-31      1568
Freq: M, Name: 利润, dtype: int32
```

Out[146]:
```
106361
```

In [147]:

Out[147]:
```
2018-06-30     19597
2018-07-31    106361
2018-08-31     84025
2018-09-30     84137
2018-10-31      1568
Freq: M, Name: 利润, dtype: int32
```

Out[147]:
```
1568
```