前言

> 資料主義（*Dataism*）認為，整個宇宙都是資料流（*data flows*）所構成；所有現象或實體的價值，全都取決於它對資料處理的貢獻……因此，資料主義可以說打破了動物（人）與機器的界限，更預期電子演算法最後終將破解、甚至超越生化演算法[1]。
>
> —— Yuval Noah Harari（以色列歷史學家）

找出正確的演算法，並以自動化的方式進行金融交易，可說是金融投資界一心想追求的「聖杯」。就在不久之前，當時還只有資金雄厚、管理大量資產的機構，才有足夠的能力運用演算法進行交易。但如今由於開放原始碼、開放資料、雲端計算、雲端儲存、線上交易平台等各領域最新的進展，讓一般小型機構與散戶交易者擁有了更公平的競爭環境，因此只要擁有一台筆記型電腦或 PC，加上可靠的網路連接，就可以進入這個引人入勝的領域。

目前，Python 及其強大的套件生態體系，已成為演算法交易的首選技術平台。除此之外，Python 還能讓你進行**高效率的資料分析**（例如使用 `pandas`：*http://pandas.pydata.org*），把機器學習的技術應用於**股市預測**（例如使用 `scikit-learn`：*http://scikit-learn.org*），甚至可透過 `TensorFlow`（*http://tensorflow.org*）導入 Google **深度學習**的技術。

1 Harari, Yuval Noah. 2015. *Homo Deus: A Brief History of Tomorrow*（人類大命運）. London: Harvill Secker.

這是一本關於如何把 Python 運用於「**演算法交易**」的書籍，主題圍繞在各種 ***alpha* 生成策略**（參見第 1 章[譯註]）。本書可說是身處兩大廣闊且令人興奮的領域交會處，因此幾乎不可能涵蓋所有相關的主題。不過，本書確實針對一些很重要的基本議題，提供了相當深入的內容。

本書內容涵蓋以下這些主題：

金融數據資料

金融數據資料是所有演算法交易專案的核心。在面對各種結構化金融數據資料時（包括每日收盤價格、盤中價格、高頻資料），只要善用 Python 和其他像是 `NumPy`、`pandas` 這樣的套件，就可以做出很好的處理。

回測（*Backtesting*）

如果交易策略未通過嚴格測試，就不應該用它來進行自動化演算法交易。本書會介紹簡單移動平均型、動量型、均值回歸型這幾種交易策略，並運用機器學習 / 深度學習的技術，來預測市場的動向。

即時資料

演算法交易必須有能力處理即時資料，而線上演算法（online algorithm）就是以即時資料為基礎，並以視覺化方式即時呈現當下的情況。本書採用 `ZeroMQ` 來說明 socket 程式設計的做法，並介紹如何以視覺化方式呈現串流資料。

線上平台

如果沒有交易平台，就無法進行交易。本書會介紹兩個相當受歡迎的電子交易平台：Oanda（*http://oanda.com*）與 FXCM（*http://fxcm.com*）。

自動化

演算法交易的優點與主要挑戰，就是交易操作的自動化。本書會示範如何在雲端部署 Python，以及如何針對自動化演算法交易，建立合適的環境。

本書具有以下這些功能與優點，可提供一些獨特的學習經驗：

涵蓋各大相關主題

本書是目前唯一針對 Python 演算法交易相關主題、兼具廣度與深度的書籍（參見後述）。

[譯註] alpha 指的是優於市場表現的「超額報酬」

可獨立運作的基礎程式碼

本書隨附一個 Git 程式碼儲存庫（*https://github.com/yhilpisch/py4at*），其中所有程式碼全都可以獨立運作執行。另外，讀者也可以在 Quant 平台（*http://py4at.pqp.io*）存取、執行本書的程式碼。

以真實交易為目標

本書會用到兩個不同的線上交易平台，讓讀者能以迅速有效的方式，開始進行理論與實際的交易操作。為了達到此目的，本書提供了許多實用而有價值的相關背景知識。

自己動手做、可自定進度的做法

由於本書所採用的資料與程式碼，全都可以各自獨立運作，而且只會用到一些標準的 Python 套件，因此讀者對於所有的處理過程、程式碼範例的使用與修改等等，全都可以充分理解與掌控。舉例來說，你不必依賴任何第三方平台，就可以連接到交易平台進行回測。有了這本書，讀者就可以靠自己輕鬆完成所有工作，並掌控每一行程式碼。

使用者論壇

雖然讀者應該有能力自行沿著道路前進，但本書作者與 Python Quants 平台還是可以隨時為你提供協助。讀者隨時都可以進入 Quant 平台（*http://py4at.pqp.io*）的使用者論壇，發表任何問題與評論（申請帳號完全免費）。

線上 / 影片訓練（需付費訂閱）

Python Quants 提供全面的線上訓練計劃（*https://oreil.ly/Qy90w*），該計劃採用本書所介紹的內容，並額外添加許多內容，涵蓋像是金融資料科學、金融人工智慧、可適用於 Excel 與資料庫的 Python 工具與技能等等重要主題。

本書的內容與結構

以下是本書各章主題與內容的快速介紹。

第 1 章：Python & 演算法交易

第一章介紹的「演算法交易」，就是以電腦演算法為基礎的自動化金融交易技術。本章會討論相關的基本概念，以及閱讀本書所需的預備知識。

第 2 章：*Python* 基礎架構

本章是隨後各章節的技術基礎，其中包括如何設定合適的 Python 環境。本章主要使用 conda 做為 Python 套件與虛擬環境的管理工具。本章也針對如何使用 Docker（*http://docker.com*）容器，以及如何在雲端部署 Python，做出了相應的說明。

第 3 章：金融數據資料的處理

對於每個演算法交易專案來說，時間序列資料總是扮演非常關鍵的角色。本章會介紹如何從不同的公開或專用資料來源，取得各種不同的金融數據資料。本章還會示範如何利用 Python，以更有效的方式儲存這些時間序列資料。

第 4 章：精通向量化回測

總體來說，向量化（Vectorization）可說是數值計算方面（尤其是金融財務分析）一種非常強大的做法。本章會介紹如何使用 NumPy 與 pandas 進行向量化操作，並把這樣的做法應用到簡單移動平均（SMA；Simple Mean Average）、動量（Momentum）、均值回歸（Mean-Reversion）等各類型交易策略的回測。

第 5 章：運用機器學習預測市場動向

本章致力於使用「機器學習」與「深度學習」技術，來預測市場的動向。這種做法主要依靠的是觀察過去報酬所隱含的特徵，再結合 TensorFlow（*https://oreil.ly/B44Fb*）、scikit-learn（*http://scikit-learn.org*） 與 Keras（*https://keras.io*） 這類的 Python 套件，藉以預測未來市場的動向。

第 6 章：打造事件型回測物件類別

雖然在程式碼簡潔性與效能表現方面，向量化回測的做法具有一定優勢，但如果需要呈現出交易策略的某些市場特徵，向量化回測的做法反而有點礙手礙腳。另一方面，只要運用物件導向程式設計技術，實作出事件型回測的做法，就可以針對各式各樣的市場特徵，進行更細化、更實際的模型化處理。本章會先針對事件型回測基礎物件類別進行介紹，然後再針對另外兩個物件類別（分別可用來回測「只做多」與「多空」交易策略）進行詳細的說明。

第 7 章：即時資料與 *Socket* 的處理

只要是有野心的演算法交易者，遲早都必須面對即時資料或串流資料的處理。以這方面來說，Socket 程式設計可說是首選的工具，因此本章介紹了 ZeroMQ（*http://zeromq.org*）這個輕量級的可擴展技術。本章還會說明如何利用 Plotly（*http://plot.ly*）建立美觀且具互動性的串流圖形。

第 *8* 章：運用 *Oanda* 交易 *CFD* 差價合約

Oanda（*http://oanda.com*）是一個外匯（forex；FX）與差價合約（CFD；Contracts for Difference）交易平台，可針對像是外匯組合、股票指數、商品、利率工具（基準債券）等投資工具進行交易。本章會使用 Python 包裝套件 `tpqoa`（*http://github.com/yhilpisch/tpqoa*），針對如何運用 Oanda 實作出自動化演算法交易策略，提供相應的指導。

第 *9* 章：運用 *FXCM* 進行外匯交易

FXCM（*http://fxcm.co.uk*）是外匯與 CFD 差價合約的另一個交易平台，最近 FXCM 還針對演算法交易發佈了最新的 RESTful API。這個平台可交易的投資工具涵蓋多種資產類別，例如外匯、股票指數、商品等等。它還提供一個 Python 包裝套件，讓演算法交易 Python 程式碼的撰寫變得相當方便而高效（*http://fxcmpy.tpq.io*）。

第 *10* 章：自動化交易操作

本章探討的是資本管理、風險分析管理，以及演算法交易操作技術自動化的一些典型任務。舉例來說，本章詳細介紹了資本配置與槓桿設定的凱利準則（Kelly Criterion）。

附錄

本附錄根據本書主要章節所呈現的內容，針對 Python、`NumPy`、`pandas` 其中最重要的一些主題，提供了簡要的介紹。這裡的介紹只是一個起點，各位可以從這裡開始，逐步增加自己的 Python 知識。

圖 P-1 顯示的是演算法交易相關的各層知識，而本書各章則是由下而上涵蓋了每一層的內容。首先必須從 Python 基礎架構（第 2 章）開始，然後再加上金融數據資料（第 3 章）、策略與向量化回測程式碼（第 4 與 5 章）的知識。到這裡為止，所有數據資料全都被當做一個整體來進行運用與操縱。事件型回測則首次引進一種想法，讓現實世界中的數據資料以陸續出現的方式來進行處理（第 6 章）。這可說是通往連線程式碼這一層的橋樑，到了此層就會涵蓋到 socket 通訊與即時資料處理的議題（第 7 章）。最重要的是，交易平台及其 API 都必須具有下單交易的能力（第 8、9 章）。本書最後則探討自動化與部署相關的重要內容（第 10 章）。從這個角度來看，本書的主要章節與圖 P-1 的各層皆有相關，而圖中各層也針對所要涵蓋的主題，排列出一個很合理的順序。

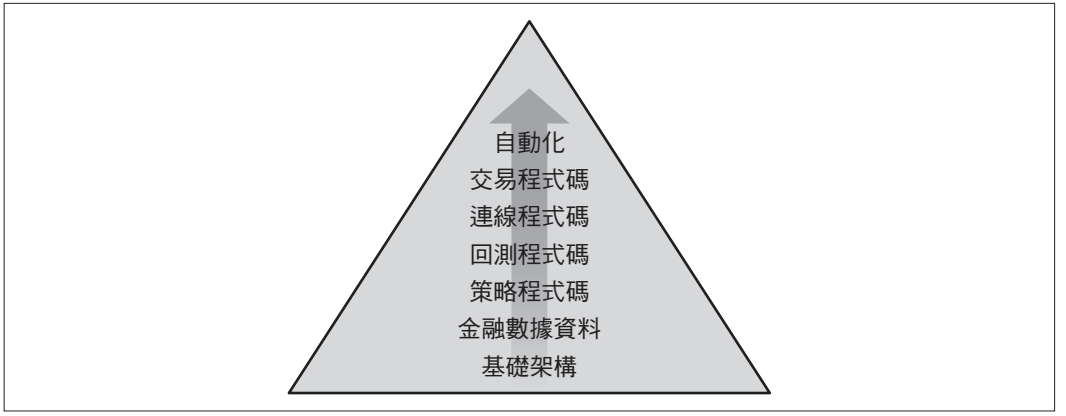

圖 P-1　Python 演算法交易的各層知識

本書適合的讀者

本書適合所有想把 Python 應用到演算法交易這個引人入勝領域的學生、學者與專業工作者。本書假設讀者在 Python 程式設計與金融交易方面，至少具有基本程度的背景知識。本書的附錄針對 Python、`NumPy`、`matplotlib`、`pandas` 介紹了一些重要的相關主題，可做為各位的參考與複習。下面還有一些很好的參考資料，可讓你對本書一些重要的 Python 相關議題獲得很好的理解。各位至少可以讀一下 Hilpisch（2018）這本書做為參考，大多數讀者應該都可以從中得到一些好處。關於可應用於演算法交易的一些機器學習與深度學習做法，Hilpisch（2020）這本書則提供了相當豐富的背景資訊與大量的具體範例。各位可以在下面這些書籍中，找到許多關於 Python 金融交易應用、資料科學與人工智慧的背景資訊：

Hilpisch, Yves. 2018. *Python for Finance: Mastering Data-Driven Finance*（Python 金融分析：掌握金融大數據）. 2nd ed. Sebastopol: O'Reilly.

Hilpisch, Yves. 2020. *Artificial Intelligence in Finance: A Python-Based Guide*（人工智慧在金融方面的應用：Python 指南）. Sebastopol: O'Reilly.

McKinney, Wes. 2017. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*（Python 資料分析：用 Pandas、NumPy、IPython 做資料分析）. 2nd ed. Sebastopol: O'Reilly.

Ramalho, Luciano. 2021. *Fluent Python: Clear, Concise, and Effective Programming*（流暢的 Python：清晰、簡潔、有效的程式設計）2nd ed. Sebastopol: O'Reilly.

第一章

Python & 演算法交易

在高盛從事股票交易的人數，已從 *2000* 年的 *600* 人，降到如今只剩 *2* 人[1]。

——《經濟學人》

本章打算針對本書所涵蓋的主題，提供相應的背景資訊與概念。雖然 Python 演算法交易是一種介於程式設計與金融投資之間的跨領域應用，但此領域進展非常迅速，其中牽涉到許多不同議題，像是 Python 部署、互動式財務分析、機器學習、深度學習、物件導向程式設計、Socket 通訊、串流資料視覺化呈現、交易平台等等。

如果想快速瞭解 Python 相關的重要概念，請先閱讀附錄。

Python 與金融界的淵源

Python 程式語言起源於 1991 年，最早是 Guido van Rossum 發佈標記為 0.9.0 的版本。1994 年，又出現了 1.0 版。不過，Python 用了將近二十年時間，才成為金融業主要的程式語言與技術平台。當然，也有人（主要是對沖基金）很早就開始採用，但是到了 2011 年左右，Python 才開始受到廣泛的運用。

金融業早期並未大量採用 Python，其中一個主要障礙在於，Python 的預設版本 CPython 屬於一種直譯式（interpret）的高階語言。一般來說，數值演算法（尤其是金融財務演算法）經常在程式碼中使用（巢狀）迴圈結構。像 C 或 C ++ 這類採用編譯式

1 “Too Squid to Fail.”（實在太會吸金，再怎樣也不會倒）*The Economist*, 29. October 2016.

（compile）做法的低階語言，執行迴圈的速度真的非常快，而 Python 靠的是直譯而非編譯，執行的速度通常就會慢很多。以結果來看，對於現實世界許多金融應用來說（例如選擇權定價或風險管理），純粹採用 Python 的做法實在是太慢了。

Python vs. 偽程式碼

雖然 Python 本來就不是針對科學與金融界而設計，但還是有很多人喜愛其優美的語法與簡潔性。過去有一段時間大家都認為，在設計或解釋（金融）演算法的過程中，先採用一些偽程式碼來做為實現技術的中間步驟，是一種很好的傳統做法。後來又有許多人認為，只要使用 Python，就可以跳過偽程式碼的步驟。事實證明，這基本上是正確的。

舉例來說，我們來考慮一下幾何布朗運動（geometric Brownian motion）的尤拉離散化公式（Euler discretization），如方程式 1-1 所示。

方程式 *1-1*　幾何布朗運動的尤拉離散化公式

$$S_T = S_0 \exp\left(\left(r - 0.5\sigma^2\right)T + \sigma z\sqrt{T}\right)$$

在編寫一些帶有數學公式的科學文件時，LaTeX 標記語言與編譯器幾十年來一直都是所謂的黃金標準。在許多方面（例如方程式佈局——如方程式 1-1 所示），Latex 語法本身就已經頗具有偽程式碼的效果。在這個特別的例子中，相應的 Latex 語法如下：

```
S_T = S_0 \exp((r - 0.5 \sigma^2) T + \sigma z \sqrt{T})
```

若採用 Python 的寫法，只要定義好相應的變數，這個方程式也可以轉換成可執行的程式碼，而且這段程式碼不但與方程式本身很接近，與 Latex 的表達方式也很相像：

```
S_T = S_0 * exp((r - 0.5 * sigma ** 2) * T + sigma * z * sqrt(T))
```

不過，速度依舊是很大的問題。像這樣的一個差分方程式，經常被用來做為隨機微分方程式的數值近似式，而在進行蒙地卡羅模擬時，通常會被用來計算價格導函數，或是被用來做為模擬的基礎，以進行風險分析與管理[2]。這些工作往往需要好幾百萬次的模擬，而且必須在一定的時間內完成（通常幾乎是即時，或至少是接近即時）。Python 做為一種直譯式的高階程式語言，在設計上天生就不具備足夠快的速度，因此很難處理好這樣的計算工作。

2　詳細訊息請參見 Hilpisch（2018，第 12 章）。

NumPy & 向量化

2006 年，Travis Oliphant 發表了 NumPy 這個 Python 套件的 1.0 版（*http://numpy.org*）。NumPy 就是 *numerical Python*（數值 Python）的意思，代表它特別適合一些對數值有嚴格要求的應用場景。原本 Python 直譯器本身的設計，只希望能夠盡量通用於許多不同的領域，但這樣的訴求經常會在執行階段造成相當大的額外開銷[3]。NumPy 則從另一個角度切入，它主要是針對一些可避免額外開銷的特定情況，提供一些專用的做法，以便在特定的應用程式場景下，達到又快又好的效果。

NumPy 最主要的物件類別就是所謂的 ndarray 物件，它其實就是 *n* 維的 *array*（陣列）物件。它是不可變的（immutable），也就是其大小不能改變，而且只能容納同一種資料型別（即所謂的 dtype）。這種專用的做法，有助於實作出簡潔而快速的程式碼。由此所衍生的其中一種主要運用方式，就是所謂的**向量化**（*vectorization*）操作。基本上，這種運用方式可以讓 Python 避免使用迴圈，因為迴圈的工作可以交給 NumPy 的專用程式碼，而這部分通常是以 C 語言實作，因此速度快很多。

我們可以先使用純粹的 Python，根據方程式 1-1 計算 1,000,000 次 S_T 的值。以下程式碼最主要的部分就是一個 for 迴圈，其中總共進行了 1,000,000 次的迭代操作：

```
In [1]: %%time
        import random
        from math import exp, sqrt

        S0 = 100  ❶
        r = 0.05  ❷
        T = 1.0  ❸
        sigma = 0.2  ❹

        values = []  ❺

        for _ in range(1000000):  ❻
            ST = S0 * exp((r - 0.5 * sigma ** 2) * T +
                          sigma * random.gauss(0, 1) * sqrt(T))  ❼
            values.append(ST)  ❽
        CPU times: user 1.13 s, sys: 21.7 ms, total: 1.15 s
        Wall time: 1.15 s
```

❶ 指數水準的初始值。

3 舉例來說，list 列表物件不只是可變的（mutable，代表其大小可以改變），而且其中的元素幾乎可以是任何種類的 Python 物件（例如 int、float、tuple 元組物件或 list 列表物件）。

❷ 固定的短期利率。

❸ 時間跨度（以年為單位）。

❹ 固定的波動率因子。

❺ 一個空的 list 列表物件，用來收集所計算出來的值。

❻ 主要的 for 迴圈。

❼ 模擬出單一個期末（end-of-period）值。

❽ 把計算出來的值放入 list 列表物件中。

只要使用 NumPy，就可以採用向量化的做法，讓 Python 完全不必執行迴圈。這樣一來程式碼也會變得更簡潔易讀，而且速度更提高了八倍左右：

```
In [2]: %%time
        import numpy as np

        S0 = 100
        r = 0.05
        T = 1.0
        sigma = 0.2

        ST = S0 * np.exp((r - 0.5 * sigma ** 2) * T +
                         sigma * np.random.standard_normal(1000000) *
                         np.sqrt(T))  ❶
        CPU times: user 375 ms, sys: 82.6 ms, total: 458 ms
        Wall time: 160 ms
```

❶ 這樣的一行 NumPy 程式碼，就可以計算出所有的值，並把結果儲存在一個 ndarray 物件中。

向量化是一種非常強大的概念，尤其是針對金融與演算法交易，可編寫出十分簡潔易讀且易於維護的程式碼。只要使用 NumPy 寫出向量化程式碼，不但可以讓程式碼更簡潔，還可以明顯加快程式碼的執行速度（例如在蒙地卡羅模擬中大約可提高八倍的速度）。

我們可以很肯定地說，Python 在科學與金融領域的成功，NumPy 肯定有絕大的貢獻。在所謂的**科學 Python 套件組合**（*scientific Python stack*）中，有許多很受歡迎的 Python 套件，都是以 NumPy 做為構建的基礎；無論是儲存或處理數值資料，NumPy 都是一種極為高效的資料結構。事實上，NumPy 其實是 SciPy 這個套件專案的產物，而 SciPy 本身則是在

科學方面提供了大量經常用到的功能。SciPy 專案意識到有必要提供一個更強大的數值資料結構，因此就把 Numeric 與 NumArray 這幾個相關的老專案，整合成 NumPy 這個全新而統一的形式。

在演算法交易領域，蒙地卡羅模擬或許還不算是程式語言最重要的應用情境。但如果你進入到演算法交易的領域，管理超大量金融時間序列資料肯定是一個非常重要的應用情境。你只要想一下（盤中）交易策略的回測，或是交易期間 tick 資料串流的處理，就可以明白我的意思了。而這正是 pandas 資料分析套件（*http://pandas.pydata.org*）可以派上用場之處。

Pandas & DataFrame 物件類別

pandas 是 2008 年由 Wes McKinney 開始進行開發，當時他在 AQR 資本管理公司（AQR Capital Management）工作，那是一家位於康乃狄克州格林威治的大型避險基金。與其他任何避險基金一樣，處理時間序列資料對於 AQR 資本管理公司來說至關重要，但當時 Python 尚未針對此類資料提供任何足夠有用的支援。Wes 的想法是建立一個可以在該領域模仿 R 統計語言（*http://r-project.org*）功能的套件。舉例來說，其主要物件類別 DataFrame 的名稱就可以反映出這種模仿的想法，因為它在 R 語言中對應的就是 data.frame。由於 AQR 資本管理公司認為，這部分的成果與該公司資金管理的核心業務並沒有很密切的關係，因此就在 2009 年開放了 pandas 專案的原始碼，而這也成為了開放原始碼資料與財務分析獲得重大成功的開端。

如今 Python 已成為資料與財務分析的主要力量，其中有一部分原因正是因為 pandas。原本使用其他各種語言、後來決定採用 Python 的許多人士，都把 pandas 視為他們當初做出決定的主要理由。如果結合像是 Quandl（*http://quandl.com*）這類的開放資料來源，pandas 甚至可以讓學生們以最低的進入門檻進行複雜的財務分析：只要有一台普通的筆記型電腦，再加上網際網路連接就足夠了。

假設有一個演算法交易者，他對交易比特幣（目前市值最大的加密貨幣）很感興趣。他的第一步很可能就是找出比特幣與美元歷史匯率的相關資料。只要使用 pandas 檢索出 Quandl 裡的資料，他就可以在不到一分鐘的時間內完成任務。圖 1-1 顯示的就是以下 Python 程式碼所生成的圖形，只用了四行的程式碼（省略了一些與繪圖風格相關的參數設定）。雖然這裡並沒有以明確的方式匯入 pandas，但在預設情況下，Quantl 這個 Python 包裝套件就會送回一個 DataFrame 物件，然後我們加上一條 100 日的簡單移動平均線（SMA），再以視覺化的方式把原始資料與 SMA 一起呈現出來：

```
In [3]: %matplotlib inline
        from pylab import mpl, plt  ❶
        plt.style.use('seaborn')  ❶
        mpl.rcParams['savefig.dpi'] = 300  ❶
        mpl.rcParams['font.family'] = 'serif'  ❶

In [4]: import configparser  ❷
        c = configparser.ConfigParser()  ❷
        c.read('../pyalgo.cfg')  ❷
Out[4]: ['../pyalgo.cfg']

In [5]: import quandl as q  ❸
        q.ApiConfig.api_key = c['quandl']['api_key']  ❸
        d = q.get('BCHAIN/MKPRU')  ❹
        d['SMA'] = d['Value'].rolling(100).mean()  ❺
        d.loc['2013-1-1':].plot(title='BTC/USD exchange rate',
                                figsize=(10, 6));  ❻
```

❶ 匯入並設定繪圖套件。

❷ 匯入 configparser 模組並讀取憑證。

❸ 匯入 Quandl 這個 Python 包裝套件，並提供 API 密鑰。

❹ 找出比特幣匯率的每日資料，並送回一個只有單一縱列的 pandas DataFrame 物件。

❺ 以向量化的方式計算出 100 天的 SMA。

❻ 從 2013 年 1 月 1 日開始選取資料，並畫出相應的圖形。

NumPy 與 pandas 這兩個套件，顯然為 Python 在金融領域的成功，做出了相當可觀的貢獻。不過，Python 整個生態體系也透過其他 Python 套件的形式，提供了許多其他的功能，其中有些可解決相當基礎的問題，有些則可用來解決某些特定的問題。本書會用到一些可檢索與儲存資料的套件（例如 PyTables、TsTables、SQLite），以及一些機器學習與深度學習相關的套件（例如 scikit-learn、TensorFlow）。在此過程中，我們也會實作出一些物件類別與模組，讓所有演算法交易專案都能因此而變得更有效率。不過在整個過程中一直都會用到的主要套件，就是 NumPy 與 pandas。

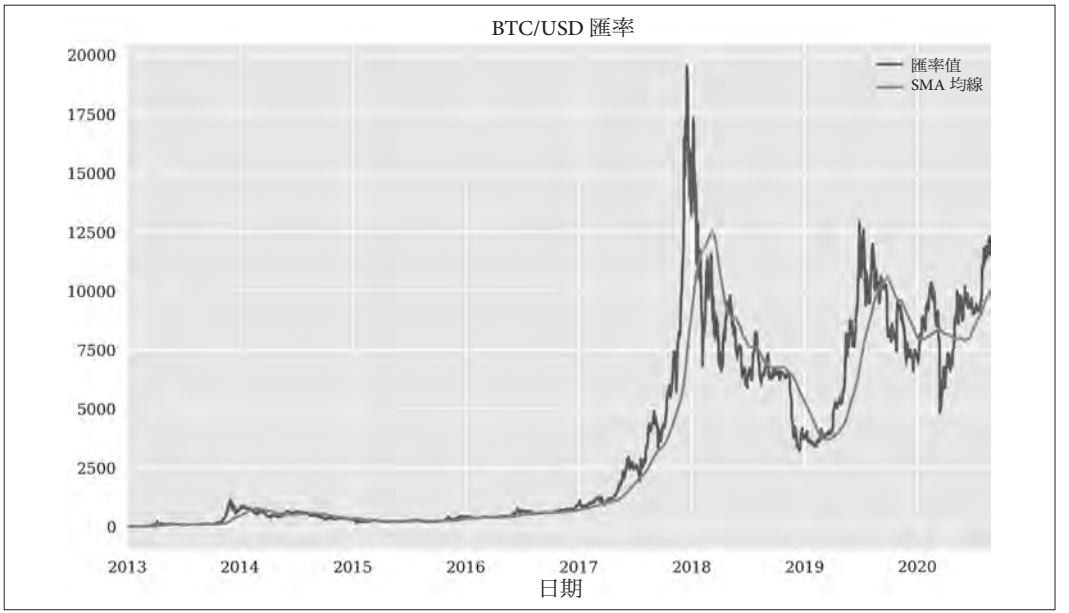

圖 1-1　從 2013 年初到 2020 年中為止，比特幣美元匯率的歷史記錄

雖然 NumPy 提供了基本的資料結構，可用來儲存數值資料並進行處理，但針對表格型資料來說，pandas 還是提供了更強大的時間序列管理能力。把其他套件的功能包裝成更容易使用的 API，這方面它也做得很好。從剛剛所提到的比特幣範例中就可以看到，只需調用 DataFrame 物件的單一方法，就可以針對兩個金融財務時間序列，製作出相應的視覺化圖形。pandas 就像 NumPy 一樣，可以讓使用者以相當簡潔的方式寫出向量化程式碼，而且 pandas 使用了大量編譯過的程式碼，因此執行的速度通常也很快。

演算法交易

演算法交易（*algorithmic trading*）這個術語既沒有唯一而獨特的定義，也不存在普遍的定義。在某種程度上，它指的就是以特定形式的演算法為基礎、針對特定的金融投資工具進行交易。而所謂的「**演算法**」，指的就是以特定順序進行一整套「數學上、技術上」的操作，以達到特定的目標。舉例來說，有一些數學演算法可以解決魔術方塊的問

題[4]。像這樣的演算法，通常可以透過逐步的程序完美解決問題。另一個演算法的例子，就是求解方程式的根（如果存在的話）。從這個意義上來說，數學演算法的目標通常都很明確，而且通常可預期會有最佳解。

但金融交易演算法的目標又是什麼呢？總體來說，這個問題並不容易回答。此時我們若退一步考慮一下一般人的交易動機，或許有點幫助。在 Dorn 等人（2008 年）的文章中寫道：

> 金融市場交易是一種很重要的經濟活動。交易需要進出市場，把目前用不到的現金投入市場，然後在需要用錢時再換回現金。交易也可以在市場內轉移資金、把某種資產換成另一種資產，藉以迴避風險或善用未來價格變動相關的一些資訊。

這裡所表達的觀點，本質上比較偏技術而非經濟，主要關注的是程序本身，而不是很在意人們交易的目的。如果要從交易的目的來看，無論是管理自己金錢的個人，或是管理他人金錢的金融機構，這裡針對大家想要交易的動機，列出了一個詳盡的列表如下：

Beta 交易

賺取市場風險溢價，例如投資 ETF（exchange traded funds）這種複製 S&P 500 指數表現的投資標的。

Alpha 生成

賺取與市場無關的風險溢價，例如放空 S&P 500 指數上市股票或 S&P 500 指數相應的 ETF。

靜態避險

針對市場風險進行避險操作，例如買進 S&P 500 的價外賣權選擇權。

動態避險

針對影響 S&P 500 選擇權的市場風險進行避險操作，例如以動態的方式交易 S&P 500 的期貨，同時保留適當的現金，或針對貨幣市場、利率工具進行操作。

資產負債管理

交易 S&P 500 指數股票與 ETF，以便能夠補償因購買人壽保險而產生的負債。

---

4 請參見《The Mathematics of the Rubik's Cube（魔術方塊的數學）》（*https://oreil.ly/16pIA*）或《Algorithms for Solving Rubik's Cube（解決魔術方塊的演算法）》（*https://oreil.ly/XM0ZP*）。

造市（*Market making*）

> 舉例來說，以不同的買進報價與賣出報價來買賣 S&P 500 選擇權，為選擇權提供一定的流動性。

以上這幾種類型的交易，全都可透過不同的方式來執行，其中人類交易者主要是靠自己來做決定，但也有人會採用演算法來提供支援，甚至在決策過程中完全用演算法取代人類進行交易。在這樣的背景下，金融交易電腦化當然扮演了非常重要的角色。早期的金融交易，一群人互相大喊大叫（公開喊價）的場內交易（floor trading）是執行交易唯一的方式，而電腦化與網路技術的出現，則徹底改變了金融業的交易方式。本章開頭的引言曾提到高盛積極從事股票交易的人數，從 2000 年到 2016 年的變化可說是令人印象深刻。如 Solomon 與 Corso（1991）所言，這是 25 年前就可以預見的趨勢：

> 電腦已徹底改變證券交易，而股票市場也處於快速轉換的過程之中。未來的市場顯然不會再像過去一樣。
>
> 由於技術上的進展，股票價格相關資訊在幾秒內就可以發送到全世界。目前經紀商已經可以直接從電腦終端連到交易所下單並執行小額的交易。如今電腦已經把各大證券交易所相連起來，這肯定有助於建立一個單一的全球證券交易市場。技術不斷改進，讓電子交易系統在全球範圍內執行交易，成為了一件可能的事。

有趣的是，在選擇權動態避險的做法中，有一個最古老、使用最廣泛的演算法。早在電腦化與電子交易開始之前，Black 與 Scholes（1973）與 Merton（1973）就發表了關於歐式選擇權定價的開創性論文，其中介紹了一種稱為「*delta* 避險」（*delta hedging*）的演算法。Delta 避險這樣的一種交易演算法，向我們展示了如何在一個簡化、完美、連續的模型化世界中，利用避險套利的方式迴避掉所有的市場風險。在現實世界中，雖然存在交易成本、離散交易、不完美的市場流動性及其他不完美的因素，但這個演算法仍被證明是個有用且穩當的做法，這可說是相當令人出乎意料。這個演算法或許無法完全消除所有影響選擇權的市場風險，但在接近理想的狀態下它確實很有用，因此至今在金融業仍受到廣泛的運用[5]。

本書會以 ***alpha* 生成策略**為背景，聚焦於相應的演算法交易。雖然 alpha 的定義很複雜，但就本書而言，我們會把 alpha 定義為某交易策略在一段時間內的報酬，與相應比較基準（有可能是單一股票、指數、加密貨幣等）兩者報酬之間的差異。舉例來說，如果 S&P 500 指數在 2018 年的報酬為 10%，而演算法策略的報酬為 12%，那麼 alpha 就

5 參見 Hilpisch（2015）的著作，其中利用 Python 詳細分析了歐式與美式選擇權的 Delta 避險策略。

是 +2% 點。如果策略的報酬為 7%，alpha 就是 -3% 點。一般來說，此類數字並不會針對風險進行調整，而其他像是回檔（drawdown）最大跌幅（與最長持續時間）這類的風險特徵，其重要性則通常被擺在第二級（如果有的話）。

本書側重於 alpha 生成策略，或是能夠生成正報酬（高於比較基準）且其表現與市場起伏無關的一些策略。本書（採用最簡單的方式）把 alpha 定義為策略超出比較基準的超額報酬。

交易演算法在其他領域也扮演著很重要的角色。**高頻交易**（HFT）就是其中之一；以高頻交易來說，速度通常就是勝負的關鍵[6]。人們從事高頻交易的動機或許各不相同，但「造市」與「alpha 生成」應該可算是其中兩個很重要的動機。**交易執行面的考量**也有可能是另一種動機，因為有時候就是必須透過某些演算法，才能以最佳的方式執行某些**非標準交易**。譬如在大量下單的過程中，如果希望盡量取得最佳的價格，或希望盡量縮小交易本身對市場價格的影響，這些全都可以算是交易執行面的動機。另外也有人會選擇在許多不同的交易所下單，藉以掩飾其下單的行動，這或許也可以算是另一種微妙的動機。

還有一個重要的問題有待解決：演算法能否取代人類的研究、經驗與判斷力，在交易時呈現出一定的優勢？這個問題很難一概而論。可以肯定的是，確實有一些人類交易者與投資組合經理，能夠在很長的一段時間內，讓平均報酬維持在優於一般投資者的水準。以這方面來說，華倫．巴菲特就是最明顯的一個例子。另一方面，根據統計分析顯示，大多數投資組合經理很少有人能始終如一擊敗相應的比較基準。2015 年時 Adam Shell 就寫道：

> 舉例來說，去年 *S&P 500* 指數的總報酬只有 *1.4%*（包括股息），可說是相當微不足道，而當時「主動管理」的大型公司股票基金，有 *66%* 的表現比該指數還差……即使看得更長遠一點，結果同樣令人感到沮喪；研究發現最近五年內，有 *84%* 的大型資本基金，其報酬比 *S&P 500* 指數還低，而過去 *10* 年也有 *82%* 的報酬比不上同一個比較基準[7]。

6 關於高頻交易（HFT），如果想找比較沒那麼偏技術性的介紹，可參見 Lewis（2015）的著作。

7 資料來源：〈66% of Fund Managers Can't Match S&P Results.〉（66% 的基金經理無法達到 S&P 的表現）*USA Today*《今日美國》，2016 年 3 月 14 日。

在 2016 年 12 月發表的一項實證研究中，Harvey 等人也寫到：

> 我們針對各大全權委託型基金與系統型避險基金，用相應的績效表現進行了對比分析。系統型基金使用的策略是以規則為基礎，幾乎沒有人會每天進行干預 ... 我們發現，在 *1996-2014* 年期間，系統型基金經理人在未調整的（原始）報酬方面，表現要比全權委託型的對手差，但針對眾所周知的風險因素進行過風險調整之後，表現則很相近。就宏觀而言，無論是未經調整或風險調整後，系統型基金的表現均優於全權委託型基金。

表 1-0 用實際的數字重現了 Harvey 等人（2016）研究的主要發現[8]。在這個表格中，所謂的特定**因素**包括傳統因素（股票、債券等）、動態因素（價值、動量等），以及波動率因素（買進價平賣權與買權）。只要讓 α 除以調整後報酬波動率，就可以得到**調整後報酬鑑定比率**（*adjusted return appraisal ratio*）。更多詳細的訊息與背景，請參見原始的研究文獻。

研究結果表示，無論有沒有針對風險進行過調整，系統型（也就是採用演算法的）宏觀避險基金都是其中表現最佳的一類。在研究期間，其年化 alpha 值為 4.85% 點。這類避險基金所採用的策略通常都具有全球性，會採用交叉資產的做法，而且通常會考慮政治與宏觀經濟因素。系統型股權避險基金則只有在調整後報酬鑑定比率方面，擊敗了全權委託型股權避險基金（0.35 對 0.25）。

	系統型宏觀	全權委託型宏觀	系統型股權	全權委託型股權
平均報酬	5.01%	2.86%	2.88%	4.09%
可歸因於特定因素的報酬	0.15%	1.28%	1.77%	2.86%
調整後平均報酬（alpha）	4.85%	1.57%	1.11%	1.22%
調整後報酬波動率	0.93%	5.10%	3.18%	4.79%
調整後報酬鑑定比率	0.44	0.31	0.35	0.25

相較於 S&P 500，避險基金在 2017 年的整體表現相當差。S&P 500 指數的報酬為 21.8%，避險基金卻只給了投資者 8.5% 的報酬（參見投資百科 *Investopedia* 裡的這篇文章：*https://oreil.ly/N59Hf*）。這也就說明了，即使有好幾百萬美元預算投入研究與技術，想要生成 alpha 還是十分困難。

8 1996 年 6 月至 2014 年 12 月期間，避險基金類別的年化表現（高於短期利率的部分）與風險衡量結果，其中包括總計 9,000 個避險基金。

Python 演算法交易

Python 已被運用於金融業許多角落，不過在演算法交易領域中，還是特別受歡迎。這其中有幾個很好的理由：

資料分析能力

每個演算法交易專案的主要需求，就是有效管理與處理金融數據資料的能力。相較於大多數其他程式語言，Python 結合了 `NumPy` 與 `pandas` 這類的套件，讓每一種演算法的交易者都變得更加輕鬆。

現代化的 *API* 處理方式

諸如 FXCM（*http://fxcm.co.uk*）與 Oanda（*http://oanda.com*）這類的現代化線上交易平台，都有提供 RESTful API 與 Socket 相關（串流）API，以存取歷史資料與即時資料。Python 通常很適合與此類 API 進行有效的互動。

專用套件

除了標準的資料分析套件之外，還有許多演算法交易領域專用的套件，例如 `PyAlgoTrade`（*https://oreil.ly/IpIt1*）與 `Zipline`（*https://oreil.ly/2cSKR*）可用來進行交易策略的回測，而 `Pyfolio`（*https://oreil.ly/KT7V8*）則可用來進行投資組合與風險分析。

廠商贊助套件

在這個領域中，有越來越多廠商發表開放原始碼的 Python 套件，以協助使用者對其產品進行存取。其中包括 Oanda 之類的線上交易平台，以及 Bloomberg（*https://oreil.ly/oSxei*）與 Refinitiv（*https://oreil.ly/1SNBN*）這些具有領導地位的資料供應商。

專用平台

舉例來說，Quantopian（*http://quantopian.com*）就提供了一個標準化的回測環境，它是一個 Web 平台，所選用的語言就是 Python，大家可以在那裡透過不同的社群網路功能，與志同道合的人交換想法。從成立到 2020 年為止，Quantopian 已經吸引了超過 30 萬的使用者。

買方與賣方皆採用

越來越多投資機構採用 Python，以促進其交易部門的開發工作更加順利。反過來說，市場也需要越來越多的 Python 熟練者，因此學習 Python 成為了一項很有價值的投資。

教育、訓練與書籍

如果某種技術或程式設計語言想要被廣泛採用，其前提就是要有足夠的學術資源與專業教育，結合專業書籍與其他資源的訓練計劃。近來整個 Python 生態體系在這方面取得了巨大的成長，有越來越多人針對金融方面的 Python 應用，接受了各式各樣的教育與訓練。我們可以預期，這一定會更加強化演算法交易領域採用 Python 的趨勢。

總而言之，我們可以很肯定地說，Python 已經在演算法交易中扮演了十分重要的角色，而且未來似乎還會有很強大的動量，變得越來越重要。因此，對於任何想進入這個領域的人來說，無論是做為一個雄心勃勃的「散戶」，或是在從事系統交易的領先金融機構中做為一個專業交易者，這都是個不錯的選擇。

預備知識與聚焦重點

本書的重點是把 Python 做為演算法交易的程式語言。書中會假設讀者已經對 Python 的使用，以及資料分析相關的 Python 流行套件有一定的經驗。像 Hilpisch（2018）、McKinney（2017）、VanderPlas（2016）這幾本都是不錯的入門書籍，可以讓你在使用 Python 進行資料分析與金融應用方面打下堅實的基礎。我們也期望讀者對於 Python 用來進行互動分析的典型工具（例如 `IPython`）有一些經驗，在這方面 VanderPlas（2016）也提供了一些介紹。

本書會針對所談論的主題（例如回測交易策略或處理串流資料）提供 Python 程式碼與相應的說明。針對各處所使用到的套件，本書也許無法一一進行全面性的介紹。不過我們還是會針對說明的主題，強調其中特別重要的套件功能（例如「用 `NumPy` 進行向量化操作」）。

本書也無法針對演算法交易相關的所有金融操作方面，提供詳盡的介紹。我們在做法上會比較側重於如何使用 Python，打造出自動化演算法交易系統必要的基礎架構。當然，本書所用到的大多數範例，均來自演算法交易這個領域。不過在處理動量型或均值回歸型策略時，或多或少只會簡單採用相應的範例，而不會進行嚴謹的（統計）驗證，或對其複雜性進行深入的討論。針對解說期間未能完全解決的問題，只要有適當的機會，本書就會提供相應的參考文獻，指引讀者深入問題的根源。

總而言之，本書是針對擁有 Python 與（演算法）交易經驗的讀者所編寫的。對於這類讀者來說，本書就是使用 Python 與其他各種套件、建立自動化交易系統的實用指南。

本書使用了許多 Python 程式設計做法（例如物件導向程式設計）與套件（例如 `scikit-learn`），無法一一詳細解釋。我們的焦點是如何把這些做法與套件應用到演算法交易程序裡的不同步驟。因此，建議所有對 Python（尤其金融應用方面）還沒有足夠經驗的人，自行參考其他更多 Python 相關的介紹文件。

交易策略

本書會以四種不同的演算法交易策略做為範例。在以下各節會進行簡要的介紹，第 4 章則會進行更詳細的說明。所有這些交易策略全都可以歸類為**追求 *alpha* 型策略**，因為其主要目標都是希望能取得與市場動向無關、高於市場表現的正向報酬。當我們談到所交易的金融投資工具時，本書最典型的範例就是**股票指數**、**單一股票**或**加密貨幣**（透過法定貨幣來表示）。本書並沒有討論到同時涉及多種金融投資工具的策略（例如成對交易策略、以整籃子為基礎的策略等）。而且本書也只討論結構化金融時間序列資料所得出交易信號衍生出來的相關策略，而不會討論像是新聞、社群媒體動態訊息等這類非結構化資料來源相關的策略。這樣的做法才能讓相關討論與 Python 實作更加簡潔而易於理解；我們之前曾提過要專注於如何使用 Python 進行演算法交易，其實這些背後的想法都是一致的[9]。

本章接下來其餘的部分，將會快速瀏覽本書所使用的四種交易策略。

9 關於演算法交易相關主題的總覽，可參見 Kissel（2013）的著作；關於動量型與均值回歸型策略更深入的討論，可參見 Chan（2013）的著作；Narang（2013）探討演算法交易的著作，則涵蓋了計量交易與高頻交易的相關介紹。

簡單移動平均

第一種交易策略就是靠著簡單移動平均（SMA）來生成交易信號，以建立市場部位。像這樣的交易策略，已被所謂的技術分析師或線圖專家廣泛採用。其基本構想就是，當短線 SMA 的價格突破長線 SMA 的價格時，就表示出現了市場偏多的訊號，相反的情況則表示出現中性或市場偏空的訊號。

動量

動量型策略背後的基本構想其實是一個假設，這個假設認為金融投資工具一般都會參考近期的表現，而傾向於讓表現繼續維持一段時間。舉例來說，如果某個股票指數在過去五天內平均報酬為負，就可以假設它明天的表現應該也會是負的。

均值回歸

在均值回歸型策略中我們會假設，如果金融投資工具當前的價格偏離某平均值或趨勢水準的距離足夠遠，它應該就會傾向於往回頭的方向折返。舉例來說，假設某股票的交易價格比它的 200 日簡單移動平均 100 美元低了 10 美元。接下來就可以預期，股價應該很快就會回到簡單移動平均的價格水準附近。

機器學習與深度學習

透過機器學習與深度學習演算法，通常就可以用更黑箱的方式來預測市場動向。由於考慮到單純性與重現性，本書的範例主要是靠著觀察歷史報酬的方式，來訓練機器學習與深度學習演算法，以預測股市的動向。

本書並沒有以系統化的方式介紹演算法交易。由於重點在於如何把 Python 應用到這個引人入勝的領域，因此讀者若不熟悉演算法交易，就應該自行探尋這些主題相應的資源，或是進一步參考本章與隨後各章所引用的資源。不過，也請各位特別留意以下事實：一般來說，演算法交易的世界總是充滿神秘的色彩，而且幾乎每個成功的人都不大願意分享自己的秘密，因為這樣才能保護他們成功的源頭（也就是他們的 alpha）。

結論

總體而言，Python 已成為金融領域不可忽視的一股力量，而且正逐漸成為演算法交易的主要力量。使用 Python 來進行演算法交易，有許多非常充分的理由，其中包括強大的套件生態體系，可有效進行各種資料分析，或使用許多現代化的 API 進行處理。學習 Python 演算法交易有很多好理由，其中一個很重要的事實就是，一些最大的交易機構在他們的交易操作中，會大量使用到 Python，而且還會不斷尋找經驗豐富的 Python 專業人士。

本書著重於如何把 Python 應用到演算法交易的不同面向，例如如何回測交易策略，或是如何與線上交易平台進行互動。本書並沒有針對 Python 本身進行全面性的介紹，也無法涵蓋所有關於交易的知識。不過本書以系統化的方式結合了這兩個引人入勝的世界，針對當今這個競爭激烈的金融與加密貨幣市場，為 alpha 生成型策略提供了寶貴的資源。

參考資料與其他資源

本章所引用的書籍與論文如下：

Black, Fischer, and Myron Scholes. 1973. “The Pricing of Options and Corporate Liabilities.”（選擇權與公司債的定價）*Journal of Political Economy* 81 (3): 638-659.

Chan, Ernest. 2013. *Algorithmic Trading: Winning Strategies and Their Rationale*（演算法交易：贏家策略及其原理）. Hoboken et al: John Wiley & Sons.

Dorn, Anne, Daniel Dorn, and Paul Sengmueller. 2008. “Why Do People Trade?（人們為什麼要交易？）” *Journal of Applied Finance* (Fall/Winter): 37-50.

Harvey, Campbell, Sandy Rattray, Andrew Sinclair, and Otto Van Hemert. 2016. “Man vs. Machine: Comparing Discretionary and Systematic Hedge Fund Performance（人類 vs. 機器：全權委託型與系統型避險基金績效表現的比較）.” *The Journal of Portfolio Management* White Paper, Man Group.

Hilpisch, Yves. 2015. *Derivatives Analytics with Python: Data Analysis, Models, Simulation, Calibration and Hedging*（Python 衍生性金融商品分析：資料分析、模型化、模型化、校正與避險）. Wiley Finance. Resources under *http://dawp.tpq.io*.

Hilpisch, Yves. 2018. *Python for Finance: Mastering Data-Driven Finance*（Python 金融分析：掌握金融大數據）. 2nd ed. Sebastopol: O'Reilly. Resources under *https://py4fi.pqp.io*.

Hilpisch, Yves. 2020. *Artificial Intelligence in Finance: A Python-Based Guide*（人工智慧在金融方面的應用：Python 指南）. Sebastopol: O'Reilly. Resources under *https://aiif.pqp.io*.

Kissel, Robert. 2013. *The Science of Algorithmic Trading and Portfolio Management*（演算法交易與投資組合管理背後的科學基礎）. Amsterdam et al: Elsevier/Academic Press.

Lewis, Michael. 2015. *Flash Boys: Cracking the Money Code*（快閃男孩：破解金錢密碼）. New York, London: W.W. Norton & Company.

McKinney, Wes. 2017. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*（Python 資料分析：用 Pandas、NumPy、IPython 做資料分析）. 2nd ed. Sebastopol: O'Reilly.

Merton, Robert. 1973. "Theory of Rational Option Pricing（理性選擇權定價理論）." *Bell Journal of Economics and Management Science* 4: 141-183.

Narang, Rishi. 2013. *Inside the Black Box: A Simple Guide to Quantitative and High Frequency Trading*（打開黑箱：計量高頻交易簡易指南）. Hoboken et al: John Wiley & Sons.

Solomon, Lewis, and Louise Corso. 1991. "The Impact of Technology on the Trading of Securities: The Emerging Global Market and the Implications for Regulation（證券交易技術衝擊：新興的全球市場與對法規的影響）." *The John Marshall Law Review* 24 (2): 299-338.

VanderPlas, Jake. 2016. *Python Data Science Handbook: Essential Tools for Working with Data*（Python 資料科學學習手冊：資料處理不可或缺的工具）. Sebastopol: O'Reilly.