Pandas 时间序列处理

发表于 2019-08-09 13:24 阅读：2316 评论：0 推荐：0

Pandas 时间序列处理

目录

1 Python 的日期和时间处理

1.1 常用模块

> datetime time calendar

- datetime，以毫秒形式存储日期和时间
- datime.timedelta，表示两个 datetime 对象的时间差
- datetime 模块中包含的数据类型

类型	说明
date	以公历形式存储日历日期（年、月、日）
time	将时间存储为时、分、秒、毫秒
datetime	存储日期和时间
timedelta	表示两个 datetime 值之间的差（日、秒、毫秒）

1.2 字符串和 datetime 转换

datetime -> str

1. str(datetime_obj)

```
dt_obj = datetime(2019, 8, 8)
str_obj = str(dt_obj)
print(type(str_obj))
print(str_obj)
```

> <class 'str'>
> 2019-08-08 00:00:00

2. datetime.strftime()

```
str_obj2 = dt_obj.strftime('%d/%m/%Y')
print(str_obj2)
```

> 08/08/2019

str -> datetime

Pandas 时间序列处理

1. datetime.strptim
   需要指定时间表

```
dt_str = '2019-08-8'
dt_obj2 = datetime.strptime(dt_str, '%Y-%m-%d')
print(type(dt_obj2))
print(dt_obj2)
```

```
<class 'datetime.datetime'>
2019-08-08 00:00:00
```

2. dateutil.parser.parse()
   可以解析大部分时间表示形式

```
from dateutil.parser import parse
dt_str2 = '8-08-2019'
dt_obj3 = parse(dt_str2)
print(type(dt_obj3))
print(dt_obj3)
```

```
<class 'datetime.datetime'>
2019-08-08 00:00:00
```

3. pd.to_datetime()
   可以处理缺失值和空字符串

   具体看这

2 Pandas 的时间处理及操作

2.1 创建与基础操作

基本类型，以时间戳为索引的 Series->DatetimeIndex

指定 index 为 datetime 的 list

```
from datetime import datetime
import pandas as pd
import numpy as np

# 指定index为datetime的list
date_list = [datetime(2017, 2, 18), datetime(2017, 2, 19),
             datetime(2017, 2, 25), datetime(2017, 2, 26),
             datetime(2017, 3, 4), datetime(2017, 3, 5)]
time_s = pd.Series(np.random.randn(6), index=date_list)
print(time_s)
print(type(time_s.index))
```

Pandas 时间序列处理

```
2017-02-18 -0.230989
2017-02-19 -0.398082
2017-02-25 -0.309926
2017-02-26 -0.179672
2017-03-04 0.942698
2017-03-05 1.053092
dtype: float64
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

索引

- 索引位置

```
print(time_s[0])
```

> -0.230988627576

- 索引值

```
print(time_s[datetime(2017, 2, 18)])
```

> -0.230988627576

- 可以被解析的日期字符串

```
print(time_s['20170218'])
```

> -0.230988627576

- 按“年份”、“月份”索引

```
print(time_s['2017-2'])
```

```
2017-02-18   -0.230989
2017-02-19   -0.398082
2017-02-25   -0.309926
2017-02-26   -0.179672
dtype: float64
```

- 切片操作

```
print(time_s['2017-2-26':])
```

```
2017-02-26   -0.179672
2017-03-04    0.942698
2017-03-05    1.053092
dtype: float64
```

过滤

- 过滤掉日期之前的

```
time_s.truncate(before='2017-2-25')
```

```
2017-02-25   -0.309926
2017-02-26   -0.179672
```

Pandas 时间序列处理

```
2017-03-04    0.942
2017-03-05    1.053
dtype: float64
```

- 过滤掉日期之后的

```
time_s.truncate(after='2017-2-25')
```

```
2017-02-18   -0.230989
2017-02-19   -0.398082
2017-02-25   -0.309926
dtype: float64
```

pd.date_range()

功能：生成日期范围

```
dates = pd.date_range('2017-02-18', # 起始日期
                      periods=5,    # 周期
                      freq='W-SAT') # 频率
print(dates)
print(pd.Series(np.random.randn(5), index=dates))
```

```
DatetimeIndex(['2017-02-18', '2017-02-25', '2017-03-04', '2017-03-11',
'2017-03-18'],
dtype='datetime64[ns]', freq='W-SAT')
2017-02-18 -1.680280
2017-02-25 0.908664
2017-03-04 0.145318
2017-03-11 -2.940363
2017-03-18 0.152681
Freq: W-SAT, dtype: float64
```

- 传入开始、结束日期，默认生成的该时间段的时间点是按天计算的

```
date_index = pd.date_range('2017/02/18', '2017/03/18')
```

- 只传入开始或结束日期，还需要传入时间段

```
print(pd.date_range(start='2017/02/18', periods=10, freq='4D'))

print(pd.date_range(end='2017/03/18', periods=10))
```

- 规范化时间戳

```
print(pd.date_range(start='2017/02/18 12:13:14', periods=10))
print(pd.date_range(start='2017/02/18 12:13:14', periods=10, normali
```

```
DatetimeIndex(['2017-02-18 12:13:14', '2017-02-19 12:13:14',
               '2017-02-20 12:13:14', '2017-02-21 12:13:14',
               '2017-02-22 12:13:14', '2017-02-23 12:13:14',
```

Pandas 时间序列处理

```
               '201                                    13:14',
               '201                                    13:14'],
              dtype='datetime64[ns]', freq='D')
DatetimeIndex(['2017-02-18', '2017-02-19', '2017-02-20', '2017-02-2
               '2017-02-22', '2017-02-23', '2017-02-24', '2017-02-2
               '2017-02-26', '2017-02-27'],
              dtype='datetime64[ns]', freq='D')
```

频率与偏移量

- 频率 Freq，由基础频率的倍数组成，基础频率包括：
  1.BM:business end of month，每个月最后一个工作日
  2.D：天，M：月等

别名	偏移量类型	说明
D	Day	每日历日
B	BusinessDay	每工作日
H	Hour	每小时
T或min	Minute	每分
S	Second	每秒
L或ms	Milli	每毫秒（即每千分之一秒）
U	Micro	每微秒（即每百万分之一秒）
M	MonthEnd	每月最后一个日历日
BM	BusinessMonthEnd	每月最后一个工作日
MS	MonthBegin	每月第一个日历日
BMS	BusinessMonthBegin	每月第一个工作日
W-MON、W-TUE…	Week	从指定的星期几（MON、TUE、WED、THU、FRI、SAT、SUN）开始算起，每周
WOM-1MON、WOM-2MON…	WeekOfMonth	产生每月第一、第二、第三或第四周的星期几。例如，WOM-3FRI表示每月第3个星期五
Q-JAN、Q-FEB…	QuarterEnd	对于以指定月份（JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV、DEC）结束的年度，每季度最后一月的最后一个日历日
BQ-JAN、BQ-FEB…	BusinessQuarterEnd	对于以指定月份结束的年度，每季度最后一月的最后一个工作日

- 偏移量，每个基础频率对应一个偏移量
  1.偏移量通过加法连接

```
sum_offset = pd.tseries.offsets.Week(2) + pd.tseries.offsets.Hour(1
print(sum_offset)
```

Pandas 时间序列处理

```
print(pd.date_range                                    sum_offset))

14 days 12:00:00
DatetimeIndex(['2017-02-18 00:00:00', '2017-03-04 12:00:00'], dtype=
```

移动数据

沿时间轴将数据前移或后移，保持索引不变

```
ts = pd.Series(np.random.randn(5), index=pd.date_range('20170218', p
print(ts)

2017-02-18   -0.208622
2017-02-25    0.616093
2017-03-04   -0.424725
2017-03-11   -0.361475
2017-03-18    0.761274
Freq: W-SAT, dtype: float64
```

向后移动一位： `print(ts.shift(1))`

```
2017-02-18         NaN
2017-02-25   -0.208622
2017-03-04    0.616093
2017-03-11   -0.424725
2017-03-18   -0.361475
Freq: W-SAT, dtype: float64
```

pd.to_datetime()

功能：字符串转成时间格式

```
import pandas as pd
s_obj = pd.Series(['2017/02/18', '2017/02/19', '2017-02-25', '2017-(
s_obj2 = pd.to_datetime(s_obj)
print(s_obj2)
```

```
0 2017-02-18
1 2017-02-19
2 2017-02-25
3 2017-02-26
Name: course_time, dtype: datetime64[ns]
```

```
# 处理缺失值
s_obj3 = pd.Series(['2017/02/18', '2017/02/19', '2017-02-25', '2017-
                    name='course_time')
print(s_obj3)
```

Pandas 时间序列处理

```
0    2017/02/18
1    2017/02/19
2    2017-02-25
3    2017-02-26
4          None
Name: course_time, dtype: object
```

时间周期计算

- Period 类，通过字符串或整数及基础频率构造
- Period 对象可进行数学运算，但要保证具有相同的基础频率
- period_range，创建指定规则的时间周期范围，生成 PeriodIndex 索引，可用于创建 Series 或 DataFrame
- 时间周期的频率转换，asfreq
    - 如：年度周期->月度周期
- 按季度计算时间周期频率

2.2 时间数据重采样

重采样（resampling）

- 将时间序列从一个频率转换到另一个频率的过程，需要 聚合
- 高频率->低频率，downsampling，相反为 upsampling
- pandas 中的 resample 方法实现重采样
    - 产生 Resampler 对象
    - reample（freq）.sum0，resampe（freq）.mean）....

```
import pandas as pd
import numpy as np

date_rng = pd.date_range('20170101', periods=100, freq='D')
ser_obj = pd.Series(range(len(date_rng)), index=date_rng)

# 统计每个月的数据总和
resample_month_sum = ser_obj.resample('M').sum()
# 统计每个月的数据平均
resample_month_mean = ser_obj.resample('M').mean()

print('按月求和：', resample_month_sum)
print('按月求均值：', resample_month_mean)

按月求和： 2017-01-31     465
2017-02-28    1246
2017-03-31    2294
```

Pandas 时间序列处理

```
2017-04-30     945
Freq: M, dtype: int
按月求均值： 2017-01-31    15.0
2017-02-28    44.5
2017-03-31    74.0
2017-04-30    94.5
Freq: M, dtype: float64
```

降采样（downsampling）

- 将数据聚合到规整的低频率
- OHLC重采样，open，high，low，close

```
# 将数据聚合到5天的频率
five_day_sum_sample = ser_obj.resample('5D').sum()
five_day_mean_sample = ser_obj.resample('5D').mean()
five_day_ohlc_sample = ser_obj.resample('5D').ohlc()
```

- 使用 groupby 降采样
  使用函数对其进行分组操作

  ```
  ser_obj.groupby(lambda x: x.month).sum()
  ```

  ```
  ser_obj.groupby(lambda x: x.weekday).sum()
  ```

升采样（upsampling）

- 将数据从低频转到高频，需要 插值 ，否则为 NaN (直接重采样会产生空值)
- 常用的插值方法

1. ffill(limit) ，空值取前面的值填充，limit 为填充个数

   ```
   df.resample('D').ffill(2)
   ```

2. bfill（limit），空值取后面的值填充

   ```
   df.resample('D').bfill()
   ```

3. fillna（fill'）或 fllna（‘bfill）

   ```
   df.resample('D').fillna('ffill')
   ```

4. interpolate，根据插值算法补全数据
   线性算法： `df.resample('D').interpolate('linear')`
   具体可以参考：pandas.core.resample.Resampler.interpolate

2.3 滑动窗口

- 在时间窗口上计算各种统计函数

Pandas 时间序列处理

Method	Desc
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of values
median()	Arithmetic median of values
min()	Minimum
max()	Maximum
std()	Bessel-corrected sample standard deviation
var()	Unbiased variance
skew()	Sample skewness (3rd moment)
kurt()	Sample kurtosis (4th moment)
quantile()	Sample quantile (value at %)
apply()	Generic apply
cov()	Unbiased covariance (binary)
corr()	Correlation (binary)

- 窗口函数（window functions）

1. 滚动统计（rolling）
   obj.rolling().func

```
import pandas as pd
import numpy as np

ser_obj = pd.Series(np.random.randn(1000),
                    index=pd.date_range('20170101', periods=1000))
ser_obj = ser_obj.cumsum()
r_obj = ser_obj.rolling(window=5)
print(r_obj)
```

> Rolling [window=5,center=False,axis=0]

2. window
   窗口大小
3. center
   窗口是否居中统计

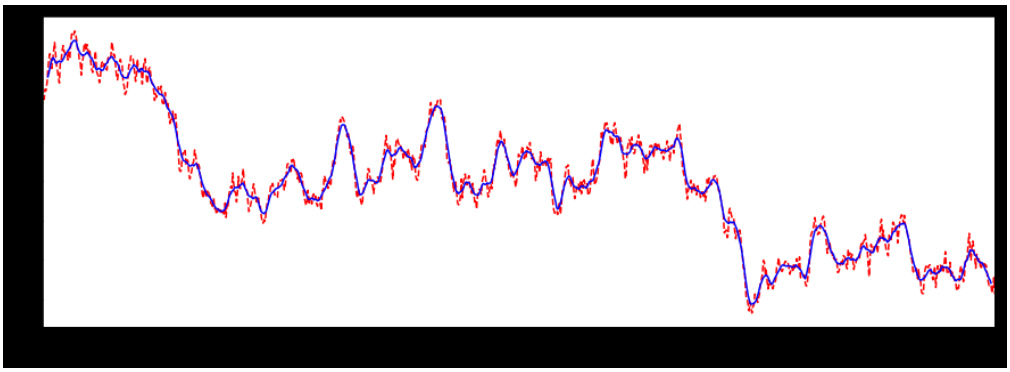
设置居中：

```
# 画图查看
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize=(15, 5))
```

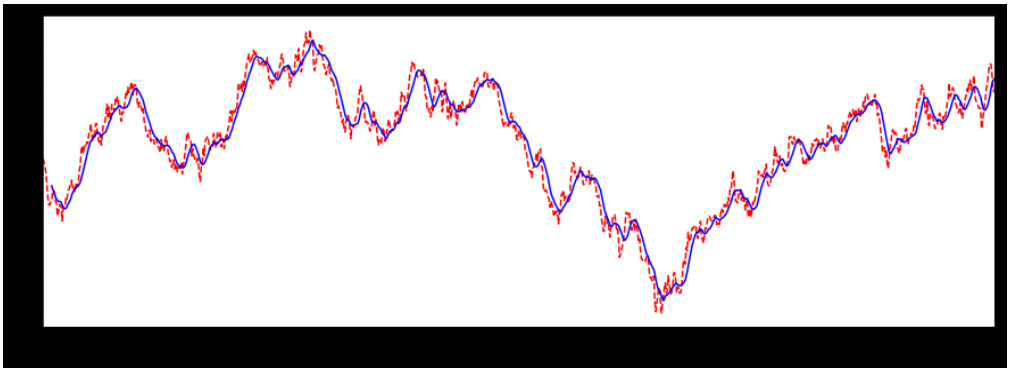
```
ser_obj.plot(style=
ser_obj.rolling(win                                    tyle='b')
```

Pandas 时间序列处理



不设置居中：

```
ser_obj.rolling(window=10, center=False).mean().plot(style='b')
```



__EOF__

本文作者：banshaohuan
本文链接：
https://www.cnblogs.com/banshaohuan/p/11326630.html

Pandas 时间序列处理

关于博主: 直接私信我。
版权声明: 本 BY-NC-SA 许可协议。转载请注明出处！
声援博主: 如果您觉得文章对您有帮助，可以点击文章右下角**【推荐】**一下。您的鼓励是博主的最大动力！

banshaohuan
关注 - 15
粉丝 - 11
+加关注

« 上一篇： Ubuntu 下使用 python3 制作读取 QR 码
» 下一篇： Python 数据分析中常用的可视化工具

posted @ 2019-08-09 13:24 banshaohuan 阅读(2316) 评论(0) 编辑 收藏 举报

Pandas 时间序列处理