Capitolo 1

Python e il trading algoritmico

> In Goldman [Sachs] il numero di persone impegnate nel trading di azioni è sceso da un picco di 600 nel 2000 a solo 2, oggi.
>
> *Too Squid to Fail*, "The Economist", 29 ottobre 2016

Questo capitolo fornisce informazioni di base e una panoramica sugli argomenti trattati nel libro. Sebbene *Python per il trading algoritmico* sia un argomento di nicchia, all'intersezione tra programmazione Python e finanza, è una nicchia in rapida ascesa, che tocca argomenti diversi come la distribuzione di codice Python, l'analisi finanziaria interattiva, il machine learning e il deep learning, la programmazione a oggetti, la comunicazione socket, la visualizzazione di dati in streaming e le piattaforme di trading.
Per un rapido aggiornamento su Python, iniziate a leggere l'Appendice.

In questo capitolo

- **Python per la finanza**
- **Trading algoritmico**
- **Python per il trading algoritmico**
- **Focus e prerequisiti**
- **Strategie di trading**
- **Conclusioni**
- **Riferimenti e ulteriori risorse**

Python per la finanza

Il linguaggio di programmazione Python è nato nel 1991 con il primo rilascio da parte di Guido van Rossum della versione 0.9.0. Nel 1994 seguì la versione 1.0. Tuttavia, ci sono voluti quasi due decenni prima che Python si affermasse come uno dei principali linguaggi di programmazione e delle più importanti piattaforme tecnologiche nel settore finanziario. Ovviamente ci sono stati gli *early adopter*, principalmente *hedge fund*, ma l'adozione più ampia è iniziata solo intorno al 2011.

Uno dei principali ostacoli all'adozione di Python nel settore finanziario è stato il fatto che la versione di default di Python, chiamata CPython, è un linguaggio di alto livello interpretato. Gli algoritmi numerici in generale, e gli algoritmi finanziari in particolare, molto spesso sono implementati sulla base di strutture a ciclo, annidate. Essendo compilati, i linguaggi di basso livello come il C o il C++ sono molto veloci nell'esecuzione di tali cicli, Python, che si basa sull'interpretazione anziché sulla compilazione, è generalmente piuttosto lento. Di conseguenza, il Python puro si è dimostrato troppo lento per molte applicazioni finanziarie reali, come il prezzo delle opzioni o la gestione del rischio.

Python e lo pseudo-codice

Sebbene Python non sia mai stato rivolto in modo specifico alle comunità scientifiche e finanziarie, molte persone provenienti da questi campi hanno comunque apprezzato la bellezza e la concisione della sua sintassi. Non molto tempo fa, era considerata buona pratica spiegare un algoritmo (finanziario) e allo stesso tempo presentare il suo pseudo-codice come passaggio intermedio verso la sua corretta implementazione tecnologica. Molti pensavano che, con Python, il passaggio dello pseudo-codice non sarebbe stato più necessario. E il più delle volte, in effetti, è così.
Considerate, per esempio, la discretizzazione di Eulero del moto browniano geometrico, come mostrato nell'Equazione 1.1.

Equazione 1.1 Discretizzazione di Eulero del moto browniano geometrico.

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

Per decenni, il linguaggio markup e il compilatore LaTeX sono stati il *gold standard* per la creazione di documenti scientifici contenenti formule matematiche. Per molti versi, la sintassi LaTeX è simile o è già pseudo-codice quando, per esempio, si strutturano le equazioni come quella nell'Equazione 1.1. In questo caso specifico, la versione LaTeX si presenta così:

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

In Python, questo si traduce in codice eseguibile, date le rispettive definizioni di variabili, che è molto simile anche alla formula finanziaria oltre alla rappresentazione LaTeX:

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

Tuttavia, rimane il problema della velocità. Tale equazione, come approssimazione numerica della rispettiva equazione differenziale stocastica, viene generalmente utilizzata per valutare le derivate dei prezzi dalla simulazione di Monte Carlo o per eseguire l'analisi e la gestione del rischio basata su simulazioni. (Per i dettagli, vedi Hilpisch (2018), Capitolo 12.) Queste attività, a loro volta, possono richiedere milioni di simulazioni, che devono essere completate in tempi accettabili, spesso quasi in tempo reale o quasi. Python, in quanto linguaggio di programmazione di alto livello interpretato, non è mai stato progettato per essere sufficientemente veloce da affrontare compiti così impegnativi, dal punto di vista computazionale.

NumPy e la vettorizzazione

Nel 2006 è stata rilasciata da Travis Oliphant la versione 1.0 del package `NumPy` per Python (`http://www.numpy.org`). `NumPy` sta per *numerical Python*, suggerendo il fatto che si rivolge a scenari numericamente impegnativi. L'interprete Python cerca di essere il più generale possibile in molte aree, il che spesso lo porta ad avere un certo sovraccarico a runtime. `NumPy`, al contrario, adotta la specializzazione come suo approccio principale, proprio per evitare i sovraccarichi ed essere il più efficace e veloce possibile in *specifici* scenari applicativi.

> **NOTA**
>
> Per esempio, gli oggetti `list` sono non solo mutabili, ovvero modificabili in termini di dimensioni, ma possono anche contenere praticamente qualsiasi altro tipo di oggetto Python, come `int`, `float`, oggetti `tuple` o anche altri oggetti `list`.

La classe principale di `NumPy` è il normale oggetto per array, chiamato `ndarray` per *array n-dimensionale*. È immutabile, il che significa che le sue dimensioni non possono essere modificate e che può contenere un solo tipo di dati, chiamato `dtype`. Questa sua specializzazione consente di implementare codice compatto e veloce. Un approccio fondamentale in questo contesto è la *vettorizzazione*. Fondamentalmente, questo approccio evita i cicli a livello di codice Python e delega i cicli al codice specializzato `NumPy`, generalmente implementato in C e, pertanto, piuttosto veloce.
Considerate la simulazione di un milione di valori di fine periodo S_T secondo l'Equazione 1.1 in Python puro. La parte principale del codice seguente è un ciclo `for` con un milione di iterazioni.

```
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))
        values.append(ST)
    CPU times: user 1.13 s, sys: 21.7 ms, total: 1.15 s
    Wall time: 1.15 s
```

❶ Il livello iniziale dell'indice.

❷ Il tasso a breve termine costante.

❸ L'orizzonte temporale in frazioni di anno.

❹ Il fattore di volatilità costante.

❺ Un oggetto `list` vuoto per raccogliere i valori simulati.

❻ Il ciclo `for` principale.

❼ La simulazione di un *singolo* valore di fine periodo.

❽ Aggiunge il valore simulato all'oggetto `list`.

Con `NumPy`, potete evitare completamente il ciclo Python usando la virtualizzazione. Il codice è molto più compatto, leggibile e anche più veloce, di circa otto volte.

```
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 Questa singola riga di codice `NumPy` simula tutti i valori e li memorizza in un oggetto `ndarray`.

SUGGERIMENTO

La vettorizzazione è un concetto per scrivere codice compatto, di facile lettura e di facile manutenzione nella finanza e nel trading algoritmico. Con `NumPy`, il codice vettorizzato non solo è più compatto, ma può anche essere più veloce, come nell'esempio di simulazione Monte Carlo: circa otto volte più veloce.

Si può dire con certezza che `NumPy` abbia contribuito in modo significativo al successo di Python nelle scienze e nella finanza. Molti altri noti package Python del cosiddetto *scientific Python stack* si basano su `NumPy` in quanto è una struttura dati efficiente e veloce per archiviare e gestire dati numerici. In realtà, `NumPy` deriva dal progetto `SciPy`, che offre un'ampia gamma di funzionalità spesso necessaria nel campo delle scienze. Il progetto `SciPy` ha riconosciuto la necessità di dotarsi di una più potente struttura per i dati numerici e ha consolidato i progetti precedenti, come `Numeric` e `NumArray` in questo campo, creando un nuovo progetto unificatore: `NumPy`.

Nel trading algoritmico, la simulazione Monte Carlo potrebbe non essere il caso d'uso più importante per un linguaggio di programmazione. Tuttavia, se si entra nel mondo di trading algoritmico, la gestione di dataset di serie storiche finanziarie di grandi o addirittura enormi dimensioni è un caso d'uso molto importante. Basti pensare al backtesting delle strategie di trading (*intraday*) o all'elaborazione dei flussi di dati durante le ore di trading. Qui entra in gioco il package di analisi dei dati `pandas` (`http://pandas.pydata.org`).

pandas e la classe DataFrame

Lo sviluppo di pandas è iniziato nel 2008 da parte di Wes McKinney, che all'epoca lavorava presso AQR Capital Management, un grande hedge fund operante a Greenwich, nel Connecticut. Come per qualsiasi altro hedge fund, lavorare con i dati delle serie storiche era di fondamentale importanza anche per AQR Capital Management, ma all'epoca Python non forniva alcun tipo di supporto per questo tipo di dati. L'idea di Wes fu quella di creare un package che imitasse le capacità del linguaggio statistico R (http://r-project.org) in questo campo. Ciò si riflette, per esempio, nella denominazione della classe principale DataFrame, la cui controparte in R è data.frame. Non considerandolo poi così vicino al core business della gestione dei capitali, AQR Capital Management rese open source il progetto pandas nel 2009, data che segna l'inizio di una grande storia di successo nei dati open source e nell'analisi finanziaria.

In parte grazie a pandas, Python è diventato una forza importante nell'analisi dei dati e finanziaria. Molte persone che adottano Python, provenienti da altri linguaggi, citano pandas come motivo principale della loro decisione. In combinazione con fonti di dati open come Quandl (http://quandl.com), pandas permette anche agli studenti di eseguire sofisticate analisi finanziarie con minime barriere di accesso: è sufficiente un normale notebook con una connessione Internet.

Supponiamo che un trader algoritmico sia interessato a fare trading di bitcoin, la criptovaluta con la maggiore capitalizzazione di mercato. Un primo passo potrebbe essere quello di recuperare i dati sul tasso di cambio storico. Utilizzando i dati Quandl e pandas, tale compito viene svolto in meno di un minuto. La Figura 1.1 mostra il grafico prodotto dal codice Python seguente, che è (escludendo le parametrizzazioni relative allo stile grafico) composto da solo quattro righe. Sebbene pandas non venga importato esplicitamente, il package wrapper per Python Quandl restituisce per default un oggetto DataFrame, che viene poi utilizzato per aggiungere una semplice media mobile (SMA) di 100 giorni, nonché per visualizzare i dati grezzi insieme alla SMA.

```
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 Importa e configura il package di stampa.

❷ Importa il modulo configparser e legge le credenziali.

❸ Importa il package wrapper Quandl Python e fornisce la chiave dell'API.
❹ Recupera i dati giornalieri per il tasso di cambio dei bitcoin e restituisce un oggetto `DataFrame` `pandas` con una singola colonna.
❺ Calcola in modo vettoriale la SMA per 100 giorni.
❻ Seleziona i dati dal 1° gennaio 2013 in poi e li traccia.

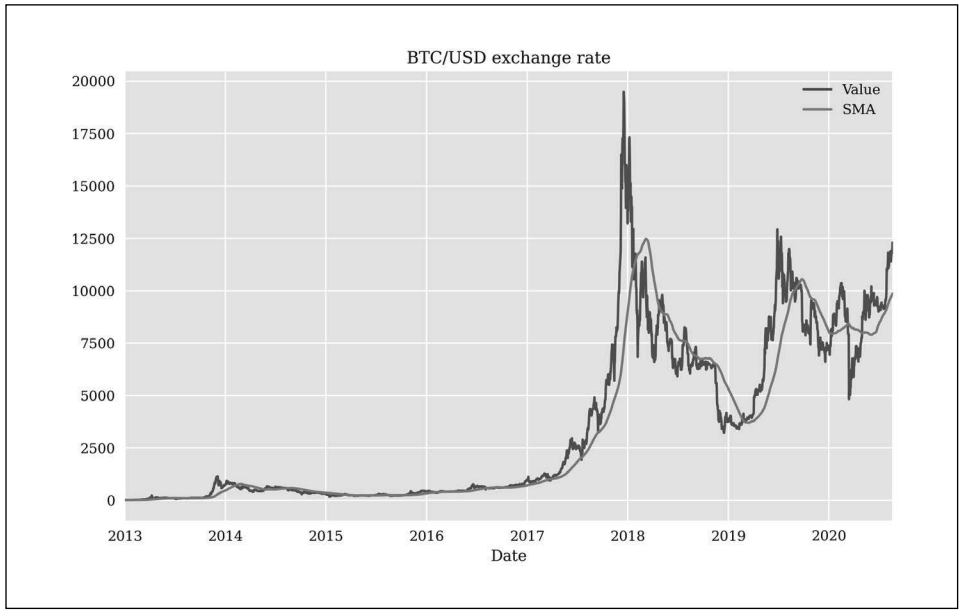

Figura 1.1 Tasso di cambio storico dei bitcoin in dollari da inizio 2013 fino alla metà 2020.

Ovviamente, `NumPy` e `pandas` contribuiscono in modo misurabile al successo di Python in finanza. Tuttavia, l'ecosistema Python ha molto più da offrire sotto forma di package Python che risolvono problemi piuttosto fondamentali e, talvolta, anche specializzati. Questo libro farà uso, tra le altre cose, di package per il recupero e l'archiviazione dei dati (per esempio `PyTables`, `TsTables`, `SQLite`) e per il machine e deep learning (per esempio `scikit-learn`, `tensorflow`), per citare solo due categorie. Nel corso dei capitoli, implementeremo anche classi e moduli che renderanno più efficiente qualsiasi progetto di trading algoritmico. Tuttavia, i package principali che utilizzeremo di più saranno `NumPy` e `pandas`.

NOTA

Mentre `NumPy` fornisce la struttura dati di base per memorizzare e manipolare i dati numerici, `pandas` offre potenti funzionalità di gestione delle serie storiche. Svolge inoltre un ottimo lavoro di wrapping di funzionalità da altri package in un'API di facile uso. L'esempio dei bitcoin appena illustrato mostra che la chiamata di un singolo metodo su un oggetto `DataFrame` è sufficiente per generare un grafico con due serie storiche. Come `NumPy`, anche `pandas` permette di produrre codice piuttosto compatto, vettoriale, che generalmente viene eseguito abbastanza velocemente, grazie a un uso intenso di codice già compilato.

Trading algoritmico

Il termine *trading algoritmico* non è né univocamente né universalmente definito. A un livello piuttosto elementare, si riferisce al trading di strumenti finanziari basato su un algoritmo formale. Un *algoritmo* è un insieme di operazioni (matematiche, tecniche) da condurre in una specifica sequenza per raggiungere un determinato obiettivo. Per esempio, esistono algoritmi matematici per risolvere il Cubo di Rubik; a tale proposito, vedi *The Mathematics of the Rubik's Cube* (`http://erikdemaine.org/papers/Rubik_ESA2011/paper.pdf`) o *Algorithms for Solving Rubik's Cube* (`http://erikdemaine.org/papers/Rubik_ESA2011/paper.pdf`). Tali algoritmi possono risolvere il problema in questione tramite una procedura sequenziale, spesso in modo perfetto. Un altro esempio è costituito dagli algoritmi per trovare la o le radici di un'equazione, se esistono. In questo senso, spesso l'obiettivo di un algoritmo matematico è ben specificato e in genere ci si aspetta una soluzione ottimale. Ma qual è l'obiettivo di un algoritmo di trading finanziario? A questa domanda non è poi così facile rispondere, in generale. Potrebbe essere utile fare un passo indietro e considerare i motivi del trading in generale. In Dorn et al. (2008), troviamo:

> Il trading nei mercati finanziari è un'importante attività economica. Il trading è necessario per operare sul mercato, per immettere del denaro in surplus nel mercato e per poi riconvertirlo di nuovo in denaro. Il trading è necessario anche per spostare i capitali all'interno del mercato, per scambiare un asset con un altro, per contenere i rischi e per sfruttare le informazioni sui movimenti futuri dei prezzi.

Il punto di vista qui espresso è di natura più tecnica che economica, concentrandosi principalmente sul processo stesso del trading e solo in parte sul *motivo* per cui le persone iniziano a fare trading. Per i nostri scopi, un elenco non esaustivo delle motivazioni del trading finanziario sia di persone sia di istituti finanziari che gestiscono il denaro per conto proprio o di altri è il seguente.

- *Beta trading*: guadagnare premi per il rischio di mercato, investendo, per esempio, in *Exchange Traded Fund* (ETF) che replicano le performance degli S&P 500.
- *Alpha generation*: guadagnare premi per rischi indipendenti dal mercato, per esempio, vendendo azioni allo scoperto quotate in S&P 500 o in ETF basati su S&P 500.
- *Static hedging*: copertura contro i rischi di mercato acquistando, per esempio, opzioni *put out-of-the-money* sull'S&P 500.
- *Dynamic hedging*: copertura contro rischi di mercato che interessano opzioni sull'indice S&P 500, per esempio, tramite il trading dinamico dei future sullo S&P 500 e adeguati strumenti di liquidità, mercato monetario o tasso.
- *Asset-liability management*: negoziazione di azioni ed ETF S&P 500 per coprire le passività derivanti, per esempio, dalla stipula di polizze assicurative sulla vita.
- *Market making*: fornire, per esempio, liquidità alle opzioni sull'S&P 500 acquistando e vendendo opzioni a prezzi *bid* e *ask* (denaro e lettera) differenti.

Tutti questi tipi di operazioni possono essere implementati con un approccio discrezionale, con il trader umano che prende le decisioni principalmente da solo, nonché algoritmi che supportano il trader umano o addirittura lo sostituiscono completamente nel processo decisionale. In questo contesto, ovviamente, l'informatizzazione del trading finanziario gioca un ruolo importante. Mentre all'inizio del trading finanziario, il *floor trading* con

un folto gruppo di persone che urlavano era l'unico modo per eseguire operazioni, l'informatizzazione e l'avvento di Internet e delle tecnologie web hanno rivoluzionato il trading nel settore finanziario. La citazione posta all'inizio di questo capitolo lo illustra in modo impressionante in termini di numero di persone impegnate nelle attività finanziarie presso Goldman Sachs nel 2000 e nel 2016. Si tratta di una tendenza già prevista 25 anni fa, come sottolineano Solomon e Corso (1991):

> I computer hanno rivoluzionato il commercio di titoli e il mercato azionario è attualmente nel mezzo di una trasformazione dinamica. È evidente che il mercato del futuro non somiglierà ai mercati del passato.
> La tecnologia ha reso possibile l'invio di informazioni sui prezzi delle azioni in tutto il mondo in pochi secondi. Attualmente, i computer instradano gli ordini ed eseguono piccole operazioni direttamente dal terminale della società di brokeraggio alla borsa. I computer ora collegano tra loro più borse, una pratica che sta contribuendo a creare un mercato globale unico per il commercio di titoli. I continui miglioramenti delle tecnologie consentiranno di eseguire trading globalmente tramite sistemi di negoziazione completamente elettronici.

È interessante notare che uno degli algoritmi più antichi e più utilizzati si trova nel dynamic hedging di opzioni. Già con la pubblicazione degli articoli fondamentali sul prezzo delle opzioni europee di Black e Scholes (1973) e Merton (1973), l'algoritmo, chiamato *delta hedging*, è stato reso disponibile molto prima che iniziasse il trading computerizzato ed elettronico. Il delta hedging come algoritmo di trading mostra come coprire tutti i rischi di mercato in un modello del mondo semplificato, perfetto e continuo. Nel mondo reale, con costi delle transazioni, negoziazione discreta, mercati liquidi in modo imperfetto e altre problematiche ("imperfezioni"), l'algoritmo ha dimostrato, forse in modo un po' sorprendente, anche la sua utilità e robustezza. Potrebbe non riuscire a consentire di coprire perfettamente i rischi di mercato che interessano le opzioni, ma è utile per avvicinarsi all'ideale e quindi è ancora utilizzato su larga scala nel settore finanziario. Vedi Hilpisch (2015) per un'analisi dettagliata delle strategie di delta hedging per le opzioni europee e americane in Python.

Questo libro si concentra sul trading algoritmico nel contesto delle *strategie di alpha generating*. Sebbene esistano definizioni più sofisticate dell'alpha, ai fini di questo libro alpha è visto come la differenza tra il rendimento di una strategia di trading in un certo periodo di tempo e il rendimento del benchmark (singola azione, indice, criptovaluta e così via). Per esempio, se l'S&P 500 dà il 10% nel 2018 e una strategia algoritmica dà il 12%, l'alpha è di +2 punti percentuali. Se la strategia restituisce il 7%, l'alpha è -3 punti percentuali. In generale, tali numeri non sono adattati al rischio, e altre caratteristiche del rischio come il prelievo massimo (nel periodo) sono tutti considerati di secondo ordine, se non del tutto ignorati.

NOTA

Questo libro si concentra sulle *strategie di alpha generating,* ovvero strategie che cercano di generare rendimenti positivi (superiori a un benchmark) indipendentemente dalla performance del mercato stesso. Alpha è definito in questo libro (nel modo più semplice) come il maggior rendimento di una strategia rispetto alla performance dello strumento finanziario di riferimento.

Ci sono altre aree in cui gli algoritmi di trading svolgono un ruolo importante. Uno è lo spazio *high frequency trading* (HFT), dove i giocatori competono in termini di velocità. Vedi il libro di Lewis (2015) per un'introduzione non tecnica all'HFT. I motivi dell'HFT sono diversi, ma probabilmente il *market making* e l'*alpha generating* giocano un ruolo preminente. Un altro è l'*esecuzione delle negoziazioni*, in cui vengono implementati algoritmi per eseguire in modo ottimale determinate operazioni non standard. I motivi in quest'area potrebbero includere l'esecuzione (ai migliori prezzi possibili) di ordini di grandi dimensioni o l'esecuzione di un ordine con il minore impatto possibile sul mercato e sui prezzi. Un motivo più subdolo potrebbe essere quello di mascherare un ordine, eseguendolo con una serie di più scambi.
Resta da affrontare una domanda importante: c'è qualche vantaggio nell'usare algoritmi per il trading invece della ricerca, dell'esperienza e della discrezione umana? È difficile rispondere in termini generali a questa domanda. Di sicuro, ci sono trader umani e gestori di portafoglio che hanno guadagnato, in media, più del loro benchmark per gli investitori per periodi di tempo più lunghi. L'esempio classico a questo riguardo è Warren Buffett. D'altro canto, le analisi statistiche mostrano che la maggior parte dei gestori di portafoglio attivi, raramente supera costantemente i benchmark più rilevanti. Riferendosi all'anno 2015, Adam Shell scrive:

> L'anno scorso, per esempio, quando l'indice azionario Standard & Poor's 500 ha registrato un misero rendimento totale dell'1,4% dividendi inclusi, il 66% dei fondi azionari di grandi aziende "gestiti attivamente" ha registrato rendimenti inferiori rispetto all'indice […] L'outlook a lungo termine è altrettanto cupo, con l'84% dei fondi large-cap che generano rendimenti più bassi rispetto al S&P 500 nell'ultimo periodo di cinque anni e l'82% è stato molto timido negli ultimi 10 anni. Questi i risultati dello studio.
> (Fonte: *66% of Fund Managers Can't Match S&P Results*, in "USA Today", 14 marzo 2016.)

In uno studio empirico pubblicato nel dicembre 2016, Harvey et al. scrivono:

> Noi analizziamo e confrontiamo le performance di hedge fund discrezionali e sistematici. I fondi sistematici utilizzano strategie basate su regole, con poco o nessun intervento quotidiano da parte degli esseri umani […] Abbiamo trovato che, per il periodo 1996-2014, i gestori azionari sistematici hanno avuto performance inferiori rispetto alle controparti discrezionali in termini di rendimenti non rettificati (grezzi), ma dopo l'aggiustamento per le esposizioni a ben noti fattori di rischio, le performance sono risultate simili. Nel caso dei fondi macro, i fondi sistematici hanno avuto performance superiori ai fondi discrezionali, sia su base non rettificata che rettificata.

La Tabella 1.1 riproduce i principali risultati quantitativi dello studio di Harvey et al. (2016), una performance annualizzata (al di sopra del tasso di interesse a breve termine) e le misure di rischio per categorie di hedge fund comprendenti un totale di 9.000 hedge fund nel periodo dal giugno 1996 al dicembre il 2014. Nella tabella, i *fattori* includono quelli tradizionali (azioni, obbligazioni e così via), quelli dinamici (valore, *momentum* e così via) e la volatilità (acquisto di put e call at-the-money). Il rapporto di stima del rendimento rettificato divide l'alpha per la volatilità del rendimento rettificato. Per maggiori dettagli e informazioni generali, si rimanda al documento originale.
I risultati dello studio dimostrano che i macro hedge fund sistematici ("algoritmici") funzionano meglio come categoria, sia in termini non rettificati che in termini rettificati

in base ai rischi. Generano un alpha annualizzato di 4,85 punti percentuali nel periodo studiato. Si tratta di fondi speculativi che implementano strategie tipicamente globali, cross-asset e spesso coinvolgono elementi politici e macroeconomici. Gli hedge fund sistematici azionari battono le loro controparti discrezionali solo sulla base del rapporto di stima del rendimento rettificato (0,35 contro 0,25).

Tabella 1.1 Performance annualizzata delle categorie di hedge fund.

	macro sistematici	macro discrezionali	equity sistematici	equity discrezionali
rendimento medio	5,01%	2,86%	2,88%	4,09%
rendimento attribuito a fattori	0,15%	1,28%	1,77%	2,86%
agg. media di ritorno (alpha)	4,85%	1,57%	1,11%	1,22%
agg. volatilità dei rendimenti	10,93%	5,10%	3,18%	4,79%
agg. rapporto di stima di ritorno	0,44	0,31	0,35	0,25

Rispetto all'S&P 500, la performance degli hedge fund nel complesso è stata piuttosto scarsa per l'anno 2017. Mentre l'indice S&P 500 ha reso il 21,8%, gli hedge fund hanno restituito solo l'8,5% agli investitori (vedi l'articolo `https://www.investopedia.com/news/2017-hedge-funds-return-less-half-sp-500` su Investopedia). Ciò dimostra quanto sia difficile, anche con budget multimilionari in termini di ricerca e tecnologia, generare alpha.

Python per il trading algoritmico

Python è utilizzato in molti ambiti del settore finanziario, ma è diventato particolarmente popolare nel mondo del trading algoritmico, e questo per alcune buone ragioni.

- *Funzionalità di analisi dei dati*: un requisito importante per ogni progetto di trading algoritmico è la capacità di gestire ed elaborare i dati finanziari in modo efficiente. Python, in combinazione con package come `NumPy` e `pandas`, semplifica le cose per ogni trader algoritmico rispetto alla maggior parte degli altri linguaggi di programmazione.
- *Manipolazione di API moderne*: moderne piattaforme di trading online come quelle di FXCM (`http://fxcm.co.uk`) e Oanda (`http://oanda.com`) offrono API (*Application Programming Interface*) RESTful e API socket (streaming) per l'accesso a dati storici e real-time. Python è, in generale, perfetto per interagire in modo efficiente con tali API.
- *Package dedicati*: oltre ai package standard di analisi di dati, sono disponibili molti package dedicati al mondo del trading algoritmico, come per esempio `PyAlgoTrade` (`http://gbeced.github.io/pyalgotrade`) e `Zipline` (`https://github.com/quantopian/zipline`), per il backtesting delle strategie di trading, e `Pyfolio` (`https://github.com/quantopian/pyfolio`) per l'esecuzione dell'analisi del portafoglio e dei rischi.
- *Package sponsorizzati dai fornitori*: sempre più fornitori nel mondo del trading rilasciano package Python open source per facilitare l'accesso alle loro offerte. Tra questi ci sono piattaforme di trading online come Oanda e i principali fornitori di dati come Bloomberg (`https://www.bloomberglabs.com/api`) e Refinitiv (`https://developers.refinitiv.com/all/api-families`).

- *Piattaforme dedicate*: Quantopian, per esempio, offre un ambiente di backtest standardizzato come una piattaforma basata sul Web in cui il linguaggio preferito è Python e dove le persone possono scambiarsi idee con altri che la pensano allo stesso modo, attraverso diverse funzionalità da social network. Dalla sua fondazione e fino al 2020, Quantopian ha attratto più di 300.000 utenti.
- *Adozione lato acquisto e lato vendita*: sempre più player istituzionali hanno adottato Python per semplificare gli sforzi di sviluppo nei loro dipartimenti di trading. Questo, a sua volta, richiede sempre più personale esperto in Python, cosa che rende l'apprendimento di Python un investimento utile.
- *Istruzione, formazione e libri*: fra i prerequisiti per un'ampia diffusione di una tecnologia o di un linguaggio di programmazione vi sono dei programmi di istruzione e formazione accademici e professionali, in combinazione con libri e altre risorse. L'ecosistema Python ha assistito a una crescita enorme di tali offerte in tempi recenti: sempre più persone sono formate nell'uso di Python per la finanza. Ci si può aspettare che questo rafforzi la tendenza all'adozione di Python nel mondo del trading algoritmico.

In sintesi, è abbastanza sicuro affermare che Python giochi già ora un ruolo importante nel trading algoritmico e che sembra avere un forte impulso per diventare sempre più importante in futuro. Python rappresenta quindi una buona scelta per chiunque cerchi di entrare nel mondo del trading, sia come ambizioso trader "retail" sia come professionista alle dipendenze di un importante istituto finanziario impegnato nel trading sistematico.

Focus e prerequisiti

Il focus di questo libro è su Python come linguaggio di programmazione per il trading algoritmico. Il libro presuppone che il lettore abbia già una certa esperienza con Python e con i package Python più utilizzati nell'analisi dei dati. Fra i migliori libri introduttivi vi sono, per esempio, Hilpisch (2018), McKinney (2017) e VanderPlas (2016), che possono essere consultati per costruire una solida base di Python nell'analisi dei dati e nella finanza. Il lettore dovrebbe inoltre avere una certa esperienza con gli strumenti utilizzati per l'analisi interattiva con Python, come `IPython`, di cui VanderPlas (2016) fornisce un'ottima introduzione.

Questo libro presenta e spiega il codice Python applicato agli argomenti in questione, come il backtesting delle strategie di trading o l'utilizzo di dati in streaming. Non può invece rappresentare un'introduzione completa a tutti i package utilizzati in altri campi. Cerca, tuttavia, di evidenziare quelle capacità dei package che sono centrali per l'esposizione dei concetti (come la vettorizzazione con `NumPy`).

Il libro, inoltre, non può rappresentare un'introduzione completa e una panoramica di tutti gli aspetti finanziari e operativi rilevanti per il trading algoritmico. L'approccio si concentra invece sull'uso di Python per costruire l'infrastruttura necessaria per sistemi di trading automatizzati e algoritmici. Ovviamente, la maggior parte degli esempi utilizzati è tratta dal mondo del trading algoritmico. Tuttavia, quando si tratta, per esempio, di strategie *momentum* o *mean reversion*, vengono solamente utilizzate, senza fornire una verifica (statistica) o una discussione approfondita delle loro complessità. Ogni volta che

sembra opportuno, vengono però forniti dei riferimenti che indirizzano il lettore a fonti che affrontano le questioni lasciate aperte durante l'esposizione.
In generale, questo libro è stato scritto per i lettori che hanno una certa esperienza sia con Python sia con il trading (algoritmico). Per un lettore di questo tipo, il libro rappresenta una guida pratica alla creazione di sistemi di trading automatizzati utilizzando Python e vari altri package aggiuntivi.

ATTENZIONE

Questo libro usa una serie di approcci alla programmazione Python (per esempio, la programmazione a oggetti) e package (per esempio, `scikit-learn`) che non possono essere trattati in dettaglio. Il focus è sull'*applicazione* di questi approcci e package alle diverse fasi di un processo di trading algoritmico. Pertanto, coloro che non hanno ancora abbastanza esperienza con Python (per la finanza) consultino anche altri testi, più introduttivi, su Python.

Strategie di trading

Negli esempi di questo libro vengono utilizzate quattro diverse strategie di trading algoritmico. Sono introdotte brevemente di seguito e poi, in modo più dettagliato, nel Capitolo 4. Tutte queste strategie di trading possono essere classificate principalmente come *strategie di alpha seeking*, in quanto il loro obiettivo principale è quello di generare rendimenti positivi e superiori al mercato, indipendentemente dalla direzione del mercato. Gli esempi canonici, in tutto il libro, quando si tratta di strumenti finanziari negoziati sono un indice azionario, una singola azione o una criptovaluta (denominata in una valuta fiat). Il libro non tratta strategie che coinvolgono più strumenti finanziari contemporaneamente (strategie di pair trading, strategie progettate su panieri e così via). Inoltre, il libro tratta solo le strategie i cui segnali di trading sono derivati da dati temporali strutturati, finanziari e non, per esempio provenienti da fonti di dati non strutturate, come i feed di notizie o dei social media. Ciò mantiene le discussioni e le implementazioni di Python più compatte e comprensibili, in linea con l'approccio (discusso in precedenza) di concentrarsi su Python per il trading algoritmico. Vedi il libro di Kissel (2013) per una panoramica degli argomenti relativi al trading algoritmico, il libro di Chan (2013) per una discussione approfondita sulle strategie *momentum* e *mean reversion*, o il libro di Narang (2013) per una copertura del trading quantitativo e HFT in generale.
Il resto di questo capitolo offre una rapida panoramica delle quattro strategie di trading utilizzate nel libro.

Simple Moving Average, medie mobili semplici

Il primo tipo di strategia di trading si basa su medie mobili semplici (SMA - *Simple Moving Average*) per generare segnali di trading e posizionamenti di mercato. Queste strategie di trading sono state rese popolari dai cosiddetti analisti o grafici tecnici. L'idea di base è che una SMA a breve termine che abbia un valore superiore rispetto a una SMA a lungo termine segnala una posizione di mercato *long* e lo scenario opposto segnala una posizione di mercato neutra o *short*.

Momentum, quantità di moto

L'idea alla base delle strategie *momentum* è che si presuppone che uno strumento finanziario funzioni in conformità con la sua performance recente per qualche tempo in più. Per esempio, se un indice azionario ha registrato un rendimento medio negativo negli ultimi cinque giorni, si presuppone che la sua performance sarà negativa anche domani.

Mean reversion

Nelle strategie *mean reversion*, si presuppone che uno strumento finanziario torni a un livello medio o di tendenza se attualmente si trova abbastanza lontano da tale livello. Per esempio, supponete che un'azione venga scambiata a 10 dollari al di sotto del livello SMA di 200 giorni di 100. Si prevede che il suo prezzo torni presto al livello SMA.

Machine e deep learning

Con gli algoritmi di machine learning e deep learning, generalmente si adotta un approccio più a "black-box" per predire i movimenti del mercato. Per semplicità e riproducibilità, gli esempi presentati in questo libro si basano principalmente sulle osservazioni storiche dei rendimenti, come caratteristiche per addestrare gli algoritmi di machine e deep learning a predire i movimenti del mercato azionario.

ATTENZIONE

Questo libro non presenta il trading algoritmico in modo sistematico. Poiché il focus è sull'applicazione di Python in questo campo così affascinante, coloro che non avessero familiarità con il trading algoritmico dovrebbero consultare altre risorse dedicate sull'argomento, alcune delle quali sono citate in questo capitolo e nei successivi. Ma siate consapevoli del fatto che il mondo del trading algoritmico, in generale, è piuttosto riluttante a svelare i suoi segreti e che quasi tutti coloro che hanno successo tendono a non condividere i propri segreti, in modo da proteggere le loro fonti del loro successo, cioè il loro alpha.

Conclusioni

Python è già una forza nella finanza in generale e sta per diventare una forza importante nel trading algoritmico. Vi sono tante buone ragioni per utilizzare Python per il trading algoritmico, tra cui il potente ecosistema di package che offrono un'analisi dei dati efficiente o la presenza di API moderne. Vi sono anche molte buone ragioni per imparare a usare Python per il trading algoritmico, prima fra tutte il fatto che alcune delle più grandi istituzioni fanno un uso massiccio di Python nelle loro operazioni di trading e cercano costantemente professionisti esperti in Python.

Questo libro si concentra sull'applicazione di Python alle diverse discipline del trading algoritmico, come il backtesting delle strategie di trading o dell'interazione con le piattaforme di trading online. Non può sostituire un'introduzione approfondita a Python né al trading in generale. Tuttavia, combina sistematicamente questi mondi così affascinanti per fornire una fonte preziosa per l'alpha generating nei mercati, oggi così competitivi, della finanza e delle criptovalute.

Riferimenti e ulteriori risorse

Libri e articoli di ricerca citati in questo capitolo.

- Fischer Black e Myron Scholes, *The Pricing of Options and Corporate Liabilities*, in "Journal of Political Economy", Vol. 81, No. 3,1973, pp. 638-659.
- Ernest Chan, *Algorithmic Trading – Winning Strategies and Their Rationale*, John Wiley & Sons, Hoboken, New Jersey 2013.
- Anne Dorn, Daniel Dorn e Paul Sengmueller, *Why do People Trade?*, in "Journal of Applied Finance", Fall/Winter 2008, pp. 37-50.
- Campbell Harvey, Sandy Rattray, Andrew Sinclair e Otto Van Hemert, *Man vs. Machine: Comparing Discretionary and Systematic Hedge Fund Performance*, White Paper, Man Group, 2016.
- Yves Hilpisch, *Artificial Intelligence in Finance – A Python-based Guide*, O'Reilly, Beijing 2020. Risorse su: `http://aiif.tpq.io`.
- Yves Hilpisch, *Python for Finance – Mastering Data-Driven Finance*, 2nd ed., O'Reilly, Beijing 2018. Risorse su: `http://pff.tpq.io`.
- Yves Hilpisch, *Derivatives Analytics with Python – Data Analysis, Models, Simulation, Calibration and Hedging*, Wiley Finance, 2015. Risorse su: `http://dawp.tpq.io`.
- Robert Kissel, *The Science of Algorithmic Trading and Portfolio Management*, Elsevier/Academic Press, Amsterdam 2013.
- Michael Lewis, *Flash Boys: Cracking the Money Code*, W.W. Norton & Company, New York, London 2015.
- Wes McKinney, *Python for Data Analysis – Data Wrangling with Pandas, NumPy, and IPython*, 2nd ed., O'Reilly, Beijing 2017.
- Robert Merton, *Theory of Rational Option Pricing*, in "Bell Journal of Economics and Management Science", Vol. 4, 1973, pp. 141-183.
- Rishi Narang, *Inside the Black Box – A Simple Guide to Quantitative and High Frequency Trading*, John Wiley & Sons, Hoboken 2013.
- Lewis Solomon, e Louise Corso, *The Impact of Technology on the Trading of Securities: The Emerging Global Market and the Implications for Regulation*, in "The John Marshall Law Review", Vol. 24, No. 2, 1991, pp. 299-338.
- Jake VanderPlas, *Python Data Science Handbook – Essential Tools for Working with Data*, O'Reilly, Beijing 2016.