Pandas处理时序数据（初学者必会）！

耿远昊 Datawhale 今天

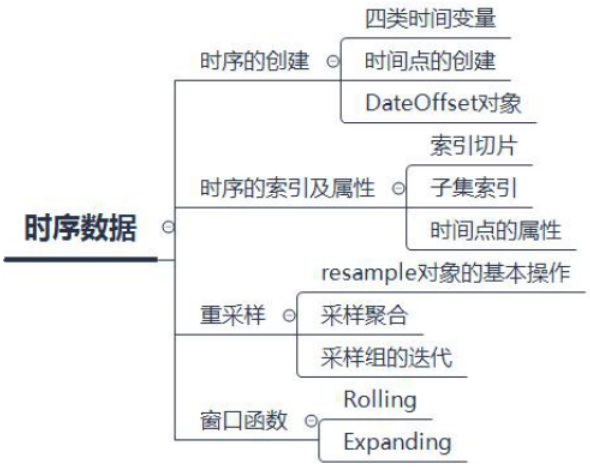
Datawhale干货

作者：耿远昊，Datawhale成员，华东师范大学

时序数据是指时间序列数据。时间序列数据是同一统一指标按时间顺序记录的数据列。在同一数据列中的各个数据必须是同口径的，要求具有可比性。时序数据可以是时期数，也可以时点数。

时间序列分析的目的是通过找出样本内时间序列的统计特性和发展规律性，构建时间序列模型，进行样本外预测。

现在，一起来学习用Pandas处理时序数据。



本文目录

```
import pandas as pd
import numpy as np
```

一、时序的创建

1.1. 四类时间变量

现在理解可能关于③和④有些困惑，后面会作出一些说明

名称	描述	元素类型	创建方式
① Date times（时间点/时刻）	描述特定日期或时间点	Timestamp	to_datetime或date_range
② Time spans（时间段/时期）	由时间点定义的一段时期	Period	Period或period_range
③ Date offsets（相对时间差）	一段时间的相对大小（与夏/冬令时无关）	DateOffset	DateOffset
④ Time deltas（绝对时间差）	一段时间的绝对大小（与夏/冬令时有关）	Timedelta	to_timedelta或timedelta_range

1.2. 时间点的创建

（a）to_datetime方法

Pandas在时间点建立的输入格式规定上给了很大的自由度，下面的语句都能正确建立同一时间点

```
pd.to_datetime('2020.1.1')
pd.to_datetime('2020 1.1')
pd.to_datetime('2020 1 1')
pd.to_datetime('2020 1-1')
pd.to_datetime('2020-1 1')
pd.to_datetime('2020-1-1')
pd.to_datetime('2020/1/1')
pd.to_datetime('1.1.2020')
pd.to_datetime('1.1 2020')
pd.to_datetime('1 1 2020')
pd.to_datetime('1 1-2020')
pd.to_datetime('1-1 2020')
pd.to_datetime('1-1-2020')
pd.to_datetime('1/1/2020')
```

```
pd.to_datetime('20200101')
pd.to_datetime('2020.0101')
```

Timestamp('2020-01-01 00:00:00')

下面的语句都会报错

```
#pd.to_datetime('2020\\1\\1')
#pd.to_datetime('2020`1`1')
#pd.to_datetime('2020.1 1')
#pd.to_datetime('1 1.2020')
```

此时可利用format参数强制匹配

```
pd.to_datetime('2020\\1\\1',format='%Y\\%m\\%d')
pd.to_datetime('2020`1`1',format='%Y`%m`%d')
pd.to_datetime('2020.1 1',format='%Y.%m %d')
pd.to_datetime('1 1.2020',format='%d %m.%Y')
```

Timestamp('2020-01-01 00:00:00')

同时，使用列表可以将其转为时间点索引

```
pd.Series(range(2),index=pd.to_datetime(['2020/1/1','2020/1/2']))
```

```
0   2020-01-01
1   2020-01-02
dtype: datetime64[ns]
```

```
type(pd.to_datetime(['2020/1/1','2020/1/2']))
```

pandas.core.indexes.datetimes.DatetimeIndex

对于DataFrame而言，如果列已经按照时间顺序排好，则利用to_datetime可自动转换

```
df = pd.DataFrame({'year': [2020, 2020],'month': [1, 1], 'day': [1, 2]})
pd.to_datetime(df)
```

```
0   2020-01-01
1   2020-01-02
dtype: datetime64[ns]
```

（b）时间精度与范围限制

事实上，Timestamp的精度远远不止day，可以最小到纳秒ns

```
pd.to_datetime('2020/1/1 00:00:00.123456789')
```

Timestamp('2020-01-01 00:00:00.123456789')

同时，它带来范围的代价就是只有大约584年的时间点是可用的

```
pd.Timestamp.min
```

Timestamp('1677-09-21 00:12:43.145225')

```
pd.Timestamp.max
```

```
Timestamp('2262-04-11 23:47:16.854775807')
```

（c）date_range方法

一般来说，start/end/periods（时间点个数）/freq（间隔方法）是该方法最重要的参数，给定了其中的3个，剩下的一个就会被确定

```
pd.date_range(start='2020/1/1',end='2020/1/10',periods=3)
```

```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-05 12:00:00',
               '2020-01-10 00:00:00'],
              dtype='datetime64[ns]', freq=None)
```

```
pd.date_range(start='2020/1/1',end='2020/1/10',freq='D')
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
               '2020-01-09', '2020-01-10'],
              dtype='datetime64[ns]', freq='D')
```

```
pd.date_range(start='2020/1/1',periods=3,freq='D')
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03'], dtype='datetime64[ns]', freq='D')
```

```
pd.date_range(end='2020/1/3',periods=3,freq='D')
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03'], dtype='datetime64[ns]', freq='D')
```

其中freq参数有许多选项，下面将常用部分罗列如下，更多选项可看这里

```
pd.date_range(start='2020/1/1',periods=3,freq='T')
```

```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-01 00:01:00',
               '2020-01-01 00:02:00'],
              dtype='datetime64[ns]', freq='T')
```

```
pd.date_range(start='2020/1/1',periods=3,freq='M')
```

```
DatetimeIndex(['2020-01-31', '2020-02-29', '2020-03-31'], dtype='datetime64[ns]', freq='M')
```

```
pd.date_range(start='2020/1/1',periods=3,freq='BYS')
```

```
DatetimeIndex(['2020-01-01', '2021-01-01', '2022-01-03'], dtype='datetime64[ns]', freq='BAS-JAN')
```

bdate_range是一个类似与date_range的方法，特点在于可以在自带的工作日间隔设置上，再选择weekmask参数和holidays参数

它的freq中有一个特殊的'C'/'CBM'/'CBMS'选项，表示定制，需要联合weekmask参数和holidays参数使用

例如现在需要将工作日中的周一、周二、周五3天保留，并将部分holidays剔除

```
weekmask = 'Mon Tue Fri'
holidays = [pd.Timestamp('2020/1/%s'%i) for i in range(7,13)]
#注意holidays
pd.bdate_range(start='2020-1-1',end='2020-1-15',freq='C',weekmask=weekmask,holi
```

```
DatetimeIndex(['2020-01-03', '2020-01-06', '2020-01-13', '2020-01-14'], dtype='datetime64[ns]', freq='C')
```

1.3. DateOffset对象

（a）DataOffset与Timedelta的区别

Timedelta绝对时间差的特点指无论是冬令时还是夏令时，增减1day都只计算24小时

DataOffset相对时间差指，无论一天是23\24\25小时，增减1day都与当天相同的时间保持一致

例如，英国当地时间 2020年03月29日，01:00:00 时钟向前调整 1 小时 变为 2020年03月29日，02:00:00，开始夏令时

```
ts = pd.Timestamp('2020-3-29 01:00:00', tz='Europe/Helsinki')
ts + pd.Timedelta(days=1)
```

```
Timestamp('2020-03-30 02:00:00+0300', tz='Europe/Helsinki')
```

```
ts + pd.DateOffset(days=1)
```

Timestamp('2020-03-30 01:00:00+0300', tz='Europe/Helsinki')

这似乎有些令人头大，但只要把tz（time zone）去除就可以不用管它了，两者保持一致，除非要使用到时区变换

```
ts = pd.Timestamp('2020-3-29 01:00:00')
ts + pd.Timedelta(days=1)
```

Timestamp('2020-03-30 01:00:00')

```
ts + pd.DateOffset(days=1)
```

Timestamp('2020-03-30 01:00:00')

（b）增减一段时间

DateOffset的可选参数包括years/months/weeks/days/hours/minutes/seconds

```
pd.Timestamp('2020-01-01') + pd.DateOffset(minutes=20) - pd.DateOffset(weeks=2)
```

Timestamp('2019-12-18 00:20:00')

（c）各类常用offset对象

```
pd.Timestamp('2020-01-01') + pd.offsets.Week(2)
```

Timestamp('2020-01-15 00:00:00')

```
pd.Timestamp('2020-01-01') + pd.offsets.BQuarterBegin(1)
```

Timestamp('2020-03-02 00:00:00')

（d）序列的offset操作

利用apply函数

```
pd.Series(pd.offsets.BYearBegin(3).apply(i) for i in pd.date_range('20200101',p
```

```
0   2023-01-02
1   2024-01-01
2   2025-01-01
dtype: datetime64[ns]
```

直接使用对象加减

```
pd.date_range('20200101',periods=3,freq='Y') + pd.offsets.BYearBegin(3)
```

```
DatetimeIndex(['2023-01-02', '2024-01-01', '2025-01-01'], dtype='datetime64[ns]', freq='A-DEC')
```

定制offset，可以指定weekmask和holidays参数（思考为什么三个都是一个值）

```
pd.Series(pd.offsets.CDay(3,weekmask='Wed Fri',holidays='2020010').apply(i)
                                    for i in pd.date_range('20200105',periods=3,f
```

```
0   2020-01-15
1   2020-01-15
2   2020-01-15
dtype: datetime64[ns]
```

二、时序的索引及属性

2.1. 索引切片

这一部分几乎与第二章的规则完全一致

```
rng = pd.date_range('2020','2021', freq='W')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts.head()
```

```
2020-01-05   -0.275349
2020-01-12    2.359218
2020-01-19   -0.447633
2020-01-26   -0.479830
2020-02-02    0.517587
Freq: W-SUN, dtype: float64
```

```
ts['2020-01-26']
```

-0.47982974619679947

合法字符自动转换为时间点

```
ts['2020-01-26':'20200726'].head()
```

```
2020-01-26   -0.479830
2020-02-02    0.517587
2020-02-09   -0.575879
2020-02-16    0.952187
2020-02-23    0.554098
Freq: W-SUN, dtype: float64
```

2.2. 子集索引

```
ts['2020-7'].head()
```

```
2020-07-05   -0.088912
2020-07-12    0.153852
2020-07-19    1.670324
2020-07-26    0.568214
Freq: W-SUN, dtype: float64
```

支持混合形态索引

```
ts['2011-1':'20200726'].head()
```

```
2020-01-05   -0.275349
2020-01-12    2.359218
2020-01-19   -0.447633
2020-01-26   -0.479830
2020-02-02    0.517587
Freq: W-SUN, dtype: float64
```

2.3. 时间点的属性

采用dt对象可以轻松获得关于时间的信息

```
pd.Series(ts.index).dt.week.head()
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

```
pd.Series(ts.index).dt.day.head()
```

```
0     5
1    12
2    19
3    26
4     2
dtype: int64
```

利用strftime可重新修改时间格式

```
pd.Series(ts.index).dt.strftime('%Y-间隔1-%m-间隔2-%d').head()
```

```
0    2020-间隔1-01-间隔2-05
1    2020-间隔1-01-间隔2-12
2    2020-间隔1-01-间隔2-19
3    2020-间隔1-01-间隔2-26
4    2020-间隔1-02-间隔2-02
dtype: object
```

对于datetime对象可以直接通过属性获取信息

```
pd.date_range('2020','2021', freq='W').month
```

```
Int64Index([ 1,  1,  1,  1,  2,  2,  2,  2,  3,  3,  3,  3,  3,  4,  4,  4,  4,
             5,  5,  5,  5,  5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  8,
             8,  9,  9,  9,  9, 10, 10, 10, 10, 11, 11, 11, 11, 11, 12, 12, 12,
            12],
           dtype='int64')
```

```
pd.date_range('2020','2021', freq='W').weekday
```

```
Int64Index([6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
            6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
            6, 6, 6, 6, 6, 6, 6, 6],
           dtype='int64')
```

三、重采样

所谓重采样，就是指resample函数，它可以看做时序版本的groupby函数

3.1. resample对象的基本操作

采样频率一般设置为上面提到的offset字符

```
df_r = pd.DataFrame(np.random.randn(1000, 3),index=pd.date_range('1/1/2020', fr
                    columns=['A', 'B', 'C'])
```

```
r = df_r.resample('3min')
r
```

```
<pandas.core.resample.DatetimeIndexResampler object at 0x7ff73ebafc10>
```

```
r.sum()
```

	A	B	C
2020-01-01 00:00:00	-8.772685	-27.074716	2.134617
2020-01-01 00:03:00	3.822484	8.912459	-15.448955
2020-01-01 00:06:00	2.744722	-8.055139	-11.364361
2020-01-01 00:09:00	4.655620	-11.524496	-10.536002
2020-01-01 00:12:00	-10.546811	5.063887	11.776490
2020-01-01 00:15:00	8.795150	-12.828809	-8.393950

```
df_r2 = pd.DataFrame(np.random.randn(200, 3),index=pd.date_range('1/1/2020', fr
                    columns=['A', 'B', 'C'])
r = df_r2.resample('CBMS')
r.sum()
```

	A	B	C
2020-01-01	5.278470	1.688588	5.904806
2020-02-03	-3.581797	7.515267	0.205308
2020-03-02	-5.021605	-4.441066	5.433917
2020-04-01	0.671702	3.840042	4.922487
2020-05-01	4.613352	9.702408	-4.928112
2020-06-01	-0.598191	7.387416	8.716921
2020-07-01	-0.327200	-1.577507	-3.956079

3.2. 采样聚合

```
r = df_r.resample('3T')
```

```
r['A'].mean()
```

```
2020-01-01 00:00:00   -0.048737
2020-01-01 00:03:00    0.021236
2020-01-01 00:06:00    0.015248
2020-01-01 00:09:00    0.025865
2020-01-01 00:12:00   -0.058593
2020-01-01 00:15:00    0.087952
Freq: 3T, Name: A, dtype: float64
```

```
r['A'].agg([np.sum, np.mean, np.std])
```

	sum	mean	std
2020-01-01 00:00:00	-8.772685	-0.048737	0.939954
2020-01-01 00:03:00	3.822484	0.021236	1.004048
2020-01-01 00:06:00	2.744722	0.015248	1.018865
2020-01-01 00:09:00	4.655620	0.025865	1.020881
2020-01-01 00:12:00	-10.546811	-0.058593	0.954328
2020-01-01 00:15:00	8.795150	0.087952	1.199379

类似地，可以使用函数lambda表达式

```
r.agg({'A': np.sum,'B': lambda x: max(x)-min(x)})
```

	A	B
2020-01-01 00:00:00	-8.772685	4.950006
2020-01-01 00:03:00	3.822484	5.711679
2020-01-01 00:06:00	2.744722	6.923072
2020-01-01 00:09:00	4.655620	6.370589
2020-01-01 00:12:00	-10.546811	4.544878
2020-01-01 00:15:00	8.795150	5.244546

3.3. 采样组的迭代

采样组的迭代和groupby迭代完全类似，对于每一个组都可以分别做相应操作

```
small = pd.Series(range(6),index=pd.to_datetime(['2020-01-01 00:00:00', '2020-0
                                                 , '2020-01-01 00:31:00','2020-
                                                 ,'2020-01-01 03:00:00','2020-0
resampled = small.resample('H')
for name, group in resampled:
    print("Group: ", name)
    print("-" * 27)
    print(group, end="\n\n")
```

```
Group:  2020-01-01 00:00:00
---------------------------
2020-01-01 00:00:00    0
2020-01-01 00:30:00    1
2020-01-01 00:31:00    2
dtype: int64

Group:  2020-01-01 01:00:00
---------------------------
2020-01-01 01:00:00    3
dtype: int64

Group:  2020-01-01 02:00:00
---------------------------
Series([], dtype: int64)

Group:  2020-01-01 03:00:00
---------------------------
2020-01-01 03:00:00    4
2020-01-01 03:05:00    5
dtype: int64
```

四、窗口函数

下面主要介绍pandas中两类主要的窗口(window)函数:rolling/expanding

```
s = pd.Series(np.random.randn(1000),index=pd.date_range('1/1/2020', periods=100
s.head()
```

```
2020-01-01    0.305974
2020-01-02    0.185221
2020-01-03   -0.646472
2020-01-04   -1.430293
2020-01-05   -0.956094
Freq: D, dtype: float64
```

4.1. Rolling

（a）常用聚合

所谓rolling方法，就是规定一个窗口，它和groupby对象一样，本身不会进行操作，需要配合聚合函数才能计算结果

```
s.rolling(window=50)
```

Rolling [window=50,center=False,axis=0]

```
s.rolling(window=50).mean()
```

```
2020-01-01         NaN
2020-01-02         NaN
2020-01-03         NaN
2020-01-04         NaN
2020-01-05         NaN
                ...
2022-09-22    0.160743
2022-09-23    0.136296
2022-09-24    0.147523
2022-09-25    0.133087
2022-09-26    0.130841
Freq: D, Length: 1000, dtype: float64
```

min_periods参数是指需要的非缺失数据点数量阀值

```
s.rolling(window=50,min_periods=3).mean().head()
```

```
2020-01-01         NaN
2020-01-02         NaN
2020-01-03   -0.051759
2020-01-04   -0.396392
2020-01-05   -0.508333
Freq: D, dtype: float64
```

count/sum/mean/median/min/max/std/var/skew/kurt/quantile/cov/corr都是常用的聚合函数。

（b）rolling的apply聚合

使用apply聚合时，只需记住传入的是window大小的Series，输出的必须是标量即可，比如如下计算变异系数

```
s.rolling(window=50,min_periods=3).apply(lambda x:x.std()/x.mean()).head()
```

```
2020-01-01          NaN
2020-01-02          NaN
2020-01-03   -10.018809
2020-01-04    -2.040720
2020-01-05    -1.463460
Freq: D, dtype: float64
```

（c）基于时间的rolling

```
s.rolling('15D').mean().head()
```

```
2020-01-01    0.305974
2020-01-02    0.245598
2020-01-03   -0.051759
2020-01-04   -0.396392
2020-01-05   -0.508333
Freq: D, dtype: float64
```

可选closed='right'（默认）\'left'\'both'\'neither'参数，决定端点的包含情况

```
s.rolling('15D', closed='right').sum().head()
```

```
2020-01-01    0.305974
2020-01-02    0.491195
2020-01-03   -0.155277
2020-01-04   -1.585570
2020-01-05   -2.541664
Freq: D, dtype: float64
```

4.2. Expanding

（a）expanding函数

普通的expanding函数等价与rolling(window=len(s),min_periods=1)，是对序列的累计计算

```
s.rolling(window=len(s),min_periods=1).sum().head()
```

```
2020-01-01    0.305974
2020-01-02    0.491195
2020-01-03   -0.155277
2020-01-04   -1.585570
2020-01-05   -2.541664
Freq: D, dtype: float64
```

```
s.expanding().sum().head()
```

```
2020-01-01    0.305974
2020-01-02    0.491195
2020-01-03   -0.155277
2020-01-04   -1.585570
2020-01-05   -2.541664
Freq: D, dtype: float64
```

apply方法也是同样可用的

```
s.expanding().apply(lambda x:sum(x)).head()
```

```
2020-01-01    0.305974
2020-01-02    0.491195
2020-01-03   -0.155277
2020-01-04   -1.585570
2020-01-05   -2.541664
Freq: D, dtype: float64
```

（b）几个特别的Expanding类型函数

cumsum/cumprod/cummax/cummin都是特殊expanding累计计算方法

```
s.cumsum().head()
```

```
2020-01-01    0.305974
2020-01-02    0.491195
2020-01-03   -0.155277
2020-01-04   -1.585570
2020-01-05   -2.541664
Freq: D, dtype: float64
```

```
1 s.cumsum().head()
```

```
2020-01-01    0.305974
2020-01-02    0.491195
2020-01-03   -0.155277
2020-01-04   -1.585570
2020-01-05   -2.541664
Freq: D, dtype: float64
```

shift/diff/pct_change都是涉及到了元素关系

① shift是指序列索引不变，但值向后移动

② diff是指前后元素的差，period参数表示间隔，默认为1，并且可以为负

③ pct_change是值前后元素的变化百分比，period参数与diff类似

```
1 s.shift(2).head()
```

```
2020-01-01         NaN
2020-01-02         NaN
2020-01-03    0.305974
2020-01-04    0.185221
2020-01-05   -0.646472
Freq: D, dtype: float64
```

```
1 s.diff(3).head()
```

```
2020-01-01         NaN
2020-01-02         NaN
2020-01-03         NaN
2020-01-04   -1.736267
2020-01-05   -1.141316
Freq: D, dtype: float64
```

```
1 s.pct_change(3).head()
```

```
2020-01-01         NaN
2020-01-02         NaN
2020-01-03         NaN
2020-01-04   -5.674559
2020-01-05   -6.161897
Freq: D, dtype: float64
```

五、问题与练习

5.1. 问题

【问题一】 如何对date_range进行批量加帧操作或对某一时间段加大时间戳密度？

```
# 不知道对不对，先这样搞出来=。=
s = pd.date_range(start='2020/1/1',end='2020/1/10',periods=3)
s
```

```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-05 12:00:00',
               '2020-01-10 00:00:00'],
              dtype='datetime64[ns]', freq=None)
```

```
a=[]
for i in s:
    a.extend(pd.date_range(start=i,freq='H',periods=3))
a
```

```
[Timestamp('2020-01-01 00:00:00', freq='H'),
 Timestamp('2020-01-01 01:00:00', freq='H'),
 Timestamp('2020-01-01 02:00:00', freq='H'),
 Timestamp('2020-01-05 12:00:00', freq='H'),
 Timestamp('2020-01-05 13:00:00', freq='H'),
 Timestamp('2020-01-05 14:00:00', freq='H'),
 Timestamp('2020-01-10 00:00:00', freq='H'),
 Timestamp('2020-01-10 01:00:00', freq='H'),
 Timestamp('2020-01-10 02:00:00', freq='H')]
```

```
pd.to_datetime(a)
```

```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-01 01:00:00',
               '2020-01-01 02:00:00', '2020-01-05 12:00:00',
               '2020-01-05 13:00:00', '2020-01-05 14:00:00',
               '2020-01-10 00:00:00', '2020-01-10 01:00:00',
               '2020-01-10 02:00:00'],
              dtype='datetime64[ns]', freq=None)
```


【问题二】 如何批量增加TimeStamp的精度？

【问题三】 对于超出处理时间的时间点，是否真的完全没有处理方法？

```
# If you have data that is outside of the Timestamp bounds, see Timestamp limitations,
# then you can use a PeriodIndex and/or Series of Periods to do computations.
# 根据官方文档来看，是转化为PeriodIndex，例如：
s = pd.Series([20121231, 20141130, 99991231])
s
```

```
0    20121231
1    20141130
2    99991231
dtype: int64
```

```
def conv(x):
    return pd.Period(year=x // 10000, month=x // 100 % 100,day=x % 100, freq='D')
pd.PeriodIndex(s.apply(conv))
```

```
PeriodIndex(['2012-12-31', '2014-11-30', '9999-12-31'], dtype='period[D]', freq='D')
```


【问题四】 给定一组非连续的日期，怎么快速找出位于其最大日期和最小日期之间，且没有出现在该组日期中的日期？

```
s = pd.date_range(start='2020/1/1',freq='W',periods=3)
s
```

```
DatetimeIndex(['2020-01-05', '2020-01-12', '2020-01-19'], dtype='datetime64[ns]', freq='W-SUN')
```

```
sl=pd.date_range(start=s.min(),end=s.max())
sl[~sl.isin(s)]
```

```
DatetimeIndex(['2020-01-06', '2020-01-07', '2020-01-08', '2020-01-09',
               '2020-01-10', '2020-01-11', '2020-01-13', '2020-01-14',
               '2020-01-15', '2020-01-16', '2020-01-17', '2020-01-18'],
              dtype='datetime64[ns]', freq=None)
```


5.2. 练习

【练习一】 现有一份关于某超市牛奶销售额的时间序列数据，请完成下列问题：

```
df1 = pd.read_csv('data/time_series_one.csv')
df1.head()
```

	日期	销售额
0	2017/2/17	2154
1	2017/2/18	2095
2	2017/2/19	3459
3	2017/2/20	2198
4	2017/2/21	2413

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   日期      1000 non-null   object
 1   销售额     1000 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 15.8+ KB
```

```
df1.head()
```

	日期	销售额
0	2017-02-17	2154
1	2017-02-18	2095
2	2017-02-19	3459
3	2017-02-20	2198
4	2017-02-21	2413

```
df1.tail()
```

	日期	销售额
995	2019-11-09	3022
996	2019-11-10	2961
997	2019-11-11	3984
998	2019-11-12	2799
999	2019-11-13	2941

（a）销售额出现最大值的是星期几？（提示：利用dayofweek函数）

```
df1['日期'] = pd.to_datetime(df1['日期'])
df11=df1.set_index("日期")
df11.head()
```

	销售额
日期	
2017-02-17	2154
2017-02-18	2095
2017-02-19	3459
2017-02-20	2198
2017-02-21	2413

```
df11.loc[df11['销售额'].idxmax()].name.dayofweek
```

6

（b）计算除去春节、国庆、五一节假日的月度销售总额

```
# 查日历看节假日。。。
hol = pd.date_range(start='20170429',end='20170501').append(
    pd.date_range(start='20171001',end='20171008')).append(
    pd.date_range(start='20180215',end='20180221')).append(
    pd.date_range(start='20180429',end='20180501')).append(
    pd.date_range(start='20181001',end='20181007')).append(
    pd.date_range(start='20190204',end='20190210')).append(
    pd.date_range(start='20190501',end='20190504')).append(
    pd.date_range(start='20191001',end='20191001'))
hol
```

```
DatetimeIndex(['2017-04-29', '2017-04-30', '2017-05-01', '2017-10-01',
               '2017-10-02', '2017-10-03', '2017-10-04', '2017-10-05',
               '2017-10-06', '2017-10-07', '2017-10-08', '2018-02-15',
               '2018-02-16', '2018-02-17', '2018-02-18', '2018-02-19',
               '2018-02-20', '2018-02-21', '2018-04-29', '2018-04-30',
               '2018-05-01', '2018-10-01', '2018-10-02', '2018-10-03',
               '2018-10-04', '2018-10-05', '2018-10-06', '2018-10-07',
               '2019-02-04', '2019-02-05', '2019-02-06', '2019-02-07',
               '2019-02-08', '2019-02-09', '2019-02-10', '2019-05-01',
               '2019-05-02', '2019-05-03', '2019-05-04', '2019-10-01'],
              dtype='datetime64[ns]', freq=None)
```

```
df11[~df11.index.isin(hol)].resample('MS').sum().head()
```

日期	销售额
2017-02-01	31740
2017-03-01	80000
2017-04-01	70427
2017-05-01	81262
2017-06-01	80750

（c）按季度计算周末（周六和周日）的销量总额

```
weekend=pd.date_range(start='2017/2/17',end='2019/11/13',freq='B')
weekend
```

```
DatetimeIndex(['2017-02-17', '2017-02-20', '2017-02-21', '2017-02-22',
               '2017-02-23', '2017-02-24', '2017-02-27', '2017-02-28',
               '2017-03-01', '2017-03-02',
               ...
               '2019-10-31', '2019-11-01', '2019-11-04', '2019-11-05',
               '2019-11-06', '2019-11-07', '2019-11-08', '2019-11-11',
               '2019-11-12', '2019-11-13'],
              dtype='datetime64[ns]', length=714, freq='B')
```

```
df11[~df11.index.isin(weekend)].resample('QS').sum().head()
```

日期	销售额
2017-01-01	32894
2017-04-01	66692
2017-07-01	69099
2017-10-01	70384
2018-01-01	74671


```
# 原来是这样用的，牛批
df = pd.read_csv('data/time_series_one.csv', parse_dates=['日期'])
result = df[df['日期'].dt.dayofweek.isin([5,6])].set_index('日期').resample('QS').sum()
result.head()
```

日期	销售额
2017-01-01	32894
2017-04-01	66692
2017-07-01	69099
2017-10-01	70384
2018-01-01	74671

（d）从最后一天开始算起，跳过周六和周一，以5天为一个时间单位向前计算销售总和

```
weekmask = 'Tue Wed Thu Fri Sun'
skip = pd.bdate_range(start='2017/2/17',end='2019/11/13',freq='C',weekmask=weekmask)
skip
```

```
DatetimeIndex(['2017-02-17', '2017-02-19', '2017-02-21', '2017-02-22',
               '2017-02-23', '2017-02-24', '2017-02-26', '2017-02-28',
               '2017-03-01', '2017-03-02',
               ...
               '2019-10-31', '2019-11-01', '2019-11-03', '2019-11-05',
               '2019-11-06', '2019-11-07', '2019-11-08', '2019-11-10',
               '2019-11-12', '2019-11-13'],
              dtype='datetime64[ns]', length=714, freq='C')
```

```
df11[-1::-1][df11.index.isin(skip)].rolling(5).sum().head()
```

日期	销售额
2019-11-13	NaN
2019-11-11	NaN
2019-11-09	NaN
2019-11-08	NaN
2019-11-07	16049.0


```
# 让我再看看。。。
df_temp = df[~df['日期'].dt.dayofweek.isin([5,6])].set_index('日期').iloc[::-1]
L_temp,date_temp = [],[0]*df_temp.shape[0]
for i in range(df_temp.shape[0]//5):
    L_temp.extend([i]*5)
L_temp.extend([df_temp.shape[0]//5]*(df_temp.shape[0]-df_temp.shape[0]//5*5))
date_temp = pd.Series([i%5==0 for i in range(df_temp.shape[0])])
df_temp['num'] = L_temp
result = pd.DataFrame({'5天总额':df_temp.groupby('num')['销售额'].sum().values},
                      index=df_temp.reset_index()[date_temp]['日期']).iloc[::-1]
result.head()
```

日期	5天总额
2017-02-22	9855
2017-03-01	12296
2017-03-08	13323
2017-03-15	13845
2017-03-22	11356

（e）假设现在发现数据有误，所有同一周里的周一与周五的销售额记录颠倒了，请计算2018年中每月第一个周一的销售额（如果该周没有周一或周五的记录就保持不动）

```
from datetime import datetime
df_temp = df.copy()
df_fri = df.shift(4)[df.shift(4)['日期'].dt.dayofweek==1]['销售额']
df_mon = df.shift(-4)[df.shift(-4)['日期'].dt.dayofweek==5]['销售额']
df_temp.loc[df_fri.index,['销售额']] = df_fri
df_temp.loc[df_mon.index,['销售额']] = df_mon
df_temp.loc[df_temp[df_temp['日期'].dt.year==2018]['日期'][
        df_temp[df_temp['日期'].dt.year==2018]['日期'].apply(
        lambda x:True if datetime.strptime(str(x).split()[0],'%Y-%m-%d').weekday() == 0
        and 1 <= datetime.strptime(str(x).split()[0],'%Y-%m-%d').day <= 7 else False)].index,:]
```

	日期	销售额
318	2018-01-01	2863.0
353	2018-02-05	2321.0
381	2018-03-05	2705.0
409	2018-04-02	2487.0
444	2018-05-07	3204.0
472	2018-06-04	2927.0
500	2018-07-02	2574.0
535	2018-08-06	2504.0
563	2018-09-03	2483.0
591	2018-10-01	2431.0
626	2018-11-05	2395.0
654	2018-12-03	2373.0

【练习二】 继续使用上一题的数据，请完成下列问题：

（a）以50天为窗口计算滑窗均值和滑窗最大值（min_periods设为1）

```
df11.rolling(50,min_periods=1).mean().head()
```

	销售额
日期	
2017-02-17	2154.000000
2017-02-18	2124.500000
2017-02-19	2569.333333
2017-02-20	2476.500000
2017-02-21	2463.800000

```
df11.rolling(50,min_periods=1).max().head()
```

	销售额
日期	
2017-02-17	2154.0
2017-02-18	2154.0
2017-02-19	3459.0
2017-02-20	3459.0
2017-02-21	3459.0

（b）现在有如下规则：若当天销售额超过向前5天的均值，则记为1，否则记为0，请给出2018年相应的计算结果

```
df = pd.read_csv('data/time_series_one.csv',index_col='日期',parse_dates=['日期'])
df.head()
```

日期	销售额
2017-02-17	2154
2017-02-18	2095
2017-02-19	3459
2017-02-20	2198
2017-02-21	2413

```
df = pd.read_csv('data/time_series_one.csv',index_col='日期',parse_dates=['日期'])
def f(x):
    if len(x) == 6:
        return 1 if x[-1]>np.mean(x[:-1]) else 0
    else:
        return 0
result_b = df.loc[pd.date_range(start='20171227',end='20181231'),:].rolling(
                                        window=6,min_periods=1).agg(f)[5:]
result_b.head()
```

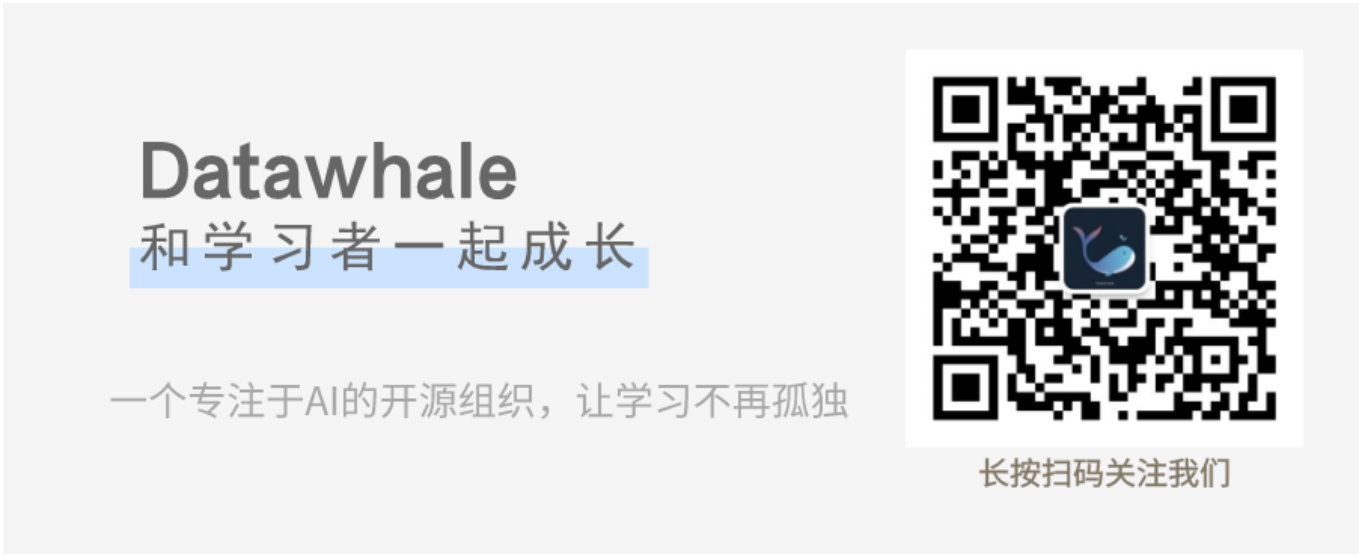
	销售额
2018-01-01	1.0
2018-01-02	0.0
2018-01-03	0.0
2018-01-04	0.0
2018-01-05	0.0

(c) 将(c)中的“向前5天”改为“向前非周末5天”，请再次计算结果

```
def f(x):
    if len(x) == 8:
        return 1 if x[-1]>np.mean(x[:-1][pd.Series([
            False if i in [5,6] else True for i in x[:-1].index.dayofweek],index=x[:-1].index)]) else 0
    else:
        return 0
result_c = df.loc[pd.date_range(start='20171225',end='20181231'),:].rolling(
                                    window=8,min_periods=1).agg(f)[7:]
result_c.head()
```

	销售额
2018-01-01	1.0
2018-01-02	0.0
2018-01-03	0.0
2018-01-04	0.0
2018-01-05	0.0

*本文电子版 后台回复 **时序数据** 获取*



“竟然学习完了，给自己点个赞↓