**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP HỒ CHÍ MINH**

**KHOA ĐIỆN ĐIỆN TỬ**


# BÁO CÁO MÔN HỌC

# CƠ SỞ KHOA HỌC DỮ LIỆU

***ĐỀ TÀI***

# FINANCIAL AND ECONOMIC DATA APPLICATIONS

**GV hướng dẫn: NGUYỄN MẠNH HÙNG**

**Nhóm sinh viên thực hiện:**

| *Họ và tên* | *MSSV* | *Mã lớp* |
|---|---|---|
| **Nguyễn Minh Tuấn** | **20139095** | **20139** |
| **Nguyễn Hữu Thiết** | **20139051** | **20139** |
| **Lê Tấn Kiên** | **20139026** | **20139** |
| **Nguyễn Vũ Tú** | **20139094** | **20139** |
| **Phan Tấn Quốc** | **20139086** | **20139** |

**Tp Hồ Chí Minh, 2022**

# Mục lục

CHAPTER 11

# Financial and Economic Data Applications

Sử dụng Python trong ngành tài chính tăng nhanh một cách đáng kể từ năm 2005, chủ yếu là do sự lớn mạnh của các thư viện (như Numpy và Pandas) và kĩ năng sẵn có của các lập trình viên Python chuyên nghiệp. Các tổ chức, cơ quan đã nhận thấy rằng Python rất phù hợp với cả vai trò là một môi trường phân tích tương tác cũng như cho phép các hệ thống mạnh mẽ được phát triển thường xuyên trong một khoảng thời gian ngắn so với Java hoặc C++. Python cũng là một ngôn ngữ lý tưởng để xây dựng giao diện Python cho các thư viện kế thừa được xây dựng trong C hoặc C++.

Trong những ví dụ này, tôi sẽ sử dụng thuật ngữ cross-section để chỉ dữ liệu tại một thời điểm cố định. Ví dụ, giá đóng cửa của toàn bộ cổ phiếu của chỉ số S&P 500 vào một ngày cụ thể tạo thành một cross-section. Dữ liệu cross-sectional tại nhiều thời điểm trên nhiều mục dữ liệu (ví dụ như giá cả với khối lượng ) tạo thành một bảng điều khiển. Dữ liệu bảng điều khiển có thành tạo thành bảng ở dưới dạng DataFrame hoặc sử dụng đối tượng pandas Panel ba chiều.

## Data Munging Topics

Nhiều công cụ tổng hợp dữ liệu hữu ích cho các ứng dụng tài chính được trải rộng trong các chương trước. Ở đây tôi sẽ nêu nổi bật một số chủ đề vì chúng liên quan đến đề này.

## 1.1 Time Series and Cross-Section Alignment

Một trong những vấn đề tốn nhiều thời gian nhất khi làm việc với dữ liệu tài chính là vấn đề liên kết dữ liệu .Hai chuỗi thời gian liên quan có thể có các chỉ số không phù hợp với nhau, hoặc hai đối tượng DataFrame có thể có các cột hoặc nhãn hàng không khớp. Người dùng MATLAB, R và các ngôn ngữ lập trình ma trận khác thường đầu tư công sức nỗ lực một cách đáng kể vào việc gói dữ liệu thành các dạng được căn chỉnh hoàn hảo. Với kinh nghiệm, phải căn chỉnh dữ liệu bằng tay và tệ hơn là phải xác minh rằng dữ liệu được căn chỉnh) là một cách cứng nhắc và nhàm chán . Nó cũng tiềm ẩn rất nhiều lỗi do kết hợp dữ liệu bị lệch không đồng nhất với nhau.

Pandas thực hiện một cách tiếp cận thay thế bằng cách tự động căn chỉnh dữ liệu trong các phép toán số học. Trên thực tế, điều này mang lại sự tự do to lớn và nâng cao năng suất của bạn. Ví dụ: hãy xem xét một vài DataFrames chứa chuỗi thời gian về giá và khối lượng cổ phiếu

In [16]: prices

Out [16]:

| | AAPL | JNJ | SPX | XOM |
|---|---|---|---|---|
| 2011-09-06 | 379.74 | 64.64 | 1165.24 | 71.15 |
| 2011-09-07 | 383.93 | 65.43 | 1198.62 | 73.65 |
| 2011-09-08 | 384.14 | 64.95 | 1185.90 | 72.82 |
| 2011-09-09 | 377.48 | 63.64 | 1154.23 | 71.01 |
| 2011-09-12 | 379.94 | 63.59 | 1162.27 | 71.84 |
| 2011-09-13 | 384.62 | 63.61 | 1172.87 | 71.65 |
| 2011-09-14 | 389.30 | 63.73 | 1188.68 | 72.64 |

In [17]: volume

Out [17]:

| | AAPL | JNJ | XOM |
|---|---|---|---|
| 2011-09-06 | 18173500 | 15848300 | 25416300 |
| 2011-09-07 | 12492000 | 10759700 | 23108400 |
| 2011-09-08 | 14839800 | 15551500 | 22434800 |
| 2011-09-09 | 20171900 | 17008200 | 27969100 |
| 2011-09-12 | 16697300 | 13448200 | 26205800 |

Giả sử bạn muốn tính giá trung bình theo khối lượng bằng cách sử dụng tất cả dữ liệu có sẵn( và đưa ra giả định đơn giản hóa rằng dữ liệu khối lượng là một tập hợp con của dữ liệu giá).Vì pandas tự động căn chỉnh dữ liệu theo số học và loại trừ dữ liệu bị thiếu trong các hàm như sum, chúng ta có thể diễn đạt điều này một cách ngắn gọn như:

In [18]: prices * volume

Out [18]:

| | AAPL | JNJ | SPX | XOM |
|---|---|---|---|---|
| 2011-09-06 | 6.901205e+09 | 1.024434e+09 | NaN | 1.808370e+09 |
| 2011-09-07 | 4.796054e+09 | 7.040072e+08 | NaN | 1.701934e+09 |
| 2011-09-08 | 5.700561e+09 | 1.010070e+09 | NaN | 1.633702e+09 |
| 2011-09-09 | 7.614489e+09 | 1.082402e+09 | NaN | 1.986086e+09 |
| 2011-09-12 | 6.343972e+09 | 8.551710e+08 | NaN | 1.882625e+09 |
| 2011-09-13 | NaN | NaN | NaN | NaN |
| 2011-09-14 | NaN | NaN | NaN | NaN |

In [19]: vwap = (prices * volume).sum() / volume.sum()

In [20]: vwap

Out [20]:

| | |
|---|---|
| AAPL | 380.655181 |
| JNJ | 64.394769 |
| SPX | NaN |
| XOM | 72.024288 |

In [21]: vwap.dropna()

Out [21]:

| | |
|---|---|
| AAPL | 380.655181 |
| JNJ | 64.394769 |
| XOM | 72.024288 |

Vì không tìm thấy SPX trong số lượng, bạn có thể chọn loại bỏ nó bất kỳ lúc nào. Nếu bạn muốn căn chỉnh bằng tay, bạn có thể sử dụng phương pháp căn chỉnh của DataFrame, trả về một loạt các phiên bản được lập chỉ mục lại của hai đối tượng:

In [22]: prices.align(volume, join='inner')

Out [22]:

| ( | AAPL | JNJ | XOM |
|---|---|---|---|
| 2011-09-06 | 379.74 | 64.64 | 71.15 |
| 2011-09-07 | 383.93 | 65.43 | 73.65 |
| 2011-09-08 | 384.14 | 64.95 | 72.82 |
| 2011-09-09 | 377.48 | 63.64 | 71.01 |
| 2011-09-12 | 379.94 | 63.59 | 71.84, |
| | AAPL | JNJ | XOM |
| 2011-09-06 | 18173500 | 15848300 | 25416300 |
| 2011-09-07 | 12492000 | 10759700 | 23108400 |
| 2011-09-08 | 14839800 | 15551500 | 22434800 |
| 2011-09-09 | 20171900 | 17008200 | 27969100 |
| 2011-09-12 | 16697300 | 13448200 | 26205800) |

Một tính năng không thể thiếu khác là xây dựng DataFrame từ một tập hợp các Series được lập chỉ mục khác nhau:

In [23]: s1 = Series(range(3), index=['a', 'b', 'c'])

In [24] s2 = pd.Series(range(4), index=['d', 'b', 'c', 'e'])

In [25]: s3 = Series(range(3), index=['f', 'a', 'c'])

In [26]: pd.DataFrame('one': s1, 'two': s2, 'three': s3)

Out [26]:

| | one | two | three |
|---|---|---|---|
| a | 0.0 | NaN | 1.0 |
| b | 1.0 | 1.0 | NaN |
| c | 2.0 | 2.0 | 2.0 |
| d | NaN | 0.0 | NaN |
| e | NaN | 3.0 | NaN |
| f | NaN | NaN | 0.0 |

Như bạn đã thấy trước đó, tất nhiên bạn có thể chỉ định rõ ràng chỉ mục của kết quả, loại bỏ phần còn lại của dữ liệu:

In [27]: pd.DataFrame('one': s1, 'two': s2, 'three': s3, index=list('face'))

Out [27]:

| | one | two | three |
|---|---|---|---|
| f | NaN | NaN | 0.0 |
| a | 0.0 | NaN | 1.0 |
| c | 2.0 | 2.0 | 2.0 |
| e | NaN | 3.0 | NaN |

## 1.2 Operations with Time Series of Different Frequencies

Chuỗi thời gian kinh tế thường có tần suất hàng năm, hàng quý, hàng tháng, hàng ngày hoặc một số tần suất chuyên biệt hơn. Một số là hoàn toàn không đều; ví dụ, các bản sửa đổi thu nhập cho một cổ phiếu có thể đến bất kỳ lúc nào. Hai công cụ chính để chuyển đổi tần số và sắp xếp lại là các phương pháp resample và reindex. Với resample giúp chuyển đổi dữ liệu thành tần suất cố định trong khi reindex phù hợp dữ liệu với một chỉ số mới. Cả hai đều hỗ trợ logic nội suy tùy chọn (chẳng hạn như điền chuyển tiếp).

Hãy xem xét một chuỗi thời gian nhỏ hàng tuần:

In [28]: ts1 = pd.Series(np.random.randn(3),

index=pd.date_range('2012-6-13', periods=3, freq='W-WED'))

In [29]: ts1

Out [29]:

2012-06-13 -0.871903

2012-06-20 0.519593

2012-06-27 -0.413622

Nếu bạn lấy lại mẫu cho tần suất làm việc hàng ngày (Thứ Hai-Thứ Sáu), bạn sẽ gặp vấn đề vào những ngày không có dữ liệu

In [30]: ts1.resample('B').mean()

Out [30]:

2012-06-13 -0.871903

2012-06-14 NaN

2012-06-15 NaN

2012-06-18 NaN

2012-06-19 NaN

2012-06-20 0.519593

2012-06-21 NaN

2012-06-22 NaN

2012-06-25 NaN

2012-06-26 NaN

2012-06-27 -0.413622

Tất nhiên, việc sử dụng 'ffill' như phương thức fillna sẽ điền các giá trị vào những khoảng trống đó. Đây là một thực tế phổ biến với dữ liệu tần suất thấp hơn khi bạn tính toán một chuỗi thời gian của các giá trị trên mỗi dấu thời gian có giá trị hợp lệ hoặc "tính đến thời điểm" mới nhất.

In [31]: ts1.resample('B').mean().fillna(method='ffill')

Out [31]:

2012-06-13 -0.871903

2012-06-14 -0.871903

2012-06-15 -0.871903

2012-06-18 -0.871903

2012-06-19 -0.871903

2012-06-20 0.519593

2012-06-21 0.519593

2012-06-22 0.519593

2012-06-25 0.519593

2012-06-26 0.519593

2012-06-27 -0.413622

Trên thực tế, việc lấy mẫu dữ liệu tần số thấp hơn lên một tần số cao hơn, thường xuyên là một giải pháp tốt, nhưng trong trường hợp chuỗi thời gian không thường xuyên nói chung, nó có thể không phù hợp. Hãy xem xét một chuỗi thời gian được lấy mẫu bất thường từ cùng một khoảng thời gian chung:

In [32]: dates = pd.DatetimeIndex([’2012-6-12’, ’2012-6-17’, ’2012-6-18’,

’2012-6-21’, ’2012-6-22’, ’2012-6-29’])

In [33]: ts2 = pd.Series(np.random.randn(6), index=dates)

In [34]: ts2

Out [34]:

2012-06-12 1.669025

2012-06-17 -0.438570

2012-06-18 -0.539741

2012-06-21 0.476985

2012-06-22 3.248944

2012-06-29 -1.021228

Nếu bạn muốn thêm các giá trị "tính đến thời điểm" trong ts1 forward filling) vào ts2. Một tùy chọn sẽ là lấy mẫu lại cả hai thành tần suất thông thường sau đó thêm vào, nhưng nếu bạn muốn duy trì chỉ số ngày trong ts2, sử dụng reindex là một giải pháp chính xác

hơn:

```
In [35]: ts1.reindex(ts2.index, method='ffill')
Out [35]:

2012-06-12         NaN
2012-06-17   -0.871903
2012-06-18   -0.871903
2012-06-21    0.519593
2012-06-22    0.519593
2012-06-29   -0.413622
In [36]: ts2 + ts1.reindex(ts2.index, method='ffill')
Out [36]:

2012-06-12         NaN
2012-06-17   -2.759194
2012-06-18    0.666666
2012-06-21    0.671619
2012-06-22    1.003184
2012-06-29   -1.556996
```

**Using periods instead of timestamps**

Các khoảng thời gian (đại diện cho các khoảng thời gian) cung cấp một phương tiện thay thế để làm việc với các chuỗi thời gian tần suất khác nhau, đặc biệt là các chuỗi thời gian tài chính hoặc kinh tế với tần suất hàng năm hoặc hàng quý có quy ước báo cáo cụ thể. Ví dụ: một công ty có thể công bố thu nhập hàng quý của mình với năm tài chính kết thúc vào tháng 6, do đó có thời hạn miễn phí Q-JUN. Xem xét một cặp chuỗi thời gian kinh tế vĩ mô liên quan đến GDP và lạm phát:

```
In [37]: gdp = pd.Series([1.78, 1.94, 2.08, 2.01, 2.15, 2.31, 2.46],
                         index=pd.period_range('1984Q2', periods=7, freq='Q-SEP'))
In [38]: infl = pd.Series([0.025, 0.045, 0.037, 0.04],
                          index = pd.period_range('1982', periods=4, freq='A-DEC'))
```

```
In [39]: gdp
Out [39]:
1984Q2    1.78
1984Q3    1.94
1984Q4    2.08
1985Q1    2.01
1985Q2    2.15
1985Q3    2.31
1985Q4    2.46
Freq: Q-SEP
```

```
In [40]: infl
Out [40]:
1982    0.025
1983    0.045
1984    0.037
1985    0.040
Freq: A-DEC
```

Không giống như chuỗi thời gian có dấu thời gian, các hoạt động giữa các chuỗi thời gian có tần suất khác nhau được lập chỉ số theo các khoảng thời gian không thể thực hiện được nếu không có các chuyển đổi rõ ràng. Trong trường hợp này, nếu chúng ta biết rằng các giá trị lạm phát đã được quan sát vào cuối mỗi năm, thì chúng ta có thể chuyển đổi sang Q-SEP để có được các khoảng thời gian phù hợp với tần suất đó:

```
In [41]: infl_q = infl.asfreq('Q-SEP', how='end')
In [42]: infl_q
Out [42]:
1983Q1    0.025
1984Q1    0.045
1985Q1    0.037
1986Q1    0.040
Freq: Q-SEP
```

Chuỗi thời gian đó sau đó có thể được lập chỉ mục lại với tính năng điền chuyển tiếp để khớp với gdp:

In [43]: infl_q.reindex(gdp.index, method='ffill')

Out [43]:

| | |
|---|---|
| 1984Q2 | 0.045 |
| 1984Q3 | 0.045 |
| 1984Q4 | 0.045 |
| 1985Q1 | 0.037 |
| 1985Q2 | 0.037 |
| 1985Q3 | 0.037 |
| 1985Q4 | 0.037 |

Freq: Q-SEP

## 1.3 Time of Day and "as of" Data Selection

Giả sử bạn có một chuỗi thời gian dài chứa dữ liệu thị trường trong ngày và bạn muốn trích xuất giá tại một thời điểm cụ thể trong ngày vào mỗi ngày của dữ liệu. Điều gì sẽ xảy ra nếu dữ liệu không đều đặn đến mức các quan sát không rơi vào đúng thời gian mong muốn thì sao? Trong thực tế, tác vụ này có thể khiến dữ liệu dễ bị lỗi nếu bạn không cẩn thận. Đây là một ví dụ cho mục đích minh họa:

In [44]: rng = pd.date_range('2012-06-01 09:30', '2012-06-01 15:59', freq='T')

In [45]: rng = rng.append([rng + pd.offsets.BDay(i) for i in range(1, 4)])

In [46]: ts = pd.Series(np.arange(len(rng), dtype=float), index=rng)

In [47]: ts

Out [47]:

```
2017-11-17      0.0
2017-11-20      1.0
2017-11-21      2.0
2017-11-22      3.0
2017-11-23      4.0
                ...
2023-06-19   5827.0
2023-06-20   5828.0
2023-06-21   5829.0
2023-06-22   5830.0
2023-06-23   5831.0
Length: 5832
```

Lập chỉ số với một đối tượng datetime.time trong Python sẽ trích xuất các giá trị tại những thời điểm đó:

```
In [48]: from datetime import time

In [49]: ts[time(10, 0)]

Out [49]:
2012-06-01 10:00:00     30.0
2012-06-04 10:00:00    420.0
2012-06-05 10:00:00    810.0
2012-06-06 10:00:00   1200.0
```

Bên dưới, điều này sử dụng một phương thức cá thể at time (khả dụng trên các chuỗi thời gian riêng lẻ và các đối tượng DataFrame như nhau):

```
In [50]: ts.at_time(time(10, 0))

Out [50]:

2012-06-01 10:00:00     30.0
2012-06-04 10:00:00    420.0
2012-06-05 10:00:00    810.0
2012-06-06 10:00:00   1200.0
```

Bạn có thể chọn các giá trị giữa hai thời điểm bằng cách sử dụng phương thức betwee_

time có liên quan:

```
In [51]: ts.between_time(time(10, 0), time(10, 1))
Out [51]:

2012-06-01 10:00:00      30.0
2012-06-01 10:01:00      31.0
2012-06-04 10:00:00     420.0
2012-06-04 10:01:00     421.0
2012-06-05 10:00:00     810.0
2012-06-05 10:01:00     811.0
2012-06-06 10:00:00    1200.0
2012-06-06 10:01:00    1201.0
```

Như đã đề cập ở trên, có thể xảy ra trường hợp không có dữ liệu nào thực sự rơi chính xác vào một thời điểm như 10 giờ sáng, nhưng bạn có thể muốn biết giá trị được biết cuối cùng vào lúc 10 giờ sáng:

```
In [53]: indexer = np.sort(np.random.permutation(len(ts))[700:])
In [54]: irr_ts = ts.copy()
In [55]: irr_ts[indexer] = np.nan
In [56]: irr_ts['2012-06-01 09:50':'2012-06-01 10:00']
Out [56]:
```

2012-06-01 09:50:00 20.0

2012-06-01 09:51:00 NaN

2012-06-01 09:52:00 22.0

2012-06-01 09:53:00 23.0

2012-06-01 09:54:00 NaN

2012-06-01 09:55:00 25.0

2012-06-01 09:56:00 NaN

2012-06-01 09:57:00 27.0

2012-06-01 09:58:00 NaN

2012-06-01 09:59:00 NaN

2012-06-01 10:00:00 30.0

Bằng cách chuyển một mảng dấu thời gian cho phương thức asof, bạn sẽ nhận được một mảng các giá trị hợp lệ cuối cùng (non-NA) tại hoặc trước mỗi dấu thời gian. Vì vậy, chúng tôi tạo phạm vi ngày lúc 10 giờ sáng cho mỗi ngày và chuyển phạm vi đó vào thời điểm:

In [57]: selection = pd.date_range('2012-06-01 10:00', periods=4, freq='B')

In [58]: irr_ts.asof(selection)

Out [58]:

2012-06-01 10:00:00 30.0

2012-06-04 10:00:00 419.0

2012-06-05 10:00:00 810.0

2012-06-06 10:00:00 1200.0

## 1.4 Splicing Together Data Sources

Trong trường hợp đầu tiên, chuyển từ tập hợp chuỗi thời gian này sang tập hợp chuỗi thời gian khác tại một thời điểm cụ thể, vấn đề là nối hai đối tượng TimeSeries hoặc DataFrame lại với nhau bằng cách sử dụng pandas.concat:

In [59]: data1 = pd.DataFrame(np.ones((6, 3), dtype=float),

columns=['a', 'b', 'c'], index=pd.date_range('6/12/2012', periods=6))

In [60]: data2 = pd.DataFrame(np.ones((6, 3), dtype=float)* 2 ,

columns=[’a’, ’b’, ’c’], index=pd.date_range(’6/13/2012’, periods=6))

In [61]: spliced = pd.concat([data1[:’2012-06-14’], data2[’2012-06-15’:]])

In [62]: spliced

Out [62]:

| | | | | |
|---|---|---|---|---|
| 2012-06-12 | 1.0 | 1.0 | 1.0 | NaN |
| 2012-06-13 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-14 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-15 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-16 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-17 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-18 | 2.0 | 2.0 | 2.0 | 2.0 |

Giả sử trong một ví dụ tương tự rằng data1 thiếu chuỗi thời gian có trong data2:

In [63]: data2 = pd.DataFrame(np.ones((6, 4), dtype=float) * 2,

columns=[’a’, ’b’, ’c’, ’d’], index=pd.date_range(’6/13/2012’, periods=6))

In [64]: spliced = pd.concat([data1[:’2012-06-14’], data2[’2012-06-15’:]])

In [65]: spliced

Out [65]:

| | a | b | c | d |
|---|---|---|---|---|
| 2012-06-12 | 1.0 | 1.0 | 1.0 | NaN |
| 2012-06-13 | 1.0 | 1.0 | 1.0 | NaN |
| 2012-06-14 | 1.0 | 1.0 | 1.0 | NaN |
| 2012-06-15 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-16 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-17 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-18 | 2.0 | 2.0 | 2.0 | 2.0 |

Sử dụng combine_first, bạn có thể nhập dữ liệu từ trước điểm nối để mở rộng cho mục ’d’:

In [66]: spliced_filled = spliced.combine_first(data2)

In [67]: spliced_filled

Out [67]:

| | a | b | c | d |
|---|---|---|---|---|
| 2012-06-12 | 1.0 | 1.0 | 1.0 | NaN |
| 2012-06-13 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-14 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-15 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-16 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-17 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-18 | 2.0 | 2.0 | 2.0 | 2.0 |

Vì data2 không có bất kỳ giá trị nào cho 2012-06-12, nên không có giá trị nào được điền vào ngày đó.

DataFrame có bản cập nhật phương pháp liên quan để thực hiện cập nhật tại chỗ. Bạn phải vượt qua overwrite = False để làm cho nó chỉ lấp đầy các lỗ:

In [68]: spliced.update(data2, overwrite=False)

In [69]: spliced

Out [69]:

| | a | b | c | d |
|---|---|---|---|---|
| 2012-06-12 | 1.0 | 1.0 | 1.0 | NaN |
| 2012-06-13 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-14 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-15 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-16 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-17 | 2.0 | 2.0 | 2.0 | 2.0 |
| 2012-06-18 | 2.0 | 2.0 | 2.0 | 2.0 |

Để thay thế dữ liệu cho một tập hợp con các ký hiệu, bạn có thể sử dụng bất kỳ kỹ thuật nào ở trên, nhưng đôi khi đơn giản hơn chỉ cần đặt các cột trực tiếp bằng lập chỉ mục DataFrame:

In [70]: cp_spliced = spliced.copy()

In [71]: cp_spliced[[’a’, ’c’]] = data1[[’a’, ’c’]]

In [72]: cp_spliced

Out [72]:

| | a | b | c | d |
|---|---|---|---|---|
| 2012-06-12 | 1.0 | 1.0 | 1.0 | NaN |
| 2012-06-13 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-14 | 1.0 | 1.0 | 1.0 | 2.0 |
| 2012-06-15 | 1.0 | 2.0 | 1.0 | 2.0 |
| 2012-06-16 | 1.0 | 2.0 | 1.0 | 2.0 |
| 2012-06-17 | 1.0 | 2.0 | 1.0 | 2.0 |
| 2012-06-18 | NaN | 2.0 | NaN | 2.0 |

## 1.5 Return Indexes and Cumulative Returns

Trong bối cảnh tài chính, lợi nhuận thường đề cập đến phần trăm thay đổi trong giá của một tài sản. Hãy xem xét dữ liệu giá của Apple trong năm 2011 và 2022:

In [73]: from pandas_datareader import data

In [74]: prices = data.DataReader(’AAPL’,’yahoo’,’2011-01-01’)[’Adj Close’]

In [75]: prices[-5:]

Out [75]:

Date

| | |
|---|---|
| 2022-11-10 | 146.869995 |
| 2022-11-11 | 149.699997 |
| 2022-11-14 | 148.279999 |
| 2022-11-15 | 150.039993 |
| 2022-11-16 | 148.789993 |

Name: Adj Close

Với Apple, công ty không có cổ tức, việc tính toán tỷ lệ hoàn vốn tích lũy giữa hai thời điểm chỉ yêu cầu tính toán phần trăm thay đổi trong giá:

In [76]: prices[1]/prices[0] - 1

Out [76]: 0.021732150681994078

Với các cổ phiếu khác có chi trả cổ tức, việc tính toán số tiền bạn kiếm được từ việc nắm giữ một cổ phiếu có thể phức tạp hơn. Tuy nhiên, các giá trị đóng được điều chỉnh được sử dụng ở đây đã được điều chỉnh cho các khoản chia tách và cổ tức. Trong mọi trường hợp, việc lấy chỉ số hoàn vốn là một chuỗi thời gian cho biết giá trị của một khoản đầu tư đơn vị (một đô la chẳng hạn) là điều khá phổ biến. Nhiều giả định có thể làm cơ sở cho chỉ số trả về. Ví dụ, một số sẽ chọn tái đầu tư lợi nhuận và những người khác thì không. Trong trường hợp của Apple, chúng ta có thể tính toán một chỉ số trả về đơn giản bằng cách sử dụng cumprod:

In [77]: returns = prices.pct_change()

In [78]: ret_index = (1 + returns).cumprod()

In [79]: ret_index[0] = 1

In [80]: ret_index

Out [80]:

| Date | |
|---|---|
| 2006-01-03 | 1.000000 |
| 2006-01-04 | 1.004736 |
| 2006-01-05 | 1.005368 |
| 2006-01-06 | 1.013734 |
| 2006-01-09 | 1.016338 |
| | ... |
| 2022-11-10 | 4.318015 |
| 2022-11-11 | 4.359807 |
| 2022-11-14 | 4.322720 |
| 2022-11-15 | 4.359588 |
| 2022-11-16 | 4.326330 |

Name: Adj Close

Với một số chỉ số trả lại, việc tính toán lợi nhuận tích lũy tại một giải pháp cụ thể trở nên đơn giản:

In [81]: m_returns = ret_index.resample('M').mean()

In [82]: m_returns['2012']

Out [82]:

Date

2012-01-31 1.161431

2012-02-29 1.209662

2012-03-31 1.244850

2012-04-30 1.243983

2012-05-31 1.205619

2012-06-30 1.192007

2012-07-31 1.226406

2012-08-31 1.268527

2012-09-30 1.307074

2012-10-31 1.303706

2012-11-30 1.266942

2012-12-31 1.295305

Freq: M

Tất nhiên, trong trường hợp đơn giản này (không tính đến cổ tức hoặc các điều chỉnh khác), chúng có thể được tính từ phần trăm hàng ngày đã thay đổi bằng cách lấy mẫu lại với tổng hợp (ở đây, thành các khoảng thời gian):

In [83]: m_rets = (1 + returns).resample('M',kind = 'period').mean() - 1

In [84]: m_rets['2012']

Out [84]:

```
Date
2012-01     0.002283
2012-02     0.002140
2012-03     0.001465
2012-04    -0.000296
2012-05    -0.002779
2012-06     0.001977
2012-07     0.000597
2012-08     0.001091
2012-09     0.001344
2012-10    -0.000852
2012-11     0.000313
2012-12     0.000470
Freq: M
```

# Group Transforms and Analysis

Trong Chương 9, bạn đã học những kiến thức cơ bản về tính toán thống kê nhóm và áp dụng các phép biến đổi của riêng bạn cho các nhóm trong tập dữ liệu.

Hãy xem xét một tập hợp các danh mục cổ phiếu giả định. Đầu tiên tôi tạo ngẫu nhiên một không gian gồm 2000 mã:

```
import random; random.seed(0)
import string
N = 1000
def rands(n):
    choices = string.ascii_uppercase
    return ''.join([random.choice(choices) for _ in range(n)])
tickers = np.array([rands(5) for _ in range(N)])
```

Sau đó, tôi tạo một DataFrame chứa 3 cột đại diện cho danh mục đầu tư giả định, nhưng ngẫu nhiên cho một tập hợp con các mã:

```
M = 500
df = pd.DataFrame({'Momentum' : np.random.randn(M) / 200 + 0.03,
                   'Value' : np.random.randn(M) / 200 + 0.08,
                   'ShortInterest' : np.random.randn(M) / 200 - 0.02},
                   index=tickers[:M])
```

Tiếp theo, hãy tạo phân loại ngành ngẫu nhiên cho các mã. Để đơn giản hóa mọi thứ, tôi sẽ chỉ giữ nó cho 2 ngành, lưu trữ ánh xạ trong một Series:

```
ind_names = np.array(['FINANCIAL', 'TECH'])
sampler = np.random.randint(0, len(ind_names), N)
industries = pd.Series(ind_names[sampler], index=tickers, name='industry')
```

Giờ đây, chúng tôi có thể nhóm theo ngành và thực hiện tổng hợp và chuyển đổi nhóm:

```
In [90]: by_industry = df.groupby(industries)
In [91]: by_industry.mean()
Out [91]:
```

| | Momentum | Value | ShortInterest |
|---|---|---|---|
| industry | | | |
| FINANCIAL | 0.030450 | 0.079967 | -0.019377 |
| TECH | 0.030053 | 0.080350 | -0.020100 |

```
In [92]: by_industry.describe()
Out [92]:
```

| | | Momentum | Value | ShortInterest |
|---|---|---|---|---|
| industry | | | | |
| FINANCIAL | count | 248.0 | | 248.0 |
| | mean | 0.029854 | ... | 0.004795 |
| | std | 0.029854 | ... | 0.004795 |
| | min | 0.029854 | ... | -0.031261 |
| | 25% | 0.026647 | ... | -0.022839 |
| | 50% | 0.029618 | ... | -0.022839 |
| | 75% | 0.03276 | ... | -0.022839 |
| | max | 0.042640 | ... | -0.022839 |
| TECH | count | 252.0 | ... | 252.0 |
| | mean | 0.029868 | ... | -0.020530 |
| | std | 0.004954 | ... | 0.005213 |
| | min | 0.015051 | ... | -0.037141 |
| | 25% | 0.026718 | ... | -0.024063 |
| | 50% | 0.029683 | ... | -0.020273 |
| | 75% | 0.029683 | ... | -0.017362 |
| | max | 0.046302 | ... | -0.007432 |

Bằng cách xác định các chức năng chuyển đổi, có thể dễ dàng chuyển đổi các danh mục đầu tư này theo ngành. Ví dụ, tiêu chuẩn hóa trong ngành được sử dụng rộng rãi trong việc xây dựng danh mục đầu tư vốn chủ sở hữu:

```
def zscore(group):
    return (group - group.mean()) / group.std()


df_stand = by_industry.apply(zscore)
```

Bạn có thể xác minh rằng mỗi ngành có trung bình 0 và độ lệch chuẩn 1:

```
In [94]: df_stand.groupby(industries).agg(['mean', 'std'])
```

Out [94]:

| | Momentum | | Value | | ShortInterest | |
|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std |
| industry | | | | | | |
| FINANCIAL | 4.174439e-16 | 1.0 | -2.313039e-15 | 1.0 | -9.414691e-17 | 1.0 |
| TECH | 4.511946e-16 | 1.0 | -9.996448e-16 | 1.0 | -3.108624e-16 | 1.0 |

Các loại biến đổi tích hợp khác, chẳng hạn như xếp hạng, có thể được sử dụng ngắn gọn hơn:

In [95]: ind_rank = by_industry.rank(ascending=False)

In [96]: ind_rank.groupby(industries).agg(['min', 'max'])

Out [96]:

| | Momentum | | Value | | ShortInterest | |
|---|---|---|---|---|---|---|
| | min | max | min | max | min | max |
| industry | | | | | | |
| FINANCIAL | 1.0 | 250.0 | 1.0 | 250.0 | 1.0 | 250.0 |
| TECH | 1.0 | 250.0 | 1.0 | 250.0 | 1.0 | 250.0 |

Trong công bằng định lượng, "xếp hạng và tiêu chuẩn hóa" là một chuỗi biến đổi phổ biến. Bạn có thể làm điều này bằng cách xâu chuỗi rank và zs_core lại với nhau như sau:

In [97]: by_industry.apply(lambda x: zs_core(x.rank()))

Out [97]:

| | Momentum | Value | ShortInterest |
|---|---|---|---|
| MYNBI | -0.891956 | -0.407949 | -1.597224 |
| QPMZJ | -1.168532 | 0.795155 | -0.504750 |
| PLSGQ | 1.306819 | 1.666368 | -0.283490 |
| EJEYD | 0.297319 | 0.988758 | -0.712182 |
| TZIRW | -1.044073 | 0.006914 | 0.560065 |
| ... | ... | ... | ... |
| JPHKQ | 0.878127 | 1.113217 | -0.449435 |
| VACPK | -0.242004 | -0.186688 | -0.076058 |
| MHNBS | -1.292991 | -1.057901 | -0.726011 |
| YBNCI | 1.237675 | 0.601552 | 1.445107 |
| GXKFD | -1.210018 | 0.698353 | -1.306819 |

500 rows × 3 columns

## 2.1 Group Factor Exposures

Phân tích nhân tố là một kỹ thuật trong quản lý danh mục đầu tư định lượng. Danh mục đầu tư nắm giữ và hiệu suất (lợi nhuận và ít hơn) được phân tách bằng cách sử dụng một hoặc nhiều yếu tố (ví dụ về mặt rủi ro) được biểu thị dưới dạng danh mục trọng số. Ví dụ: giá cổ phiếu đồng chuyển động với một điểm chuẩn (như chỉ số S&P 500) được gọi là phiên bản thử nghiệm của nó, một yếu tố rủi ro phổ biến. Chúng ta hãy xem xét một ví dụ giả định về một danh mục đầu tư được xây dựng từ 3 yếu tố được tạo sẵn (thường được gọi là hệ số tải) và một số trọng số.

```
from numpy.random import rand
fac1, fac2, fac3 = np.random.rand(3, 1000)
ticker_subset = tickers.take(np.random.permutation(N)[:1000])

port = pd.Series(0.7 * fac1 - 1.2 * fac2 + 0.3 * fac3 + rand(1000), index=ticker_subset)

factors = pd.DataFrame('f1': fac1, 'f2': fac2, 'f3': fac3, index=ticker_subset)
```

Tương quan vectơ giữa mỗi yếu tố và danh mục đầu tư có thể không chỉ ra quá nhiều:

In [99]: factors.corrwith(port)

Out [99]:

f1 -0.027825

f2 0.020046

f3 -0.016751

Cách tiêu chuẩn để tính toán các yếu tố rủi ro là bằng hồi quy bình phương nhỏ nhất:

```
from sklearn.linear_model import LinearRegression
factors_array = list(np.round(LinearRegression().fit(factors, port).coef,6))
intercept = LinearRegression().fit( factors, port).intercept_
intercept = round(intercept,6)
factors_array.append(intercept)
index = ['f1','f2','f3','intercept']
lm = pd.Series(factors_array, index = index)
```

In [100]: lm

Out [100]:

f1 0.029736

f2 0.085512

f3 -0.114180

intercept 0.391194

## 2.2 Decile and Quartile Analysis

Phân tích dữ liệu dựa trên số lượng mẫu là một công cụ quan trọng khác để phân tích tài chính. Ví dụ: hiệu suất của danh mục cổ phiếu có thể được chia thành các phần tư (bốn phần có kích thước bằng nhau) dựa trên giá trên thu nhập của mỗi cổ phiếu. Sử dụngpan das.qcut kết hợp với chia nhóm làm cho phân tích lượng tử trở nên đơn giản một cách hợp lý.

Ví dụ, hãy xem xét một xu hướng đơn giản sau hoặc chiến lược giao dịch chỉ số S&P 500 thông qua quỹ giao dịch hối đoái SPY. Bạn có thể tải xuống lịch sử giá từ Yahoo!Finance:

In [105]: from pandas_datareader import data as web

In [106]: data = web.DataReader('SPY','yahoo','2006-01-01')

In [107]: data.count()

Out [107]:

| | |
|---|---|
| High | 4250 |
| Low | 4250 |
| Open | 4250 |
| Close | 4250 |
| Volume | 4250 |
| Adj Close | 4250 |

In [108]: data.isnull().sum()

Out [108]:

| | |
|---|---|
| High | 0 |
| Low | 0 |
| Open | 0 |
| Close | 0 |
| Volume | 0 |
| Adj Close | 0 |

Bây giờ, chúng ta sẽ tính toán lợi nhuận hàng ngày và một hàm để chuyển đổi lợi nhuận thành tín hiệu xu hướng được hình thành từ tổng di chuyển có độ trễ:

```
px = prices['Adj Close']

returns = px.pct_change()

def to_index(rets):
    index = (1 + rets).cumprod()
    first_loc = max(index.notnull().argmax() - 1, 0)
    index.values[first_loc] = 1
    return index

def trend_signal(rets, lookback, lag):
    signal = pd.Series.rolling(rets, lookback, min_periods=lookback - 5).sum()
    return signal.shift(lag)
```

Sử dụng chức năng này, chúng ta có thể (một cách đơn giản) tạo và thử nghiệm một chiến lược giao dịch đưa ra tín hiệu xung lượng:

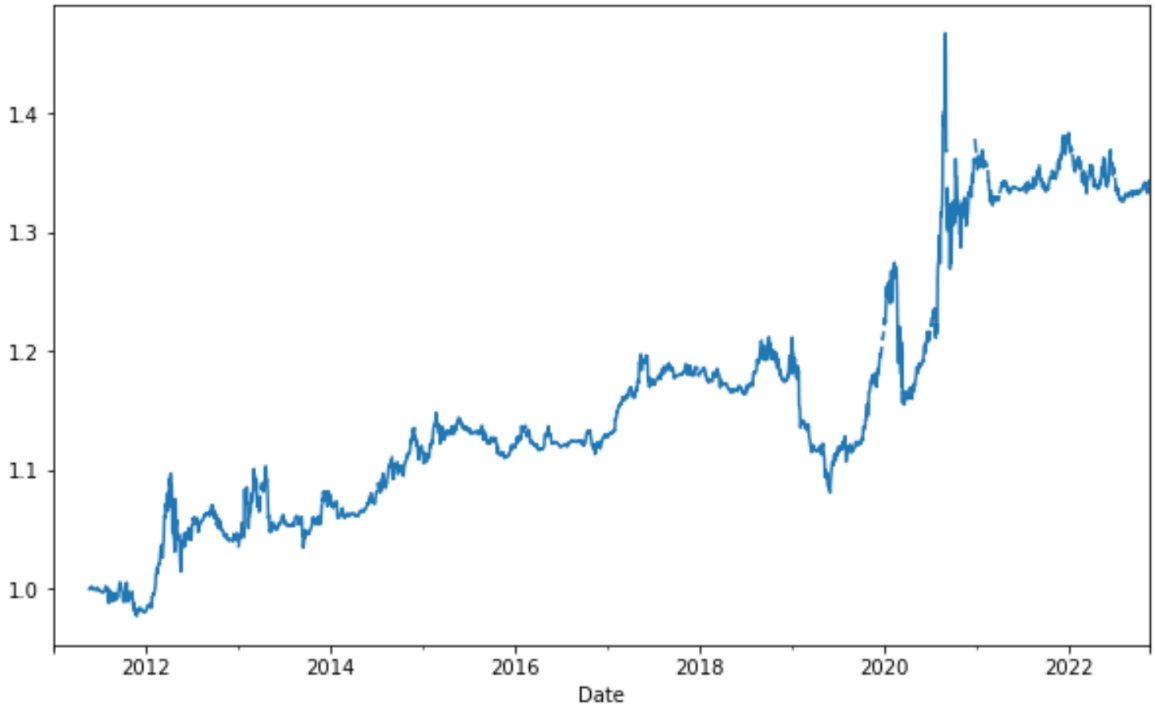
In [109]: signal = trend_signal(returns, 100, 3)

In [110]: trade = signal.resample('B').fillna(method='ffill')

In [111]: trade_rets = trade.shift(1) * returns

Sau đó, chúng tôi có thể chuyển đổi chiến lược trả về thành chỉ mục trả lại và vẽ biểu đồ của chúng (Hình 1):

In [112]: to_index(trade_rets).plot()




Hình 1: Chỉ số hoàn vốn chiến lược SPY

Giả sử bạn muốn phân tách hiệu suất chiến lược thành các giai đoạn giao dịch ngày càng ít biến động hơn. Theo dõi độ lệch chuẩn hàng năm theo dõi một năm là một phép đo đơn giản về sự biến động và chúng ta có thể tính toán tỷ lệ Sharpe để đánh giá tỷ lệ phần thưởng trên rủi ro trong các chế độ biến động khác nhau:

```
vol = pd.Series.rolling(returns, 250, min_periods=200).std() * np.sqrt(250)
def sharpe(rets, ann=250):
    return rets.mean() / rets.std() * np.sqrt(ann)
```

Bây giờ, phân chia vào tứ phân vị với qcut và tổng hợp sharpe chúng tôi đạt được:

```
In [114]: trade_rets.groupby(pd.qcut(vol, 4)).agg(sharpe)
Out [114]:

(0.066, 0.118]     0.754687
(0.118, 0.153]    -0.787331
(0.153, 0.197]     0.717905
(0.197, 0.457]    -0.288824
```

Những kết quả này cho thấy rằng chiến lược hoạt động tốt nhất trong thời kỳ mà sự biến động là cao nhất.

# More Example Applications

Dưới đây là một tập hợp nhỏ các ví dụ bổ sung.

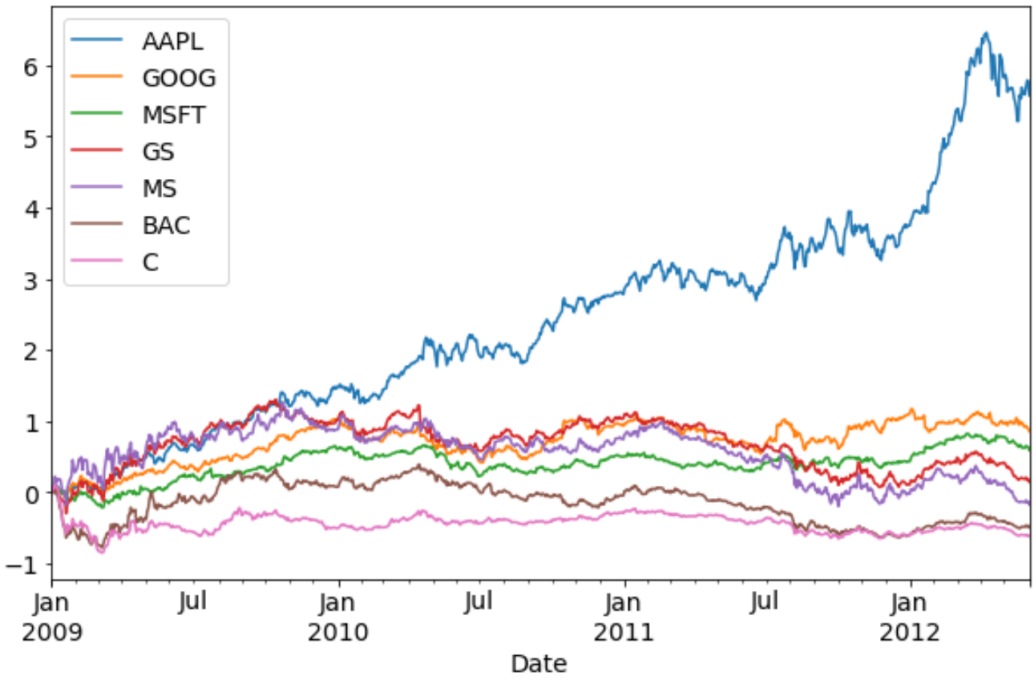
## 3.1 Signal Frontier Analysis

Trong phần này, tôi sẽ mô tả một danh mục đầu tư động lượng mặt cắt ngang đơn giản và cho thấy cách bạn có thể khám phá một mạng lưới các tham số hóa mô hình. Đầu tiên, tôi sẽ tải giá lịch sử cho một danh mục các cổ phiếu tài chính và công nghệ:

```
import pandas_datareader.data as web
names = ['AAPL', 'GOOG', 'MSFT', 'GS', 'MS', 'BAC', 'C']
def get_px(stock, start, end):
    return web.get_data_yahoo(stock, start, end)['Adj Close']
px = pd.DataFrame(n: get_px(n, '1/1/2009', '6/1/2012') for n in names)
```

Chúng tôi có thể dễ dàng vẽ biểu đồ lợi nhuận tích lũy của mỗi cổ phiếu:

```
In [117]: px = px.asfreq('B').fillna(method='pad')
In [118]: rets = px.pct_change()
In [119]: ((1 + rets).cumprod() - 1).plot()
```




Hình 2: Lợi nhuận tích lũy cho mỗi cổ phiếu

Đối với việc xây dựng danh mục đầu tư, chúng ta sẽ tính toán động lượng qua một lần xem lại nhất định, sau đó xếp hạng theo thứ tự giảm dần và chuẩn hóa:

```
def calc_mom(price, lookback, lag):
    mom_ret = price.shift(lag).pct_change(lookback)
    ranks = mom_ret.rank(axis=1, ascending=False)
    demeaned = ranks - ranks.mean(axis=1)
    return demeaned / demeaned.std(axis=1)
```

Với chức năng chuyển đổi này, chúng ta có thể thiết lập một chức năng kiểm tra chiến lược để tính toán một danh mục đầu tư cho một khoảng thời gian xem lại và nắm giữ cụ thể (ngay giữa các phiên giao dịch), trả về tỷ lệ Sharpe tổng thể:

```
compound = lambda x : (1 + x).prod() - 1
daily_sr = lambda x: x.mean() / x.std()

def strat_sr(prices, lb, hold):
    # Compute portfolio weights
    freq = '%dB' % hold
    port = calc_mom(prices, lb, lag=1)
```

```
    daily_rets = prices.pct_change()

    # Compute portfolio returns
    port = port.shift(1).resample(freq, how='first')
    returns = daily_rets.resample(freq, how=compound)
    port_rets = (port * returns).sum(axis=1)

    return daily_sr(port_rets) * np.sqrt(252 / hold)
```

Khi được gọi với prices và kết hợp tham số, hàm này trả về một giá trị vô hướng:

```
In [122]: strat_sr(px, 70, 30)
Out[122]: 0.27421582756800583
```

Từ đó, bạn có thể đánh giá hàm trên hoạt động trên một lưới các tham số, lưu trữ chúng khi bạn đi vào sự kết luận và cuối cùng đưa kết quả vào DataFrame:

```
from collections import defaultdict

lookbacks = range(20, 90, 5)
holdings = range(20, 90, 5)
dd = defaultdict(dict)
for lb in lookbacks:
    for hold in holdings:
        dd[lb][hold] = strat_sr(px, lb, hold)
ddf = DataFrame(dd)
ddf.index.name = 'Holding Period'
ddf.columns.name = 'Lookback Period'
```

Để trực quan hóa kết quả và có ý tưởng về những gì đang xảy ra, đây là một hàm sử dụng matplotlib để tạo bản đồ nhiệt với một số trang trí:

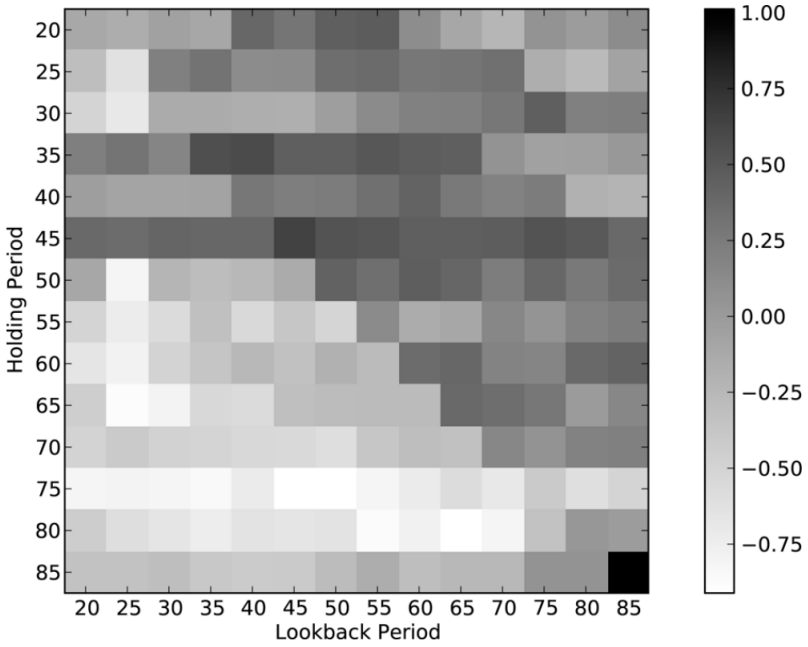
```
import matplotlib.pyplot as plt

def heatmap(df, cmap=plt.cm.gray_r):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    axim = ax.imshow(df.values, cmap=cmap, interpolation='nearest')
```

```
    ax.set_xlabel(df.columns.name)
    ax.set_xticks(np.arange(len(df.columns)))
    ax.set_xticklabels(list(df.columns))
    ax.set_ylabel(df.index.name)
    ax.set_yticks(np.arange(len(df.index)))
    ax.set_yticklabels(list(df.index))
    plt.colorbar(axim)
```

Gọi hàm này trên kết quả kiểm tra lại, chúng tôi nhận được:

```
In [125]: heatmap(ddf)
```




Hình 3: Sơ đồ nhiệt của chiến lược động lượng tỷ lệ Sharpe (cao hơn là tốt hơn) qua các lần xem lại khác nhau và giữ thời gian

## 3.2 Future Contract Rolling

Hợp đồng tương lai là một dạng hợp đồng phái sinh phổ biến; đó là một thỏa thuận để nhận một tài sản nhất định (chẳng hạn như dầu, vàng hoặc cổ phiếu của chỉ số FTSE 100) vào một ngày cụ thể. Trong thực tế, mô hình hóa và giao dịch các hợp đồng tương lai trên cổ phiếu, tiền tệ ,hàng hóa, trái phiếu và các loại tài sản khác rất phức tạp do tính

chất có thời hạn của mỗi hợp đồng. Ví dụ: tại bất kỳ thời điểm nào cho một loại tương lai (chẳng hạn như hợp đồng tương lai bạc hoặc đồng), nhiều hợp đồng với các ngày hết hạn có thể được giao dịch. Trong nhiều trường hợp, hợp đồng tương lai sẽ hết hạn tiếp theo (ở gần hợp đồng) sẽ có tính thanh khoản cao nhất (khối lượng cao nhất và chênh lệch giá mua-bán thấp nhất).

Đối với mục đích mô hình hóa và dự báo, có thể dễ dàng hơn nhiều để làm việc với chỉ số hoàn vốn liên tục cho thấy lợi nhuận và thua lỗ liên quan đến việc luôn giữ hợp đồng gần. Chuyển đổi từ hợp đồng hết hạn sang hợp đồng tiếp theo (hoặc xa hơn) được gọi là rolling. Tính toán một chuỗi liên tục trong tương lai từ dữ liệu cá nhân không nhất thiết phải là một bài tập đơn giản và thường đòi hỏi sự hiểu biết sâu sắc hơn về thị trường và cách các công cụ được giao dịch. Ví dụ: trong thực tế, khi nào và bao lâu bạn sẽ giao dịch ra khỏi một hợp đồng sắp hết hạn và vào hợp đồng tiếp theo? Ở đây tôi mô tả một quá trình như vậy.

Đầu tiên, tôi sẽ sử dụng giá quy mô cho quỹ giao dịch trao đổi SPY làm đại diện cho chỉ số S&P 500:

```
In [127]: import pandas_datareader.data as web

In [128]: px = web.get_data_yahoo('SPY')['Adj Close'] * 10

In [129]: px

Out [129]:
```

```
Date
2017-11-20          2370.312347
2017-11-21          2385.820770
2017-11-22          2383.710022
2017-11-24          2389.216461
2017-11-27          2388.022919
                                ...
2022-11-11          3985.100098
2022-11-14          3951.199951
2022-11-15          3984.899902
2022-11-16          3954.500122
2022-11-17          3942.399902
Name: Adj Close    Length: 1258
```

Bây giờ, một chút thiết lập. Tôi đã đặt một số hợp đồng tương lai và ngày hết hạn của S&P 500 vào một Series:

```
from datetime import datetime
expiry = 'ESU2': datetime(2012, 9, 21), 'ESZ2': datetime(2012, 12, 21)
expiry = pd.Series(expiry).sort_values()
```

expiry sẽ trông như thế này:

```
In [131]: expiry
Out [131]:
 ESU2   2012-09-21
 ESZ2   2012-12-21
```

Sau đó, tôi sử dụng Yahoo! Giá tài chính cùng với một bước đi ngẫu nhiên và một số nhiễu để mô phỏng hai hợp đồng trong tương lai:

```
N = 200
np.random.seed(12347)
walk = (np.random.randint(0, 200, size=N) - 100) * 0.25
perturb = (np.random.randint(0, 20, size=N) - 10) * 0.25
```

```
walk = walk.cumsum()

rng = pd.date_range(px.index[0], periods=len(px) + N, freq='B')

near = np.concatenate([px.values, px.values[-1] + walk])

far = np.concatenate([px.values, px.values[-1] + walk + perturb])

prices = pd.DataFrame('ESU2': near, 'ESZ2': far, index=rng)
```

prices sau đó có hai chuỗi thời gian cho các hợp đồng khác nhau một lượng ngẫu nhiên:

```
In [133]: prices.tail()

Out [133]:
                 ESU2         ESZ2
2023-06-14  3983.750122  3985.500122
2023-06-15  3970.000122  3972.250122
2023-06-16  3978.000122  3979.750122
2023-06-19  3994.500122  3993.750122
2023-06-20  3974.500122  3972.250122
```

Một cách để nối các chuỗi thời gian với nhau thành một chuỗi liên tục là xây dựng ma trận trọng số. Các hợp đồng đang hoạt động sẽ có trọng số là 1 cho đến khi ngày hết hạn gần kề. Tại thời điểm đó, bạn phải quyết định một quy ước cuộn. Đây là một hàm tính toán một ma trận trọng số với phân rã tuyến tính trong một số khoảng thời gian dẫn đến hết hạn:

```
def get_roll_weights(start, expiry, items, roll_periods=5):
    # start : first date to compute weighting DataFrame
    # expiry : Series of ticker -> expiration dates
    # items : sequence of contract names


    dates = pd.date_range(start, expiry[-1], freq='B')
    weights = pd.DataFrame(np.zeros((len(dates), len(items))),
                           index=dates, columns=items)

    prev_date = weights.index[0]
    for i, (item, ex_date) in enumerate(expiry.iteritems()):
```

```
        if i < len(expiry) - 1:
            weights.loc[prev_date:ex_date - pd.offsets.BDay(), item] = 1
            roll_rng = pd.date_range(end=ex_date - pd.offsets.BDay(),
                                     periods=roll_periods + 1, freq='B')
            decay_weights = np.linspace(0, 1, roll_periods + 1)
            weights.loc[roll_rng, item] = 1 - decay_weights
            weights.loc[roll_rng, expiry.index[i + 1]] = decay_weights
        else:
            weights.loc[prev_date:, item] = 1
    prev_date = ex_date
    return weights
```

Các trọng số trông như thế này xung quanh ESU2 expiry:

```
In [135]: weights = get_roll_weights('6/1/2012', expiry, prices.columns)

In [136]: weights.loc['2012-09-12':'2012-09-21']

Out [136]:
                ESU2   ESZ2
2012-09-12       1.0    1.0
2012-09-13       1.0    1.0
2012-09-14       0.8    1.0
2012-09-17       0.6    1.0
2012-09-18       0.4    1.0
2012-09-19       0.2    1.0
2012-09-20       0.0    1.0
2012-09-21       0.0    1.0
```

Cuối cùng, lợi nhuận tương lai được chuyển nhượng chỉ là một tổng trọng số của lợi nhuận hợp đồng:

```
In [137]: rolled_returns = (prices.pct_change() * weights).sum(1)
```

## 3.3 Rolling Correlation and Linear Regression

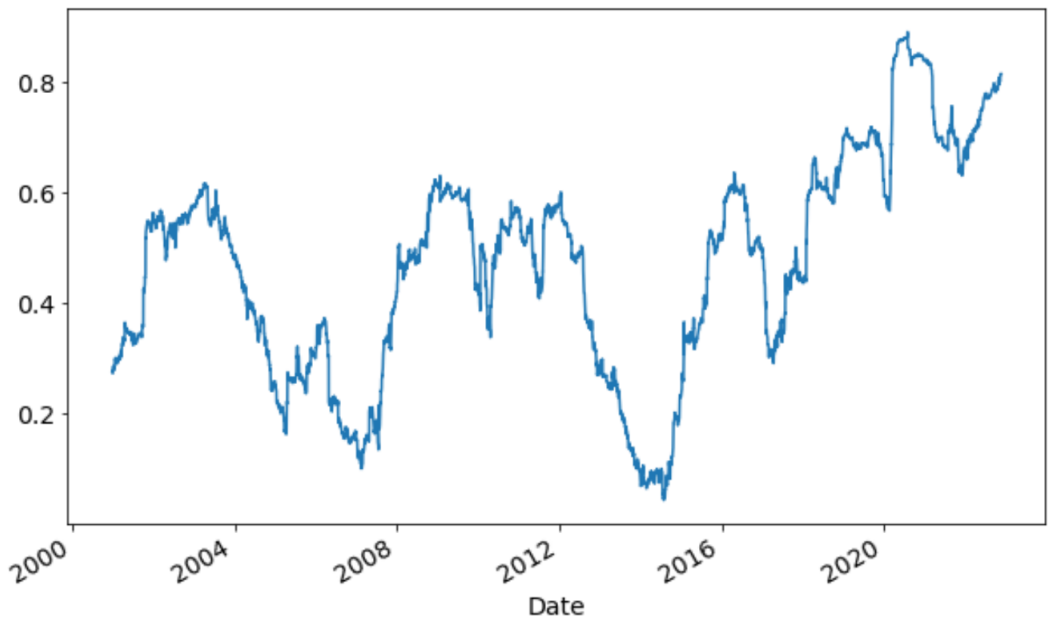
Các mô hình động đóng một vai trò quan trọng trong mô hình tài chính vì chúng có thể được sử dụng để mô phỏng các quyết định giao dịch trong một giai đoạn lịch sử. Di chuyển cửa sổ và chức năng chuỗi thời gian có trọng số theo cấp số nhân là một ví dụ về các công cụ được sử dụng cho các mô hình động.

Tương quan là một cách để xem xét sự đồng chuyển động giữa những thay đổi trong hai chuỗi thời gian của nội dung. Hàm rolling-cor có thể được gọi với hai chuỗi trả về để tính toán tương quan cửa sổ chuyển động. Đầu tiên, tôi tải một số chuỗi giá từ Yahoo!Finance và tính toán lợi nhuận hàng ngày:

```
aapl = web.get_data_yahoo('AAPL', '2000-01-01')['Adj Close']
msft = web.get_data_yahoo('MSFT', '2000-01-01')['Adj Close']
msft_rets = msft.pct_change()
aapl_rets = aapl.pct_change()
```

Sau đó, tôi tính toán và vẽ biểu đồ tương quan biến động trong một năm:

```
aapl_rets.rolling(250).corr(msft_rets).plot()
```




Hình 4: Tương quan một năm của Apple với Microsoft

GHI CHÚ:

- Phần code ở trang 28, 29, 30 không thể chạy được thư viện không còn được hỗ trợ, phần nội dung ở trang được lấy từ kết quả trong sách.

# Tài Liệu Tham Khảo

Python for Data Analysis của Wes McKinney. Bản quyền © 2013 Wes McKinney. Nhà xuất bản O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

# Đường dẫn tới link github

https://github.com/s2nmt/CSDL.git