Introduction to Pairs Trading 筆記

配對交易是一個經典的方法，主要是尋找走勢相近的資產，當走勢偏離時，我們下單來謀取報酬，因為我們相信之後價格又會走回相似的價差。

Section1 基本知識

要判斷那些資產有相似走勢，我們必須有一些共整合(cointegration)以及定態(stationary)時間序列的知識，讓我們看看以下的解釋。

我們先載入相關的套件庫

```
import numpy as np
import pandas as pd

import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint, adfuller

import matplotlib.pyplot as plt
```

然後我們定義一個函數

```
def generate_datapoint(params):
    mu = params[0]
    sigma = params[1]
    return np.random.normal(mu, sigma)
```

這個函數作用是產生隨機數，這個函數最後會回傳一個常態分配的實現，這個函數接受一個像列表的輸入，第一個參數決定回傳常態分配的期望值，第二個決定這個分配的變異數。

下面我們用程式生成兩個時間序列。

Series A

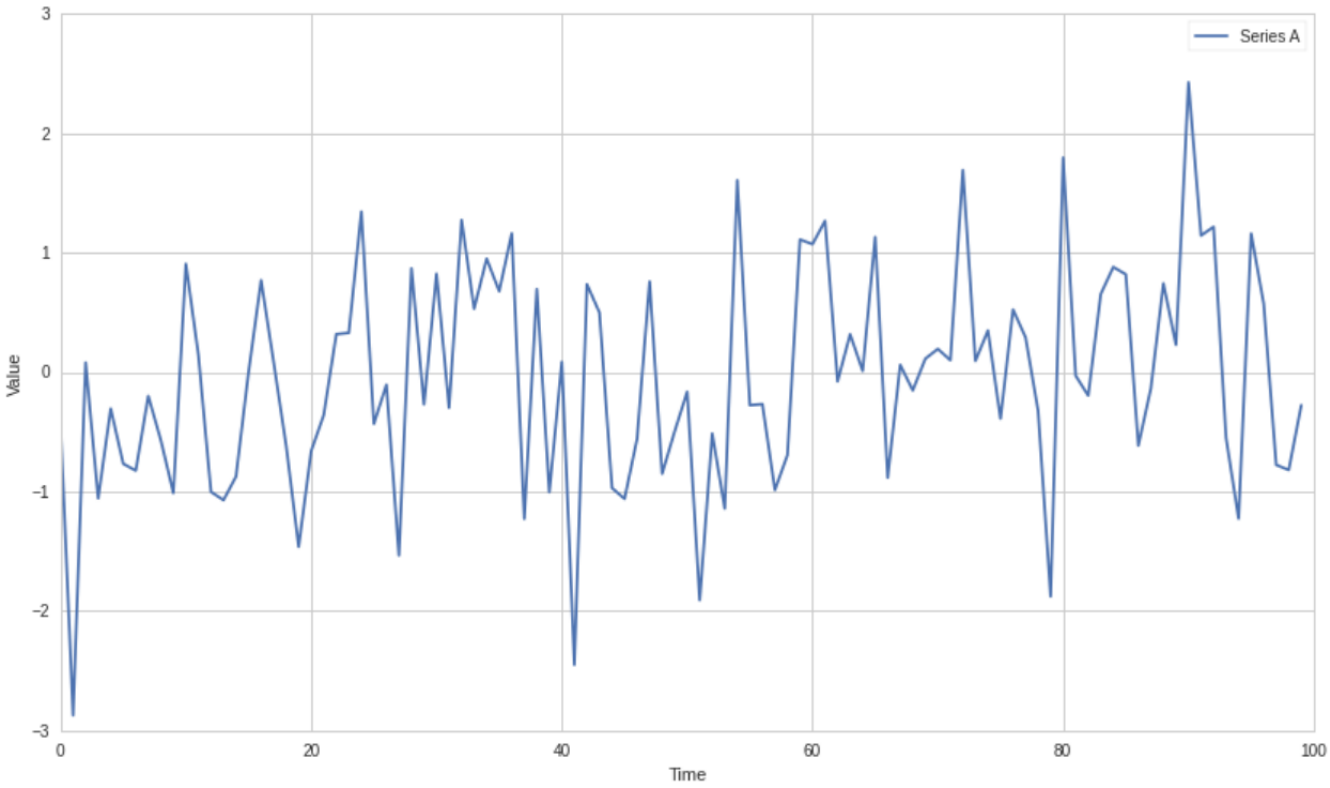
第一個時間序列產生100個隨機數(常態分配(0,1))，並存入A這個series裡面，並將其畫出。

```
# Set the parameters and the number of datapoints
params = (0, 1)
T = 100

A = pd.Series(index=range(T))
A.name = 'A'

for t in range(T):
    A[t] = generate_datapoint(params)
```

```
plt.plot(A)
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend(['Series A']);
```



Series B

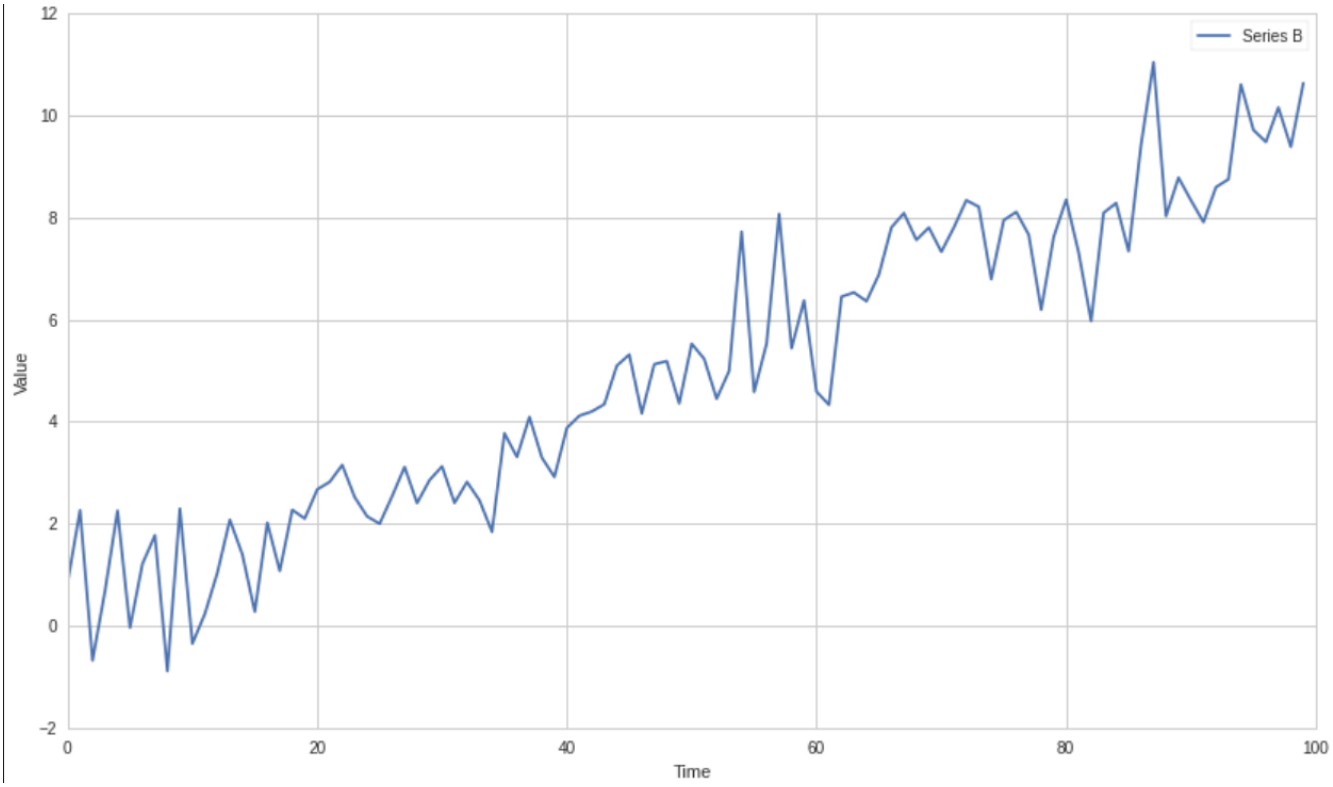
第二個數列，我們加入時間趨勢，期常態分配期望值參數為$t \times 0.1$ ，代表當時間增大，其抽出來的數平均更大，期數列有向上跑得趨勢。

```
# Set the number of datapoints
T = 100

B = pd.Series(index=range(T))
B.name = 'B'

for t in range(T):
    # Now the parameters are dependent on time
    # Specifically, the mean of the series changes over time
    params = (t * 0.1, 1)
    B[t] = generate_datapoint(params)

plt.plot(B)
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend(['Series B']);
```



Testing for Stationarity

我們先回憶一下stationary 數列的定義(weak version)

A time random process $\{X_t\}$ is weak sense stationarity has the following restrictions on its mean function $m_x(t) = E[x_t]$ and autocovariance function $K_{XX}(t_1, t_2) = E[(X_{t_1} - m_X(t_1))(X_{t_2} - m_X(t_2))]$:

- $m_X(t) = m_X(t + \tau)$ for all $\tau \in R$
- $K_{XX}(t_1, t_2) = K_{XX}(t_1 - t_2, 0)$ for all $t_1, t_2 \in R$
- $E[|X(t)|^2] < \infty$

簡單來說，其產生的資料點的分配滿足平均隨時間不變，與相關結構只與時間差有關兩項特性。而常用用來檢測數列是否為定態數列的檢定方法為單根檢定，下面我們引用statsmodel裡面寫好的單跟檢定來進行檢定。

```
def check_for_stationarity(X, cutoff=0.01):
    # H_0 in adfuller is unit root exists (non-stationary)
    # We must observe significant p-value to convince ourselves that the series is
stationary
    pvalue = adfuller(X)[1]
    if pvalue < cutoff:
        print 'p-value = ' + str(pvalue) + ' The series ' + X.name +' is likely
stationary.'
        return True
    else:
        print 'p-value = ' + str(pvalue) + ' The series ' + X.name +' is likely non-
stationary.'
        return False
```

```
check_for_stationarity(A);
check_for_stationarity(B);
```

Cointegration

回顧完怎麼檢定定態隨機序列，我們學習怎麼定義共整合數列。

定義**1**:我們將定態時間序列稱為零階整合(integrated of order zero)序列，簡稱$I(0)$，如果一個序列經過一階差分後為定態，則稱此序列為$I(1)$

定義**2**： For a set of the time series $(X_1, X_2, \cdots, X_k)$, if all series are $I(1)$, and some linear combination of them is $I(0)$, we say the set of time series is cointegrated

簡單來說，兩數列本身都不定態數列，但是在某種線性組合後會變成定態數列，我們就稱這組數列為共正和數列，這可以理解這組數列有某種長期均衡關係。

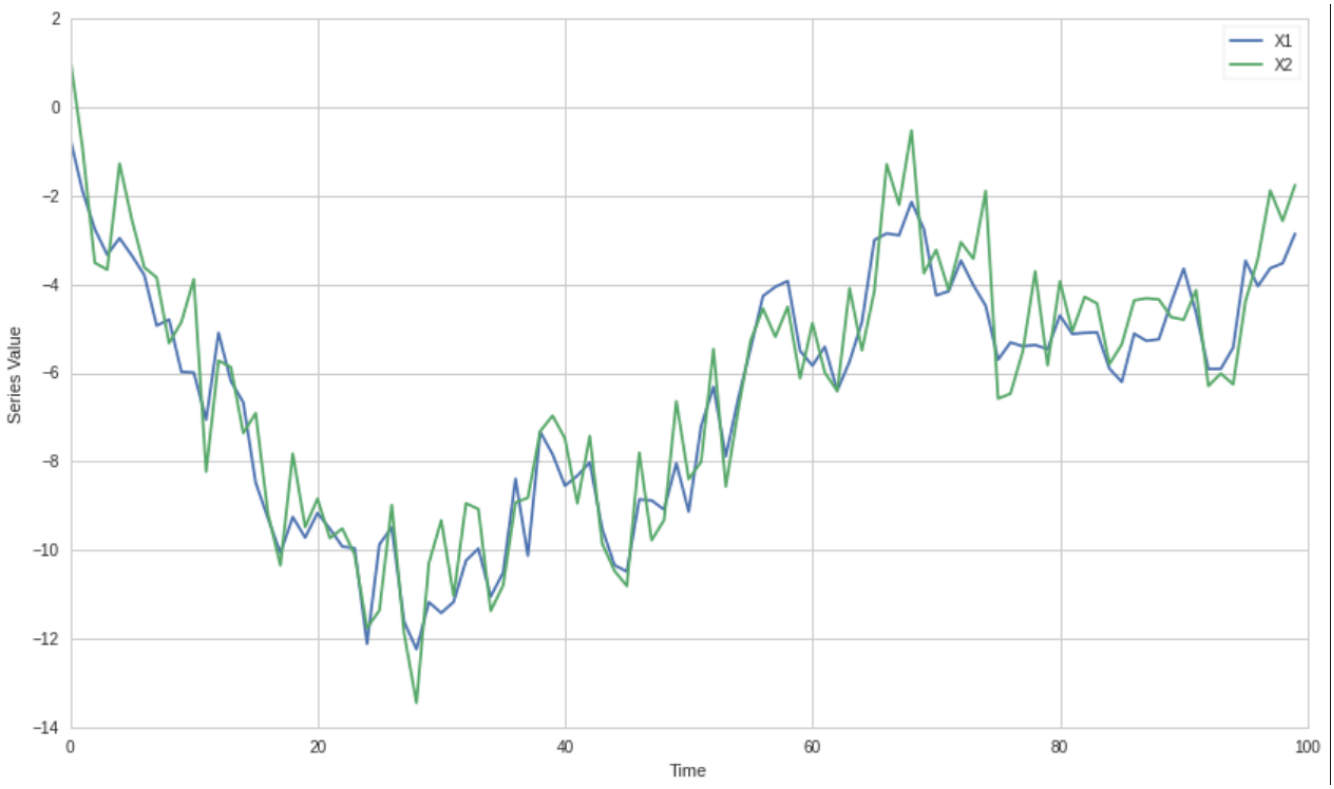
讓我們產生一個共整合數列。

```
# Length of series
N = 100
# Generate a stationary random X1
X1 = np.random.normal(0, 1, N)
# Integrate it to make it I(1)
X1 = np.cumsum(X1)
X1 = pd.Series(X1)
X1.name = 'X1'

# Make an X2 that is X1 plus some noise
X2 = X1 + np.random.normal(0, 1, N)
X2.name = 'X2'
```

可以看到X_2是X_1加白噪音

```
plt.plot(X1)
plt.plot(X2)
plt.xlabel('Time')
plt.ylabel('Series Value')
plt.legend([X1.name, X2.name]);
```




```
Z = X2.diff()[1:]
Z.name = 'Z'

check_for_stationarity(Z);
```

Testing for Cointegration

一般來說，我們使用 X_2對X_1先去跑回歸

$X_2 = \alpha + \beta X_1 + \varepsilon$

然後減去X_1影響

$X_2 - \beta X_1 = \alpha + \varepsilon$

應該要是stationary，讓我們看看一個實際案例。

```
symbol_list = ['ABGB', 'FSLR']
prices = get_pricing(symbol_list, fields=['price']
                                , start_date='2014-01-01', end_date='2015-01-01')['price']
prices.columns = map(lambda x: x.symbol, prices.columns)
X1 = prices[symbol_list[0]]
X2 = prices[symbol_list[1]]
plt.plot(X1.index, X1.values)
plt.plot(X1.index, X2.values)
plt.xlabel('Time')
plt.ylabel('Series Value')
plt.legend([X1.name, X2.name]);
X1 = sm.add_constant(X1)
results = sm.OLS(X2, X1).fit()
```

```
# Get rid of the constant column
X1 = X1[symbol_list[0]]

results.params
b = results.params[symbol_list[0]]
Z = X2 - b * X1
Z.name = 'Z'

plt.plot(Z.index, Z.values)
plt.xlabel('Time')
plt.ylabel('Series Value')
plt.legend([Z.name]);

check_for_stationarity(Z);
```

這邊已經有存在一個寫好的共整合檢定程式，我們可以直接調用

```
from statsmodels.tsa.stattools import coint

coint(X1, X2)
```

Section 2 Pair Trading

在這邊，我們想要找到一對資產，利用它們價格有一個長期的關係進行套利，而用來檢定一對資產是否有長期關係的工具就是正整合檢定。

我們底下先虛擬一個pair trading的例子

```
import numpy as np
import pandas as pd

import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint
# just set the seed for the random number generator
np.random.seed(107)

import matplotlib.pyplot as plt

X_returns = np.random.normal(0, 1, 100) # Generate the daily returns
# sum them and shift all the prices up into a reasonable range
X = pd.Series(np.cumsum(X_returns), name='X') + 50
X.plot();

some_noise = np.random.normal(0, 1, 100)
Y = X + 5 + some_noise
Y.name = 'Y'
pd.concat([X, Y], axis=1).plot();
```

我們假設Y與X之間有某種經濟相關，譬如A公司與B公司可能是母子公司的關係等等，所以產生這兩序列，接下來我們畫出價差。

```
(Y - X).plot() # Plot the spread
plt.axhline((Y - X).mean(), color='red', linestyle='--') # Add the mean
plt.xlabel('Time')
plt.legend(['Price Spread', 'Mean']);

# compute the p-value of the cointegration test
# will inform us as to whether the spread between the 2 timeseries is stationary
# around its mean
score, pvalue, _ = coint(X,Y)
print pvalue
```

很多人搞不懂correlation與cointegration之間的關聯，接下來我們給兩個小例子

```
X_returns = np.random.normal(1, 1, 100)
Y_returns = np.random.normal(2, 1, 100)

X_diverging = pd.Series(np.cumsum(X_returns), name='X')
Y_diverging = pd.Series(np.cumsum(Y_returns), name='Y')

pd.concat([X_diverging, Y_diverging], axis=1).plot();

print 'Correlation: ' + str(X_diverging.corr(Y_diverging))
score, pvalue, _ = coint(X_diverging,Y_diverging)
print 'Cointegration test p-value: ' + str(pvalue)
```

```
Y2 = pd.Series(np.random.normal(0, 1, 1000), name='Y2') + 20
Y3 = Y2.copy()

# Y2 = Y2 + 10
Y3[0:100] = 30
Y3[100:200] = 10
Y3[200:300] = 30
Y3[300:400] = 10
Y3[400:500] = 30
Y3[500:600] = 10
Y3[600:700] = 30
Y3[700:800] = 10
Y3[800:900] = 30
Y3[900:1000] = 10

Y2.plot()
Y3.plot()
plt.ylim([0, 40]);

# correlation is nearly zero
print 'Correlation: ' + str(Y2.corr(Y3))
score, pvalue, _ = coint(Y2,Y3)
print 'Cointegration test p-value: ' + str(pvalue)
```

我們正在尋找一套太陽能公司的股票，看看它們是否是一體化的。我們首先要定義我們想要查看的證券清單。然後我們將獲得2014年每個證券的定價數據。

我們這裡的方法是在我們之前提到的頻譜中間的某個地方。我們已經制定了一個經濟假設，即能源部門內的一部分證券之間存在某種聯繫，我們想要測試是否存在任何協整的貨幣對。與搜索數百種證券相比，這導致多重比較偏差顯著減少，而不是形成單獨測試的假設。

注意：我們在數據中包含市場。這是因為市場推動了許多證券的流動，你經常會發現兩個看似協整的證券，但實際上它們並沒有整合，只是與市場協整。這被稱為混雜變量，檢查您發現的任何關係中的市場參與是很重要的。

```
def find_cointegrated_pairs(data):
    n = data.shape[1]
    score_matrix = np.zeros((n, n))
    pvalue_matrix = np.ones((n, n))
    keys = data.keys()
    pairs = []
    for i in range(n):
        for j in range(i+1, n):
            S1 = data[keys[i]]
            S2 = data[keys[j]]
            result = coint(S1, S2)
            score = result[0]
            pvalue = result[1]
            score_matrix[i, j] = score
            pvalue_matrix[i, j] = pvalue
            if pvalue < 0.05:
                pairs.append((keys[i], keys[j]))
    return score_matrix, pvalue_matrix, pairs
```

```
symbol_list = ['ABGB', 'ASTI', 'CSUN', 'DQ', 'FSLR','SPY']
prices_df = get_pricing(symbol_list, fields=['price']
                                , start_date='2014-01-01', end_date='2015-01-01')['price']
prices_df.columns = map(lambda x: x.symbol, prices_df.columns)

# Heatmap to show the p-values of the cointegration test between each pair of
# stocks. Only show the value in the upper-diagonal of the heatmap
scores, pvalues, pairs = find_cointegrated_pairs(prices_df)
import seaborn
seaborn.heatmap(pvalues, xticklabels=symbol_list, yticklabels=symbol_list,
cmap='RdYlGn_r'
                , mask = (pvalues >= 0.05)
                )
print pairs
```

接下來我們要畫兩者之間的價差。

```
S1 = sm.add_constant(S1)
results = sm.OLS(S2, S1).fit()
S1 = S1['ABGB']
b = results.params['ABGB']
```

```
spread = S2 - b * S1
spread.plot()
plt.axhline(spread.mean(), color='black')
plt.legend(['Spread']);

ratio = S1/S2
ratio.plot()
plt.axhline(ratio.mean(), color='black')
plt.legend(['Price Ratio']);
```

為了妥善定義我們的策略，我們先把數列進行標準化。

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

zscore(spread).plot()
plt.axhline(zscore(spread).mean(), color='black')
plt.axhline(1.0, color='red', linestyle='--')
plt.axhline(-1.0, color='green', linestyle='--')
plt.legend(['Spread z-score', 'Mean', '+1', '-1']);
```

則我們可以把策略定為:

- 當價差小於-1.0時作多
- 當價差大於1.0時做空
- 出場，當價差接近0

這只是冰山一角，只是一個非常簡單的例子來說明這些概念。在實踐中，您可能希望計算針對S1和S2保持多少份額的更佳權重。關於配對交易的一些額外資源列在本筆記本的末尾 使用不斷更新的統計數據進

一般來說，對整個樣本量進行統計可能很糟糕。例如，如果市場向上移動，並且兩種證券都在上漲，那麼過去3年的平均價格可能無法代表今天。出於這個原因，交易者經常使用依賴於最新數據的滾動窗口的統計數據。 移動平均線

移動平均線只是過去n的平均值 每個給定時間的數據點。對於前n個，它將是未定義的

我們系列中的數據點。較短的移動平均線將更加跳躍並且不太可靠，但會快速響應新信息。較長的移動平均線將更加平滑，但需要更多時間來整合新信息。

我們還需要使用滾動測試版，這是對我們的點差應該如何計算的滾動估計，以便使我們的所有參數保持最新。

```
# Get the spread between the 2 stocks
# Calculate rolling beta coefficient
rolling_beta = pd.ols(y=S1, x=S2, window_type='rolling', window=30)
spread = S2 - rolling_beta.beta['x'] * S1
spread.name = 'spread'

# Get the 1 day moving average of the price spread
spread_mavg1 = pd.rolling_mean(spread, window=1)
spread_mavg1.name = 'spread 1d mavg'

# Get the 30 day moving average
spread_mavg30 = pd.rolling_mean(spread, window=30)
```

```
spread_mavg30.name = 'spread 30d mavg'

plt.plot(spread_mavg1.index, spread_mavg1.values)
plt.plot(spread_mavg30.index, spread_mavg30.values)

plt.legend(['1 Day Spread MAVG', '30 Day Spread MAVG'])

plt.ylabel('Spread');
```

```
# Take a rolling 30 day standard deviation
std_30 = pd.rolling_std(spread, window=30)
std_30.name = 'std 30d'

# Compute the z score for each day
zscore_30_1 = (spread_mavg1 - spread_mavg30)/std_30
zscore_30_1.name = 'z-score'
zscore_30_1.plot()
plt.axhline(0, color='black')
plt.axhline(1.0, color='red', linestyle='--');

# Plot the prices scaled down along with the negative z-score
# just divide the stock prices by 10 to make viewing it on the plot easier
plt.plot(S1.index, S1.values/10)
plt.plot(S2.index, S2.values/10)
plt.plot(zscore_30_1.index, zscore_30_1.values)
plt.legend(['S1 Price / 10', 'S2 Price / 10', 'Price Spread Rolling z-Score']);
```

樣本外測試

現在我們已經適當地構建了我們的傳播並且知道我們將如何進行交易，現在是時候進行一些樣本測試了。 我們的整個模型基於這些證券是協整的前提，但我們是根據特定時期的信息建立的。 如果我們真的想要實現這個模型，我們需要進行一個樣本外測試，以確認我們模型的原理在未來仍然有效。

由於我們最初在2014年至2015年期間建立了該模型，讓我們看看這種協整關係是否適用於2015年至2016年。歷史結果並不能保證未來的結果，因此這是一個完整性檢查，以確定我們所做的工作是否具有強大的作用。

```
symbol_list = ['ABGB', 'FSLR']
prices_df = get_pricing(symbol_list, fields=['price']
                                , start_date='2015-01-01', end_date='2016-01-01')['price']
prices_df.columns = map(lambda x: x.symbol, prices_df.columns)

S1 = prices_df['ABGB']
S2 = prices_df['FSLR']

score, pvalue, _ = coint(S1, S2)
print 'p-value: ', pvalue
```

不幸的是，因為我們的p值高於0.05的截止值

，我們得出結論，由於我們選擇的證券之間缺乏協整，我們的模型將不再有效。如果我們嘗試在沒有基本假設的情況下部署此模型，我們就沒有理由相信它實際上會起作用。樣本測試是確保我們的工作在市場上真正可行的關鍵步驟。履行

當實際實施配對交易策略時，您通常希望一次交易許多不同的配對。如果通過分析數據找到良好的配對關係，則無法保證這種關係將在未來繼續存在。交易許多不同的貨幣對創造了多元化的投資組合，以減輕個別貨幣對“脫離”協整的風險。

本講座附有一個模板算法，展示瞭如何在我們的平台上實現配對交易的示例。請隨意檢查並使用您自己的配對進行修改，看看是否可以改進它。 進一步的研究

這個筆記本包含一些簡單的介紹方法。在實踐中，應該使用更複雜的統計數據，其中一些列在此處。