TRƯỜNG ĐẠI HỌC SPKT TP.HỒ CHÍ MINH
KHOA ĐIỆN - ĐIỆN TỬ


MÔN HỌC: CƠ SỞ KHOA HỌC DỮ LIỆU

# Time Series

*Sinh viên:*
Võ Văn Linh - 20139080
Nguyễn Minh Nhật - 20139082
Nguyễn Huy Hoàng - 20139031
Nguyễn Thị Thảo - 20139090
Mai Huỳnh Tuấn Vũ - 20139098
Trần Ngọc Tường Vân - 20139097
Trần Quang Diệu - 20139066

*Giảng viên:*
TS Nguyễn Mạnh Hùng

# Contents

**1 Abstract** **3**

**2 Date and Time Data Types and Tools** **3**
2.1 Converting between string and datetime . . . . . 5

**3 Time Series Basics** **6**
3.1 Indexing, Selection, Subsetting . . . . . 10
3.2 Time Series with Duplicate Indices . . . . . 14

**4 Date Ranges, Frequencies, and Shifting** **17**
4.1 Generating Date Ranges . . . . . 18
4.2 Frequencies and Date Offsets . . . . . 20
4.3 Shifting (Leading and Lagging) Data . . . . . 25

**5 Time Zone Handling** **29**
5.1 Localization and Conversion . . . . . 30
5.2 Operations with Time Zoneaware Timestamp Objects . . . . . 33
5.3 Operations between Different Time Zones . . . . . 35

**6 Periods and Period Arithmetic** **36**
6.1 Period Frequency Conversion . . . . . 39
6.2 Quarterly Period Frequencies . . . . . 41
6.3 Converting Timestamps to Periods (and Back) . . . . . 44
6.4 Creating a PeriodIndex from Arrays . . . . . 46

**7 Resampling and Frequency Conversion** **47**
7.1 Downsampling . . . . . 49
7.1.1 Open-High-Low-Close (OHLC) resampling . . . . . 51
7.1.2 Resampling with GroupBy . . . . . 52
7.2 Upsampling and Interpolation . . . . . 53
7.3 Resampling with Periods . . . . . 55
7.4 Time Series Plotting . . . . . 57

**8 Moving Window Functions** **59**
8.1 Exponentially-weighted functions . . . . . 62
8.2 Binary Moving Window Functions . . . . . 63
8.3 User-Defined Moving Window Functions . . . . . 64

**9 Performance and Memory Usage Notes** **65**

**10 Conlusion** **67**

# 1 Abstract

Trong toán học, Time Series là tập hợp các điểm trong một khoảng thời gian nhất định. Hay nói một cách khác, time series là một chuỗi được thực hiện tại các điểm cách đều nhau liên tiếp trong thời gian.

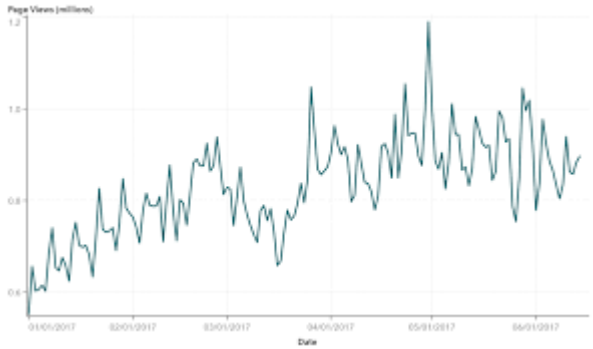

Figure 1: Time Series

Time series gồm có 2 lĩnh vực chính: Time series analysis và Time series forecasting.

Time series analysis là quá trình phân tích dữ liệu thu thập từ những khoảng thời gian trước đó. Khoảng thời gian phân tích có thể từ mili giây cho đến vài năm. Mục đích cuối cùng của time series analysis là nhận thức được sự biến động của các xu hướng từ đó nghiên cứu tìm ra nguyên nhân thay đổi.

Time series forecasting là kỹ thuật dự đoán sự thay đổi tiếp theo của môt sự kiện thông qua phân tích và xu hướng dữ liệu trong quá khứ.

Trong bài này chúng ta sẽ tìm hiểu về time series data, một trong những cấu trúc dữ liệu quan trọng áp dụng trên nhiều lĩnh vực của time series, thông qua *Timestamps*, *Fixed periods*, *Intervals of time*.

# 2 Date and Time Data Types and Tools

Trong Python có các thư viện chuẩn bao gồm kiểu dữ liệu cho ngày(date) và thời gian(time), đi kèm thì nó cũng có chức năng liên quan đến lịch. Modules chính bắt đầu là *datetime*, *time*.

```
from datetime import datetime
now = datetime.now()
now
```

Out

```
datetime.datetime(2022, 11, 9, 22, 20, 42, 899144)
```

```
now.year, now.month, now.day
```

Out

```
(2022, 11, 9)
```

Chúng ta nhận được kết quả khi chạy đoạn code trên tương ứng với năm, tháng, ngày, giờ, phút, giây, and micro giây. Như vậy module *datetime* có cá method sẽ trả về những thông tin của đối tượng *date*.

```
delta = datetime(2011, 1, 7) - datetime(2008, 6, 24, 8, 15)
delta
```

Out

```
datetime.timedelta(926, 56700)
```

```
delta.days
```

Out

```
926
```

```
delta.seconds
```

Out

```
56700
```

Với kiểu dữ liệu *datetime* có thể cộng trừ như các kiểu dữ liệu khác thông qua *timedelta* như ví trên khi ta trừ từ ngày 7 tháng 1 năm 2011 đến ngày 24 tháng 6 năm 2008 thì được 926. Để hiểu rõ hơn về *timedelta* xét thêm một vài ví dụ.

```
from datetime import timedelta
start = datetime(2011, 1, 7)
start + timedelta(12)
start - 2 * timedelta(12)
```

Out

```
datetime.datetime(2011, 1, 19, 0, 0)

datetime.datetime(2010, 12, 14, 0, 0)
```

Xét theo cú pháp *timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)* thì ở ví dụ trên *timedelta(12)* tương ứng với 12 ngày cho nên kết quả từ ngày 7 tháng 11 năm 2011 sẽ cộng thêm 12 ngày sẽ thành ngày 19 thàng 11 năm 2011, tương ứng với phép toán còn lại. Vậy ta thấy rằng *timedelta* thường được sử dụng để tính toán sự khác biệt về ngày tháng và cũng có thể được sử dụng để thao tác ngày tháng trong Python.

| Type | Description |
|---|---|
| date | Lưu trữ dữ liệu theo lịch (ngày, tháng, năm) |
| time | Lưu trữ kiểu dữ liệu theo giờ, phút, giây, milis giây |
| datetime | Lưu trữ cả *date* và *time* |
| timedelta | sự khác biệt giữa 2 giá trị *datetime* |

Table 1: Types in datetime module

## 2.1 Converting between string and datetime

Từ dữ liệu *datetime* có thể chuyển về *string* thông qua phương thức str() hoặc strftime().

```
stamp = datetime(2011, 1, 3)
str(stamp)
stamp.strftime('%Y-%m-%d')
```

Out

```
'2011-01-03 00:00:00'
```

```
'2011-01-03'
```

```
value = '2011-01-03'
datetime.strptime(value, '%Y-%m-%d')
datestrs = ['7/6/2011', '8/6/2011']
[datetime.strptime(x, '%m/%d/%Y') for x in datestrs]
```

Out

```
[datetime.datetime(2011, 7, 6, 0, 0), datetime.datetime(2011, 8, 6, 0, 0)]
```

Trong phương thức strftime() các đing dạng "%Y-%m-%d" tương ứng với năm, tháng, ngày, có thể thay đổi thứ tự giữa các định dạng. Ngoài ra còn có các định dạng khác của giờ, phút, giây.

| Type | Description |
|---|---|
| %Y | 4-digit year |
| %y | 2-digit year |
| %m | 2-digit month [01, 12] |
| %d | 2-digit day [01, 31] |

# 3 Time Series Basics

Một đối tượng *timeseries* chứa dữ liệu và thông tin thời gian trong các thuộc tính của nó mô tả một quá trình động.Có thể sử dụng các hàm của đối tượng *timeseries* để tạo, sửa đổi và phân tích hành động của một time series.

| Type | Description |
|---|---|
| %H | Hour (24-hour clock) [00, 23] |
| %I | Hour (12-hour clock) [01, 12] |
| %M | 2-digit minute [00, 59] |
| %S | Second [00, 61] (seconds 60, 61 account for leap seconds) |
| %w | Weekday as integer [0 (Sunday), 6] |
| %U | Week number of the year [00, 53]. Sunday is considered the first day of the week, and days before the first Sunday of the year are "week 0". |
| %W | Week number of the year [00, 53]. Monday is considered the first day of the week, and days before the first Monday of the year are "week 0". |
| %z | UTC time zone offset as +HHMM or -HHMM, empty if time zone naive |
| %F | Shortcut for %Y-%m-%d, for example 2012-4-18 |
| %D | Shortcut for %m/%d/%y, for example 04/18/12 |

Table 2: Datetime format specification (ISO C89 compatible)

Căn bản nhất của đối tượng *timeseries* trong *pandas* là một series indexed theo dấu thời gian (*timeseries*).

```
from datetime import datetime
dates = [datetime(2011, 1, 2), datetime(2011, 1, 5),
         datetime(2011, 1, 7), datetime(2011, 1, 8),
         datetime(2011, 1, 10), datetime(2011, 1, 12)]
ts = pd.Series(np.random.randn(6), index=dates)
ts
```

Out

```
2011-01-02   0.690002

2011-01-05   1.001543

2011-01-07   -0.503087

2011-01-08   -0.622274
```

```
2011-01-10   -0.921169

2011-01-12   -0.726213
```

Đưa các đối tượng *datetime()* chứa thông tin vào một list là biến *dates*.Ta tạo một series biến *ts* và thêm các giá trị ngẫu nhiên với từng index thuộc biến *dates*

Ta dùng hàm *type()* để kiểm tra biến *ts* và thấy biến thuộc loại TimeSeries:

```
1 type(ts)
```

Out

```
pandas.core.series.TimeSeries
```

```
1 ts.index
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2011-01-02 00:00:00, ..., 2011-01-12 00:00:00]

Length: 6, Freq: None, Timezone: None
```

*Pandas.Series.index* cho thấy chỉ mục (nhãn trục) của Series các đối tượng *datetime* này đã được đưa vào một DatetimeIndex.

Không cần thiết phải sử dụng phương thức khởi tạo TimeSeries một cách rõ ràng; khi tạo Series với DatetimeIndex, pandas biết rằng đối tượng đó là 1 time series.

Giống như các series khác, các phép toán số học giữa các time series indexed khác nhau sẽ tự động căn chỉnh vào các dates:

```
1 ts + ts[::2]
```

Out

```
2011-01-02    1.380004
2011-01-05         NaN
2011-01-07   -1.006175
2011-01-08         NaN
2011-01-10   -1.842337
2011-01-12         NaN
```

Ở phép tính trên ts[::2] chỉ chứa những index từ 0 với bước nhảy là 2 nên ta thấy output những giá trị tại index lẻ là NaN

Pandas lưu trữ *timestamps* bằng cách sử dụng loại dữ liệu datetime64 của NumPy ở độ phân giải nano giây:

```
1 ts.index.dtype
```

Out

```
dtype('datetime64[ns]')
```

Giá trị vô hướng từ DatetimeIndex là đối tượng pandas Timestamp:

```
1 stamp = ts.index[0]
2 stamp
```

Out

```
<Timestamp: 2011-01-02 00:00:00>
```

Ở trên ta xem Timestamp của biến *ts* tại index đầu tiên, vì ta không đặt *time* cho index đầu tiên nên *time* lúc in ra là 00:00:00.

Timestamp có thể được thay thế ở bất kỳ đâu bạn sử dụng đối tượng datetime. Ngoài ra , nó có thể lưu trữ thông tin tần suất (nếu có) và hiểu cách thực hiện múi giờ chuyển đổi và các loại thao tác khác.( Tìm hiểu thêm về cả hai điều này sau).

Hàm timestamp() của mô-đun datetime trả về dấu thời gian POSIX(1 January 1970) tương ứng với trường hợp datetime. Giá trị trả về là float.

## 3.1 Indexing, Selection, Subsetting

TimeSeries là một lớp con của Series và do đó hoạt động theo cùng một cách đối với idexing và chọn dữ liệu dựa trên nhãn:

```
stamp = ts.index[2]
ts[stamp]
```

Out

```
-0.50308739136034464
```

Để thuận tiện hơn có thể ghi thành một chuỗi nhưng vẫn có thể hiểu được ngày tháng năm :

```
ts['1/10/2011']
```

Out

```
-0.92116860801301081
```

Ta có thể ghi ngày tháng năm cách nhau bằng dấu / như trên

```
ts['20110110']
```

Out

```
-0.92116860801301081
```

Hoặc ta ghi liền như trên (nhưng phải đúng thứ tự) chương trình vẫn có thể hiểu được ngày tháng năm trong chuỗi

Đối với chuỗi thời gian dài hơn, một năm hoặc chỉ một năm và tháng có thể được thông qua để dễ dàng lựa chọn các phần dữ liệu:

```
longer_ts = pd.Series(np.random.randn(1000),
                      index=pd.date_range('1/1/2000',
                      periods=1000))
longer_ts
```

Out

```
2000-01-01    0.222896
2000-01-02    0.051316
2000-01-03   -1.157719
2000-01-04    0.816707
...
2002-09-23   -0.395813
2002-09-24   -0.180737
2002-09-26   -0.416584
Freq: D, Length: 1000
```

Ta tạo một *series* với các giá trị random (1000 giá trị) và index là một dải thời gian bắt đầu từ 1/1/2000 với số lượng thời gian tạo ra(1000).

```
1 longer_ts['2001']
```

Out

```
2001-01-01   -1.499503
2001-01-02    0.545154
2001-01-03    0.400823
2001-01-04   -1.946230
...
2001-12-28   -1.568139
2001-12-29   -0.900887
2001-12-30    0.652346
2001-12-31    0.871600
Freq: D, Length: 365
```

Ở trên ta chọn với thời gian là năm (*longer_ts['2001']*)

```
longer_ts['2001-05']
```

Out

```
2001-05-01   1.662014
2001-05-02   -1.189203
2001-05-03   0.093597
2001-05-04   -0.539164
...
2001-05-28   -0.683066
2001-05-29   -0.950313
2001-05-30   0.400710
2001-05-31   -0.126072
Freq: D, Length: 31
```

Ở trên ta chọn với thời gian là năm và tháng (*longer_ts['2001-05']*)

Việc cắt (slicing) dữ liệu hoạt động như một Series thông thường khi ta cắt bắt đầu từ khoảng thời gian 1/7/2011 trong biến *ts*

```
ts[datetime(2011, 1, 7):]
```

Out

```
2011-01-07   -0.503087
2011-01-08   -0.622274
2011-01-10   -0.921169
2011-01-12   -0.726213
```

Bởi vì hầu hết dữ liệu time series được sắp xếp theo thứ tự thời gian, bạn có thể cắt bằng timestamps không nằm trong time series để thực hiện truy vấn phạm vi:

Ở đây ta lấy trong khoảng từ 1/6/2011 đến 1/11/2011 tuy 2 mốc thời gian đó không có trong biến *ts* nhưng ta vẫn truy vấn được phạm vi giữa 2 mốc đó thuộc biến *ts*

```
ts['1/6/2011':'1/11/2011']
```

Out

```
2011-01-07    -0.503087

2011-01-08    -0.622274

2011-01-10    -0.921169
```

Có một phương thức tương đương cắt ngắn một TimeSeries giữa hai *date*,cắt theo cách này tạo ra các chế độ xem trên nguồn time series giống như cắt mảng NumPy

```
ts.truncate(after='1/9/2011')
```

Out

```
2011-01-02    0.690002

2011-01-05    1.001543

2011-01-07    -0.503087

2011-01-08    -0.622274
```

Ở trên hàm truncate(after='1/9/2011') cho phép ta cắt đi khoảng thời gian sau 1/9/2011

Tất cả những điều trên đúng với DataFrame, indexing trên các hàng của nó:

```
dates = pd.date_range('1/1/2000', periods=100,
                      freq='W-WED')
long_df = pd.DataFrame(np.random.randn(100, 4),
                    index=dates, columns=
                    ['Colorado', 'Texas', 'New York', 'Ohio'])
long_df.ix['5-2001']
```

Out

| | Colorado | Texas | New York | Ohio |
|---|---|---|---|---|
| 2001-05-02 | 0.943479 | -0.349366 | 0.530412 | -0.508724 |
| 2001-05-09 | 0.230643 | -0.065569 | -0.248717 | -0.587136 |
| 2001-05-16 | -1.022324 | 1.060661 | 0.954768 | -0.511824 |
| 2001-05-23 | -1.387680 | 0.767902 | -1.164490 | 1.527070 |
| 2001-05-30 | 0.287542 | 0.715359 | -0.345805 | 0.470886 |

Ở trên ta tạo biến *date* với pd.date_range() lấy một dải thời gian với số lượng là 100 và tần suất là thứ tư hàng tuần (freq='W-WED') và tạo dataframe *long_ df* với index là biến *date* với các giá trị ngẫu nhiên và cột,ix[] ở đây hoạt động như khi ta chọn dữ liệu và ở đây ta chọn thời gian là trong tháng 5/2001.

## 3.2 Time Series with Duplicate Indices

Trong một số ứng dụng, có thể có nhiều sự chú ý dữ liệu rơi vào một timestamp. Đây là một ví dụ:

```
dates = pd.DatetimeIndex(['1/1/2000', '1/2/2000',
                          '1/2/2000','1/2/2000','1/3/2000'])
dup_ts = pd.Series(np.arange(5), index=dates)
dup_ts
```

Out

```
2000-01-01    0

2000-01-02    1

2000-01-02    2

2000-01-02    3

2000-01-03    4
```

Về pd.DatetimeIndex hàm này giống như mảng (ndarray) bất biến của dữ liệu datetime64.

Được biểu diễn bên trong dưới dạng int64 và có thể được gom lại cho các đối tượng Timestamp là các lớp con của datetime và mang siêu dữ liệu.

Chúng ta có thể biết rằng index không phải là duy nhất bằng cách kiểm tra thuộc tính is_unique của nó:

```
1 dup_ts.index.is_unique
```

Out

```
False
```

Indexing vào time series này bây giờ sẽ tạo ra các giá trị vô hướng hoặc các phần cắt tùy thuộc vào việc liệu một timestamp có bị trùng lặp hay không:

```
1 dup_ts['1/3/2000']    #not  duplicated
```

Out

```
4
```

Như ở trên ta thấy chỉ in ra một giá trị. Còn bên dưới in ra một phần đã được cắt của một timestamp đã bị trùng.

```
1 dup_ts['1/2/2000']    #duplicated
```

Out

```
2000-01-02   1

2000-01-02   2

2000-01-02   3
```

Giả sử bạn muốn tổng hợp dữ liệu có timestamps không phải là duy nhất. Có một cách để làm điều này là sử dụng groupby và đặt mức = 0 (mức index):

```
grouped = dup_ts.groupby(level=0)

grouped.mean()
```

Out

```
2000-01-01  0

2000-01-02  2

2000-01-03  4
```

Tính giá trị trung bình(mean())

```
grouped.count()
```

Out

```
2000-01-01  1

2000-01-02  3

2000-01-03  1
```

Đếm tần suất xuất hiện các index (count())

# 4 Date Ranges, Frequencies, and Shifting

Chuỗi thời gian chung trong Pandas được coi là không đều; nghĩa là chúng không có tần suất cố định. Đối với nhiều ứng dụng như vậy là đủ. Tuy nhiên, bạn nên có tần suất cố định, chẳng hạn như hàng ngày, hàng tháng hoặc 15 phút một lần, ngay cả khi điều đó có nghĩa là đưa các giá trị còn thiếu vào một chuỗi thời gian. May mắn thay, Pandas có một bộ đầy đủ các tần số chuỗi thời gian tiêu chuẩn và các công cụ để lấy mẫu lại, suy luận tần số và tạo phạm vi ngày tần số cố định. Ví dụ: trong chuỗi thời gia , việc chuyển đổi nó thành tần suất cố định hàng ngày có thể được thực hiện bằng cách gọi resample:

```
ts
```

Out

```
2011-01-02    0.690002

2011-01-05    1.001543

2011-01-07    -0.503087

2011-01-08    -0.622274

2011-01-10    -0.921169

2011-01-12    -0.726213
```

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

Out

```
2011-01-02    0.690002

2011-01-03         NaN

2011-01-04         NaN

2011-01-05    1.001543
```

```
2011-01-06         NaN

2011-01-07    -0.503087

2011-01-08    -0.622274

2011-01-09         NaN

2011-01-10    -0.921169

2011-01-11         NaN

2011-01-12    -0.726213

Freq: D
```

Chuyển đổi giữa các tần số hoặc lấy mẫu lại là một chủ đề đủ lớn để có phần riêng về sau. Ở đây sẽ hướng dẫn cách sử dụng các tần số cơ sở và bội số của chúng.

## 4.1 Generating Date Ranges

Ta đã sử dụng nó trước đây mà không giải thích, bạn có thể đoán rằng *pandas.date_ range* chịu trách nhiệm tạo Datetime Index với độ dài được chỉ định theo tần suất cụ thể:

```
1 index = pd.date_range("4/1/2012", "6/1/2012")
2 index
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2012-04-01 00:00:00, ..., 2012-06-01 00:00:00]

Length: 62, Freq: D, Timezone: None
```

Theo mặc định, date_range tạo dấu thời gian ngày. Nếu bạn chỉ truyền ngày bắt đầu hoặc ngày kết thúc

```
pd.date_range(start='4/1/2012', periods=20)
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2012-04-01 00:00:00, ..., 2012-04-20 00:00:00]

Length: 20, Freq: D, Timezone: None
```

```
pd.date_range(end='6/1/2012', periods=20)
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2012-05-13 00:00:00, ..., 2012-06-01 00:00:00]

Length: 20, Freq: D, Timezone: None
```

Ngày bắt đầu và ngày kết thúc xác định ranh giới nghiêm ngặt cho chỉ mục ngày được tạo. Ví dụ, nếu bạn muốn có chỉ mục ngày chứa ngày làm việc cuối cùng của mỗi tháng, bạn sẽ chuyển tần suất 'BM' (cuối tháng làm việc) và chỉ những ngày rơi vào hoặc trong khoảng thời gian ngày mới được đưa vào:

```
pd.date_range('1/1/2000', '12/1/2000', freq='BM')
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2000-01-31 00:00:00, ..., 2000-11-30 00:00:00]

Length: 11, Freq: BM, Timezone: None
```

date_range theo mặc định bảo toàn thời gian (nếu có) của dấu thời gian bắt đầu hoặc kết thúc:

```
pd.date_range('5/2/2012 12:56:31', periods=5)
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2012-05-02 12:56:31, ..., 2012-05-06 12:56:31]

Length: 5, Freq: D, Timezone: None
```

Đôi khi, bạn sẽ có ngày bắt đầu hoặc ngày kết thúc với thông tin thời gian nhưng muốn tạo một tập hợp dấu thời gian được chuẩn hóa thành nửa đêm theo quy ước. Để làm điều này, có một tùy chọn chuẩn hóa :

```
pd.date_range('5/2/2012 12:56:31', periods=5, normalize=True)
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2012-05-02 00:00:00, ..., 2012-05-06 00:00:00]

Length: 5, Freq: D, Timezone: None
```

## 4.2 Frequencies and Date Offsets

Tần số trong Pandas bao gồm tần số cơ sở (base frequency) và số nhân (multiplier). Tần số cơ sở thường được gọi bằng bí danh chuỗi, chẳng hạn như 'M' cho hàng tháng hoặc 'H' cho hàng giờ.

```
from pandas.tseries.offsets import Hour, Minute
hour = Hour()
hour
```

Out

```
<1 Hour>
```

**Week of month dates**

Bạn có thể xác định bội số của phần bù bằng cách chuyển một số nguyên:

```
pd.date_range('1/1/2000', '1/3/2000 23:59', freq='4h')
```

```
<class'pandas.tseries.index.DatetimeIndex'>

[2000-01-01 00:00:00, ..., 2000-01-03 20:00:00]

Length: 18, Freq: 4H, Timezone: None
```

Bạn có thể xác định bội số của phần bù bằng cách chuyển một số nguyên

```
four_hours = Hour(4)
four_hours
```

Out

```
<4 Hours>
```

Trong hầu hết các ứng dụng, bạn sẽ không bao giờ cần tạo một trong các đối tượng này một cách rõ ràng, thay vào đó sử dụng bí danh chuỗi như 'H' hoặc '4H'. Đặt một số nguyên trước tần số cơ sở sẽ tạo ra một bội số:

```
pd.date_range('1/1/2000', '1/3/2000 23:59', freq='4h')
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2000-01-01 00:00:00, ..., 2000-01-03 20:00:00]

Length: 18, Freq: 4H, Timezone: None
```

Nhiều độ lệch có thể được kết hợp với nhau bằng phép cộng:

```
Hour(2) + Minute(30)
```

Out

```
<150 Minutes>
```

Tương tự, bạn có thể chuyển các chuỗi tần số như '2h30min' sẽ được phân tích cú pháp thành cùng một biểu thức một cách hiệu quả:

```
pd.date_range('1/1/2000', periods=10, freq='1h30min')
```

Out

```
<class'pandas.tseries.index.DatetimeIndex'>

[2000-01-01 00:00:00, ..., 2000-01-01 13:30:00]

Length: 10, Freq: 90T, Timezone: None
```

Một số tần số mô tả các thời điểm không cách đều nhau. Ví dụ: 'M' (cuối tháng theo lịch) và 'BM' (ngày làm việc cuối/ngày trong tuần của tháng) phụ thuộc vào số ngày trong tháng và trong trường hợp sau, tháng có kết thúc vào cuối tuần hay không. Vì thiếu một thuật ngữ tốt hơn, tôi gọi đây là các độ lệch cố định.

Xem Bảng để biết danh sách các mã tần suất và các lớp bù ngày có sẵn trong Pandas.

Người dùng có thể xác định các lớp tần suất tùy chỉnh của riêng họ để cung cấp logic ngày không có sẵn trong Pandas, mặc dù chi tiết đầy đủ về điều đó nằm ngoài phạm vi của cuốn sách này.

| Alias | Loại offset | Mô tả |
|---|---|---|
| D | Day | Lịch hàng ngày |
| B | BusinessDay | Ngày làm việc |
| H | Hour | Hàng giờ |
| T hoặc min | Minute | Từng phút |
| S | Second | Từng giây |
| L hoặc ms | Milli | Mili giây (1/1000 của 1 giây) |
| U | Micro | Micro giây (1/1000000 của 1 giây) |
| M | MonthEnd | Ngày cuối cùng của tháng (theo lịch) |
| BM | BusinessMonthEnd | Ngày làm việc cuối cùng của tháng (là một ngày trong tuần) |

| | | |
|---|---|---|
| MS | MonthBegin | Ngày đầu tiên của tháng (theo lịch) |
| BMS | BusinessMonthBegin | Ngày đầu tuần của tháng |
| W-MON, W-TUE, ... | Week | Vào ngày nhất định trong tuần: Thứ Hai, Thứ Ba, Thứ Tư, Thứ Năm, Thứ Sáu, Thứ Bảy hoặc Chủ Nhật |
| WOM-1MON, WOM-2MON, ... | WeekOfMonth | Tạo ngày hàng tuần trong tuần đầu tiên, thứ hai, thứ ba hoặc thứ tư của tháng. Ví dụ: WOM-3FRI cho Thứ Sáu của tuần thứ 3 hàng tháng. |
| Q-JAN, Q-FEB, ... | QuarterEnd | Các ngày hàng quý được cố định vào ngày dương lịch cuối cùng của mỗi tháng, cho năm kết thúc vào tháng được chỉ định: JAN, FEB, MAR, APR, May, JUN, JUL, AUG, SEP, OCT, NOV hoặc DEC |
| BQ-JAN, BQ-FEB, ... | BusinessQuarterEnd | Ngày hàng quý được cố định vào ngày cuối tuần của mỗi tháng, cho năm kết thúc vào tháng được chỉ định |
| QS-JAN, QS-FEB, ... | QuarterBegin | Ngày hàng quý được cố định vào ngày dương lịch đầu tiên của mỗi tháng, cho năm kết thúc vào tháng được chỉ định |
| BQS-JAN, BQS-FEB, ... | BusinessQuarterBegin | Các ngày hàng quý được cố định vào ngày trong tuần đầu tiên của mỗi tháng, cho năm kết thúc vào tháng được chỉ định. |
| A-JAN, A-FEB, ... | YearEnd | Các ngày hàng năm được cố định vào ngày dương lịch cuối cùng của tháng đã cho: JAN, FEB, MAR, APR, May, JUN, JUL, AUG, SEP, OCT, NOV , hoặc tháng 12 |

| | | |
|---|---|---|
| BA-JAN, BA-FEB, ... | BusinessYearEnd | Ngày hàng năm được cố định vào ngày trong tuần cuối cùng của tháng nhất định |
| AS-JAN, AS-FEB, ... | YearBegin | Ngày hàng năm được neo vào ngày đầu tiên của tháng nhất định |
| BAS-JAN, BAS-FEB, ... | BusinessYearBegin | Ngày hàng năm được cố định vào ngày trong tuần đầu tiên của tháng nhất định |

Table 3: Base Time Series Frequencies

### Week of month dates

Một loại tần suất hữu ích là "tuần trong tháng", bắt đầu với WOM. Điều này cho phép bạn có được những ngày như thứ Sáu thứ ba của mỗi tháng:

```
rng = pd.date_range('1/1/2012', '9/1/2012', freq='WOM-3FRI')
```

Out

```
[<Timestamp:  2012-01-20 00:00:00>,

 <Timestamp:   2012-02-17 00:00:00>,

 <Timestamp:   2012-03-16 00:00:00>,

 <Timestamp:   2012-04-20 00:00:00>,

 <Timestamp:   2012-05-18 00:00:00>,

 <Timestamp:   2012-06-15 00:00:00>,
```

```
<Timestamp:    2012-07-20 00:00:00>,

<Timestamp:    2012-08-17 00:00:00>]
```

Các nhà giao dịch quyền chọn vốn chủ sở hữu Hoa Kỳ sẽ nhận ra những ngày này là ngày hết hạn hàng tháng tiêu chuẩn.

## 4.3 Shifting (Leading and Lagging) Data

"Shifting" đề cập đến việc di chuyển dữ liệu tiến và lùi theo thời gian. Cả Series và DataFrame đều có phương pháp dịch chuyển để thực hiện các dịch chuyển tiến hoặc lùi đơn giản, khiến chỉ mục không bị sửa đổi:

```
1 ts = Series(np.random.randn(4), index=pd.date_range('1/1/2000' periods=4, freq='M'))
2 ts
```

Out:

```
2000-01-31    0.575283

2000-02-29    0.304205

2000-03-31    1.814582

2000-04-30    1.634858

Freq: M
```

```
1 ts.shift(2)
```

Out

```
2000-01-31       NaN

2000-02-29       NaN

2000-03-31  0.575283

2000-04-30  0.304205

Freq: M
```

```
ts.shift(-2)
```

Out

```
2000-01-31    1.814582
2000-02-29    1.634858
2000-03-31         NaN
2000-04-30         NaN
Freq: M
```

Một cách sử dụng phổ biến củ ashift là tính toán phần trăm thay đổi trong một chuỗi thời gian hoặc nhiều chuỗi thời gian dưới dạng các cột DataFrame. Điều này được thể hiện như

```
ts / ts.shift(1) - 1
```

Bởi vì những thay đổi này khiến chỉ mục không được sửa đổi, một số dữ liệu sẽ bị loại bỏ. Do đó, nếu biết tần số, nó có thể được chuyển sang shift để tăng dấu thời gian thay vì chỉ đơn giản là dữ liệu:

```
ts.shift(2, freq='M')
```

Out

```
2000-03-31    0.575283
2000-04-30    0.304205
2000-05-31    1.814582
2000-06-30    1.634858
Freq: M
```

Các tần số khác cũng có thể được chuyển, giúp bạn linh hoạt hơn trong cách hướng đến và làm chậm dữ liệu:

```
ts.shift(3, freq='D')
```

Out

```
2000-02-03    0.575283
2000-03-03    0.304205
2000-04-03    1.814582
2000-05-03    1.634858
```

```
ts.shift(1, freq='3D')
```

Out

```
2000-02-03  0.575283
2000-03-03  0.304205
2000-04-03  1.814582
2000-05-03  1.634858
```

```
ts.shift(1, freq='90T')
```

Out

```
2000-01-31 01:30:00   0.575283
2000-02-29 01:30:00   0.304205
2000-03-31 01:30:00   1.814582
2000-04-30 01:30:00   1.634858
```

**Shifting dates with offsets**

các offsets ngày của Pandas cũng có thể được sử dụng với các đối tượng datetime hoặc Timestamp :

```
from pandas.tseries.offsets import Day, MonthEnd
now = datetime(2011, 11, 17)
now + 3 * Day()
```

Out

```
datetime.datetime(2011, 11, 20, 0, 0)
```

Nếu bạn thêm một Offset cố định như MonthEnd, thì số đầu tiên sẽ chuyển tiếp một ngày sang ngày tiếp theo theo quy tắc tần suất:

```
now + MonthEnd()
```

Out

```
datetime.datetime(2011, 11, 30, 0, 0)
```

```
now + MonthEnd(2)
```

Out

```
datetime.datetime(2011, 12, 31, 0, 0)
```

Độ lệch cố định có thể "cuộn" ngày về phía trước hoặc phía sau một cách rõ ràng bằng cách sử dụng các phương pháp roll forWard và rollback của chúng , tương ứng:

```
offset = MonthEnd()
offset.rollforward(now)
```

Out

```
datetime.datetime(2011, 11, 30, 0, 0)
```

```
offset.rollback(now)
```

Out

```
datetime.datetime(2011, 10, 31, 0, 0)
```

Một cách sử dụng offset ngày hay là sử dụng các phương thức này với groupby:

```
ts = Series(np.random.randn(20), index=pd.date_range('1/15/2000', periods=20, freq='4d'))
ts.groupby(offset.rollforward).mean()
```

Out

```
2000-01-31    -0.448874

2000-02-29  -0.683663

2000-03-31  0.251920
```

Tất nhiên, một cách dễ dàng hơn và nhanh hơn để làm điều này là sử dụng resample:

```
ts.resample('M', how='mean')
```

Out

```
2000-01-31    -0.448874

2000-02-29    -0.683663

2000-03-31     0.251920

Freq: M
```

# 5 Time Zone Handling

Trong Python, thông tin múi giờ đến từ thư viện pytz của bên thứ 3, thư viện này hiển thị cơ sở dữ liệu Olson, một tập hợp thông tin múi giờ thế giới. Điều này đặc biệt quan trọng đối với dữ liệu lịch sử vì ngày chuyển đổi DST (và thậm chí cả thời gian bù giờ UTC) đã bị thay đổi nhiều lần tùy thuộc vào ý tưởng bất chợt của chính quyền địa phương. Tại Hoa Kỳ, thời gian chuyển đổi DST đã được thay đổi nhiều lần kể từ năm 1900!

Để biết thông tin chi tiết về thư viện pytz, bạn sẽ cần xem tài liệu của thư viện đó. Theo như cuốn sách này có liên quan, pandas bao hàm chức năng của pytz nên bạn có thể bỏ qua API của nó bên ngoài tên múi giờ. Tên múi giờ có thể được tìm thấy tương tác và trong tài liệu:

```
import pytz
pytz.common_timezones[-5:]
```

Out

```
['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

Để lấy một đối tượng múi giờ từ pytz, sử dụng pytx.timezone:

```
tz = pytz.timezone('US/Eastern')
tz
```

Out

```
<DstTzInfo 'US/Eastern' EST-1 day, 19:00:00 STD>
```

Các phương thức trong pandas sẽ chỉ chấp nhận tên timezone hoặc các đối tượng. Tôi đề xuất chỉ sử dụng tên.

## 5.1 Localization and Conversion

Theo mặc định, time seiries trong pandas là timezone naive. xem xét time series sau:

```
rng = pd.date_range('3/9/2012 9:30', periods=6, freq='D')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out:

| | | |
|---|---|---|
| 2012-03-09 | 09:30:00 | -0.783408 |
| 2012-03-10 | 09:30:00 | -1.797685 |
| 2012-03-11 | 09:30:00 | -0.172670 |
| 2012-03-12 | 09:30:00 | 0.680215 |
| 2012-03-13 | 09:30:00 | 1.607578 |
| 2012-03-14 | 09:30:00 | 0.200381 |

Freq: D, dtype: float64

pd.date_ra sẽ trả về phạm vi các thời điểm cách đều nhau theo một tần số cố định.Trong ví dụ trên, '3/9/2012 9:30' là thời gian bắt đầu,chương trình tạo ra periods=6 điểm thời gian, mỗi điểm chênh lệch nhau theo tần số đã chỉ định là freq = 'D'

Index's tz field là None:

```
print(ts.index.tz)
```

Out: None

Date ranges có thể được tạo với time zone set:

```
pd.date_range('3/9/2012 9:30', periods=10, freq='D', tz='UTC')
```

Out

<class 'pandas.tseries.index.DatetimeIndex'>
[2012-03-09 09:30:00, ..., 2012-03-18 09:30:00]
Length: 10, Freq: D, Timezone: UTC

Chuyển đổi từ dạng naive sang localized được xử lý bởi e tz_localize:

```
ts_utc = ts.tz_localize('UTC')
ts_utc
```

Out:

| | | |
|---|---|---|
| 2012-03-09 | 09:30:00+00:00 | 0.414615 |
| 2012-03-10 | 09:30:00+00:00 | 0.427185 |
| 2012-03-11 | 09:30:00+00:00 | 1.172557 |
| 2012-03-12 | 09:30:00+00:00 | -0.351572 |
| 2012-03-13 | 09:30:00+00:00 | 1.454593 |
| 2012-03-14 | 09:30:00+00:00 | 2.043319 |

Freq: D

Hàm ts.tz_localize trên đã đặt múi giờ của đối tượng ts thành UTC

```
ts_utc.index}
```

Out;

<class 'pandas.tseries.index.DatetimeIndex'>
[2012-03-09 09:30:00, ..., 2012-03-14 09:30:00]
Length: 6, Freq: D, Timezone: UTC

Sau khi một chuỗi thời gian đã được bản địa hóa thành một múi giờ cụ thể, nó có thể được chuyển đổi thành múi giờ khác sử dụng tz_convert. Dưới đây là hàm chuyển đổi thành múi giờ miền Đông (Bắc Mỹ):

```
ts_utc.tz_convert('US/Eastern')
```

Out:

| | | |
|---|---|---|
| 2012-03-09 | 04:30:00-05:00 | 0.414615 |
| 2012-03-10 | 04:30:00-05:00 | 0.427185 |
| 2012-03-11 | 05:30:00-04:00 | 1.172557 |
| 2012-03-12 | 05:30:00-04:00 | -0.351572 |
| 2012-03-13 | 05:30:00-04:00 | 1.454593 |
| 2012-03-14 | 05:30:00-04:00 | 2.043319 |

Freq: D

Trong trường hợp chuỗi thời gian ở trên, nằm giữa sự chuyển đổi DST ở múi giờ Miền Đông, chúng tôi có thể bản địa hóa thành Eastern và chuyển đổi sang UTC hoặc giờ Berlin:

```
ts_eastern = ts.tz_localize('US/Eastern')
ts_eastern.tz_convert('UTC')
```

Out:

| | | |
|---|---|---|
| 2012-03-09 | 14:30:00+00:00 | 0.414615 |
| 2012-03-10 | 14:30:00+00:00 | 0.427185 |
| 2012-03-11 | 13:30:00+00:00 | 1.172557 |
| 2012-03-12 | 13:30:00+00:00 | -0.351572 |
| 2012-03-13 | 13:30:00+00:00 | 1.454593 |
| 2012-03-14 | 13:30:00+00:00 | 2.043319 |

Freq: D

Và bên dưới là chuyển đổi từ múi giờ EST sang Berlin:

```
ts_eastern.tz_convert('Europe/Berlin')
```

Out:

| | | |
|---|---|---|
| 2012-03-09 | 15:30:00+01:00 | 0.414615 |
| 2012-03-10 | 15:30:00+01:00 | 0.427185 |
| 2012-03-11 | 14:30:00+01:00 | 1.172557 |
| 2012-03-12 | 14:30:00+01:00 | -0.351572 |
| 2012-03-13 | 14:30:00+01:00 | 1.454593 |
| 2012-03-14 | 14:30:00+01:00 | 2.043319 |

Freq: D

tz_localize và tz_convert đều là những phương thức thể hiện trên DatetimeIndex:

```
ts.index.tz_localize('Asia/Shanghai')
```

Out:

<class 'pandas.tseries.index.DatetimeIndex'>
[2012-03-09 09:30:00, ..., 2012-03-14 09:30:00]
Length: 6, Freq: D, Timezone: Asia/Shanghai

## 5.2 Operations with Time Zoneaware Timestamp Objects

Tương tự như time series và date ranges, các đối tượng Timestamp riêng lẻ tương tự có thể được bản địa hóa từ naive sang time zone-aware và chuyển đổi từ múi giờ này sang múi giờ khác:

```
stamp = pd.Timestamp('2011-03-12 04:00')
stamp_utc = stamp.tz_localize('utc')
stamp_utc.tz_convert('US/Eastern')
```

Out:

```
Timestamp('2011-03-11 23:00:00-0500', tz='America/New_York')
```

pd.Timestamp được sử dụng cho các mục tạo nên Datetime Index và các cấu trúc dữ liệu định hướng chuỗi thời gian khác trong pandas. stamp.tz_localize chuyển đổi naive Timestamp sang múi giờ địa phương UTC, Sau khi chuyển thành UTC, một lần nữa chuyển đổi về múi giờ Miền Đông Hoa Kỳ.

Bạn cũng có thể vượt qua một múi giờ khi tạo Timestamp:

```
stamp_moscow = pd.Timestamp('2011-03-12 04:00', tz='Europe/Moscow')
stamp_moscow
```

Out:

```
Timestamp: 2011-03-12 04:00:00+0300 MSK, tz=Europe/Moscow
```

Các đối tượng Timestamp nhận biết múi giờ lưu trữ nội bộ giá trị dấu thời gian UTC dưới dạng nano giây kể từ kỷ nguyên UNIX (ngày 1 tháng 1 năm 1970); giá trị UTC này là bất biến giữa các lần chuyển đổi múi giờ:

```
stamp_utc.value
```

Out

```
1299902400000000000
```

cho dù có chuyển đổi sang múi giờ Miền Đông Hoa Kỳ thì giá trị này cũng không thay đổi

```
stamp_utc.tz_convert('US/Eastern').value
```

Out

```
1299902400000000000
```

Khi thực hiện số học thời gian bằng các đối tượng DateOffset của Pandas, quá trình chuyển đổi thời gian tiết kiệm ánh sáng ban ngày được tuân thủ nếu có thể:

```
# 30 minutes before DST transition
from pandas.tseries.offsets import Hour
stamp = pd.Timestamp('2012-03-12 01:30', tz='US/Eastern')
stamp
```

Out:

```
Timestamp: 2012-03-12 01:30:00-0400 EDT, tz=US/Eastern
```

Thực hiện phép cộng số học với 1 giờ đồng hồ:

```
stamp + Hour()
```

Out:

```
Timestamp: 2012-03-12 02:30:00-0400 EDT, tz=US/Eastern
```

```
# 90 minutes before DST transition
stamp = pd.Timestamp('2012-11-04 00:30', tz='US/Eastern')
stamp
```

Out:

```
Timestamp: 2012-11-04 00:30:00-0400 EST, tz=US/Eastern
```

```
stamp + 2 * Hour()
```

Out:

```
Timestamp: 2012-03-12 01:30:00-0500 EDT, tz=US/Eastern
```

## 5.3 Operations between Different Time Zones

Nếu kết hợp hai chuỗi thời gian có múi giờ khác nhau, kết quả sẽ là UTC. Vì các timestamps được lưu trữ dưới dạng UTC nên đây là một thao tác đơn giản và không yêu cầu chuyển đổi xảy ra:

```
rng = pd.date_range('3/7/2012 9:30', periods=10, freq='B')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out:

| | | |
|---|---|---|
| 2012-03-07 | 09:30:00 | -1.749309 |
| 2012-03-08 | 09:30:00 | -0.387235 |
| 2012-03-09 | 09:30:00 | -0.208074 |
| 2012-03-12 | 09:30:00 | -1.221957 |
| 2012-03-13 | 09:30:00 | -0.067460 |
| 2012-03-14 | 09:30:00 | 0.229005 |
| 2012-03-15 | 09:30:00 | -0.576234 |
| 2012-03-16 | 09:30:00 | 0.816895 |
| 2012-03-19 | 09:30:00 | -0.772192 |
| 2012-03-20 | 09:30:00 | -1.333576 |

Freq: D

Hàm pd.sỉries Ndarray bất biến của dữ liệu datetime64, được biểu diễn bên trong dưới dạng int64 và có thể được đóng hộp cho các Timestamp objects là các lớp con của datetime và mang siêu dữ liệu như thông tin tần số.

```
ts1 = ts[:7].tz_localize('Europe/London')
ts2 = ts1[2:].tz_convert('Europe/Moscow')
result = ts1 + ts2
result.index
```

Out:

<class 'pandas.tseries.index.DatetimeIndex'>
[2012-03-07 09:30:00, ..., 2012-03-15 09:30:00]
Length: 7, Freq: B, Timezone: UTC

# 6 Periods and Period Arithmetic

Khoảng thời gian biểu thị thời gian như ngày, tháng, quý hoặc năm. Lớp Period cần được truyền vào 1 chuỗi hoặc số nguyên và tham số freq:

```
p = pd.Period('2007', freq='A-DEC')
p
```

Out:

Period('2007', 'A-DEC')

Trong trường hợp này, đối tượng Period biểu diễn toàn bộ khoảng thời gian từ 1/1/2007 đến 31/12/2007. Và cái việc này nó rất là tiện lợi cho việc cộng trừ số nguyên từ các period có tác dụng dịch chuyển theo tần số của chúng.

```
p + 5
```

Out: Period('2012', 'A-DEC')

```
p - 2
```

Out:

Period('2005', 'A-DEC')

Nếu hai chu kỳ có cùng tần số, sự khác nhau giữa chúng là số những đơn vị giữa chúng:

```
pd.Period('2014', freq='A-DEC') - p
```

Out:

7

Phạm vi chu kỳ thông thường có thể được tạo bằng cách sử dụng hàm period_range:

```
rng = pd.period\_range('1/1/2000', '6/30/2000', freq='M')
rng
```

Out:

```
<class 'pandas.tseries.period.PeriodIndex'>
freq: M
[2000-01, ..., 2000-06]
length: 6
```

Lớp PeriodIndex lưu trữ một chuỗi các khoảng thời gian và có thể phục vụ như chỉ số trục trong bất kỳ cấu trúc dữ liệu Pandas nào:

```
Series(np.random.randn(6), index=rng)
```

Out:

```
2000-01    -0.309119
2000-02     0.028558
2000-03     1.129605
2000-04    -0.374173
2000-05    -0.011401
2000-06     0.272924
Freq: M
```

Nếu bạn có một mảng các chuỗi, bạn cũng có thể kháng nghị chính lớp PeriodIndex:

```
values = ['2001Q3', '2002Q2', '2003Q1']
index = pd.PeriodIndex(values, freq='Q-DEC')
index
```

Out:

```
<class 'pandas.tseries.period.PeriodIndex'>
freq: Q-DEC
freq: Q-DEC
length: 3
```

## 6.1 Period Frequency Conversion

Các đối tượng Chu kỳ có thể được chuyển đổi sang tần số khác bằng cách sử dụng phương thức asfreq. Ví dụ: giả sử chúng tôi có một khoảng thời gian hàng năm và muốn chuyển đổi nó vào một khoảng thời gian hàng tháng vào đầu hoặc cuối năm. Đây là phường khá đơn giản:

```
p=pd.Period('2007', freq='A-DEC')
p.asfreq('M', how='start')
```

Out:

Period('2007-01', 'M')

```
p=pd.Period('2007', freq='A-DEC')
p.asfreq('M', how='end')
```

Out:

Period('2007-12', 'M')

Ta có thể thấy Period('2007', 'A-DEC') là một con trỏ trỏ đến khoảng thời gian được chia nhỏ cho hằng tháng. Đối với những năm kết thúc vào tháng mà không phải tháng 12, các kỳ , giai đoạn của tháng khác nhau:

```
p = pd.Period('2007', freq='A-JUN')
p.asfreq('M', 'start')
```

Out:

Period('2006-07', 'M')

```
p = pd.Period('2007', freq='A-JUN')
p.asfreq('M', 'end')
```

Out:

Period('2007-07', 'M')

Khi chuyển đổi từ tần số cao sang thấp, siêu chu kỳ sẽ được xác định tại nơi thuộc về các chu kỳ con. Ví dụ, vào chu kỳ A-JUN tháng 8 năm 2007 thực chất là một phần của năm 2008:

```
p = pd.Period('2007-08', 'M')
p.asfreq('A-JUN')
```

Out:

Period('2008', 'A-JUN')

Toàn bộ các đối tượng PeriodIndex hay TimeSeries có thể chuyển đổi cùng một cách:

```
rng = pd.period\_range('2006', '2009', freq='A-DEC')
ts = Series(np.random.randn(len(rng)), index=rng)
ts
```

Out:

```
2006  -0.601544
2007  0.574265
2008  0.194115
2009  0.202225
Freq: A-DEC
```

```
ts.asfreq('M', how='start')
```

Out:

```
2006-01    -0.601544
2007-01    0.574265
2008-01    -0.194115
2009-01     0.202225
Freq: M
```

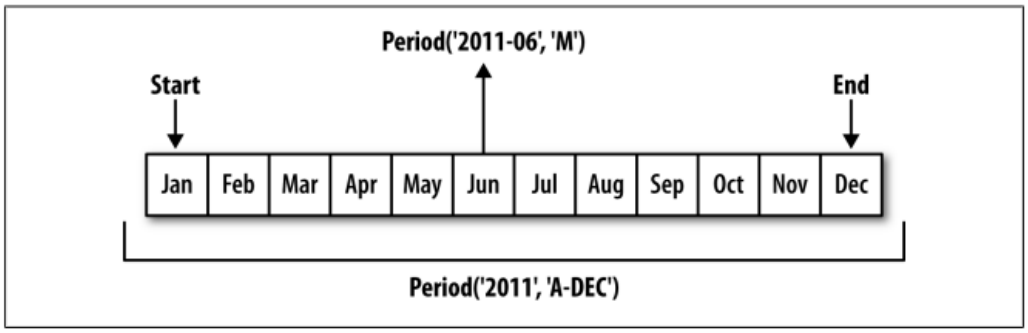


Figure 2: Hình minh họa chuyển đổi tần số chu kỳ

```
ts.asfreq('B', how='end')
```

Out:

| | |
|---|---|
| 2006-12-29 | -0.601544 |
| 2007-12-31 | 0.574265 |
| 2008-12-31 | -0.194115 |
| 2009-12-31 | 0.202225 |

Freq: B

## 6.2 Quarterly Period Frequencies

Dữ liệu hàng quý là tiêu chuẩn trong kế toán, tài chính và các lĩnh vực khác. Nhiều dữ liệu hàng quý được báo cáo liên quan đến ngày kết thúc năm tài chính, thường là ngày dương lịch hoặc ngày làm việc cuối cùng của một của 12 tháng trong năm. Như vậy, giai đoạn 2012 Q4 có một ý nghĩa khác khi chờ kết thúc năm tài chính. Pandas hỗ trợ tất cả 12 tần suất hàng quý có thể có từ Q JAN đến Q-DEC:

```
t p = pd.Period('2012Q4', freq='Q-JAN')
p
```

Out:

Period('2012Q4', 'Q-JAN')

Trong trường hợp năm tài chính kết thúc vào tháng 1, quý 4 năm 2012 kéo dài từ tháng 11 đến tháng 1, bạn có thể kiểm tra quý này bằng cách chuyển đổi sang tần suất hàng ngày. Xem Hình 10-2 để biết hình minh họa:

```
p.asfreq('D', 'start')
```

Out:

Period('2011-11-01', 'D')

```
p.asfreq('D', 'end')
```

Out:

Out[481]: Period('2012-01-31', 'D')

Do đó, có thể thực hiện phép tính thời gian rất dễ dàng; ví dụ: để lấy dấu thời gian lúc 4 giờ chiều ngày làm việc thứ 2 đến ngày làm việc cuối cùng của quý, bạn có thể thực hiện:

```
p4pm = (p.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60
p4pm
```

Out:

Period('2012-01-30 16:00', 'T')

```
p4pm.to_timestamp()
```

Out:

<Timestamp: 2012-01-30 16:00:00>

Tạo phạm vi hàng quý hoạt động như bạn mong đợi bằng cách sử dụng period_range.

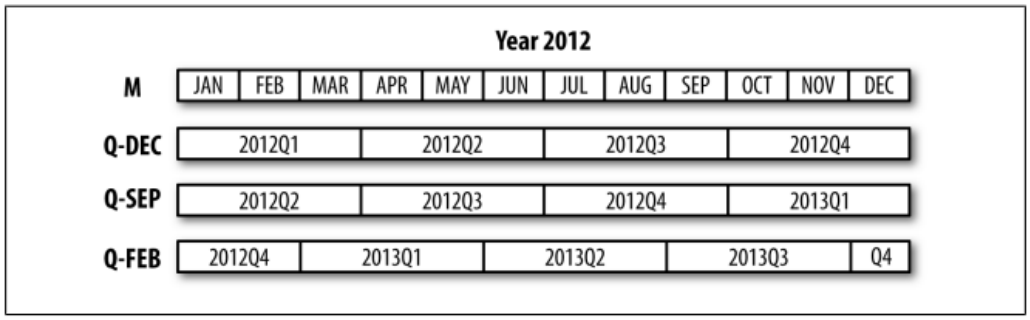


Figure 3: Các quy ước tần suất hàng quý khác nhau

```
rng = pd.period_range('2011Q3', '2012Q4', freq='Q-JAN')
ts = Series(np.arange(len(rng)), index=rng)
ts
```

Out:

```
2011Q3    0
2011Q4    1
2011Q1    2
2011Q2    3
2011Q3    4
2011Q4    5
Freq: Q-JAN
```

```
new_rng = (rng.asfreq('B', 'e') - 1).asfreq('T', 's') + 16 * 60
ts.index = new_rng.to_timestamp()
ts
```

Out:

```
2010-10-28 16:00:00    0
2010-01-28 16:00:00    1
2010-04-28 16:00:00    2
2010-07-28 16:00:00    3
2010-10-28 16:00:00    4
```

2010-01-30 16:00:00 5

## 6.3 Converting Timestamps to Periods (and Back)

Các đối tượng Series và DataFrame được lập chỉ mục theo dấu thời gian có thể được chuyển đổi thành các khoảng thời gian bằng cách sử dụng phương thức to_period:

```
rng = pd.date_range('1/1/2000', periods=3, freq='M')
ts = Series(randn(3), index=rng)
ts = ts.to\_period()
ts
```

Out:

2000-01-31 -0.505124
2000-02-29 2.954439
2000-03-31 -2.630247
Freq: M

```
rng = pd.date_range('1/1/2000', periods=3, freq='M')
ts = Series(randn(3), index=rng)
ts = ts.to\_period()
pts
```

Out:

2000-01 -0.505124
2000-02 2.954439
2000-03 -2.630247
Freq: M

Vì các khoảng thời gian luôn đề cập đến các khoảng thời gian không chồng chéo nên dấu thời gian chỉ có thể thuộc về đến một khoảng thời gian duy nhất cho một tần số nhất định. Trong khi tần suất của PeriodIndex mới là được suy ra từ dấu thời gian theo mặc định, bạn có thể chỉ định bất kỳ tần suất nào bạn muốn. Ở đó cũng không có vấn đề gì với việc có các

khoảng thời gian trùng lặp trong kết quả:

```
rng = pd.date_range('1/29/2000', periods=6, freq='D')
ts2 = Series(randn(6), index=rng)
ts2.to_period('M')
```

Out:

```
2000-01    -0.352453
2000-01    -0.477808
2000-01     0.161594
2000-01     1.686833
2000-01     0.821965
2000-01    -0.667406
Freq: M
```

Để chuyển đổi ngược lại, ta dùng hàm to_timestamp:

```
pts = ts.to_period()
pts
```

Out:

```
2000-01    -0.505124
2000-02     2.954439
2000-03    -2.630247
Freq: M
```

```
pts.to_timestamp(how='end')
```

Out:

```
2000-01-31    -0.505124
2000-02-29     2.954439
2000-03-31     -2.630247
Freq: M
```

## 6.4 Creating a PeriodIndex from Arrays

Các bộ dữ liệu tần số cố định đôi khi được lưu trữ với thông tin về khoảng thời gian trên nhiều cột. Ví dụ: trong bộ dữ liệu kinh tế vĩ mô này, năm và quý nằm trong các cột khác nhau:

```
data = pd.read_csv('ch08/macrodata.csv')
data.year
```

Out:

```
0 1959
1 1959
2 1959
3 1959
...
199 2008
200 2009
201 2009
202 2009
Name: year, Length: 203
```

```
data = pd.read_csv('ch08/macrodata.csv')
data.quarter
```

Out:

```
0 1
1 2
2 3
3 4
...
199 4
200 1
201 2
202 3
Name: quarter, Length: 203
```

Bằng cách chuyển các mảng này tới Period với tần suất, chúng có thể được kết hợp để tạo thành một chỉ mục cho DataFrame:

```
index = pd.PeriodIndex(year=data.year, quarter=data.quarter, freq='Q-DEC')
index
```

Out:

<class 'pandas.tseries.period.PeriodIndex'>
freq: Q-DEC
[1959Q1, ..., 2009Q3]
length: 203

```
data.index = index
data.infl
```

Out:

1959Q1 0.00
1959Q2 2.34
1959Q3 2.74
1959Q4 0.27
...
2008Q4 -8.79
2009Q1 0.94
2009Q2 3.37
2009Q3 3.56
Freq: Q-DEC, Name: infl, Length: 203

# 7 Resampling and Frequency Conversion

Lấy mẫu lại đề cập đến quá trình chuyển đổi một chuỗi thời gian từ tần số này sang tần số khác, Pandas cung cấp phương thức .resample() có rất nhiều tùy chọn có thể tùy chỉnh.Có 2 contexts cần chú ý :

+ Downsampling: Giảm tần suất lấy mẫu bằng cách tăng thời gian lấy mẫu từ vài phút đến vài giờ.
+ Upsampling: Tăng tần suất lấy mẫu bằng cách giảm thời gian lấy mẫu từ vài giờ xuống vài phút.
Cú pháp:

```
Series.resample(self, rule, how=None, axis=0, fill_method=None,
closed=None, label=None, convention='start', kind=None,
loffset=None, limit=None, base=0, on=None, level=None)

```

Ví dụ:

```
rng = pd.date_range('1/1/2000', periods=100, freq='D')
ts = Series(randn(len(rng)), index=rng)
ts.resample('M', how='mean')
```

Out:

```
2000-01-31    0.170876
2000-02-29    0.165020
2000-03-31    0.095451
2000-04-30    0.363566
Freq: M
```

```
ts.resample('M', how='mean', kind='period')
```

Out:

```
2000-01    0.170876
2000-02    0.165020
2000-03    0.095451
2000-04    0.363566
Freq: M
```

Resample là một phương pháp linh hoạt và hiệu suất cao có thể được sử dụng để xử lý chuỗi thời gian rất lớn.

## 7.1 Downsampling

Dữ liệu bạn đang tổng hợp không cần phải được sửa thường xuyên, tần số mong muốn xác định các cạnh bin được sử dụng để chia chuỗi thời gian thành nhiều phần để tổng hợp lại.

Ví dụ:Để chuyển đổi thành hàng tháng, 'M' hoặc 'BM', dữ liệu cần được chia nhỏ thành các khoảng thời gian một tháng. Mỗi khoảng được cho là nửa mở.

Một điểm dữ liệu chỉ có thể thuộc về một khoảng thời gian và sự kết hợp của các khoảng thời gian đó phải tạo nên toàn bộ khung thời gian.

Hai điều cần suy nghĩ khi sử dụng resample để giảm dữ liệu mẫu:

• Phía nào của mỗi khoảng được đóng lại

• Cách dán nhãn cho mỗi thùng tổng hợp, có thể bắt đầu khoảng thời gian hoặc kết thúc

Minh hoạ: hãy xem một số dữ liệu dài một phút:

```
1 rng = pd.date_range('1/1/2000', periods=12, freq='T')
2 ts = Series(np.arange(12), index=rng)
3 ts
```

Out:

```
2000-01-01    00:00:00    0

2000-01-01    00:01:00    1

2000-01-01    00:02:00    2

2000-01-01    00:03:00    3

2000-01-01    00:04:00    4

2000-01-01    00:05:00    5

2000-01-01    00:06:00    6

2000-01-01    00:07:00    7

2000-01-01    00:08:00    8
```

```
2000-01-01    00:09:00    9

2000-01-01    00:10:00    10

2000-01-01    00:11:00    11

Freq: T
```

Giả sử muốn tổng hợp dữ liệu này thành các phần dài năm phút hoặc các thanh bằng cách lấy tổng của mỗi nhóm:

```
ts.resample('5min', how='sum')
```

Out:

```
2000-01-01    00:00:00    0

2000-01-01    00:05:00    5

2000-01-01    00:10:00    40

2000-01-01    00:15:00    11

Freq: 5T
```

Theo mặc định, cạnh thùng bên phải được inclusive, vì vậy giá trị 00:05 được bao gồm trong khoảng thời gian từ 00:00 đến 00:05. 1 Passing closed= 'left' thay đổi khoảng thời gian được đóng ở bên trái:

```
ts.resample('5min', how='sum', closed = 'left')
```

Out:

```
2000-01-01    00:05:00    10

2000-01-01    00:10:00    35

2000-01-01    00:15:00    21

Freq: 5T
```

Kết quả chuỗi thời gian được gắn nhãn bằng dấu thời gian từ phía bên phải của mỗi thùng. Bằng cách chuyển label = 'left', có thể gắn nhãn chúng với cạnh thùng bên trái:

```
ts.resample('5min', how='sum', closed='left', label='left')
```

Out:

```
2000-01-01    00:05:00    10

2000-01-01    00:10:00    35

2000-01-01    00:15:00    21

Freq: 5T
```

Có thể thay đổi chỉ số kết quả một số lượng, chẳng hạn như trừ đi một giây từ cạnh bên phải để làm rõ hơn khoảng thời gian nào mà dấu thời gian đề cập đến. Để thực hiện việc này, chuyển một chuỗi hoặc bù ngày vào loffset:

```
ts.resample('5min', how='sum', loffset='-1s')
```

Out:

```
1999-12-31    23:59:59    0

2000-01-01    00:04:59    15

2000-01-01    00:09:59    40

2000-01-01    00:14:59    11

Freq: 5T
```

### 7.1.1 Open-High-Low-Close (OHLC) resampling

Trong tài chính, một cách phổ biến để tổng hợp chuỗi thời gian là tính toán bốn giá trị cho mỗi nhóm:

· Giá trị đầu tiên (mở)
· Cuối cùng (đóng)
· Tối đa (cao)
· Tối thiểu (thấp).

Bằng cách chuyển how = 'ohlc', sẽ nhận được DataFrame có các cột chứa bốn tổng hợp này, được tính toán hiệu quả trong một lần quét dữ liệu:

```
ts.resample('5min', how='ohlc')
```

Out:

| | open | high | low | close |
|---|---|---|---|---|
| 2000-01-01 00:00:00 | 0 | 0 | 0 | 0 |
| 2000-01-00 00:05:00 | 1 | 5 | 1 | 5 |
| 2000-01-00 00:10:00 | 6 | 10 | 6 | 10 |
| 2000-01-01 00:15:00 | 11 | 11 | 11 | 11 |

Freq: 5T

### 7.1.2 Resampling with GroupBy

Một cách thay thế để giảm mẫu là sử dụng chức năng phân nhóm Groupby có thể được sử dụng để nhóm một lượng lớn dữ liệu và tính toán các hoạt động trên các nhóm này..
Cú pháp:

```
DataFrame.groupby(by=None, axis=0, level=None, as_index=True,sort=True,
group_keys=_NoDefault.no_default, squeeze=_NoDefault.no_default,
observed=False, dropna=True)
```

Ví dụ: có thể nhóm theo tháng hoặc ngày trong tuần bằng cách chuyển một hàm truy cập các trường đó trên chỉ mục của chuỗi thời gian:

```
rng = pd.date\_range('1/1/2000', periods=100, freq='D')
ts = Series(np.arange(100), index=rng)
ts.groupby(lambda x: x.month).mean()
```

Out:

```
1  15
2  45
```

```
3  75
4  95
```

```
ts.groupby(lambda x: x.weekday).mean()
```

Out:

```
0  47.5
1  48.5
2  49.5
3  50.5
4  51.5
5  49.0
6  50.0
```

## 7.2 Upsampling and Interpolation

Khi chuyển đổi từ tần số thấp sang tần số cao hơn, không cần tổng hợp. Xem ví dụ sau:

```
frame = DataFrame(np.random.randn(2, 4),
index=pd.date_range('1/1/2000', periods=2, freq='W-WED'),
columns=['Colorado', 'Texas', 'New York', 'Ohio'])
frame
```

Out:

| | Colorado | Texas | New York | Ohio |
|---|---|---|---|---|
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-12 | -0.263206 | 1.141350 | -0.101937 | -0.07666 |

Khi lấy mẫu lại tần suất hàng ngày, các giá trị bị thiếu theo mặc định sẽ được đưa vào:

```
df_daily = frame.resample('D')
df_daily
```

Out:

| | Colorado | Texas | New York | Ohio |
|---|---|---|---|---|
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-06 | NaN | NaN | NaN | NaN |
| 2000-01-07 | NaN | NaN | NaN | NaN |
| 2000-01-08 | NaN | NaN | NaN | NaN |
| 2000-01-09 | NaN | NaN | NaN | NaN |
| 2000-01-10 | NaN | NaN | NaN | NaN |
| 2000-01-11 | NaN | NaN | NaN | NaN |
| 2000-01-12 | -0.263206 | 1.141350 | -0.101937 | -0.07666 |

Giả sử muốn chuyển tiếp từng giá trị hàng tuần vào các ngày không phải thứ Tư. Các phương pháp điền hoặc nội suy tương tự có sẵn trong các phương pháp điền và lập chỉ mục là có sẵn để lấy mẫu lại:

```
frame.resample('D', fill\_method='ffill')
```

Out:

| | Colorado | Texas | New York | Ohio |
|---|---|---|---|---|
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |

| | | | | |
|---|---|---|---|---|
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-05 | -0.609657 | -0.268837 | 0.195592 | 0.85979 |
| 2000-01-12 | -0.263206 | 1.141350 | -0.101937 | -0.07666 |

Tương tự, cũng có thể chọn chỉ điền vào một số khoảng thời gian nhất định về phía trước để giới hạn khoảng cách tiếp tục sử dụng một giá trị quan sát:

## 7.3 Resampling with Periods

Lấy mẫu lại dữ liệu được lập chỉ mục theo các khoảng thời gian khá đơn giản.
Việc lấy mẫu ngược có nhiều sắc thái hơn vì bạn phải đưa ra quyết định về điểm cuối của khoảng thời gian trong tần suất mới để đặt các giá trị trước khi lấy mẫu lại, giống như phương pháp asfreq. Đối số quy ước mặc định là 'end' nhưng cũng có thể là 'start':

```
1 annual_frame.resample('Q-DEC', fill_method='ffill')
```

Out:

| | Colorado | Texas | New York | Ohio |
|---|---|---|---|---|
| 2000Q0 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q1 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q2 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q3 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q4 | 0.158207 | 0.042967 | -0.360755 | 0.184687 |

```
annual_frame.resample('Q-MAR', fill_method='ffill')
```

Out:

| | Colorado | Texas | New York | Ohio |
|---|---|---|---|---|
| 2000Q1 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q2 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q3 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q4 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q1 | 0.158207 | 0.042967 | -0.360755 | 0.184687 |

Các quy tắc về lấy mẫu lên và lấy mẫu xuống:
• Trong lấy mẫu xuống, tần số đích phải là tần số con của tần số nguồn.
• Trong upsampling, tần số đích phải là một siêu nghiệm của tần số nguồn. Nếu các quy tắc này không được thỏa mãn, một ngoại lệ sẽ được đưa ra. Điều này chủ yếu ảnh hưởng đến tần suất hàng quý, hàng năm và hàng tuần; ví dụ: các khoảng thời gian được xác định bởi Q-MAR chỉ phù hợp với A-MAR, A-JUN, A-SEP và A-DEC:

```
annual_frame.resample('Q-DEC', fill_method='ffill', convention='start')
```

Out:

| | Colorado | Texas | New York | Ohio |
|---|---|---|---|---|
| 2000Q3 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q4 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q1 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q2 | 0.352070 | -0.553642 | 0.196642 | -0.094099 |
| 2001Q3 | 0.158207 | 0.042967 | -0.360755 | 0.184687 |

## 7.4 Time Series Plotting

Các biểu đồ của pandas time-series đã cải thiện định dạng ngày so với biểu đồ của matplotlib. Ví dụ: Chúng ta tải xuống một số dữ liệu về giá cổ phiếu của một số cổ phiếu phổ biến của Hoa Kỳ từ Yahoo! Finance:

```
close_px_all = pd.read_csv('ch09/stock_px.csv', parse_dates=True, index_col=0)
close_px = close_px_all[['AAPL', 'MSFT', 'XOM']]
close_px = close_px.resample('B', fill_method='ffill')
close_px
```

Out

```
<class'pandas.core.frame.DataFrame'>

DatetimeIndex: 2292 entries, 2003-01-02 00:00:00 to 2011-10-14 00:00:00

Freq: B

Data columns:
```

| | | |
|---|---|---|
| AAPL | 2292 | non-null values |
| MSFT | 2292 | non-null values |
| XOM | 2292 | non-null values |

```
dtypes: float64(3)
```

Việc gọi một trong các cột của biểu đồ trên sẽ tạo thành một biểu đồ đơn giản, thấy như trong Figure 4.

```
close_px['AAPL'].plot()
```

Khi được gọi trên DataFrame, tất cả các time-series được vẽ trên một ô con duy nhất với chú thích cho biết đó là ô nào. Sơ đồ dữ liệu năm 2009 giúp bạn có thể thấy cách mà cả tháng và năm được định dạng trên trục X, xem Figure 5.

```
close_px.ix['2009'].plot()
```

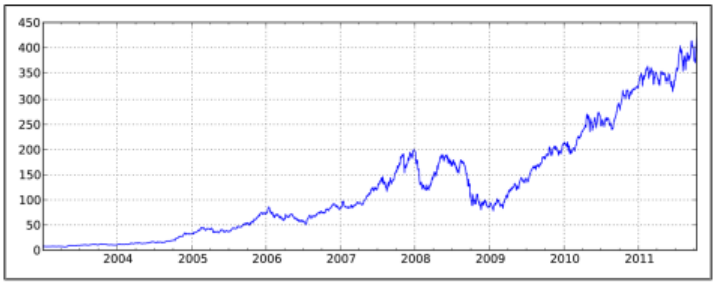


Figure 4: AAPL Daily Price

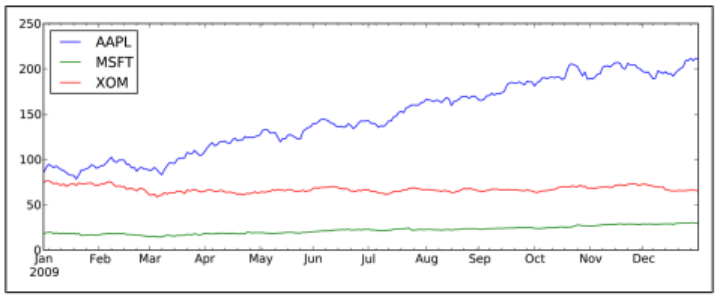


Figure 5: Stock Prices in 2009

```
close_px['AAPL'].ix['01-2011':'03-2011'].plot()
```

Dữ liệu tần suất hàng quý cũng được định dạng tốt hơn với điểm đánh dấu hàng quý, việc mà sẽ tốn nhiều thời gian nếu ta làm thủ công. Xem Figure 7.

```
appl_q = close_px['AAPL'].resample('Q-DEC', fill_method='ffill')
appl_q.ix['2009':].plot()
```

Một tính năng cuối cùng của biểu đồ time-series trong pandas là bằng cách nhấp chuột phải và kéo để phóng to và thu nhỏ, ngày tháng sẽ được mở

rộng hoặc thu nhỏ và định dạng lại tùy thuộc vào khoảng thời gian có trong chế độ xem biểu đồ. Với matplotlib, điều này chỉ đúng khi sử dụng ở chế độ tương tác.

# 8 Moving Window Functions

Một lớp phổ biến của phép biến đổi mảng cho các phép toán time-series là thống kê, các hàm khác được đánh giá qua sliding window ( cửa sổ trượt ) hoặc đánh giá với trọng số giảm dần theo cấp số nhân. Ta gọi đây là *moving window functions*. Giống như các chức năng thống kê khác, các chức năng này cũng tự động loại trừ các dữ liệu bị thiếu.

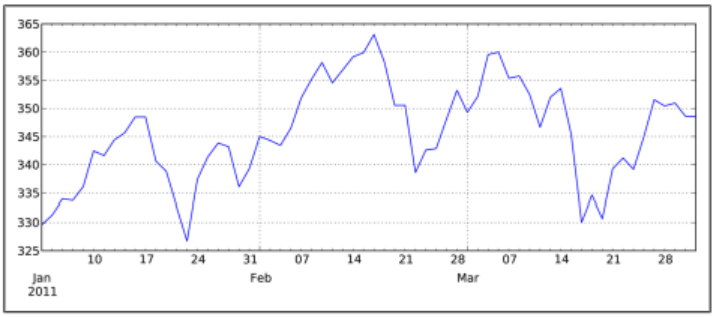


Figure 6: Apple Daily Price in 1/2011-3/2011

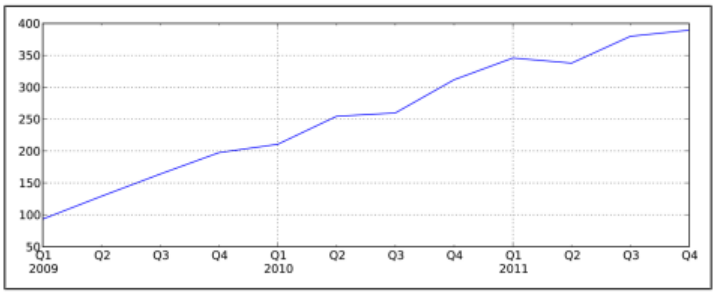


Figure 7: Apple Quarterly Price 2009-2011

*rolling_mean* chỉ đơn giản là giá trị trung bình của một số khoảng thời gian nhất định trước đó trong một time-series, mất một TimeSeries hoặc

DataFrame cùng với một window để biểu thị một số Khoảng thời gian:

```
close_px.AAPL.plot()
<matplotlib.axes.AxesSubplot at 0x1099b3990>
pd.rolling_mean(close_px.AAPL, 250).plot()
```

Xem Figure 8 . Theo mặc định, các hàm như rolling_mean yêu cầu số lượng non-NA được chỉ định. Hoạt động này có thể được dùng để giải thích cho dữ liệu bị thiếu, thực tế là bạn sẽ có ít hơn window periods của dữ liệu ở đầu time-series (xem figure 9):

```
appl_std250 = pd.rolling_std(close_px.AAPL, 250, min_periods=10)
appl_std250[5:12]
```

Out

| | |
|---|---|
| 2003-01-09 | NaN |
| 2003-01-10 | NaN |
| 2003-01-13 | NaN |
| 2003-01-14 | NaN |
| 2003-01-15 | 0.077496 |
| 2003-01-16 | 0.074760 |
| 2003-01-17 | 0.112368 |

Freq: B

```
appl_std250.plot()
```

Để tính giá trị *expanding window meanư*, bạn có thể thấy rằng expanding window (cửa sổ mở rộng ) chỉ là một trường hợp đặc biệt trong đó window là độ dài của time-series, nhưng chỉ cần một hoặc nhiều periods để tính giá trị:

```
# Define expanding mean in terms of rolling_mean
expanding_mean = lambda x: rolling_mean(x, len(x), min_periods=1)
```

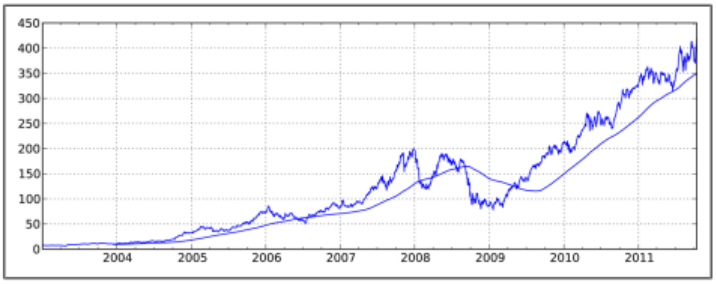


Figure 8: Apple Price with 250-day MA

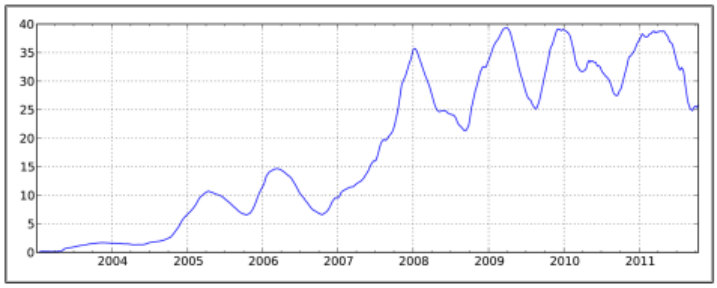


Figure 9: Apple 250-day daily return standard deviation

Việc gọi *rolling_mean* và friends trên DataFrame sẽ áp dụng phép biến đổi cho từng cột (xem Figure 10):

```
1 pd.rolling_mean(close_px, 60).plot(logy=True)
```

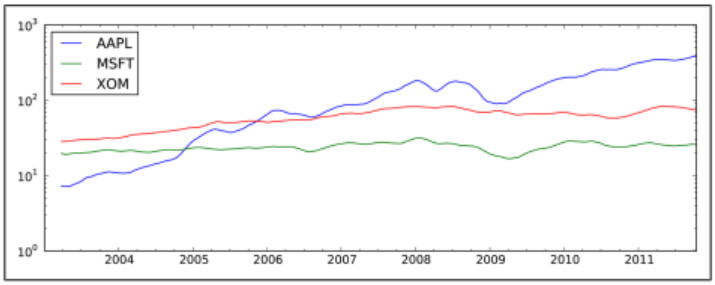


Figure 10: Stocks Prices 60-day MA (log Y-axis)

Xem bảng bên dưới để biết danh sách các chức năng liên quan của pandas.

| Function | Description |
|---|---|
| rolling_count | Trả về số lượng non-NA trong mỗi trailing window |
| rolling_sum | Trả về tổng trên window |
| rolling_mean | Trả về trung bình trên window |
| rollingl_median | Trả về trung vị trên window |
| rolling_var, Roll_std | Trả về phương sai và độ lệch chuẩn của window . Sử dụng n - 1 mẫu số |
| rolling_skew, rolling_kurt | Độ lệch của window ((3rd moment) và kurtosis (4th moment) |
| rolling_min, rolling_max | Tối thiểu và tối đa của Moving window |
| rolling_quantile | Điểm ở phân vị / lượng tử mẫu của moving window |
| rolling_corr, rolling_cov | Tương quan và hiệp phương sai của moving window. |
| rollingl_apply | Áp dụng hàm mảng chung trên một moving window |
| ewma | Đường trung bình động có trọng số theo cấp số nhân |
| ewmvar, ewmstd | Phương sai và độ lệch chuẩn moving có trọng số theo cấp số nhân |
| ewmcorr, ewmcov | Tương quan và hiệp phương sai moving có trọng số theo cấp số nhân. |

## 8.1 Exponentially-weighted functions

Một giải pháp thay thế cho việc sử dụng kích thước window tĩnh với các observations có trọng số bằng nhau là chỉ định *decay factor* (hệ số phân rã) không đổi để tạo thêm trọng lượng cho các observations gần đây hơn. Theo thuật ngữ toán học, nếu mat là kết quả trung bình động tại thời điểm t và x là chuỗi thời gian được đề cập, thì mỗi giá trị trong kết quả được tính là mat = a * mat - 1 + (a - 1) * x_t, trong đó a là *decay factor*. Có một số cách để chỉ định *decay factor*, một cách phổ biến là sử dụng một*span*, làm cho kết quả có thể so sánh với một hàm moving window đơn giản với kích thước window bằng span.

Vì một thống kê có trọng số theo cấp số nhân đặt nhiều trọng lượng hơn

vào các observations gần đây, nên nó "thích ứng" nhanh hơn với những thay đổi so với phiên bản có trọng số bằng nhau. Dưới đây là một ví dụ so sánh đường trung bình động trong 60 ngày của giá cổ phiếu Apple với đường trung bình động EW với span = 60 (xem Figure 11):

```
fig, axes = plt.subplots(nrows=2, ncols=1, sharex=True, sharey=True,figsize=(12, 7))
aapl_px = close_px.AAPL['2005':'2009']
ma60 = pd.rolling_mean(aapl_px, 60, min_periods=50) ewma60 = pd.ewma(aapl_px, span=60)
aapl_px.plot(style='k-', ax=axes[0])
ma60.plot(style='k--', ax=axes[0])
aapl_px.plot(style='k-', ax=axes[1])
ewma60.plot(style='k--', ax=axes[1])
axes[0].set_title('Simple MA')
axes[1].set_title('Exponentially-weighted MA')
```

## 8.2 Binary Moving Window Functions

Một số toán tử thống kê, như tương quan và hiệp phương sai, cần hoạt động trên hai time-series. Ví dụ: các nhà phân tích tài chính thường quan tâm đến mối tương quan của cổ phiếu với chỉ số chuẩn như SP 500. Chúng ta có thể tính toán điều đó bằng cách tính toán phần trăm thay đổi và sử dụng *rolling_ corr* (xem Figure 12):

```
spx_rets = spx_px / spx_px.shift(1) - 1
returns = close_px.pct_change()
corr = pd.rolling_corr(returns.AAPL, spx_rets, 125, min_periods=100)
corr.plot()
```

Hàm *pct_ change()* được dùng để tính toán phần trăm thay đổi giữa phần tử hiện tại và phần tử trước đó . Theo mặc định, hàm này tính toán phần trăm thay đổi so với hàng ngay trước đó ( Chức năng này chủ yếu hữu ích trong dữ liệu Time-Series).

Giả sử bạn muốn tính toán mối tương quan của chỉ số SP 500 với nhiều cổ phiếu cùng một lúc. Viết một vòng lặp và tạo một DataFrame mới sẽ dễ dàng nhưng có thể lặp lại, vì vậy nếu bạn chuyển một TimeSeries và một

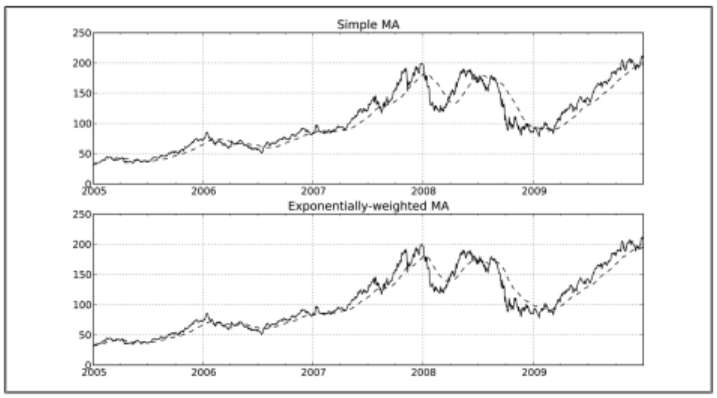


Figure 11: Simple moving average versus exponentially-weighted

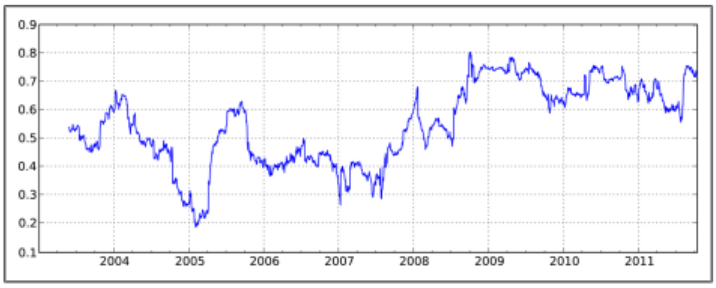


Figure 12: Six-month AAPL return correlation to SP 500

DataFrame, một hàm như *rolling_corr* sẽ tính toán mối tương quan của TimeSeries (trong trường hợp này là *spx_rets*) với mỗi cột trong DataFrame. Xem Figure 13 để biết kết quả:

```
1 corr = pd.rolling_corr(returns, spx_rets, 125, min_periods=100)
2 corr.plot()
```

## 8.3 User-Defined Moving Window Functions

Hàm *rolling_apply* cung cấp một phương tiện để áp dụng một hàm mảng do chính bạn tạo ra trên một moving window. Yêu cầu duy nhất là hàm tạo ra một giá trị duy nhất (giảm) từ mỗi phần của mảng. Ví dụ: trong khi chúng ta có thể tính toán lượng tử mẫu bằng cách sử dụng*rolling_quantile*, chúng

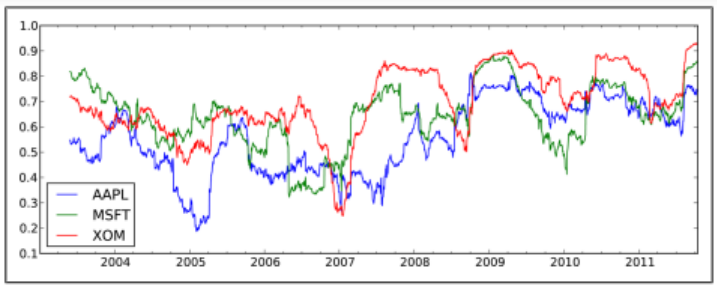


Figure 13: Six-month return correlations to SP 500

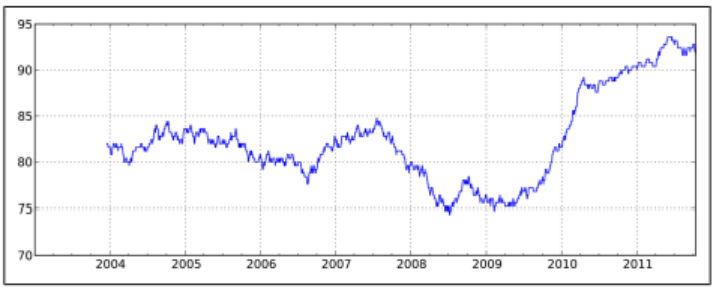


Figure 14: Percentile rank of 2% AAPL return over 1 year window

ta có thể quan tâm đến thứ hạng phần trăm của một giá trị cụ thể trên mẫu. Hàm *scipy.stats.percentileofscore* thực hiện điều này:

```
from scipy.stats import percentileofscore
score_at_2percent = lambda x: percentileofscore(x, 0.02)
result = pd.rolling_apply(returns.AAPL, 250, score_at_2percent)
result.plot()
```

# 9 Performance and Memory Usage Notes

Timestamps (Dấu thời gian) và periods (khoảng thời gian) được biểu thị dưới dạng số nguyên 64-bit sử dụng loại ngày time64 của NumPy. Điều này có nghĩa là đối với mỗi điểm dữ liệu, có 8 byte bộ nhớ được liên kết trên mỗi Timestamps. Do đó, một time-series với 1 triệu điểm dữ liệu float64 có

dung lượng bộ nhớ xấp xỉ 16 megabyte. Vì pandas cố gắng hết sức để chia sẻ các index giữa các time-series, nên việc tạo các chế độ xem trên time-series hiện có sẽ không gây ra bất kỳ bộ nhớ nào được sử dụng nữa. Ngoài ra, các indexes cho tần số thấp hơn (hàng ngày trở lên) được lưu trữ trong bộ đệm trung tâm, do đó, bất kỳ index tần số cố định nào đều là chế độ xem trên bộ đệm ngày. Do đó, nếu bạn có một bộ sưu tập lớn các time-series tần số thấp, vùng nhớ của các index sẽ không đáng kể.

Về hiệu suất, pandas đã được tối ưu hóa cao cho các hoạt động căn chỉnh dữ liệu (công việc hậu trường của ts1 + ts2 được lập index khác nhau) và lấy mẫu lại. Dưới đây là một ví dụ về tổng hợp các điểm dữ liệu 10MM thành OHLC:

```
rng = pd.date_range('1/1/2000', periods=10000000, freq='10ms')
ts = Series(np.random.randn(len(rng)), index=rng)
ts
```

Out

```
2000-01-01  00:00:00           -1.402235
2000-01-01  00:00:00.010000     2.424667
2000-01-01  00:00:00.020000    -1.956042
2000-01-01  00:00:00.030000    -0.897339
2000-01-02  03:46:39.960000     0.495530
  ...
  ...
2000-01-02  03:46:39.960000    0.495530
2000-01-02  03:46:39.970000    0.574766
2000-01-02  03:46:39.980000    1.348374

Freq: 10L, Length: 10000000
```

```
ts.resample('15min', how='ohlc')
```

Out

```
<class'pandas.core.frame.DataFrame'>

DatetimeIndex: 113 entries, 2000-01-01 00:00:00 to 2000-01-02 04:00:00
```

```
Freq: 15T

Data columns:

    open 113   non-null   values
    high 113   non-null   values
     low 113   non-null   values
   close 113   non-null   values

dtypes: float64(4)
```

```
%timeit ts.resample('15min', how='ohlc') 10 loops, best of 3: 61.1 ms per loop
```

Thời gian chạy có thể phụ thuộc một chút vào kích thước tương đối của kết quả tổng hợp; không ngạc nhiên khi tổng hợp tần số cao hơn mất nhiều thời gian hơn để tính toán:

```
rng = pd.date_range('1/1/2000', periods=10000000, freq='1s')
ts = Series(np.random.randn(len(rng)), index=rng)
%timeit ts.resample('15s', how='ohlc') 1 loops, best of 3: 88.2 ms per loop
```

# 10 Conlusion

Như vậy chúng ta đã tìm hiểu các ứng dụng và sử dụng các kĩ thuật ứng dụng vào time series. Cùng với các thư viện cơ bản của python và pandas có thể dễ dàng *slice, aggregate, resample, fixed frequency*. Hy vọng rằng nó có thể giúp bạn ứng dụng vào các lĩnh vực thực tế của như tài chính, kinh tế. Để xem phần lập trình có thể truy cập đường github (https://github.com/Vo-Linh/FDS_TimeSeries)