Python for Algorithmic Trading

The Python Quants GmbH <training@tpq.io>

Table of Contents

Copyright

Preface

> Dataism says that the universe consists of data flows, and the value of any phenomenon or entity is determined by its contribution to data processing. … Dataism thereby collapses the barrier between animals [humans] and machines, and expects electronic algorithms to eventually decipher and outperform biochemical algorithms.
>
> — Yuval Noah Harari (Homo Deus)

Finding the right algorithm to automatically and successfully trade in financial markets is the *holy grail in finance*. Not too long ago, **Algorithmic Trading** was only available for institutional players with deep pockets and lots of assets under management. Recent developments in the areas of open source, open data, cloud compute and storage as well as online trading platforms have leveled the playing field for smaller institutions and individual traders — making it possible to get started in this fascinating discipline being equipped with a modern notebook and an Internet connection only.

Nowadays, **Python** and its eco-system of powerful packages is the technology platform of choice for algorithmic trading. Among others, Python allows you to do *efficient data analytics* (with e.g. `pandas`), to apply *machine learning* to stock market prediction (with e.g. `scikit-learn`) or even make use of Google's *deep learning* technology (with `tensorflow`).

This is a course about **Python for Algorithmic Trading**. Such a course at the intersection of two vast and exciting fields can hardly cover all topics of relevance. However, it can cover a range of important meta topics in-depth:

- **financial data**: financial data is at the core of every algorithmic trading project; Python and packages like `NumPy` and `pandas` do a great job in handling and working with structured financial data of any kind (end-of-day, intraday, high frequency)
- **backtesting**: no automated, algorithmic trading without a rigorous testing of the trading strategy to be deployed; the course covers, among others, trading strategies bases on simple moving averages, momentum, mean-reversion and machine/deep learning based prediction

- **real-time data**: algorithmic trading requires dealing with real-time data, online algorithms based on it and visualization in real-time; the course introduces to socket programming with `ZeroMQ` and streaming visualization with Plotly
- **online platforms**: no trading without a trading platform; the course covers three popular electronic trading platforms: Oanda (CFD trading), Interactive Brokers (stock and options trading) and Gemini (cryptocurrency trading); it also provides convenient wrapper classes in Python to get up and running within minutes
- **automation**: the beauty as well as some major challenges in algorithmic trading result from the automation of the trading operation; the course shows how to deploy Python in the cloud and how to set up an environment appropriate for automated, algorithmic trading

The course offers a unique learning experience with the following features and benefits.

coverage of relevant topics

It is the only course covering such a breadth and depth with regard to relevant topics in Python for Algorithmic trading.

self-contained code base

The course is accompanied by a Git repository with all codes in a self-contained, executable form (3,000+ lines of code).

book version as PDF

In addition to the online version, there is also a book version as PDF (450+ pages).

online/video training (optional)

The Python Quants offer an online and video training class (not included) based on this course/book that provides an interactive learning experience (e.g. to see the code executed live, to ask individual questions) as well as a look at additional topics or at topics from a different angle.

real trading as the goal

The coverage of three different online trading platforms puts the student in the position to start both paper and live trading efficiently. This course equips the student with relevant, practical and valuable background knowledge.

do-it-yourself & self-paced approach

Since the material and the codes are self-contained and only relying on standard Python packages, the student has full knowledge of and full control over what is going on, how to use the code examples, how to change them, etc. There is no need to rely on third-party platforms, for instance, to do the backtesting or to connect to the trading platforms. The student can do all this on his/her own with this course — at a pace that is most convenient — and has every single line of code to do so available.

user forum

Although you are supposed to *be able* to do it all by yourself, we are there to help you. You can post questions and comments in our user forum at any time. We aim to get back within 24 hours.

The course assumes that the student has — at least on a fundamental level — some background knowledge both in Python programming as well as in financial trading. The course material includes Appendix A: Python, NumPy, matplotlib, pandas that introduces important Python, NumPy, matplotlib and pandas topics. Good references to get a sound understanding of the Python topics important for the course are:

- Hilpisch, Yves (2014): *Python for Finance*. O'Reilly, Beijing et al.
- McKinney, Wes (2017): *Python for Data Analysis*. 2nd ed., O'Reilly, Beijing et al.
- Ramalho, Luciano (2016): *Fluent Python*. O'Reilly, Beijing et al.
- VanderPlas, Jake (2016): *Python Data Science Handbook*. O'Reilly, Beijing et al.

Background information about algorithmic trading can be found, for instance, in these books:

- Chan, Ernest (2009): *Quantitative Trading*. John Wiley & Sons, Hoboken et al.
- Chan, Ernest (2013): *Algorithmic Trading*. John Wiley & Sons, Hoboken et al.
- Kissel, Robert (2013): *Algorithmic Trading and Portfolio Management*. Elsevier/Academic Press, Amsterdam et al.

- Narang, Rishi (2013): *Inside the Black Box*. John Wiley & Sons, Hoboken et al.

Enjoy your journey through the Algorithmic Trading world with Python and get in touch under training@tpq.io if you have questions or comments.

Chapter 1. Python and Algorithmic Trading

> At Goldman [Sachs] the number of people engaged in trading shares has fallen from a peak of 600 in 2000 to just two today. [2: “Too Squid to Fail.” The Economist, 29. October 2016.]
>
> — The Economist

1.1. Introduction

This chapter provides background information for, and an overview of, the topics covered in this book. Although Python for Algorithmic Trading is a niche at the intersection of Python programming and finance, it is a fast-growing one that touches on such diverse topics as Python deployment, interactive financial analytics, machine and deep learning, object oriented programming, socket communication, visualization of streaming data, and trading platforms.

For a quick refresher on important Python topics, read Appendix A: Python, NumPy, matplotlib, pandas first.

1.2. Python for Finance

The Python programming language originated in 1991 with the first release by Guido van Rossum of a version labeled 0.9.0. In 1994, version 1.0 followed. However, it took almost two decades for Python to establish itself as a major programming language and technology platform in the financial industry. Of course, there were early adopters, mainly hedge funds, but widespread adoption probably started only around 2011.

One major obstacle to the adoption of Python in the financial industry has been the fact that the default Python version, called CPython, is an interpreted, high level language. Numerical algorithms in general and financial algorithms in particular are quite often implemented based on (nested) loop structures. While compiled, low level languages like C or C++ are really fast at executing such loops, Python — which relies on interpretation instead of compilation — is generally quite slow at doing so.

As a consequence, pure Python proved too slow for many real-world financial applications, such as option pricing or risk management.

Although Python was never specifically targeted towards the scientific and financial communities, many people from these fields nevertheless liked the beauty and conciseness of its syntax. Not too long ago, it was generally considered good tradition to explain a (financial) algorithm and at the same time present some pseudo-code as an intermediate step towards its proper technological implementation. Many felt that, with Python, the pseudo-code step would not be necessary anymore. And they were proven mostly correct.

Consider, for instance, the Euler discretization of the geometric Brownian motion as in Euler discretization of geometric Brownian motion.

$$S_T = S_0 \exp((r - 0.5\sigma^2)T + \sigma z \sqrt{T}) \qquad (1)$$

Euler discretization of geometric Brownian motion

For decades, the Latex markup language and compiler have been the gold standard for authoring scientific documents containing mathematical formulae. In many ways, Latex syntax is similar to or already like pseudo-code when, for example, layouting equations as in Euler discretization of geometric Brownian motion. In this particular case, the Latex version looks like this:

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

In Python, this translates to executable code — given respective variable definitions — that is also really close to the financial formula as well as to the Latex representation:

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

However, the speed issue remains. Such a difference equation, as a numerical approximation of the respective stochastic differential equation, is generally used to price derivatives by Monte Carlo simulation or to do risk analysis and management based on simulation. These tasks in turn can require millions of simulations that need to be finished in due time — often in almost real-time or at least near-real time.

Interpreted Python per se was never designed to be fast enough to tackle such computationally demanding tasks.

In 2006, version 1.0 of the `NumPy` Python package was released by Travis Oliphant. `NumPy` stands for *numerical Python*, suggesting that it targets scenarios that are numerically demanding. The base Python interpreter tries to be as general as possible in many areas, which often leads to quite a bit of overhead at run-time. [3: For example, `list` objects are not only mutable, i.e. they can be changed in size, they can also contain almost any other kind of Python object, like `int`, `float`, `tuple` objects or `list` objects themselves.] `NumPy`, on the other hand, uses specialization as its major approach to avoid overhead and to be as good and as fast as possible in certain application scenarios.

The major class of `NumPy` is the regular array object, called ndarray object for *n-dimensional array*. It is immutable, i.e. it cannot be changed in size, and can only accommodate a single data type, called dtype. This specialization allows for the implementation of concise and fast code. One central approach in this context is *vectorization*. Basically, this approach avoids looping on the Python level and delegates the looping to specialized `NumPy` code, implemented in general in C and therefore rather fast.

Consider the simulation of 1,000,000 end of period values S_T according to Euler discretization of geometric Brownian motion with pure Python. The major part of the code below is a for loop with 1,000,000 iterations:

```
%%time
import random
from math import exp, sqrt

S_0 = 100  ①
r = 0.05  ②
T = 1.0  ③
sigma = 0.2  ④

values = []  ⑤

for _ in range(1000000):  ⑥
    S_T = S_0 * exp((r - 0.5 * sigma ** 2) * T +
                    sigma * random.gauss(0, 1) * sqrt(T))  ⑦
    values.append(S_T)  ⑧

CPU times: user 1.41 s, sys: 20.4 ms, total: 1.43 s
Wall time: 1.44 s
```

① The initial index level.

② The constant short rate.

③ The time horizon in year fractions.

④ The constant volatility factor.

⑤ An empty `list` object to collect simulated values.

⑥ The main `for` loop.

⑦ The simulation of a *single* end-of-period value.

⑧ Appends the simulated value to the `list` object.

With `NumPy`, you can avoid looping on the Python level completely by the use of vectorization. The code is much more concise, more readable, and faster by a factor of about 25:

```
%%time
import numpy as np

S_0 = 100
r = 0.05
T = 1.0
sigma = 0.2

values = S_0 * np.exp((r - 0.5 * sigma ** 2) * T +
                     sigma * np.random.standard_normal(1000000) * np.sqrt(T))  ①

CPU times: user 43.6 ms, sys: 6.13 ms, total: 49.8 ms
Wall time: 48.8 ms
```

① This single line of `NumPy` code simulates all the values and stores them in an `ndarray` object.

Vectorization is a powerful concept for writing concise, easy-to-read and easy-to-maintain code in finance and algorithmic trading. With `NumPy`, vectorized code does not only make code more concise, it also can speed up code execution considerably, like in the Monte Carlo simulation example by a factor of about 25.

It's safe to say that `NumPy` has significantly contributed to the success of Python in science and finance. Many other popular Python packages from the so-called *scientific Python stack* build on `NumPy` as an efficient, performing data structure to store and handle numerical data. In fact, `NumPy` is an outgrowth of the `SciPy` package project, which provides a wealth of functionality frequently needed in science. The `SciPy` project recognized the need for a more powerful numerical data structure and consolidated older projects like Numeric and NumArray in this area into a new, unifying one in the form of `NumPy`.

In algorithmic trading, Monte Carlo simulation might not be the most important use case for a programming language. However, if you enter the algorithmic trading space, the management of larger or even big financial time series data sets is, for example, a very important use case. Just think of the backtesting of (intraday) trading strategies or the processing of tick data streams during trading hours. This is where the `pandas` data analysis package comes into play (`pandas` home page).

Development of `pandas` began in 2008 by Wes McKinney, who back then was working at AQR Capital Management, a big hedge fund operating out of Greenwich, Connecticut. Like for any other hedge fund, working with time series data is of paramount importance for AQR Capital Management, but back then Python did not provide any kind of appealing support for this type of data. Wes's idea was to create a package that mimics the capabilities of the R statistical language in this area. This is reflected, for example, in naming the major class DataFrame, whose counterpart in R is called data.frame. Not being considered close enough to the core business of money management, AQR Capital Management open sourced the `pandas` project in 2009, which marks the beginning of a major success story in open source-based data and financial analytics.

Partly due to `pandas`, Python has become a major force in data and financial analytics. Many people who adopt Python, coming from diverse other languages, cite `pandas` as a major reason for their decision. In combination with open data sources like Quandl, `pandas` even allows students to do sophisticated financial analytics with the lowest barriers of entry ever: a regular notebook with an Internet connection suffices.

Assume an algorithmic trader is interested in trading Bitcoins, the cryptocurrency with the largest market capitalization. A first step might be to retrieve data about the historical exchange rate in USD. Using Quandl data and `pandas`, such a task is accomplished in less than a minute. Historical Bitcoin exchange rate in USD from the beginning of 2013 until the 04. November 2017 shows the plot that results from the Python code below, which is (omitting some plotting style related parameterizations) only four lines. Although `pandas` is not explicitly imported, the Quandl Python wrapper package by default returns a DataFrame object which is then used to add a simple moving average (SMA) of 100 days, as well as to visualize the raw data alongside the SMA.

```
import quandl as q  ①
d = q.get('BCHAIN/MKPRU')  ②
d['SMA'] = d['Value'].rolling(100).mean()  ③
d.loc['2013-1-1':].plot(title='BTC/USD exchange rate',
                        figsize=(10, 6))  ④
```

① Imports the Quandl Python wrapper package.

② Retrieves daily data for the Bitcoin exchange rate and returns a `pandas` DataFrame object with a single column.

③ Calculates the SMA for 100 days in vectorized fashion.

④ Selects data from January 1, 2013 on and plots it.

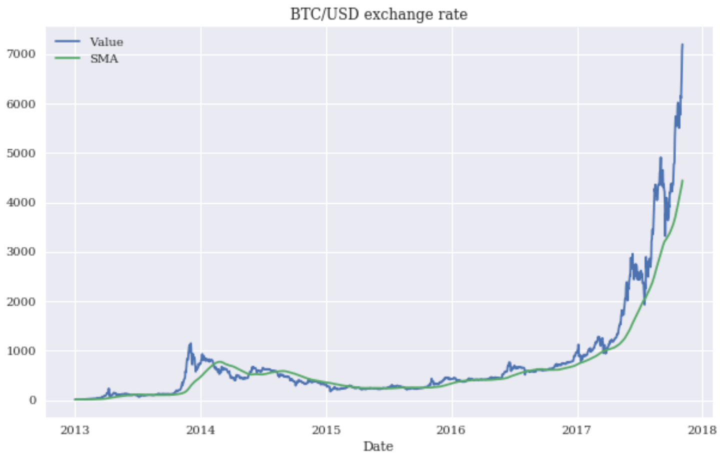

Figure 1. Historical Bitcoin exchange rate in USD from the beginning of 2013 until the 04. November 2017

Obviously, `NumPy` and `pandas` measurably contribute to the success of Python in finance. However, the Python ecosystem has much more to offer in the form of additional Python packages that solve rather fundamental problems and sometimes also specialized ones. In this book, we will make use of, among others, packages for data retrieval and storage (e.g. `PyTables`, TsTables, `SQLite`) and for machine and deep learning (e.g. scikit-learn, tensorflow) — to name just two categories. Along the way, we will also implement classes and modules that will make any algorithmic trading project more efficient. But the main packages used throughout will be `NumPy` and `pandas`.

While `NumPy` provides the basic data structure to store numerical data and work with it, `pandas` brings powerful time series management capabilities to the table. It also does a great job of wrapping functionality from other packages into an easy-to-use API. The Bitcoin example just described shows that a single method call on a DataFrame object is enough to generate a plot with two financial time series visualized. Like `NumPy`, `pandas` allows for rather concise, vectorized code that is also generally executed quite fast due to heavy use of compiled code under the hood.

1.3. Algorithmic Trading

The term *algorithmic trading* is neither uniquely nor universally defined. On a rather basic level, it refers to the trading of financial instruments based on some formal algorithm. An *algorithm* is a set of operations (mathematical, technical) to be conducted in a certain sequence to achieve a certain goal. For example, there are mathematical algorithms to solve a Rubik's cube. [4: See The Mathematics of the Rubik's Cube or Algorithms for Solving Rubik's Cube.] Such an algorithm can solve the problem at hand via a step-by-step procedure, often perfectly. Another example is algorithms for finding the root(s) of an equation if it (they) exist(s) at all. In that sense, the objective of a mathematical algorithm is often well specified and an optimal solution is often expected.

But what about the objective of financial trading algorithm? This question is not that easy to answer in general. It might help to step back for a moment and consider motives for trading in general. In Dorn et al. (2008), they write:

> Trading in financial markets is an important economic activity. Trades are necessary to get into and out of the market, to put unneeded cash into the market, and to convert back into cash when the money is wanted. They are also needed to move money around within the market, to exchange one asset for another, to manage risk, and to exploit information about future price movements.

The view expressed here is more technical than economic in nature, focusing mainly on the process itself and only partly on *why* people initiate trades in the first place. For our purposes, a non-exhaustive list of financial trading motives of people and also of financial institution managing money of their own or for others includes:

- **beta trading**: earning market risk premia by investing, for instance, in exchange traded funds (ETFs) that replicate the performance of the S&P 500
- **alpha generation**: earning risk premia independent of the market by, for example, selling short stocks listed in the S&P 500 or ETFs on the S&P 500
- **static hedging**: hedging against market risks by buying, for example, out-of-the-money put options on the S&P 500
- **dynamic hedging**: hedging against market risks affecting options on the S&P 500 by, for example, dynamically trading futures on the S&P 500 and appropriate cash, money market, or rate instruments
- **asset-liability management**: trading S&P 500 stocks and ETFs to be able to cover liabilities resulting from, for example, writing life insurance policies
- **market making**: providing, for example, liquidity to options on the S&P 500 by buying and selling options at different bid and ask prices

All these types of trades can be implemented by a discretionary approach, with the human trader making decisions mainly on his or her own. as well as based on algorithms supporting the human trader or even replacing him completely in the decision making process. In this context, computerization of financial trading of course plays an important role. While in the beginning of financial trading, floor trading with a large group of people shouting at each other ("open outcry") was the only way of executing trades, computerization and the advent of the Internet and web technologies have revolutionized trading in the financial industry. The quote at the beginning of this chapter illustrates this impressively in terms of the number of people actively engaged in financial at Goldman Sachs in 2000 and in 2016. It is a trend that was foreseen 25 years ago, as Solomon and Corso (1991) point out:

> Computers have revolutionized the trading of securities and the stock market is currently in the midst of a dynamic transformation. It is clear that the market of the future will not resemble the markets of the past.
>
> Technology has made it possible for information regarding stock prices to be sent all over the world in seconds. Presently, computers route orders and execute small trades directly from the brokerage firm’s terminal to the exchange. Computers now link together various stock exchanges, a practice which is helping to create a single global market for the trading of securities. The continuing improvements in technology will make it possible to execute trades globally by electronic trading systems.

Interestingly, one of the oldest and most widely used algorithms is found in dynamic hedging of options. Already with the publication of the seminal papers about the pricing of European options by Black and Scholes (1973) and Merton (1973), the algorithm, called *delta hedging*, was made available — long before computerized and electronic trading even started. Delta hedging as a trading algorithm shows how to hedge away all market risks in a simplified, perfect, continuous model world. In the real world, with transaction costs, discrete trading, imperfectly liquid markets, and other frictions ("imperfections"), the algorithm has proven — somewhat surprisingly maybe — its usefulness and robustness as well. It might not allow to perfectly hedge away market risks affecting options, but it is useful in getting close to the ideal and is therefore still used on a large scale in the financial industry. [5: See Hilpisch (2015) for a detailed analysis of delta hedging strategies for European and American options using Python.]

This book focuses on algorithmic trading in the context of *alpha generating strategies*. Although there are more sophisticated definitions for alpha, for the purposes of this book alpha is seen as the difference between a trading strategy's return over some period of time and the return of the benchmark (single stock, index, cryptocurrency, etc.). For example, if the S&P 500 returns 10% in 2018 and an algorithmic strategy returns 12%, then alpha is +2% points. If the strategy returns 7%, then alpha is -3% points. In general, such numbers are not adjusted for risk, and other risk characteristics like maximal drawdown (period) are usually considered to be of second order importance, if at all.

This book focuses on alpha-generating strategies, i.e. strategies that try to generate positive returns (above a benchmark) independent of the market's performance itself. Alpha is defined in this book in the simplest way as the excess return of a strategy over the benchmark financial instrument.

There are other areas where trading-related algorithms play an important role. One is the *high frequency trading* (HFT) space, where speed is typically the discipline in which players compete. [6: See the book by Lewis (2015) for a non-technical introduction to HFT.] The motives for HFT are diverse, but market making and alpha generation probably play a prominent role. Another one is *trade execution*, where algorithms are deployed to optimally execute certain non-standard trades. Motives in this area might include the execution (at best possible prices) of large orders or the execution of an order with as little market and price impact as possible. A more subtle motive might be to disguise an order by executing it on a number of different exchanges.

An important question remains to be addressed: is there any advantage to using algorithms for trading instead of human research, experience, and discretion? This question can hardly be answered in any generality. For sure, there are human traders and portfolio managers who have earned, on average, more than their benchmark for investors over longer periods of time. The paramount example in this regard is Warren Buffett. On the other hand, statistical analyses show that the majority of active portfolio managers rarely beat relevant benchmarks consistently. Referring to the year 2015, Adam Shell writes:

> Last year, for example, when the Standard & Poor's 500-stock index posted a paltry total return of 1.4% with dividends included, 66% of "actively managed" large-company stock funds posted smaller returns than the index … The longer-term outlook is just as gloomy, with 84% of large-cap funds generating lower returns than the S&P 500 in the latest five year period and 82% falling shy in the past 10 years, the study found. [8: Source: "66% of Fund Managers Can't Match S&P Results." USA Today, March 14, 2016.]

In an empirical study published in December 2016, Harvey et al. (2016) write:

> We analyze and contrast the performance of discretionary and systematic hedge funds. Systematic funds use strategies that are rules-based, with little or no daily intervention by humans … We find that, for the period 1996-2014, systematic equity managers underperform their discretionary counterparts in terms of unadjusted (raw) returns, but that after adjusting for exposures to well-known risk factors, the risk-adjusted performance is similar. In the case of macro, systematic funds outperform discretionary funds, both on an unadjusted and risk-adjusted basis.

Annualized performance of hedge fund categories reproduces the major quantitative findings of the study by Harvey et al. (2016). [9: Annualized performance (above the short term interest rate) and risk measures for hedge fund categories comprising a total of 9,000 hedge funds over the period from June 1996 to December 2014.] In the table, *factors* include traditional ones (equity, bonds, etc.), dynamic ones (value, momentum, etc.), and volatility (buying at-the-money puts and calls). The *adjusted return appraisal ratio* divides alpha by the adjusted return volatility. For more details and background, see the paper itself.

The study’s results illustrate that systematic ("algorithmic") macro hedge funds perform best as a category, both in unadjusted and risk-adjusted terms. They generate an annualized alpha of 4.85% points over the period studied. These are hedge funds implementing strategies that are typically global, cross-asset, and often involve political and macroeconomic elements. Systematic equity hedge funds only beat their discretionary counterparts on the basis of the adjusted return appraisal ratio (0.35 vs. 0.25).

Table 1. Annualized performance of hedge fund categories

	systematic macro	**discretionary macro**	**systematic equity**	**discretionary equity**
return average	5.01%	2.86%	2.88%	4.09%
return attributed to factors	0.15%	1.28%	1.77%	2.86%
adj. return average (alpha)	4.85%	1.57%	1.11%	1.22%
adj. return volatility	10.93%	5.10%	3.18%	4.79%
adj. return appraisal ratio	0.44	0.31	0.35	0.25

1.4. Python for Algorithmic Trading

Python is used in many corners of the financial industry, but has become particularly popular in the algorithmic trading space. There are a few good reasons for this:

- **data analytics capabilities**: A major requirement for every algorithmic trading project is the ability to manage and process financial data efficiently. Python, in combination with packages like `NumPy` and `pandas`, makes life easier in this regard for every algorithmic trader than most other programming languages.
- **handling of modern APIs**: Modern online trading platforms like Oanda and Gemini offer RESTful application programming interfaces (APIs) and socket

(streaming) APIs to access historical and live data. Python is really good at interacting with such APIs.

- **dedicated packages**: In addition to the standard data analytics packages, there are multiple packages available that are dedicated to the algorithmic trading space, such as PyAlgoTrade and Zipline for the backtesting of trading strategies, and Pyfolio for performing portfolio and risk analysis.
- **vendor sponsored packages**: More and more vendors in the space release open source Python packages to facilitate access to their offerings; among them are online trading platforms like Oanda as well as the leading data providers like Bloomberg and Thomson Reuters.
- **dedicated platforms**: Quantopian, for example, offers a standardized backtesting environment as a web-based platform where the language of choice is Python and where people can exchange ideas with like-minded others via different social network features. Near the end of 2016, Quantopian reported that it had attracted more than 100,000 users.
- **buy- and sell-side adoption**: More and more institutional players have adopted Python to streamline development efforts in their trading departments. This, in turn, requires more and more staff proficient in Python, which makes learning Python a worthwhile investment.
- **education, training, and books**: Prerequisites for the wide-spread adoption of a technology or programming language are academic and professional education and training programs in combination with specialized books and other resources. The Python ecosystem has seen a tremendous growth in such offerings recently, educating and training more and more people in the use of Python for finance. This can be expected to reinforce the trend of Python adoption in the algorithmic trading space.

In summary, it is rather safe to say that Python plays an important role in algorithmic trading already, and seems to have strong momentum to become even more important in the near future. It is therefore a good choice for anyone trying to enter the space, be it as an ambitious “hobby” trader or as a professional employed by a leading financial institution engaged in automated trading.

1.5. Focus and Prerequisites

The focus of this book is on Python as a programming language *for* algorithmic trading. The book assumes that the reader already has some experience with Python and popular Python packages used for data analytics. Good introductory books are, for example, Hilpisch (2014), McKinney (2017), and VanderPlas (2016), which all can be consulted to build a solid foundation in Python for data analysis and finance. The reader is also expected to have some experience with typical tools used for interactive analytics with Python, such as IPython, to which VanderPlas (2016) also provides an introduction.

This book presents and explains Python code that is applied to the topics at hand, like backtesting trading strategies or working with streaming data. It cannot provide a thorough introduction to all packages used in different places. It tries, however, to highlight those capabilities of the packages that are central to the exposition (such as vectorization with `NumPy`).

The book also cannot provide a thorough introduction and overview of all financial and operational aspects relevant for algorithmic trading. The approach instead focuses on the use of Python to build the necessary infrastructure for automated, algorithmic trading systems. Of course, the majority of examples used are taken from the algorithmic trading space. However, when dealing with, say, momentum or mean-reversion strategies, they are more or less simply used without providing (statistical) verification or an in-depth discussion of their intricacies. Whenever it seems appropriate, references are given that point the reader to sources that address issues left open during the exposition.

All in all, this book is written for readers who have some experience with both Python and (algorithmic) trading. For such a reader, the book is a practical guide to the creation of automated trading systems using Python and additional packages.

This book uses a number of Python programming approaches (e.g. object oriented programming) and packages (e.g. scikit-learn) that cannot be explained in detail. The focus is on *applying* these approaches and packages to different steps in an algorithmic trading process. It is therefore recommended that those who do not yet have enough Python (for finance) experience additionally consult more introductory Python texts.

1.6. Trading Strategies

Throughout this book, four different algorithmic trading strategies are used as examples. They are introduced briefly below and in some more detail in Mastering Vectorized Backtesting. All these trading strategies can be classified as mainly *alpha seeking strategies* since their main objective is to generate positive, above-market returns independent of the market direction. Canonical examples throughout the book when it comes to financial instruments traded are a *stock index, a single stock*, or *a cryptocurrency* (denominated in a fiat currency). The book does not cover strategies involving multiple financial instruments at the same time (pair trading strategies, strategies based on baskets, etc.). It also covers only strategies whose trading signals are derived from structured, financial time series data and not, for instance, from unstructured data sources like news or social media feeds. This keeps the discussions and the Python implementations concise and easier to understand, in line with the approach (discussed earlier) of focusing on Python *for* algorithmic trading. [10: See the book by Kissel (2013) for an overview of topics related to algorithmic trading, the book by Chan (2013) for an in-depth discussion of momentum and mean-reversion strategies, or the book by Narang (2013) for a coverage of quantitative and HFT trading in general.]

The remainder of this section gives a quick overview of the four trading strategies used in this book.

1.6.1. Simple Moving Averages

The first type of trading strategy relies on simple moving averages (SMAs) to generate trading signals and market positionings. These trading strategies have been popularized by so-called technical analysts or chartists. The basic idea is that a shorter-term SMA being higher in value than a longer term SMA signals a long market position and the opposite scenario signals a neutral or short market position.

1.6.2. Momentum

The basic idea behind momentum strategies is that a financial instrument is assumed to perform in accordance with its recent performance for some additional time. For example, when a stock index has seen a negative return on average over the last five days, it is assumed that its performance will be negative tomorrow as well.

1.6.3. Mean-Reversion

In mean-reversion strategies, a financial instrument is assumed to revert to some mean or trend level if it is currently far enough away from such a level. For example, assume that a stock trades 10 USD under its 200 days SMA level of 100. It is then expected that the stock price will return to its SMA level sometime soon.

1.6.4. Machine and Deep Learning

With machine and deep learning algorithms, one generally takes a more black box-like approach to predicting market movements. In this book, we mainly rely on historical return observations as features to train machine and deep learning algorithms to predict stock market movements.

This book does not introduce to algorithmic trading in a systematic fashion. Since the focus lies on applying Python in this fascinating field, readers not familiar with algorithmic trading should consult other, dedicated resources on the topic, some of which are cited in this chapter and the others that follow. But be aware of the fact that the algorithmic trading world in general is secretive and that almost everybody who is successful there is naturally reluctant to share his or her secrets in order to protect their sources of success, i.e. alpha.

1.7. Overview

Here’s a quick overview of the topics presented in each chapter:

Setting up the Python Environment

Lays the foundation for all subsequent chapters in that it shows how to set up a proper Python environment. This chapter mainly uses conda as a package and environment manager, and illustrates Python deployment via Docker containers and in the cloud.

Working with Financial Data

Financial times series data is central to every algorithmic trading project. This chapter shows you how to retrieve financial data from different public data and also proprietary data sources. It also demonstrates how to store financial time series data efficiently with Python.

Mastering Vectorized Backtesting

Vectorization is a powerful approach in numerical computation in general and for financial analytics in particular. This chapter introduces vectorization with `NumPy` and `pandas`, and applies that approach to backtesting SMA-based, momentum, and mean-reversion strategies.

Predicting Market Movements with Machine Learning

This chapter is dedicated to generating market predictions by the use of machine learning and deep learning approaches. By mainly relying on past return observations as features, approaches are presented for predicting tomorrow’s market direction by using such Python packages as scikit-learn and tensorflow.

Building Classes for Event-based Backtesting

While vectorized backtesting has advantages when it comes to conciseness of code and performance, it’s limited with regard to the representation of certain market features of trading strategies; on the other hand, event-based backtesting—technically implemented by the use of object oriented programming—allows for a rather granular and more realistic modeling of such features. This chapter presents and explains in detail a base class as well as two classes for the backtesting of long-only and long-short trading strategies.

Working with Real-Time Data and Sockets

Needing to cope with real-time or streaming data is a reality even for the ambitious individual algorithmic trader. The tool of choice is socket programming, for which this chapter introduces ZeroMQ as a lightweight and scalable technology. The chapter also illustrates how to make use of Plotly to create nice looking, interactive, streaming plots. It also presents a wrapper class that simplifies the creation of such plots in cases where multiple data streams need to be visualized simultaneously (e.g. in a dashboard-like manner).

CFD Trading with Oanda

Oanda is a leading Contracts for Difference (CFD) trading platform offering a multitude of tradable instruments, e.g. based on foreign exchange pairs, stock indices, commodities or rates instruments (benchmark bonds). This chapter provides guidance on how to implement automated, algorithmic trading strategies with Oanda.

Stock Trading with Interactive Brokers

Interactive Brokers is a leading online brokerage platform that focuses on stocks and options trading. The chapter deals with the Interactive Brokers API which is technologically based on the Trader Workstation application. It introduces a Python wrapper class that makes life quite convenient and efficient in this context.

Trading Cryptocurrencies with Gemini

Cryptocurrencies and related technologies, like blockchains, have been a rather popular topic in technology as well as financial circles recently. The chapter covers Gemini as one of the modern platforms that allow for the automated trading of cryptocurrencies, like Bitcoin or Ether. The chapter presents Python wrapper classes to simplify most of the typical operations in algorithmic trading considerably.

Automating Trading Operations

This chapter deals with capital management, risk analysis and management as well as with typical tasks in the technical automation of algorithmic trading operations. It covers, for instance, the Kelly criterion for capital allocation and leverage in detail.

Appendix A: Python, NumPy, matplotlib, pandas

This appendix provides a concise introduction to the most important Python, `NumPy` and `pandas` topics in the context of the material presented in the main chapters. It represents a starting point from which one can add to one's own Python knowledge over time.

1.8. Conclusions

Python is already a force in finance in general, and is on its way to becoming a major force in algorithmic trading. There are a number of good reasons to use Python for algorithmic trading, among them the powerful ecosystem of packages that allow for efficient data analysis or the handling of modern APIs. There are also a number of good reasons to learn Python for algorithmic trading, chief among them the fact that some of the biggest buy- and sell-side institutions make heavy use of Python in their trading operations and constantly look for seasoned Python professionals.

This book and online course focuses on applying Python to the different disciplines in algorithmic trading, like backtesting trading strategies or interacting with online trading platforms. It cannot replace a thorough introduction to Python itself nor to trading in general. However, it systematically combines these two fascinating worlds to provide a valuable source for the generation of alpha in today's competitive financial and cryptocurrency markets.

1.9. Further Resources

Research papers cited in this chapter:

- Black, Fischer and Myron Scholes (1973): "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy*, Vol. 81, No. 3, 638-659.
- Harvey, Campbell, Sandy Rattray, Andrew Sinclair and Otto Van Hemert (2016): "Man vs. Machine: Comparing Discretionary and Systematic Hedge Fund Performance." White Paper, Man Group.
- Dorn, Anne, Daniel Dorn, and Paul Sengmueller (2008): "Why do People Trade?" *Journal of Applied Finance*, Fall/Winter, 37-50.

- Merton, Robert (1973): “Theory of Rational Option Pricing.” *Bell Journal of Economics and Management Science*, Vol. 4, 141-183.
- Solomon, Lewis and Louise Corso (1991): “The Impact of Technology on the Trading of Securities: The Emerging Global Market and the Implications for Regulation.” *The John Marshall Law Review*, Vol. 24, No. 2, 299-338.

Books cited in this chapter:

- Chan, Ernest (2013): *Algorithmic Trading*. John Wiley & Sons, Hoboken et al.
- Kissel, Robert (2013): *Algorithmic Trading and Portfolio Management*. Elsevier/Academic Press, Amsterdam et al.
- Lewis, Michael (2015): *Flash Boys*. W.W. Norton & Company, New York & London.
- Hilpisch, Yves (2014): *Python for Finance*. O’Reilly, Bejing et al. 2nd ed. coming out in 2018. Resources under http://pff.tpq.io.
- Hilpisch, Yves (2015): *Derivatives Analytics with Python*. Wiley Finance. Resources under http://dawp.tpq.io.
- McKinney, Wes (2017): *Python for Data Analysis*. 2nd ed., O’Reilly, Bejing et al.
- Narang, Rishi (2013): *Inside the Black Box*. John Wiley & Sons, Hoboken et al.
- VanderPlas, Jake (2016): *Python Data Science Handbook*. O’Reilly, Bejing et al.

Author Biography

Dr. Yves J. Hilpisch is founder and managing partner of The Python Quants, a group focusing on the use of open source technologies for financial data science, algorithmic trading and computational finance. He is the author of the books

- Python for Finance (O'Reilly, 2014),
- Derivatives Analytics with Python (Wiley, 2015) and
- Listed Volatility and Variance Derivatives (Wiley, 2017).

Yves lectures on computational finance at the CQF Program, on data science at htw saar University of Applied Sciences and is the director for the online training program leading to the first Python for Finance & Python for Algorithmic Trading University Certificates (awarded by htw saar).

Yves has written the financial analytics library DX Analytics and organizes meetups and conferences about Python for quantitative finance in Frankfurt, London and New York. He has also given keynote speeches at technology conferences in the United States, Europe and Asia.