Hands-On Machine Learning for Algorithmic Trading

Design and implement smart investment strategies to analyze market behavior using the Python ecosystem

Packt

www.packt.com

Stefan Jansen

Table of Contents

Preface

Availability of diverse data has increased the demand for expertise in algorithmic trading strategies. With this book, you will select and apply machine learning to a broad range of data sources and create powerful algorithmic strategies.

This book will start with introducing you to essential elements like evaluating datasets, accessing data API's using Python, using Quandl to access financial data, and managing prediction error. We then cover various machine learning techniques and algorithms that can be used to build and train algorithmic models using pandas, seaborn, statsmodels, and sklearn. We will then build, estimate and interpret AR(p), MA(q) and ARIMA(p, d, q) models using statsmodels. You will apply Bayesian concepts of prior, evidence, and posterior, to distinguish the concept of uncertainty using PyMC3. We will then utilize NLTK, sklearn and SpaCy to assign sentiment scores to financial news, classify documents to extract trading signals. We will learn to design, build, tune and evaluate feed-forward neural network, RNN, CNN using Keras to design sophisticated algorithms. You will apply transfer learning on satellite image data to predict economic activity. Finally, we will apply reinforcement learning for optimal trading results.

By the end, you will learn to adopt algorithmic trading to implement smart investing strategies.

Who this book is for

The book is for data analysts, data scientists, python developers as well as investment analysts and portfolio managers working within the finance and investment industry. If you want to perform efficient algorithmic trading by developing smart investigating strategies using machine learning algorithms, this is the book you need! Some understanding of Python and machine learning techniques is mandatory.

What this book covers

Chapter 1, From Trade Idea to Execution, This chapter identifies the focus of the book by outlining how machine learning matters in generating and evaluating signals for the design and execution of a trading strategy. It outlines the strategy process from hypothesis generation and modeling, data selection and back testing to evaluation and execution in a portfolio context incl. risk management.

Chapter 2, Market and Fundamental Data, This chapter provides a framework for navigating the proliferating supply of data sources that are combined in algorithmic trading (AT). It introduces criteria to evaluate datasets against specific investment objectives, summarizes conventional data sources, and explains how new sources are produced by individuals, business processes and sensor data.

Chapter 3, Working with Alternative Data, This chapter provides a framework for navigating the proliferating supply of data sources that are combined in algorithmic trading (AT). It introduces criteria to evaluate datasets against specific investment objectives, summarizes conventional data sources, and explains how new sources are produced by individuals, business processes and sensor data.

Chapter 4, Alpha Factors: Research and Evaluation, AT happens in a portfolio context. This chapter introduces the portfolio (PF) concept and reviews the Sharpe Ratio (main PF performance measure). It outlines the CAPM model as a framework to discuss sources of alpha, contrasts it with the Arbitrage Pricing Theory, and summarizes the Efficient Market Hypothesis

Chapter 5, Strategy Evaluation and Portfolio Management, AT happens in a portfolio context. This chapter introduces the portfolio (PF) concept and reviews the Sharpe Ratio (main PF performance measure). It outlines the CAPM model as a framework to discuss sources of alpha, contrasts it with the Arbitrage Pricing Theory, and summarizes the Efficient Market Hypothesis

Chapter 6, The Machine Learning Process, This chapter introduces key concepts of machine learning by contrasting the objectives of regression and classification, and the basic linear models to address each. It explains how errors are measured in each case, and derives the linear and logistic regression. It compares in-sample training & performance measures (incl. statistical significance) using statsmodels with predictive performance using sklearn. It also reviews key data considerations, visualization techniques and transformations during data preparation for each model.

Chapter 7, Generalized Linear Models, In this chapter, you will learn how linear regression works and which assumptions it makes, how to train and diagnose linear regression model and use linear regression to predict future returns. You will also learn to use regularization to improve the predictive performance

Chapter 8, Linear Time Series Models, This chapter reviews autocorrelation, moving average and stationarity using the pandas and statsmodels libraries and introduces the AR(I)MA models for univariate time series that build on these concepts. It explains how to estimate each model, interpret the results, generate and evaluate predictions, and how to select between alternatives.

Chapter 9, Decision Trees and Random Forests, This chapter introduces tree-based models to build non-linear models of either high transparency or high accuracy. It lays out the objective function of decision tree models for regression and classification, and visualizes the decision boundary and explains the Gini, Entropy and misclassification error metrics for the classification case. It then shows to train & tune/regularize decision trees using sklearn, and visualize the result using simple financial data.

Chapter 10, Gradient Boosting Machines, This chapter introduces gradient boosting, a second ensemble model that often performs best on structured data typical in financial markets. It builds on the tree-model logic laid out in the previous chapter and explains how gradient boosting continues to add new trees to the ensemble by optimizing against errors of prior trees. It proceeds to provide a thorough tutorial to catBoost, the most recent state-of-the-art implementation.

Chapter 11, Bayesian Machine Learning for Algorithmic Trading using PyMC3, This chapter adds the Bayesian perspective on machine learning using the PyMC3 library. It reviews basic concepts of Bayesian statistics like prior, evidence, and posterior distribution and compares it to frequentist statistics. It then introduces Bayesian Machine Learning and the PyMC3 library using linear regression, illustrates how to build, train and evaluate a model, and contrasts the results to the conventional approach.

Chapter 12, From text to numbers: the document-text matrix and text classification, This chapter introduces the bag-of-words model represented by the document-term matrix and its use in text classification. It demonstrates how to use sklearn to tokenize text (optionally using SpaCy for lemmatization) and produce a matrix that represents documents as sparse term vectors.

Chapter 13, Making sense of very large language data: Topic Modeling using gensim, This chapter introduces unsupervised topic models that aim to represent text more efficiently than document-term matrices. It outlines the evolution from Latent Semantic Analysis to Latent Dirichlet Allocation (LDA). It demonstrates how LDA works in theory and applied using gensim. It introduces pyLDAvis to visualize topics found by the algorithm, and topic coherence as a quantitative metric to evaluate the quality of topics.

Chapter 14, State-of-the-art text features: word2vec models, This chapter introduces the state-of-the art approach to represent individual tokens as continuous vectors. The goal is to convert text data into more informative features that capture more semantic information than the bag-of-word model. The value consists in the training of domain-specific models that capture relationship of words specific to, e.g., a financial context. Word2vec and FastText are the two most popular approaches to generate word vectors.

Chapter 15, Introduction to Deep Neural Networks, This chapter introduces key elements of neural network model architectures, training methods, and applications. It explains how individual neurons operate, how they connect to form a network, and how the parameters of the model are learned from data using back-propagation. It lays out the challenges in training deep networks, explains strategies to manage this process, including regularization, data augmentation, early stopping, dropout and initialization strategies.

Chapter 16, Recurrent Neural Networks, This chapter introduces recurrent neural networks (RNN). It lays out how RNN work, and how RNN are much more general sequence-to-sequence model than the linear time series models covered in the previous part, in particular through their ability to capture non-linear effects. It shows how RNN faced similar challenges in learning deeper relationships, and how the LSTM architecture allowed RNN to overcome this barrier.

Chapter 17, Convolutional Neural Networks: Image Recognition for Satellite Data, This chapter introduces convolutional network architectures for image recognition. It outlines how various forms of image (satellite) data are being used to predict specific economic activity. It then explains how convolutional layers operate, why they are useful to extract features from images in a hierarchical fashion, and how this process can be visualized. It also highlights which architectures have proven successful for various relevant tasks.

Chapter 18, Autoencoders, In this chapter, we will go through autoencoders.

Chapter 19, Reinforcement learning, This chapter introduces reinforcement learning that enables algorithms to dynamically learn from experience. Reinforcement learning solves a Markov Decision Problem that consists of states, actions, transitions between states, and rewards and aims to find a policy that maximizes the reward for a given time horizon. Trading in markets is a reinforcement-learning problem with optimal trading as the solution

Chapter 20, Next Steps, this chapter covers the summary of all the previous chapters.

To get the most out of this book

1. Inform the reader of the things that they need to know before they start, and spell out what knowledge you are assuming.
2. Any additional installation instructions and information they need for getting set up.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit www.packt.com/support and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at www.packt.com.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at https://github.com/PacktPublishing/Book-Name. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at https://github.com/PacktPublishing/. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: http://www.packtpub.com/sites/default/files/downloads/Bookname_ColorImages.pdf.

Conventions used

There are a number of text conventions used throughout this book.

CodeInText: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Mount the downloaded WebStorm-10*.dmg disk image file as another disk in your system."

A block of code is set as follows:

```
html, body, #map {
 height: 100%;
 margin: 0;
 padding: 0
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Any command-line input or output is written as follows:

```
$ mkdir css
$ cd css
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Select **System info** from the **Administration** panel."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at customercare@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
From Trade Idea to Execution

Algorithmic trading relies on computer programs that execute algorithms to automate some or all elements of a trading strategy. Algorithms are a sequence of steps or rules to achieve a goal and can take many forms. In the case of machine learning, algorithms pursue the objective of learning other algorithms, namely rules to achieve an objective based on data, such as minimizing a prediction error.

These algorithms encode various activities of a portfolio manager who observes market transactions and analyzes relevant data to decide on placing buy or sell orders. The sequence of orders defines the portfolio holdings that over time aim to produce returns that are attractive to the providers of capital given their appetite for risk.

Ultimately, the goal of active investment management consists in achieving **alpha**, that is, returns in excess of the benchmark used for evaluation. The fundamental law of active management uses the **Information Ratio** (**IR**) to express the value of active management as the ratio of portfolio returns above the returns of a benchmark, usually an index, to the volatility of those returns. It approximates the information ratio as the product of the **information coefficient** (**IC**), which measures the quality of forecast as their correlation with outcomes, and the breadth of a strategy expressed as the square root of the number of bets (`Grienold, Kahn 1989`).

Hence, the key to generating alpha is **forecasting**. That requires superior information or superior ability to process public information. Algorithms facilitate optimization throughout the investment process, from asset allocation to idea generation, trade execution and risk management. The use of machine learning for algorithmic trading, in particular, aims for more efficient use of conventional data and alternative data, with the goal of producing both better and more actionable forecasts, hence improving the value of active management.

Historically, algorithmic trading used to be more narrowly defined as the automation of trade execution to minimize costs as offered by the sell side, but we will take a broader perspective since the use of algorithms and machine learning, in particular, has come to impact a broader range of activities, including on the buy side of the market.

This chapter looks at the bigger picture of how the use of machine learning has emerged as a key source of competitive advantage in the investment industry and where it fits into the investment process to enable algorithmic trading strategies.

We will be covering the following topics in the chapter:

- Industry trends
- Designing a trading strategy
- Algorithmic strategies and signals
- How machine learning fits into the process
- Focus and organization of the book

It summarizes industry trends in quantitative investment and algorithmic trading, identifies central elements of a trading strategy, and highlights where and how machine learning adds value to the process of designing, testing and executing a trading strategy. It also outlines the organization of the book, identifies the target group and outlines the prerequisites to get the most out of it to successfully apply machine learning for algorithmic trading in practice.

Industry trends

The investment industry has evolved dramatically over the last several decades and continues to do so amid increased competition, technology advances, and a challenging economic environment. This section will review several key trends that have shaped the investment environment in general, and the context for algorithmic trading more specifically, and related themes will recur throughout this book.

The trends that have propelled algorithmic trading and machine learning to current prominence include:

- Changes in the market microstructure, such as the spread of electronic trading and the integration of markets across asset classes and geographies

- The development of investment strategies framed in terms of risk factor exposure as opposed to asset classes, and

- The revolutions in computing power, data generation and management, and analytic methods, and

- The outperformance of the pioneers in algorithmic traders relative to human, discretionary investors

In addition, the financial crises of 2001 and 2008 have affected how investors approach diversification and risk management and have given rise to low-cost passive investment vehicles in the form of exchange-traded funds (**ETF**). Amid low yield and low volatility after the 2008 crisis, cost-conscious investors shifted $2trn from actively managed mutual funds to into passively managed ETF. Competitive pressure is also reflected in lower hedge fund fees that dropped from the traditional 2% annual management fee and 20% take of profits to an average of 1.48% and 17.4%, respectively in 2017 (Barron).

From electronic to high-frequency trading

Electronic trading has advanced dramatically in terms of capabilities, volume and coverage of asset classes and geographies since networks started routing prices to computer terminals in the 1960s and a bit later began to also transmit orders and trades.

Equity markets have led this trend worldwide. The 1997 order handling rules by the **Securities and Exchange Commission** (**SEC**) introduced competition to exchanges through **Electronic Communication Networks** (**ECN**). ECNs are automated **alternative trading systems** (**ATS**) that match buy and sell orders at specified prices, primarily for equities and currencies and are registered as broker-dealers. It allows major brokerages and individual traders in different geographic locations to trade directly without intermediaries, both on exchanges and after hours. Dark pools are another type of ATS that allow investors to place orders and trade without publicly revealing their information as in the order book maintained by an exchanged. Dark pools have grown since a 2007 SEC ruling, are often housed within large banks, and subject to SEC regulation.

With the rise of electronic trading, algorithms for cost-effective execution have developed rapidly and adoption has spread quickly from sell side to buy side and across asset classes. Automated trading emerged around 2000 as a sell-side tool aimed at cost-effective trade execution that spread orders over time to limit the market impact. These tools spread to the buy side and became increasingly sophisticated by taking into account. For example, transaction costs and liquidity, as well as short-term price and volume forecasts.

Direct Market Access (**DMA**) gives a trader greater control over execution by allowing it to send orders directly to the exchange using the infrastructure and market participant identification of a broker who is a member of an exchange. Sponsored access removes pre-trade risk controls by the brokers and forms the basis for **high-frequency trading** (**HFT**).

HFT refers to automated trades in financial instruments that are executed with extremely low latency in the microsecond range and where participants hold positions for very short periods of time. The goal is to detect and exploit inefficiencies in the market microstructure, the institutional infrastructure of trading venues. HFT has grown substantially over the past 10 years and is estimated to make up roughly 55% of trading volume in U.S. equity markets and about 40% in European equity markets. HFT has also grown in futures markets to roughly 80% of foreign exchange futures volumes and two-thirds of both interest rate and Treasury 10-year futures volumes (FAS 2016).

HFT strategies aim to earn small profits per trade using passive or aggressive strategies. **Passive strategies** include arbitrage trading to profit from very small price differentials for the same asset or its derivatives traded on different venues, and passive market making that profits from spreads between bid and ask prices. **Aggressive strategies** include order anticipation or momentum ignition. Order anticipation, also known as liquidity detection, involves algorithms that submit small exploratory orders to detect hidden liquidity from large institutional investors and trade ahead of a large order to benefit from subsequent price movements. Momentum ignition implies an algorithm executing and canceling a series of orders to spoof other HFT algorithms into buying (or selling) more aggressively and benefit from the resulting price changes.

Regulators have expressed concern over the potential link between certain aggressive HFT strategies and increased **market fragility and volatility**, such as that experienced during the May 2010 Flash Crash, the October 2014 Treasury market volatility, and the sudden crash by over 1,000 points of the Dow Jones Industrial Average on August 24, 2015. At the same time, market liquidity has increased with trading volumes due to the presence of HFT, which has lowered overall transaction costs.

The combination of reduced trading volumes amid lower volatility and rising costs of the technology and access to both data and trading venues have led to **financial pressure**. Aggregate HFT revenues from US stocks have been estimated to drop below US$ 1bn for the first time since 2008, down from US$ 7.9bn in 2009 (FT, 1/1/2018).

This trend has led to **industry consolidation** with various acquisitions by. For example, the largest listed proprietary trading firm Virtu Financial, and shared infrastructure investments like the new Go West ultra-low latency route between Chicago and Tokyo. Simultaneously, startups like Alpha Trading Lab make HFT trading infrastructure and data available to democratize HFT by crowdsourcing algorithms in return for a share of profits.

Factor investing and smart beta funds

The return provided by an asset is a function of the uncertainty or risk associated with the financial investment. An equity investment implies. For example, assuming a company's business risk, and a bond investment implies assuming default risk.

To the extent that specific risk characteristics predict returns, identifying and forecasting the behavior of these risk factors becomes a primary focus when designing an investment strategy. It yields valuable trading signals and is the key to superior active management results. The industry's understanding of risk factors has evolved very substantially over time and has impacted how machine learning is used for algorithmic trading.

Modern Portfolio Theory (MPT) introduced the distinction between idiosyncratic and systematic sources of risk for a given asset. Idiosyncratic risk can be eliminated through diversification, but systematic risk cannot. In the early 1960s, the **Capital Asset Pricing Model** (**CAPM**) identified a single factor driving all asset returns: the return on the market portfolio in excess of T-bills. The market portfolio consisted of all tradeable securities, weighted by their market value. The systematic exposure of an asset to the market is measured by beta, the correlation between the returns of the asset and the market portfolio.

The recognition that the risk of an asset does not depend on the asset in isolation, but rather how it moves relative to other assets and to the market as a whole was a major conceptual breakthrough. In other words, assets do not earn a risk premium because of their specific, idiosyncratic characteristics, but rather because of their exposure to underlying factor risk.

However, a large body of academic literature and long investing experience have disproved the CAPM prediction that asset risk premiums depend only on its exposure to a single factor measured by the asset's beta. Instead, numerous **additional risk factors** have since been discovered. A factor is a quantifiable signal, attribute, or any variable that has historically correlated with future stock returns and is expected to remain correlated in future.

These risk factors were labeled **anomalies** since they contradicted the **Efficient Market Hypothesis** (**EMH**), which sustained that market equilibrium would always price securities according to the CAPM so that no other factors should have predictive power. The economic theory behind factors can be either rational, where factor risk premiums compensate for low returns during bad times, or behavioral, where agents fail to arbitrage away excess returns.

Well-known anomalies include the value, size, and momentum effects that help predict returns while controlling for the CAPM market factor. The **size effect** rests on small firms systematically outperforming large firms, discovered by Banz (1981) and Reinganum (1981). The **value effect** (Basu 1982) consists in firms with low price multiples like the price-to-earnings or the price-to-book ratios performing better than their more expensive peers (as suggested by the inventors of value investing Benjamin Graham and David Dodd and popularized by Warren Buffet).

The **momentum effect**, discovered in the late 1980s by, among others, Clifford Asness, founding partner of AQR, states that stocks with good momentum in terms of recent 6-12 months returns have higher returns going forward than poor momentum stocks with similar market risk. Researchers also found that value and momentum factors explain returns for stocks outside the US as well as for other asset classes like bonds, currencies, and commodities, and added additional risk factors.

In fixed income, the value strategy is called **riding the yield curve** and is a form of the duration premium. In commodities, it is called the roll return, with a positive return for an upward-sloping futures curve and a negative return otherwise. In foreign exchange, the value strategy is called carry.

There is also an **illiquidity premium**. Securities that are more illiquid trade at low prices and have high average excess returns, relative to their more liquid counterparts. Bonds with higher default risk tend to have higher returns on average, reflecting a credit risk premium. Since investors are willing to pay for insurance against high volatility when returns tend to crash, sellers of volatility protection in options markets tend to earn high returns.

Multifactor models define risks in broader and more diverse terms than just the market portfolio. In 1976, Stephen Ross proposed **Arbitrage Pricing Theory** that asserted that investors are compensated for multiple systematic sources of risk that cannot be diversified away. The three most important macro factors are growth, inflation, and volatility in addition to productivity, demographic, and political risk. In 1992, Eugene **Fama** and Kenneth **French** combined the equity risk factors size and value with a market factor into a single model that better explained cross-sectional stock returns. They later added a model that also included bond risk factors two simultaneously explain returns for both asset classes.

A particularly attractive aspect of risk factors is their **low or negative correlation**. Value and momentum risk factors, for instance, are negatively correlated, reducing the risk and increasing risk-adjusted returns above and beyond the benefit implied by the risk factors. Furthermore, using leverage and long-short strategies, factor strategies can be combined into market-neutral approaches. The combination of long positions in securities exposed to positive risks with underweight or short positions in the securities exposed to negative risks allows for the collection of dynamic risk premiums.

As a result, the factors that explained returns above and beyond the CAPM were incorporated into investment styles that tilt portfolios in favor of one or more factors, and assets began to migrate into **factor-based portfolios**. The 2008 financial crisis underlined how asset class labels can be highly misleading and create a false sense of diversification when investors do not look at the underlying factor risks as asset classes came crashing down together.

Over the past several decades, quantitative factor investing has evolved from a simple approach based on two or three styles to **multi-factor smart or exotic beta** products. Smart beta funds have crossed $1tn AUM in 2017, testifying to the popularity of the hybrid investment strategy that combines active and passive management. Smart beta funds take a passive strategy but modify it according to one or more factors, such as cheaper stocks or screening them according to dividend payouts, in order to generate better returns. This growth has coincided with increasing criticism of the high fees charged by traditional active managers as well as heightened scrutiny of their performance (FT 10/01/2018).

The ongoing discovery and successful forecasting of risk factors that, either individually or in combination with other risk factors, significantly impact future asset returns across asset classes is a **key driver of the surge in machine learning** in the investment industry and will be a key theme throughout this book.

Algorithmic pioneers outperform humans at scale

The track record and growth of **assets under management** (**AUM**) of firms that spearheaded algorithmic trading has played a key role in generating investor interest and subsequent industry efforts to replicate their success. Systematic funds differ from HFT in that trades may be held significantly longer while seeking to exploit arbitrage opportunities as opposed to advantages from sheer speed.

Systematic strategies that mostly or exclusively rely on algorithmic decision-making were most famously introduced by mathematician James Simons who founded Renaissance Technologies in 1982 and built it into the premier quant firm. Its secretive Medallion Fund, which is closed to outsiders, has earned an estimated annualized return of 35 percent since 1982 (II 07/07/2017).

DE Shaw, Citadel and Two Sigma, three of the most prominent quantitative hedge funds that use systematic strategies based on algorithms rose to the all-time top 20 performers for the first time in 2017 in terms of total dollars earned for investors, after fees, and since inception.

DE Shaw, founded in 1988 with $47bn AUM in 2018 joined the list at number 3. Citadel started in 1990 by Kenneth Griffin, manages $29bn and ranks 5, and Two Sigma started only in 2001 by DE Shaw alumni John Overdeck and David Siegel and has grown from $8 bn AUM in 2011 to $52 bn in 2018. Bridgewater started in 1975 with over $150bn AUM continues to lead due to its Pure Alpha Fund that also incorporates systematic strategies (FT 01/02/2017).

Similarly, on Institutional Investors 2017 Hedge Fund 100 list, five of the top six firms rely largely or completely on computers and trading algorithms to make investment decisions — and all of them have been growing their assets in an otherwise challenging environment. Several quantitatively focused firms climbed several ranks and in some cases grew their assets by double-digit percentages. No. 2-ranked Applied Quantitative Research (AQR) grew its hedge fund assets 48 percent in 2017 to $69.7 billion and managed $187.6 billion firm-wide.

Among all hedge funds, ranked by compounded performance over the last three years, the quant-based funds run by Renaissance Technologies achieved ranks 6 and 24, Two Sigma rank 11, D.E. Shaw no 18 and 32; and Citadel ranks 30 and 37. Beyond the top performers, algorithmic strategies have worked well in the last several years. In the past five years, quant-focused hedge funds gained about 5.1% per year while the average hedge fund rose 4.3% per year in the same period.

Data and machine-learning driven funds attract $1trn AUM

The familiar three revolutions in computing power, data, and machine learning methods have made the adoption of systematic, data-driven strategies not only more compelling and cost-effective but a key source of competitive advantage.

As a result, algorithmic approaches are not only finding wider application in the hedge fund industry that pioneered these strategies but across a broader range of asset managers and even passively managed vehicles like ETFs. In particular, predictive analytics using machine learning and algorithmic automation play an increasingly prominent role in all steps of the investment process across asset classes from idea generation and research to strategy formulation and portfolio construction, trade execution, and risk management.

Estimates of industry size vary because there is no objective definition of a quantitative or algorithmic fund, and many traditional hedge funds or even mutual funds and ETFs are introducing computer-driven strategies or integrating them into a discretionary environment in a **human-plus-machine** approach.

Morgan Stanley estimated in 2017 that algorithmic strategies have grown at 15 percent per year over the past six years and control about $1.5tn between hedge funds, mutual funds, and smart beta ETFs. Other reports suggest the quantitative hedge fund industry was about to exceed $1tn of assets under management (AUM), nearly doubling its size since 2010 amid outflows from traditional hedge funds. (FT 08/01/2018). In contrast, total hedge fund industry capital hit $3.21 trillion according to the latest global Hedge Fund Research report.

The market research firm Preqin estimates that almost 1,500 hedge funds make a majority of their trades with help from computer models. Quantitative hedge funds are now responsible for 27% of all U.S. stock trades by investors, up from 14% in 2013. But many use data scientists—or **quants**, that in turn use machines to build large statistical models (WSJ).

In recent years, however, funds have moved towards true machine learning, where artificially intelligent systems can analyze large amounts of data at speed and improve themselves through such analysis. Recent examples include Rebellion Research, Sentient, and Aidyia that rely on evolutionary algorithms and deep learning to devise fully automatic AI-driven investment platforms.

From the core hedge fund industry, the adoption of algorithmic strategies has spread to mutual funds and even passively managed exchange-traded funds in the form of smart beta funds, and to discretionary funds in the form of quantamental approaches.

The rise of quantamental funds

Two distinct approaches have evolved in active investment management: systematic (or quant) and discretionary investing. Systematic approaches rely on algorithms for a repeatable and data-driven approach to identify investment opportunities across many securities; in contrast, a discretionary approach involves an in-depth analysis of a smaller number of securities. These two approaches are becoming more similar as fundamental managers take more data-science driven approaches.

Even fundamental traders now arm themselves with quantitative techniques, accounting for $55 billion of systematic assets, according to Barclays. Agnostic to specific companies, quantitative funds trade patterns and dynamics across a wide swath of securities. Quants now account for about 17 percent of total hedge fund assets, data compiled by Barclays show.

Point72 Asset Management, with $12 billion in assets, has been shifting about half of its portfolio managers to a **man-plus-machine** approach. Point72 is also investing tens of millions of dollars into a group that analyzes large amounts of alternative data (see below) and passes the results on to traders (Bloomberg 25/08/2018).

Investments in strategic capabilities

Rising investments in related capabilities - technology, data and, most importantly, skilled humans - highlight how significant algorithmic trading using machine learning has become for competitive advantage, especially in light of the rising popularity of passive, indexed investment vehicles like Exchange Traded Funds since the 2008 financial crisis.

Morgan Stanley noted that only 23 percent of its quant clients say they are not considering using or not already using machine learning, down from 44 per cent in 2016.

Guggenheim Partners LLC built what it calls a **supercomputing cluster** for $1 million at the Lawrence Berkeley National Laboratory in California to help crunch numbers for

Guggenheim's quant investment funds. Electricity for the computers costs another $1 million a year.

AQR is a quantitative investment group that relies on academic research to identify and systematically trade factors that have over time proven to beat the broader market. The firm used to eschew the purely computer-powered strategies of quant peers such as Renaissance Technologies or DE Shaw. More recently, however, AQR has begun to seek profitable patterns in markets using machine learning to parse through novel data sets such as satellite pictures of shadows cast by oil wells and tankers.

The leading firm BlackRock with over $5tn AUM also bets on algorithms to beat discretionary fund managers by heavily investing in SAE, a systematic trading firm it acquired during the financial crisis.

Franklin Templeton bought Random Forest Capital, a debt-focused, data-led investment company for an undisclosed amount, hoping that its technology can support the wider asset manager.

Machine learning and alternative data

Hedge funds have long looked for alpha through informational advantage and the ability to uncover new uncorrelated signals. Historically, this included, e.g., proprietary surveys of shoppers or voters ahead of elections or referendums. Occasionally, the use of company insiders, doctors, and expert networks to expand knowledge of industry trends or companies crossed legal lines: a series of prosecutions of traders, portfolio managers and analysts for using insider information after 2010 has shaken the industry.

In contrast, the informational advantage from exploiting conventional and alternative data sources using machine learning is not related to expert and industry networks or access to corporate management, but rather the ability to collect large quantities of data and analyze them in real time.

Three trends have revolutionized the use of data in algorithmic trading strategies and may further shift the investment industry trends from discretionary to quantitative investment styles:

- The exponential increase in the amount of data available
- The increase in computing power and data storage capacity, at a reduced cost
- The advances in Machine Learning methods to analyze complex datasets

Conventional data includes economic statistics, trading data, or corporate reports. **Alternative data** is much broader and includes sources like satellite images, credit card sales, sentiment analysis, mobile geo-location data and website scraping, as well as the conversion of data generated in the ordinary course of business into valuable intelligence.

For instance, sales data from an insurance company on policies on new cars proxies not only new car sales but can be broken down further into brands or geographies. Many vendors scrape websites for valuable data, ranging from app downloads and user reviews to airlines and hotel bookings. Social media sites can also be scraped for hints on consumer views and trends.

Typically, the data sets are large and require storage, access, and analysis using **scalable data solutions** like Hadoop and Spark: there are more than 1bn websites with more than 10tn individual web pages, with 500 exabytes (or 500bn gigabytes) of data, according to Deutsche Bank. And more than 100m websites are added to the internet every year.

Insights into a company's prospect **in real-time**, long before their results are released, can be gleaned from a decline in job listings on its website, the internal rating of its chief executive by employees on the recruitment site Glassdoor, or a dip in the average price of clothes on its website. This could be combined with satellite images of car parks and geolocation data from mobile phones that indicate how many people are visiting stores.

The most valuable data directly reveals **consumer expenditures**, with credit card companies as a primary source. This data only offers a partial view of sales trends, but can offer vital insights when combined with other data. Point72, for instance, analyzes 80m credit card transactions every day. We will explore the various sources, their use cases, and how to evaluate them in detail in the next chapter.

Investment groups have more than doubled their **spending on alternative sets and data scientists** in the past two years, as the asset management industry has tried to reinvigorate its fading fortunes. There are over 200 alternative data providers listed on alternativedata.org (sponsored by provider Yipit).

Asset managers last year spent a total of $373m on data sets and hiring new employees to parse them, up 60% on 2016, and will probably spend a total of $616m this year, according to a survey of investors by alternativedata.org. It forecasts that overall expenditures will climb to over $1bn by 2020. Some estimates are even higher: Optimus, a consultancy, estimates that investors are spending about $5bn per year on alternative data, and expects the industry to grow 30 percent per year over the coming years.

Exclusivity arrangements are a key feature of data source contracts in order to maintain the informational advantage. At the same time, regulators have begun to start looking at the currently largely unregulated data provider industry as privacy concerns are mounting.

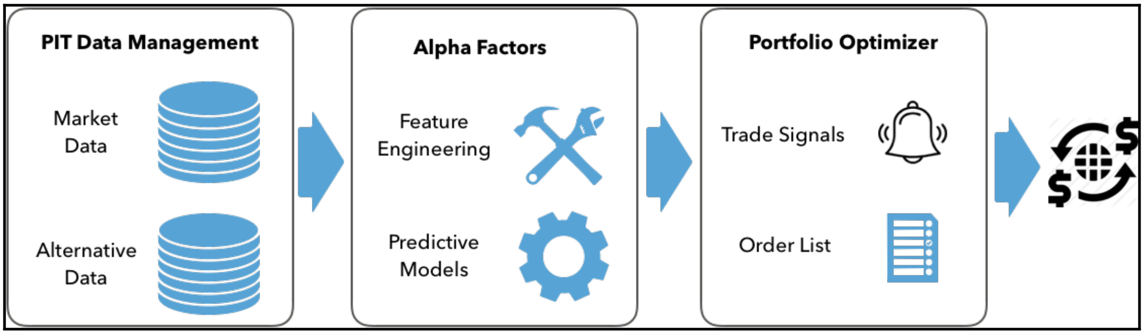
Crowdsourcing of trading algorithms

Relatively recently, several algorithmic trading firms have begun to offer investment platforms that provide access to data and a programming environment to crowd-source risk factors that become part of an investment strategy, or entire trading algorithms. Key examples include WorldQuant, Quantopian, and, most recently, Alpha Trading Labs.

WorldQuant manages more than $5 billion for Millennium Management with $34.6bn AUM since 2007 and announced in 2018 that it will launch its first public fund. It employs hundreds of scientists and many more part-time workers around the world in its Alpha Factory that organizes the investment process as a quantitative assembly line. This factory claims to have produced four million successfully tested alpha factors for inclusion in more complex trading strategies and is aiming for 100 million. Each alpha factor is an algorithm that seeks to predict a future asset price change. Other teams then combine alpha factors into strategies and strategies into portfolios, allocate funds between portfolios and manage risk while avoiding that strategies cannibalize each other.

Designing a trading strategy

An algorithmic trading strategy is based on alpha factors that transform one or several data sources into signals that predict future asset returns and trigger buy or sell orders. Execution of the strategy takes place in the context of the portfolio management process that optimes portfolio weights to exploit interactions among positions in line with the risk-return profile established by the overall investment objectives.



Let's take a brief look at these steps that we will discuss in depth in the following chapters.

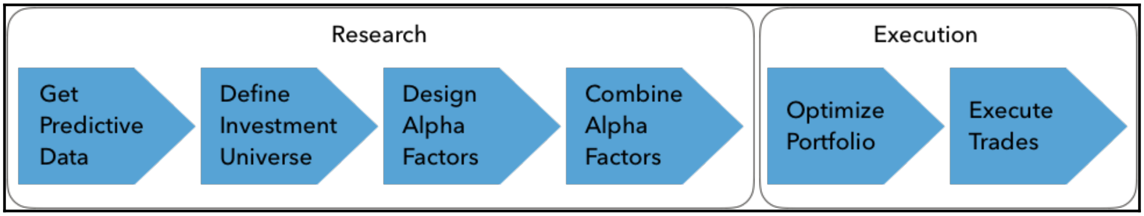
Sourcing and managing data

The explosion of data in terms of volume, variety, and velocity is one of the reasons machine learning to leverage machine learning for algorithmic trading. Dealing with the proliferating supply of data requires several steps:

- A key first step is to identify and evaluate market, fundamental, and alternative data sources containing alpha signals that do not decay too quickly.
- Sourcing and managing data for fast yet flexible access require the deployment of scalable data infrastructure, e.g. Hadoop and analytical tools, e.g. Spark.
- To identify trading signals, data needs to be analyzed in the context of historical trade ticks or time bars that aggregate time periods. Hence, data needs to be adjusted to the desired frequency on a point-in-time (PIT) basis so that it reflects only information available and known at the given time to avoid look-ahead bias.

Alpha factor research and evaluation

Alpha factors are designed to extract signals from the available data that predict assets returns for the given investment universe over the trading horizon. A factor takes on a single value for each asset when evaluated, but may combine one or several input variables.



The research phase of the algorithmic trading strategy workflow includes the design, evaluation, and combination of alpha factors. Machine learning plays a large role in this process because the complexity of factors has increased as investors react to both the signal decay of simpler factors and the much richer data available today.

Identifying and testing alpha factors

The development of predictive alpha factors resembles the building and testing of machine learning models. It requires the exploration of relationships between input data and the target returns, creative feature engineering, and the testing and fine-tuning of data transformations to optimize the predictive power of the inputs.

The data transformations range from simple non-parametric rankings to complex ensemble models or deep neural nets, depending on the amount of information contained in the input data and the complexity of the relationship between the input data and the target. Many of the simpler factors have emerged from academic research and have been increasingly widely used in the industry over the last several decades. We will introduce key traditional factors below and focus on more complex model-based factors throughout this book.

To minimize the risks of false discoveries due to data mining and because finance has been subject to decades of very active research that has been awarded several Nobel prizes, investors prefer to rely on factors that align with theories about financial markets and investor behavior. Laying out these theories is beyond the scope of this book, but the references will highlight avenues to dive deeper into this important aspect of algorithmic trading strategies.

To validate the signal content of an alpha factor candidate, it is necessary to obtain a robust estimate of its predictive power in environments that are representative of the market regime during which the factor would be used in a strategy. There are numerous methodological and practical mistakes to avoid for a reliable estimate, such as data that induces survivorship, look-ahead or other biases by not reflecting realistic point-in-time information, or the failure to correct for a bias due to multiple tests on the same data.

Combining factors into a trading strategy

Signals derived from alpha factors are often individually weak but sufficiently powerful when combined with other factors or data sources, e.g. to modulate the signal as a function of the market or economic context.

The aggregation of individual factor signals into an ensemble of factors to emit a trading signal is another key area to leverage the ability of machine learning model to process, weight and exploit information to maximize predictive accuracy.

Machine learning training and testing

The testing and tuning of machine learning models typically rely on cross-validation to estimate the generalization error rate from historical data.

The time-series nature of financial data requires modifications to the standard approach to avoid look-ahead bias or otherwise contaminate the data used for training, validation, and testing. In addition, the limited availability of historical data has given rise to alternative approaches that use synthetical data.

We will demonstrate various methods to test machine learning models using market, fundamental and alternative that obtain sound estimates of out-of-sample errors.

Portfolio optimization and risk management

The entry and exit signals emitted by alpha factors lead to buy or sell orders, and their execution results in portfolio holdings. The portfolio positions typically have different risk profiles that interact to create a specific portfolio risk profile.

Portfolio management involves the optimization of position weights to achieve the desired portfolio risk and return profile that aligns with the overall investment objectives. This process is highly dynamic to incorporate continuously evolving market data.

The execution of trades during this process requires balancing the trader's dilemma: fast execution tends to drive up costs due to market impact, whereas slow execution may create implementation shortfall when the realized price deviates from the price that prevailed when the decision was taken.

Risk management occurs throughout the portfolio management process to adjust holdings or assume hedges depending on observed or predicted changes in the market environment that impact the portfolio risk profile.

Strategy backtesting

The incorporation of an investment idea into an algorithmic strategy requires extensive research and testing with a scientific approach that attempts to reject the idea based on its performance in alternative out-of-sample market scenarios. This may include testing the idea on simulated data to capture scenarios deemed possible but not reflected in historic data.

A strategy backtesting engine needs to simulate the execution of a strategy in a realistic way to achieve unbiased performance and risk estimates. In addition to the potential biases introduced by the data or a flawed use of statistics, the backtest engine needs to accurately represent the practical aspects of trade signal evaluation, order placement, and execution in line with market conditions.

Algorithmic strategies and signals

There are several ways to categorize trading strategies that rely on the automatic execution of trading rules by means of algorithms. Quantitative strategies have evolved historically and become more sophisticated in three waves.

In the 1980s and 1990s, simple signals dominated that often emerged from academic research and were based on a single or very few inputs derived from market and fundamental data. These signals are by now largely commoditized and available as ETF like basic mean-reversion strategies or simple risk factor implementations.

in the 2000s, factor-based investing proliferated and large funds used algorithms to identify stocks exposed to risk factors like value or momentum that made them likely to under- or outperform. These funds would seek arbitrage opportunities using large amounts of leverage exploit small differences in performance. Redemptions during the early days of the financial crisis triggered the quant quake of August 2007 that cascaded through the factor-based fund industry. These strategies are now also available as long-only smart-beta funds that tilt portfolios according to a given set of risk factors.

The third era is driven by investments in machine learning capabilities and alternative data to generate profitable signals that can be used for repeatable trading strategies. Factor decay is a major challenge: as academics continue to find new anomalies, the excess returns have been shown to drop by a quarter from discovery to publication, and by over 50% after publication due to competition and crowding

Key algorithmic strategies

Strategies can be grouped into short-term trades that aim to benefit from small price movements, behavioral strategies that aim to anticipate larger price moves based on expected the behavior of other market participants, programs that aim to optimize trade execution, and a large group of trading based on predicted pricing.

The HFT funds introduced above most prominently rely on very short holding periods to benefit from small price movements. Expectations for these small price movements can be based on, e.g., bid-ask arbitrage or statistical arbitrage where prices are expected to revert to their equilibrium price. Behavioral algorithms usually operate in environments with lower liquidity and aim to anticipate moves by a larger player that is like to significantly impact the price. The expectation of the price impact is based on sniffing algorithms that generate insights into other market participants strategies, or market patterns like forced trades by ETFs.

Trade execution programs aim to limit the market impact of trades and range from the simple slicing of trades to match time-weighted average pricing (TWAP) or volume-weighted average pricing (VWAP). Simple algorithms leverage historical patterns, whereas more sophisticated algorithms take into account transaction costs, implementation shortfall or predicted price movements. These algorithms can operate at the security or portfolio level, e.g. to implement multi-leg derivative or cross-asset trades.

Trading on predicted price movements or market conditions is the main rationale for using machine learning for algorithmic trading. The portfolio management process includes numerous forecasts to arrive at capital market expectations, their impact on security prices, and the correlation among securities. The predictions can then be used to capture specific risk factors like value or volatility or implement technical approaches like trend following or mean revision.

How machine learning fits into the process

To be completed once the ML chapters are written.

Machine learning adds value by extracting signals from a wide range of market, fundamental, and alternative data and can be applied at all steps of the algorithmic trading strategy process.

Trade idea generation

- Machine learning plays a key role in the analysis and mining of data, and the generation of features, risk factors or alphas.
- Key applications include, e.g., unsupervised learning algorithms like dimensionality reduction and clustering.
- With respect to natural language data, topic modeling or word vector representation learning are important methods to extract features from unstructured data as input for supervised learning models.
- Numerous supervised learning algorithms can be used to learn functional relationships between input and output data that generalize out of sample. These include
    - linear models for cross-sectional data that we will use in chapter 7 to identify risk factors.

- Linear models for time series are important to identify significant patterns for a given time series or among multiple series that repeat over time and that we will cover in chapter 8.
- Non-linear time-series using recurrent neural networks have achieved superior performance in several contexts and will be the subject of chapter 12.

Trade idea testing

- Backtesting is a critical step to select successful algorithmic trading strategies. Cross-validation using synthetic data is a key machine learning technique to generate reliable out-of-sample results when combined with appropriate methods to correct for multiple testing.

Asset allocation:

- machine learning has been used to allocate portfolios based on decision tree models that compute a hierarchical risk parity. As a result, risk characteristics are driven by patterns in asset prices rather than by asset classes and achieve superior risk-return characteristics (see the chapter on Ensemble Learning)

Online learning

- Reinforcement learning

Focus and organization of the book

This book does not provide investment advice. It illustrates how to develop and test algorithmic trading strategies that use machine learning, not which specific algorithms will be successful going forward.

We will be using historical data that is freely available.

Who should read this book

Prior Knowledge: some finance, some machine learning, fluent in python

How this book is organized

This book provides a comprehensive introduction to how machine learning can add value to the design and execution of algorithmic trading strategies. To this end, it is organized in four parts. (To be completed as produced)

Part I: Industry Trends, Data Sources, and Strategy Design

The first part provides a framework for the development of algorithmic trading strategies, introduces the rapidly evolving data sources that power machine learning models at the heart of the strategies discussed in this book and highlight where machine learning can be applied throughout the algorithmic trading process.

The next two chapters introduce market and fundamental data, as well as the burgeoning supply of alternative data, how to evaluate data sources and supplies, and how to work with these data sources using python. The following chapter focuses on the design and evaluation of alpha factors using the `zipline` and `alphalens` libraries and introduces the Information Coefficient to evaluate an alpha factor's predictive performance.

The last chapter of the first part introduces strategy backtesting and portfolio management. We will simulate trading strategies using historical data with `zipline` both offline and on the Quantopian platform, introduce key performance and risk metrics, and how to compute them using the `pyfolio` library. We will use this backtesting framework throughout the book as we add machine learning models to our trading strategies. We will also address numerous practical and methodological challenges of backtests, and how to improve their reliability. Furthermore, we will introduce various methods to optimize portfolio holdings from a risk perspective, and demonstrate how unsupervised machine learning can be used to learn hierarchical risk characteristics from portfolio candidates to develop a risk parity allocation that has been shown to outperform out-of-sample.

Part 2: Machine Learning Fundamentals

Chapter 5: Linear Models for Regression & Classification
Chapter 6: Time Series Models: ARIMA & VAR Models

Chapter 7: Unsupervised Learning: Filters, Clustering and Dimensionality Reduction
Chapter 8: Decision Trees & Random Forests to predict Asset Prices
Chapter 9: Tuning & Interpreting Gradient Boosting Machines with LightGBM
Chapter 10: Bayesian Machine Learning for Algorithmic Trading using PyMC3

Part 3: Natural Language Processing

Chapter 11: From text to numbers: the document-text matrix and text classification
Chapter 12: Making sense of very large language data: Topic Modeling using gensim
Chapter 13: State-of-the-art text features: word2vec models

Part 4: Deep and Reinforcement Learning

Chapter 14: Deep Neural Networks: Design, Training & Tuning
Chapter 15: Recurrent NN: LSTM for State-of-the-Art Time Series Models
Chapter 16: Convolutional Neural Networks: Image Recognition for Satellite Data
Chapter 17: Transfer Learning: Leapfrogging based on Pre-Trained Models using keras
Chapter 18: Reinforcement Learning: Key Principles, Q-Learning and the Dyna Architecture

Technical requirements

There are Jupyter Notebooks and Labs to illustrate many of the concepts and models in more detail. Jupyter notebooks are a great tool for creating reproducible computational narratives, and they enables users to create and share documents that combine live code with narrative text, mathematical equations, visualizations, interactive controls, and other rich output. It also provides building blocks for interactive computing with data: a file browser, terminals, and a text editor.

Python libraries

- we will use python 3.6
- library installation instructions referenced in GitHub repo

Summary

In this chapter, we have introduced algorithmic trading strategies and how machine learning has become a key ingredient for the design and combination of alpha factors that in turn are the key drivers of portfolio performance. We have covered various industry trends around algorithmic trading strategies, the emergence of alternative data and the use of machine learning to exploit these new sources of informational advantage

Furthermore, we have introduced the algorithmic trading strategy design process, important types of alpha factors, and how we will use machine learning to design and execute our strategies.

In the next two chapters, we will take a closer look at the oil that fuels any algorithmic trading strategy using machine learning: market, fundamental, and alternative data sources.

2
Market and Fundamental Data

Data has always been an essential driver of trading, and traders have long made efforts to gain an advantage by access to superior information. These efforts date back at least to the rumors that the House Rothschild benefited handsomely from bond purchases upon advance news about the British victory at Waterloo carried by pigeons across the Canal.

Today, investments in faster data access take the shape of the Go West consortium of leading HFT firms that connects the **Chicago Mercantile Exchange** (**CME**) with Tokyo. The round-trip latency between the CME and the BATS exchange in New York has dropped to close to the theoretical limit of 8 ms as traders compete to exploit arbitrage opportunities.

Traditionally, investment strategies mostly relied on publicly available data, with limited efforts to create or acquire private data sets. In the case of equities, fundamental strategies used financial models built on reported financials, possibly combined with industry or macro data. Strategies motivated by technical analysis extract signals from market data like prices and volumes.

Machine learning algorithms are able to exploit market and fundamental data more efficiently, in particular when combined with alternative data, the topic of the next chapter. We will address several techniques that focus on market and fundamental data in later chapters, such as classic and modern time series techniques including recurrent neural networks.

This chapter introduces market and fundamental data sources and the environment in which they are created. Because backtests of a trading strategy simulate the market environment, basic familiarity with various types of orders and the trading infrastructure is important. We also illustrate how to use python to access and work with trading and financial statement data.

In particular, this chapter will cover the following topics:

- How market microstructure shapes market data
- How to reconstruct the order book from tick data using NASDAQ ITCH
- How to summarize tick data using various types of bars

- How to work with XBRL-encoded electronic filings
- How to parse and combine market and fundamental data to create a P/E series
- How to access various market and fundamental data source using python

Market data

Market data results from the placement and processing of buy and sell orders in the course of the trading of financial instruments on the many marketplaces. The data reflects the institutional environment of trading venues, including the rules and regulations that govern orders, trade execution and price formation. Algorithmic traders use machine learning algorithms to analyze the flow of buy and sell orders and the resulting volume and price statistics to extract trade signals or features that capture insights into, for example, demand-supply dynamics or the behavior of certain market participants.

We will first provide some context because institutional features are important when simulating a trading strategy during a backtest. Then, we will take a look at how tick data can be reconstructed from the order book source. Next, we will highlight several methods that regularize tick data and aim to maximize the information content. Finally, we will illustrate how to access various market data provider interfaces and highlight several providers.

Market microstructure

Market microstructure is the branch of financial economics that investigates the trading process and the organization of related markets. The institutional details are quite complex and diverse across asset classes and their derivatives, trading venues, and geographies. We will only give a brief overview of key concepts before we dive into the data generated by trading. The references contain several sources that treat this important subject in great detail, see e.g. Harris (2003).

Market places

Trading in financial instruments occurs in organized, mostly electronic exchanges and over the counter. An exchange is a central marketplace where buyers and sellers meet, and buyers compete with each other for the highest bid while sellers compete for the lowest offer.

There are many exchanges and alternative trading venues across the US and abroad. The below table lists some of the larger global exchanges and the trading volumes for the 12 months concluded 03/2018 in various asset classes, including derivatives. Typically, a minority of financial instruments accounts for most trading:

Exchange	Stocks				
	Market Cap (USD mn)	# Listed Companies	Volume / Day (USD mn)	# Shares / Day ('000)	# Options / Day ('000)
NYSE	23,138,626	2,294	78,410	6,122	1,546
Nasdaq - US	10,375,718	2,968	65,026	7,131	2,609
Japan Exchange Group Inc.	6,287,739	3,618	28,397	3,361	1
Shanghai Stock Exchange	5,022,691	1,421	34,736	9,801	
Euronext	4,649,073	1,240	9,410	836	304
Hong Kong Exchanges and Clearing	4,443,082	2,186	12,031	1,174	516
LSE Group	3,986,413	2,622	10,398	1,011	
Shenzhen Stock Exchange	3,547,312	2,110	40,244	14,443	
Deutsche Boerse AG	2,339,092	506	7,825	475	
BSE India Limited	2,298,179	5,439	602	1,105	
National Stock Exchange of India Limited	2,273,286	1,952	5,092	10,355	
BATS Global Markets - US					1,243
Chicago Board Options Exchange					1,811
International Securities Exchange					1,204

Exchanges may rely on bilateral trading or order-driven systems where buy and sell orders are matched according to certain rules. Price formation may occur through auctions, like in the NYSE, where the highest bid and lowest offer are matched, or through dealers who buy from sellers and sell to buyers.

Many exchanges use intermediaries that provide liquidity, ie, the ability to trade, by making markets in certain securities. The New York Stock Exchange (NYSE), e.g., usually has a single designated market maker who ensures orderly trading for each security, while the NASDAQ has several. Intermediaries can act as dealers that trade as principals on their own behalf, or brokers that trade as agents on behalf of others.

Exchanges used to be member-owned but have often moved to corporate ownership as market reforms have increased competition. The NYSE dates back to 1792, whereas the National Association of Securities Dealers Automated Quotations (NASDAQ) started 1971 as the world's first electronic stock market and took over most stock trades that had been executed OTC. In U.S. equity markets alone, trading is fragmented across 13 exchanges and over 40 alternative trading venues, each reporting trades to the consolidated tape, but at different latencies (de Prado 2018).

Types of orders

Traders can place various types of buy or sell orders. Some orders guarantee immediate execution, while others may state a price threshold or other conditions that trigger execution. Orders are typically valid for the same trading day unless specified otherwise.

A market order guarantees immediate execution of the order upon arrival to the trading venue, at the price that prevails at that moment. In contrast, a limit order only executes if the market price is higher (lower) than the limit for a sell (buy) limit order. A stop order, in turn, only becomes active when the market price rises above (falls below) a specified price for a buy (sell) stop order. A buy stop order can be used to limit losses of short sales. Stop orders may also have limits.

Numerous other conditions can be attached to orders: All or None Orders prevent partial execution and are only filled if a specified number of shares is available, and can be valid for the day or longer. They require special handling and are not visible to market participants. Fill or Kill Orders also prevent partial execution but cancel if not executed immediately. Immediate or Cancel Orders immediately buy or sell the number of shares that are available and cancels the remainder. Not Held Orders allow the broker to decide on time and price of execution. Finally, Market on Open / Close Orders executes on or near the opening or closing of the market. Partial executions are allowed.

Working with trading data

The primary source of market data is the order book that is continuously updated in real time throughout the day to reflect all trading activity. Exchanges typically offer this data as a real-time service and may provide some historical data for free.

The trading activity is reflected in numerous messages about trade orders sent by market participants. These messages typically conform to the electronic Financial Information eXchange (FIX) communications protocol for real-time exchange of securities transactions and market data or a native exchange protocol.

The FIX protocol

Just like SWIFT is the message protocol for back office (e.g. for trade settlement) messaging, the FIX Protocol is the de facto messaging standard for communication prior to and during trade execution between exchanges, banks, brokers, clearing firms, and other market participants. Fidelity Investments and Salomon Brothers introduced FIX in 1992 to facilitate electronic communication between broker-dealers and institutional clients who by then exchanged information over the phone.

It became popular in global equity markets before expanding into foreign exchange, fixed income and derivatives markets and further into post-trade to support straight-through processing. Exchanges provide access to FIX messages as a real-time data feed that is parsed by algorithmic traders to track market activity and, e.g., identify the footprint of market participants and anticipate their next move.

The sequence of messages allows for the reconstruction of the order book. The scale of transactions across numerous exchanges creates a large amount (~10TB) of unstructured data that is challenging to process and, hence, can be a source of competitive advantage.

The FIX protocol, currently at version 5.0, is a free and open standard with a large community of affiliated industry professionals. It is self-describing like the more recent XML, and a FIX session is supported by the underlying Transmission Control Protocol (TCP) layer. The community constantly adds `new functionality`.

The protocol supports pipe-separated key-value pair as well as a tag-based FIXML syntax. A sample message that requests server login would look as follows:

```
8=FIX.5.0|9=127|35=A|59=theBroker.123456|56=CSERVER|34=1|32=20180117-
08:03:04|57=TRADE|50=any_string|98=2|108=34|141=Y|553=12345|554=passw0rd!|1
0=131|
```

There are a few open-source FIX implementations in python that can be used to formulate and parse FIX messages. Interactive Brokers offers a FIX-based computer-to-computer interface (CTCI) for automated trading (see resources section for this chapter in the GitHub repo).

Working with NASDAQ TotalView-ITCH order book data

While the FIX Protocol has a very large market share, exchanges also offer **native protocols**. The NASDAQ offers a **TotalView ITCH** direct data-feed protocol that allows subscribers to track individual orders for equity instruments from placement to execution or cancellation.

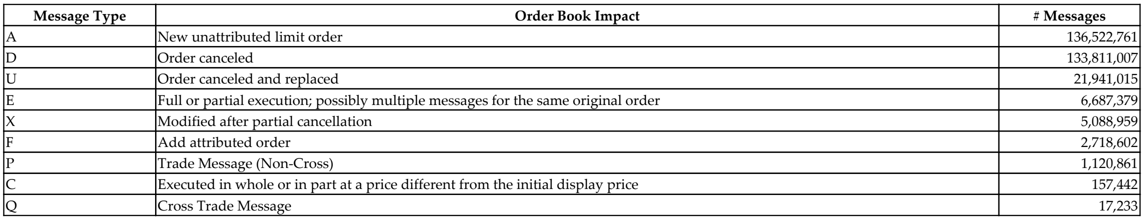
As a result, it allows for the reconstruction of the order book that keeps track of the list of active buy and sell orders for a specific security or financial instrument. The order book reveals the **market depth** throughout the day by listing the number of shares being bid or offered at each price point. It may also identify the market participant responsible for a specific buy and sell orders unless it is placed anonymously. Market depth is an important indicator of liquidity and the potential price impact of sizable market orders.

In addition to matching market and limit orders, the NASDAQ also operates auctions or crosses that execute a large number of **trades at market opening and closing**. Crosses are becoming more important as passive investing continues to grow and traders look for opportunities to execute larger blocks of stock. TotalView also disseminates the Net Order Imbalance Indicator (NOII) for the Nasdaq Opening and Closing Crosses and Nasdaq IPO/Halt Cross.

Parsing binary ITCH messages

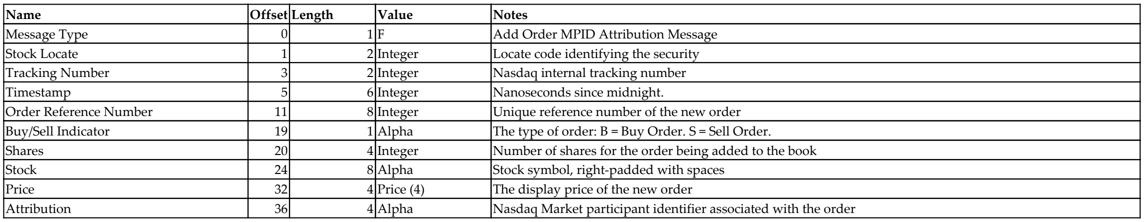
The `ITCH v5.0 specification` declares over 20 message types related to system events, stock characteristics, the placement and modification of limit orders, and trade execution. It also contains information about the net order imbalance prior to the open and closing cross.

The NASDAQ provides access to sample daily `historical binary files` for several months. The GitHub repository for the book contains code to parse the messages to reconstruct both the executed trades and the order book for any given tick. The below table contains selected messages used for this purpose and the number of messages for each type on March 29, 2018.



Message Type	Order Book Impact	# Messages
A	New unattributed limit order	136,522,761
D	Order canceled	133,811,007
U	Order canceled and replaced	21,941,015
E	Full or partial execution; possibly multiple messages for the same original order	6,687,379
X	Modified after partial cancellation	5,088,959
F	Add attributed order	2,718,602
P	Trade Message (Non-Cross)	1,120,861
C	Executed in whole or in part at a price different from the initial display price	157,442
Q	Cross Trade Message	17,233

For each message, the specification lays out the components and their respective length and data types:



Name	Offset	Length	Value	Notes
Message Type	0	1	F	Add Order MPID Attribution Message
Stock Locate	1	2	Integer	Locate code identifying the security
Tracking Number	3	2	Integer	Nasdaq internal tracking number
Timestamp	5	6	Integer	Nanoseconds since midnight.
Order Reference Number	11	8	Integer	Unique reference number of the new order
Buy/Sell Indicator	19	1	Alpha	The type of order: B = Buy Order. S = Sell Order.
Shares	20	4	Integer	Number of shares for the order being added to the book
Stock	24	8	Alpha	Stock symbol, right-padded with spaces
Price	32	4	Price (4)	The display price of the new order
Attribution	36	4	Alpha	Nasdaq Market participant identifier associated with the order

Python provides the `struct module` to parse binary data using format strings that identify the message elements by indicating length and type of the various components of the byte string as laid out in the specification. The ITCH parser relies on the message specifications provided as a csv file and assembles `format strings` according to the dictionary `formats`:

```
formats = {
    ('integer', 2): 'H',  # int of length 2 => format string 'H'
    ('integer', 4): 'I',
    ('integer', 6): '6s', # int of length 6 => parse as string, convert
later
```

```
        ('integer', 8): 'Q',
        ('alpha', 1)  : 's',
        ('alpha', 2)  : '2s',
        ('alpha', 4)  : '4s',
        ('alpha', 8)  : '8s',
        ('price_4', 4): 'I',
        ('price_8', 8): 'Q',
    }
```

The parser translates the message specs into format strings and `namedtuples` to capture the messages:

```
    # Get ITCH specs and create formatting (type, length) tuples
    specs = pd.read_csv('message_types.csv')
    specs['formats'] = specs[['value', 'length']].apply(tuple,
    axis=1).map(formats)

    # Formatting for alpha fields
    alpha_fields = specs[specs.value == 'alpha'].set_index('name')
    alpha_msgs = alpha_fields.groupby('message_type')
    alpha_formats = {k: v.to_dict() for k, v in alpha_msgs.formats}
    alpha_length = {k: v.add(5).to_dict() for k, v in alpha_msgs.length}

    # Generate message classes as named tuples and format strings
    message_fields, fstring = {}, {}
    for t, message in specs.groupby('message_type'):
        message_fields[t] = namedtuple(typename=t,
    field_names=message.name.tolist())
        fstring[t] = '>' + ''.join(message.formats.tolist())
```

Fields of type alpha require postprocessing as defined in the function `format_alpha`:

```
    def format_alpha(mtype, data):
        for col in alpha_formats.get(mtype).keys():
            if mtype != 'R' and col == 'stock': # stock name only in summary
    message 'R'
                data = data.drop(col, axis=1)
                continue
            data.loc[:, col] = data.loc[:, col].str.decode("utf-8").str.strip()
            if encoding.get(col):
                data.loc[:, col] = data.loc[:, col].map(encoding.get(col)) #
    int encoding
        return data
```

The binary file for a single day contains over 300 million messages worth over 9GB. To handle memory constraints, the script appends the parsed result iteratively to a file in the fast `HDF5 format` (more on this format below):

```
def store_messages(m):
    with pd.HDFStore('/path/to/hdfstore.h5') as store:
        for mtype, data in m.items():
            data = pd.DataFrame(data)
            data.timestamp = data.timestamp.apply(int.from_bytes,
byteorder='big')
            data.timestamp = pd.to_timedelta(data.timestamp)
            if mtype in alpha_formats.keys():
                data = format_alpha(mtype, data)

            msize = alpha_length.get(mtype)
            if s:
                s = {c: msize.get(c) for c in data.columns}
            store.append(mtype, data,
                         format='t',
                         min_itemsize=msize,
                         data_columns=['stock_locate'])
```

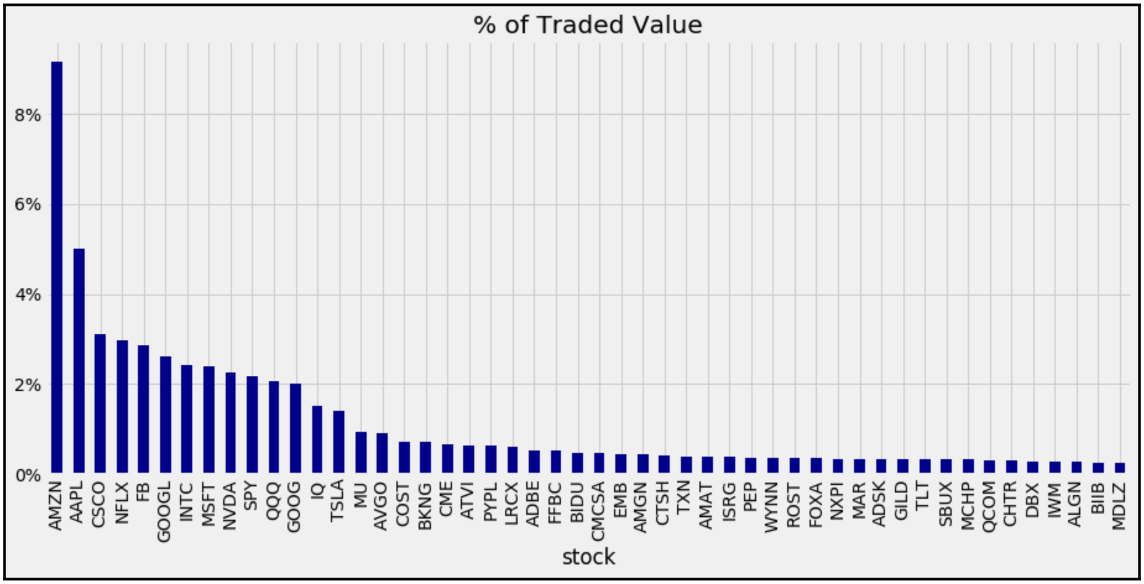
The following code processes the binary file and produces the parsed orders stored by message type:

```
messages = {}
message_count = 0
message_type_counter = Counter() # keep track of messages by type

with (data_path / file_name).open('rb') as data:
    while True:
        message_size = int.from_bytes(data.read(2), byteorder='big',
signed=False)
        message_type = data.read(1).decode('ascii')
        if not messages.get(message_type):
            messages[message_type] = []
        message_type_counter.update([message_type])

        record = data.read(message_size - 1)
        message =
message_fields[message_type]._make(unpack(fstring[message_type], record))
        messages[message_type].append(message)

        if message_type == 'S': # system event
            timestamp = int.from_bytes(message.timestamp, byteorder='big')
            if message.event_code.decode('ascii') == 'C': # last message
per day
                break

        message_count += 1
        if message_count % 2.5e7 == 0:
            timestamp = int.from_bytes(message.timestamp, byteorder='big')
            store_messages(messages)
            messages = {}
```

As expected, a small number of the over 8,500 equity securities traded on this day account for most trades:

```
with pd.HDFStore(hdf_store) as store:
    stocks = store['R'].loc[:, ['stock_locate', 'stock']]
    trades = store['P'].append(store['Q'].rename(columns={'cross_price':
'price'}).merge(stocks)
trades['value'] = trades.shares.mul(trades.price)
trades['value_share'] = trades.value.div(trades.value.sum())
trade_summary =
trades.groupby('stock').value_share.sum().sort_values(ascending=False)
trade_summary.iloc[:50].plot.bar(figsize=(14, 6), color='darkblue',
title='% of Traded Value')
plt.gca().yaxis.set_major_formatter(FuncFormatter(lambda y, _:
'{:.0%}'.format(y)))
```

We get the following plot for the graph:



Reconstructing Trades & the Order Book

The parsed messages let us reconstruct the order flow for the given day. The `'R'` type contains a listing of all stocks traded during a given day, including information about IPOs, trading restrictions, etc. New orders are added, and orders that are executed and canceled are removed from the order book. The proper accounting for messages that reference orders placed on a prior date would require tracking the order book over multiple days, but we are ignoring this aspect here.

The following function illustrates how to collect the orders for a single stock that affect trading (refer to the ITCH specification for details about each message):

```
def get_messages(date, stock=stock):
    with pd.HDFStore(itch_store) as store:
        stock_locate = store.select('R', where='stock =
stock').stock_locate.iloc[0]
        target = 'stock_locate = stock_locate'

        data = {}
        messages = ['A', 'F', 'E', 'C', 'X', 'D', 'U', 'P', 'Q']
        for m in messages:
            data[m] = store.select(m, where=target).drop('stock_locate',
axis=1).assign(type=m)

    order_cols = ['order_reference_number', 'buy_sell_indicator', 'shares',
'price']
    orders = pd.concat([data['A'], data['F']], sort=False,
ignore_index=True).loc[:, order_cols]

    for m in messages[2: -3]:
        data[m] = data[m].merge(orders, how='left')

    data['U'] = data['U'].merge(orders, how='left',
                                right_on='order_reference_number',
                                left_on='original_order_reference_number',
                                suffixes=['', '_replaced'])
    data['Q'].rename(columns={'cross_price': 'price'}, inplace=True)
    data['X']['shares'] = data['X']['cancelled_shares']
    data['X'] = data['X'].dropna(subset=['price'])

    data = pd.concat([data[m] for m in messages], ignore_index=True,
sort=False)
    data['date'] = pd.to_datetime(date)
    data.timestamp = data['date'].add(data.timestamp)
    data = data[data.printable != 0]

    drop_cols = ['tracking_number', 'order_reference_number',
'original_order_reference_number',
                 'cross_type', 'new_order_reference_number', 'attribution',
'match_number',
                 'printable', 'date', 'cancelled_shares']
    return data.drop(drop_cols,
axis=1).sort_values('timestamp').reset_index(drop=True)
```

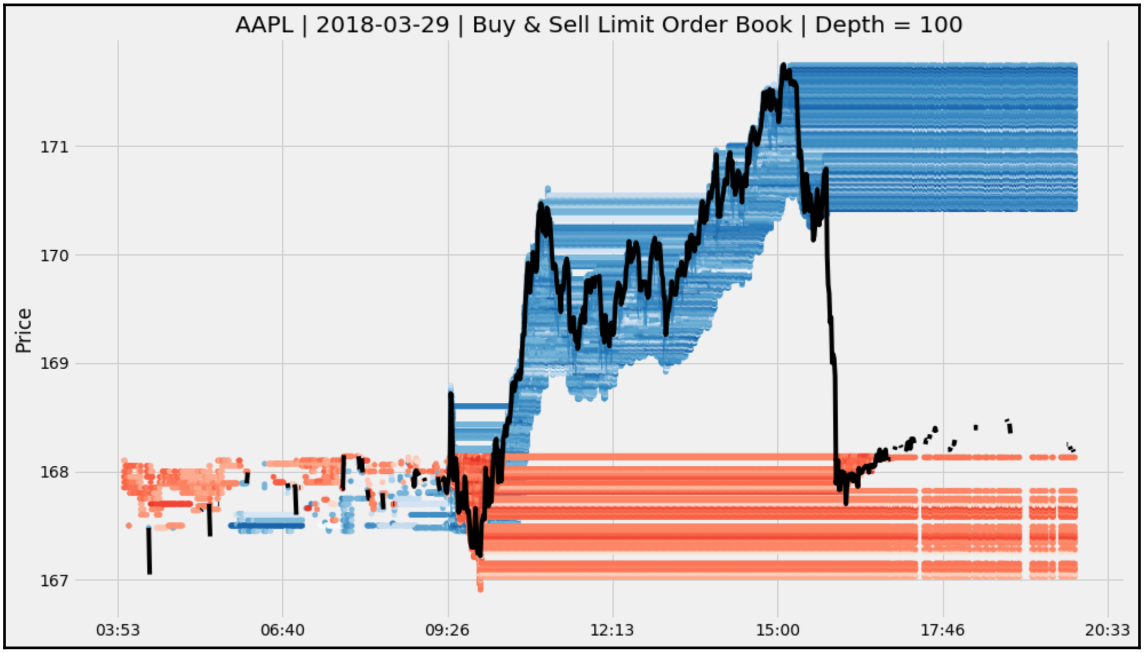
Reconstructing the trades that took place from trade-related message types `C, E, P, Q` is relatively straightforward:

```
    def get_trades(m):
        """Combine C, E, P and Q messages into trading records"""
        trade_dict = {'executed_shares': 'shares', 'execution_price': 'price'}
        cols = ['timestamp', 'executed_shares']
        trades = pd.concat([m.loc[m.type == 'E', cols +
    ['price']].rename(columns=trade_dict),
                            m.loc[m.type == 'C', cols +
    ['execution_price']].rename(columns=trade_dict),
                            m.loc[m.type == 'P', ['timestamp', 'price',
    'shares']],
                            m.loc[m.type == 'Q', ['timestamp', 'price',
    'shares']].assign(cross=1),
                            ], sort=False).dropna(subset=['price']).fillna(0)
        return trades.set_index('timestamp').sort_index().astype(int)
```

The order book keeps track of limit orders, and the various price levels for buy and sell orders constitute the depth of the order book. The following code reconstructs the order book over the course of the trading day for a given level of depth:

```
    order_book = {-1: {}, 1: {}}
    current = {-1: Counter(), 1: Counter()}
    message_counter = Counter()
    nlevels = 100

    def add_orders(orders, buysell, nlevels):
        new_order = []
        items = sorted(orders.copy().items())
        if buysell == 1:
            items = reversed(items)
        for i, (p, s) in enumerate(items, 1):
            new_order.append((p, s))
            if i == nlevels:
                break
        return orders, new_order

    for message in messages.itertuples():
        if np.isnan(message.buy_sell_indicator):
            continue
        message_counter.update(message.type)

        buysell = message.buy_sell_indicator
        price, shares = None, None
        if message.type in ['A', 'F', 'U']:
            price = int(message.price)
            shares = int(message.shares)

            current[buysell].update({price: shares})
            current[buysell], new_order = add_orders(current[buysell], buysell,
    nlevels)
            order_book[buysell][message.timestamp] = new_order

        if message.type in ['E', 'C', 'X', 'D', 'U']:
            if message.type == 'U':
                if not np.isnan(message.shares_replaced):
                    price = int(message.price_replaced)
                    shares = -int(message.shares_replaced)
            else:
                if not np.isnan(message.price):
                    price = int(message.price)
                    shares = -int(message.shares)

            if price is not None:
                current[buysell].update({price: shares})
                if current[buysell][price] <= 0:
                    currentfor message in messages.itertuples():
        i = message[0]
        if i % 1e5 == 0 and i > 0:
            print('{:,.0f}\t{}'.format(i, timedelta(seconds=time() - start)))
            # save_orders(order_book, append=True)
            # order_book = {-1: {}, 1: {}}
            start = time()
        if np.isnan(message.buy_sell_indicator):
            continue
        message_counter.update(message.type)

        buysell = message.buy_sell_indicator
        price, shares = None, None

        if message.type in ['A', 'F', 'U']:
            price = int(message.price)
            shares = int(message.shares)
            current_orders[buysell].update({price: shares})
            current_orders[buysell], new_order = add_orders(current[buysell],
    buysell, nlevels)
            order_book[buysell][message.timestamp] = new_order

        if message.type in ['E', 'C', 'X', 'D', 'U']:
            if message.type == 'U':
                if not np.isnan(message.shares_replaced):
                    price = int(message.price_replaced)
                    shares = -int(message.shares_replaced)
            else:
                if not np.isnan(message.price):
                    price = int(message.price)
                    shares = -int(message.shares)

            if price is not None:
                current_orders[buysell].update({price: shares})
                if current_orders[buysell][price] <= 0:
                    current_orders[buysell].pop(price)
                current_orders[buysell], new_order =
    add_orders(current[buysell], buysell, nlevels)
                order_book[buysell][message.timestamp] =
    new_order_orders[buysell].pop(price)
                current[buysell], new_order = add_orders(current[buysell],
    buysell, nlevels)
                order_book[buysell][message.timestamp] = new_order
```

The number of orders at different price levels, highlighted below using different intensities for buy and sell orders, visualize the depth of liquidity at any given point in time. The dark line tracks the prices for executed trades during market hours as well as prior to and after market hours:



Regularizing Tick Data

The trade data is indexed by nanoseconds and very noisy. The bid-ask bounce, for instance, causes the price to oscillate between the bid and ask prices when trade initiation alternates between buy and sell market orders. To improve the noise-signal ratio and improve the statistical properties, we need to resample and regularize the tick data by aggregating the trading activity.

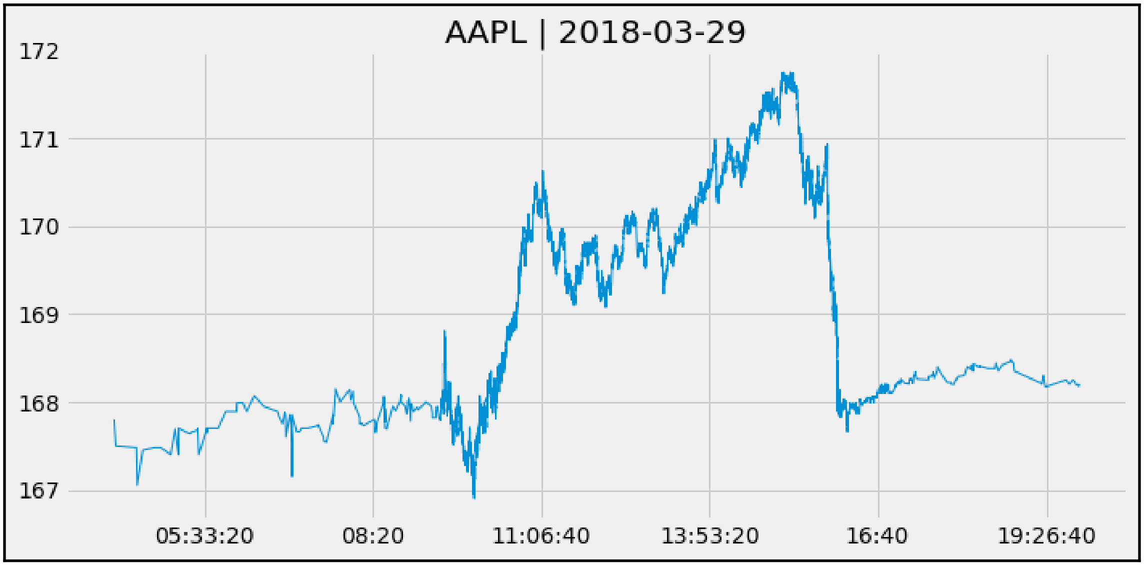
We typically collect the Open (first), Low, High and Closing (last) price for the aggregated period, alongside the volume-weighted average price (VWAP), the number of shares traded, and timestamp associated with the data.

Tick Bars

A plot of the raw tick price and volume data for AAPL looks as follows:

```
stock, date = 'AAPL', '20180329'
title = '{} | {}'.format(stock, pd.to_datetime(date).date()

with pd.HDFStore(itch_store) as store:
    s = store['S'].set_index('event_code') # system events
    s.timestamp = s.timestamp.add(pd.to_datetime(date)).dt.time
    market_open = s.loc['Q', 'timestamp']
    market_close = s.loc['M', 'timestamp']

with pd.HDFStore(stock_store) as store:
    trades = store['{}/trades'.format(stock)].reset_index()
trades = trades[trades.cross == 0] # excluding data from open/close
crossings
trades.price = trades.price.mul(1e-4)

trades.price = trades.price.mul(1e-4) # format price
trades = trades[trades.cross == 0]    # exclude crossing trades
trades = trades.between_time(market_open, market_close) # market hours only

tick_bars = trades.set_index('timestamp')
tick_bars.index = tick_bars.index.time
tick_bars.price.plot(figsize=(10, 5), title=title), lw=1)
```

We get the following plot for the preceding code:



The tick returns are far from normally distributed as evidenced by the low p-value of the `scipy.stats.normaltest`:

```
from scipy.stats import normaltest
normaltest(tick_bars.price.pct_change().dropna())

NormaltestResult(statistic=62408.76562431228, pvalue=0.0)
```

Time Bars

Time bars involve trade aggregation by time period.

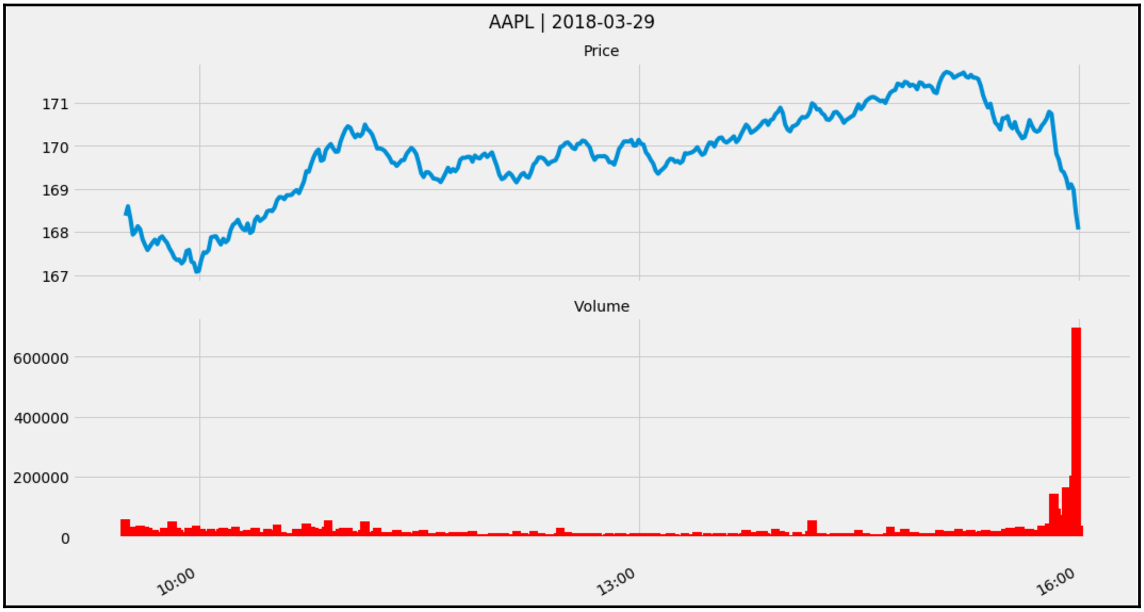
```
def get_bar_stats(agg_trades):
    vwap = agg_trades.apply(lambda x: np.average(x.price,
weights=x.shares)).to_frame('vwap')
    ohlc = agg_trades.price.ohlc()
    vol = agg_trades.shares.sum().to_frame('vol')
    txn = agg_trades.shares.size().to_frame('txn')
    return pd.concat([ohlc, vwap, vol, txn], axis=1)

resampled = trades.resample('1Min')
time_bars = get_bar_stats(resampled)
```

We can display the result as Price-Volume Chart:

```
def price_volume(df, price='vwap', vol='vol', suptitle=title):
    fig, axes = plt.subplots(nrows=2, sharex=True, figsize=(15, 8))
```

```
        axes[0].plot(df.index, df[price])
        axes[1].bar(df.index, df[vol], width=1 / (len(df.index)), color='r')

        xfmt = mpl.dates.DateFormatter('%H:%M')
        axes[1].xaxis.set_major_locator(mpl.dates.HourLocator(interval=3))
        axes[1].xaxis.set_major_formatter(xfmt)
        axes[1].get_xaxis().set_tick_params(which='major', pad=25)
        axes[0].set_title('Price', fontsize=14)
        axes[1].set_title('Volume', fontsize=14)
        fig.autofmt_xdate()
        fig.suptitle(suptitle)
        fig.tight_layout()
        plt.subplots_adjust(top=0.9)

    price_volume(time_bars)
```

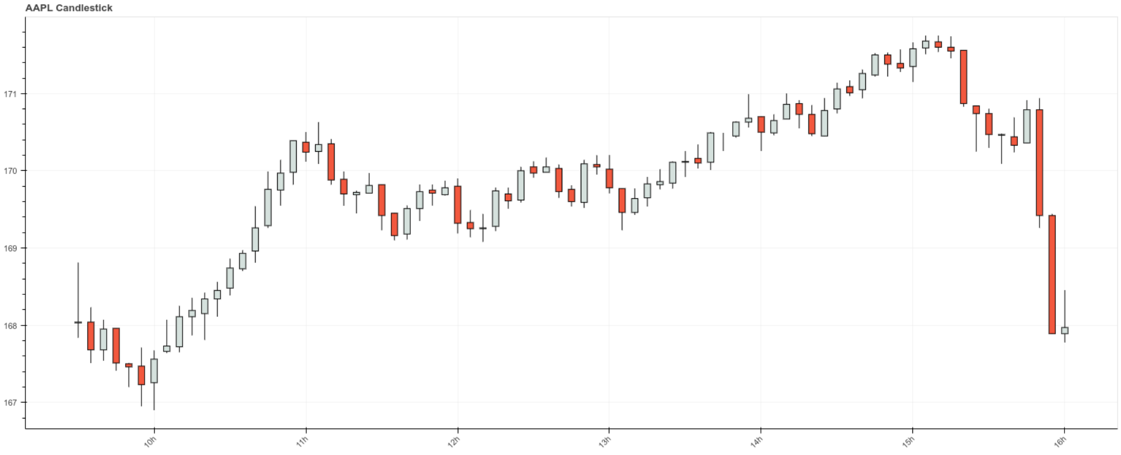
We get the following plot for the preceding code:



Or as candlestick chart using the `bokeh` plotting library:

```
    resampled = trades.resample('5Min') # 5 Min bars for better print
    df = get_bar_stats(resampled)

    increase = df.close > df.open
    decrease = df.open > df.close
    w = 2.5 * 60 * 1000 # 2.5 min in ms
```

```
WIDGETS = "pan, wheel_zoom, box_zoom, reset, save"

p = figure(x_axis_type='datetime', tools=WIDGETS, plot_width=1500, title =
"AAPL Candlestick")
p.xaxis.major_label_orientation = pi/4
p.grid.grid_line_alpha=0.4

p.segment(df.index, df.high, df.index, df.low, color="black")
p.vbar(df.index[increase], w, df.open[increase], df.close[increase],
fill_color="#D5E1DD", line_color="black")
p.vbar(df.index[decrease], w, df.open[decrease], df.close[decrease],
fill_color="#F2583E", line_color="black")
show(p)
```



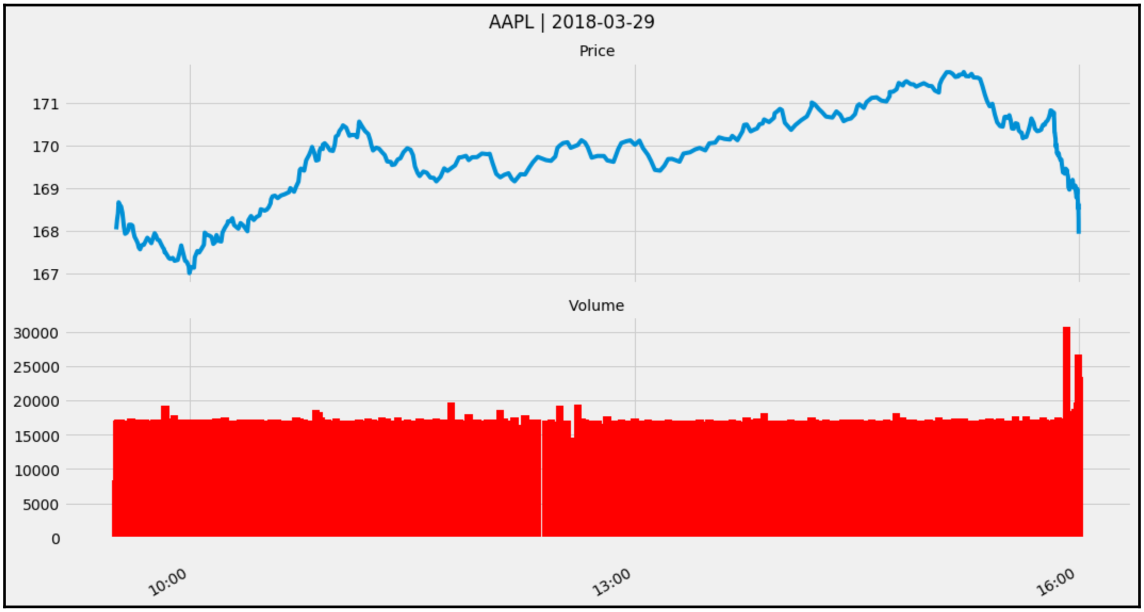
Volume Bars

Time bars smooth some of the noise contained in the raw tick data but may fail to account for the fragmentation of orders. Execution-focused algorithmic trading may aim to match the VWAP over a given period and will divide a single order into multiple trades and place orders according to historical patterns. Time bars would treat the same order differently, even though no new information has arrived in the market.

Volume bars offer an alternative by aggregating trade data according to volume. We can accomplish this as follows:

```
trades_per_min = trades.shares.sum()/(60*7.5) # min per trading day
trades['cumul_vol'] = trades.shares.cumsum()
df = trades.reset_index()
by_vol = df.groupby(df.cumul_vol.div(trades_per_min).round().astype(int))
vol_bars = pd.concat([by_vol.timestamp.last().to_frame('timestamp'),
get_bar_stats(by_vol)], axis=1)
price_volume(vol_bars.set_index('timestamp'))
```

We get the following plot for the preceding code:



Dollar Bars

When comparing trading behavior for different periods with significantly changed asset prices or around stock splits, volume bars do not reflect that the number of shares required to exchange a certain amount of value has changed as well. In this case, the volume bar method should be adjusted to utilize the product of shares and price instead of the number of shares alone.

Remote Data Access via pandas

The `pandas library` enables access to data displayed on websites via the function `read_html` and access to the API endpoints of various data providers through the related `pandas-datareader library`. The download of the content of one or more html tables works as follows, for instance for the constituents of the S&P500 index from `wikipedia`:

```
    sp_url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'
```

```
sp = pd.read_html(sp_url, header=0) # returns a list for each table
sp[0].info()

RangeIndex: 505 entries, 0 to 504
Data columns (total 9 columns):
Ticker symbol            505 non-null object
Security                 505 non-null object
SEC filings              505 non-null object
GICS Sector              505 non-null object
GICS Sub Industry        505 non-null object
Location                 505 non-null object
Date first added[3][4]   398 non-null object
CIK                      505 non-null int64
Founded                  139 non-null object
```

`pandas` used to facilitate access to data providers APIs directly but this functionality has moved to the related library `pandas-datareader`. The stability of the APIs varies with provider policies, and as of June 2o18 at version 0.6, the following sources are available:

SourceType of DataNotes

Google Finance	Historical EOD Prices	Unstable
Tiingo	EOD prices on equities, mutual funds and ETF	Free API key required; limited to 500 symbols
Morningstar	Daily OHLC & Volume Data	Limited to 5 years
The Investors Exchange (IEX)	Daily OHLC & Volume Data, Order Book Summary	Limited to 5 years
Robinhood	Daily OHLC & Volume Data, Order Book Summary	Limited to 1 year
Quandl	Access to free datasets listed on website	Mixed data quality
NASDAQ Trader Symbols	Latest list of traded stocks	
Stooq	Worldwide index data	

Access and retrieval of data follow a similar API for all sources:

```
import pandas_datareader.data as web
from datetime import datetime

start = '2014' # datetime objects or time strings
end = datetime.today() # default

symbols = ['FB', 'AAPL'] # string or list
web.DataReader(symbols, 'morningstar', start).info()

MultiIndex: 2322 entries, (FB, 2014-01-01 00:00:00) to (AAPL, 2018-06-13
00:00:00)
Data columns (total 5 columns):
Close     2322 non-null float64
High      2322 non-null float64
Low       2322 non-null float64
Open      2322 non-null float64
Volume    2322 non-null int64
dtypes: float64(4), int64(1)
memory usage: 106.7+ KB
```

The Investor Exchange (IEX)

IEX is an alternative exchange started in response to the High-Frequency Trading controversy and portrayed in Michael Lewis' controversial Flash Boys. It aims to slow down the speed of trading to create a more level playing field and has been growing rapidly since launch in 2016 while still small with a market share of around 2.5% in June 2018.

In addition to historical EOD price and volume data, IEX provides real-time depth of book quotations that offer an aggregated size of orders by price and side. This service also includes last trade price and size information:

```
book = web.get_iex_book('AAPL')
orders = pd.concat([pd.DataFrame(book[side]).assign(side=side) for side in
['bids', 'asks']])
orders.sort_values('timestamp').head()

  price  size timestamp      side
4 140.00  100  1528983003604 bids
3 175.30  100  1528983900163 bids
3 205.80  100  1528983900163 asks
1 187.00  200  1528996876005 bids
2 186.29  100  1528997296755 bids
```

Quantopian

Quantopian is an investment firm that offers a research platform to crowd-source trading algorithms. Upon free registration, it enables members to research trading ideas using a broad variety of data sources. In addition, it offers an environment to backtest the algorithm against historical data, as well forward test it out-of-sample with live data. It awards investment allocations for top-performing algorithms whose authors are entitled to a 10% (at time of writing) profit share.

The Quantopian research platform consists of a Jupyter notebook environment for R&D for alpha factor research and performance analysis. In addition, there is an interactive development environment (IDE) for coding algorithmic strategies and backtesting the result using historical data since 2002 with minute bar frequency.

Users can also simulate algorithms with live data, which is known as paper trading. Quantopian provides various market datasets, including US equity and futures price and volume data at one-minute frequency, as well as US equity corporate fundamentals, and integrates numerous alternative datasets.

We will dive into the Quantopian platform in much more detail in the chapter on

Backtesting and rely on its functionality throughout the book, so feel free to open an account right away (see GitHub repo for more details).

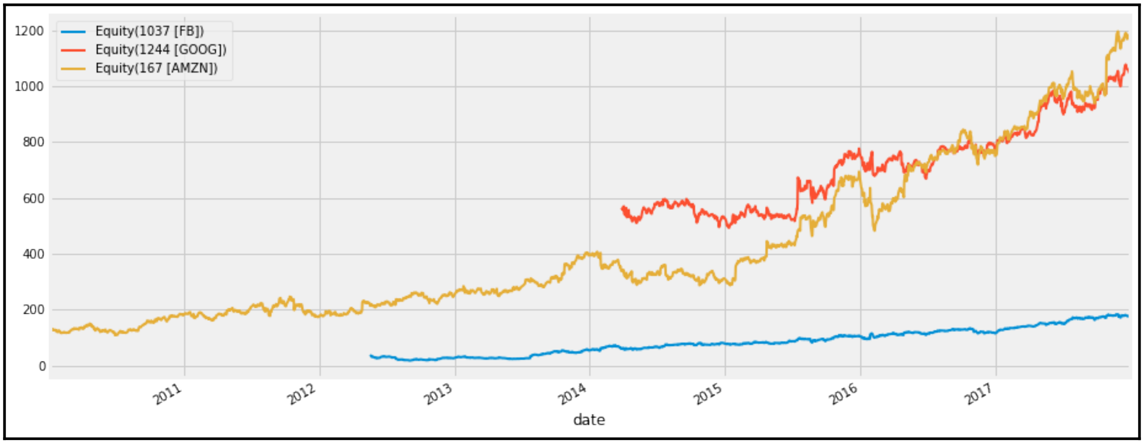
Zipline

Zipline is the algorithmic trading `library` that powers the Quantopian backtesting and live-trading platform. It is also available offline to develop a strategy using a more limited number of free data bundles that can be ingested and used to test the performance of trading ideas before porting the result to the online Quantopian platform for paper and live trading.

The following code illustrates how `zipline` permits to access daily stock data for a range of companies. You can run zipline scripts in the jupyter notebook using the magic function of the same name. First, you need to initialize the context with the desired security symbols. We'll also use a counter variable. Then, `zipline` calls the `handle_data` where we use the `data.history()` method to look back a single time period and append the data for the last day to a csv file.

```
%load_ext zipline
%%zipline --start 2010-1-1 --end 2018-1-1 --data-frequency daily
from zipline.api import order_target, record, symbol

def initialize(context):
    context.i = 0
    context.assets = [symbol('FB'), symbol('GOOG'), symbol('AMZN')]
def handle_data(context, data):
    df = data.history(context.assets, fields=['price', 'volume'],
bar_count=1, frequency="1d")
    df = df.to_frame().reset_index()
    if context.i == 0:
        df.columns = ['date', 'asset', 'price', 'volumne']
        df.to_csv('stock_data.csv', index=False)
    else:
        df.to_csv('stock_data.csv', index=False, mode='a', header=None)
    context.i += 1

df = pd.read_csv('stock_data.csv')
df.date = pd.to_datetime(df.date)
df.set_index('date').groupby('asset').price.plot(lw=2, legend=True,
figsize=(14, 6));
```

We get the following plot for the preceding code:



We will explore the capabilities of zipline and in particular the online Quantopian platform in more detail in the coming chapters.

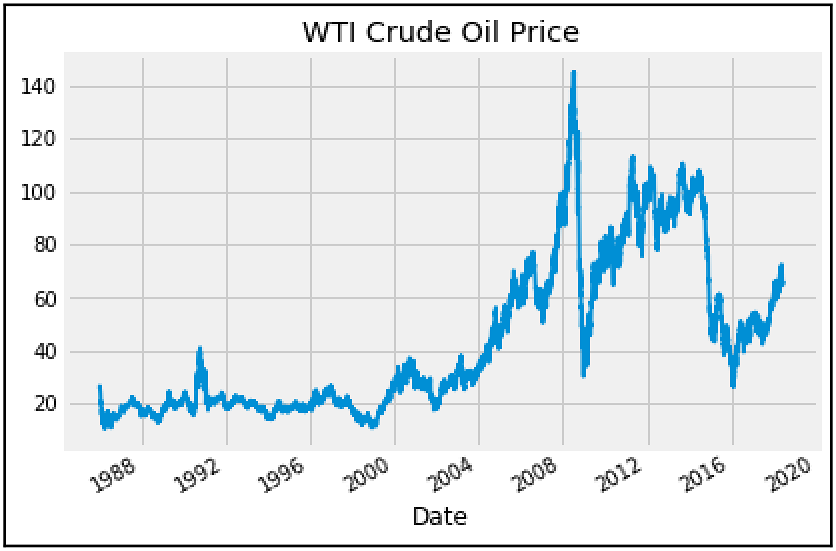
Quandl

Quandl provides a broad range of data sources both free and as a subscription via a `python API`. Register and obtain a free API key to make more than 50 calls/day. Quandl data covers multiple asset classes beyond equities and includes FX, fixed income, indexes, futures and options, and commodities.

API usage is straightforward, `well documented` and flexible with numerous methods beyond single-series downloads, e.g. including bulk download or metadata search. The below call obtains the oil prices since 1986 as quoted by the US Department of Energy:

```
import quandl
oil = quandl.get('EIA/PET_RWTC_D').squeeze()
oil.plot(lw=2, title='WTI Crude Oil Price')
```

We get the following plot for the preceding code:



Other Market Data Providers

There is both a large number and broad variety market data providers beyond those already mentioned. Rather than a comprehensive survey, here are a few categories and interesting example:

- Exchanges derive a growing share of their revenues from an ever broader range of data services, typically via subscription
- Data aggregators have long been led by Bloomberg and Thomson Reuters with a combined share of over 55% in the $28.5bn financial data market. Smaller rivals like FactSet are growing (`FT 03/22/18`), or emerging like `money.net` and `Quandl` as well as `Trading Economics` or `Barchart`.
- Specialist data providers abound. One example is `LOBSTER` that aggregates NASDAQ order book data in real time as illustrated using the historical sample above.
- Free data providers (at least for now) in addition to those above include `Alpha Vantage` with python APIs for real-time equity, FX, and crypto-currency market data, as well as technical indicators.
- Crowd-sourced investment firms that provide research platforms with data access include, in addition to Quantopian, the `Alpha Trading Labs`, launched in March 2018 that provide High-Frequency Trading infrastructure and data.

Fundamental Data

Fundamental data pertain to the economic drivers that determine the value of securities. The nature of the data depends on the asset class:

- for equities and corporate credit, it includes corporate financials as well as industry and economy-wide data
- for government bonds, it includes international macro data and foreign exchange, and
- for commodities, asset-specific demand- and supply determinants, like weather data for agricultural crops.

We will focus on equity fundamentals, and in particular on the US where data access is easier. There are some 13000+ public companies worldwide that generate 2m pages of annual reports and 30,000+ hours of earnings calls. In algorithmic trading, fundamental data and features engineered from this data may be used to derive trading signals directly, e.g. as value indicators, and are important inputs for predictive models, including machine learning models.

Financial Statements Data

The Securities and Exchange Commission (SEC) requires US issuers, i.e., listed companies and securities, including mutual funds to file 3 quarterly financial statements (Form **10-Q**) and 1 annual report (Form **10-K**), in addition to various other regulatory filing requirements.

Since the early 1990s, the SEC makes these filings available through its Electronic Data Gathering, Analysis, and Retrieval (**EDGAR**) system. They constitute the primary data source for the fundamental analysis of equity and other securities like corporate credit where the value depends on the business prospects and financial health of the issuer.

Automated Processing: XBRL

Automated analysis of regulatory filings has become much easier since the SEC introduced the eXtensible Business Reporting Language (XBRL), a free and open and global standard for the electronic representation and exchange of business reports. XBRL is based on XML and relies on **taxonomies** that define the meaning of the elements of a report and map to tags that highlight the corresponding information in the electronic version of the report. One such taxonomy represents the US Generally Accepted Accounting Principles (GAAP).

The SEC introduced voluntary XBRL filings in 2005 in response to accounting scandals before requiring this format for all filers since 2009 and continues to expand the mandatory coverage to other regulatory filings. The SEC maintains a `website` that lists the current taxonomies that shape the content of different filings and can be used to extract specific items.

The following data sets provide information extracted from EX-101 attachments submitted to the Commission in a flattened data format to assist users in consuming the data for analysis. The data reflects selected information from the XBRL tagged financial statements. It currently includes numeric data from the quarterly and annual financial statements, as well as certain additional fields (e.g. Standard Industrial Classification (SIC)).

There are several avenues to track and access fundamental data reported to the SEC.

- As part of the EDGAR Public Dissemination Service, electronic feeds of accepted filings are `available for a fee.`
- The SEC updates `Really Simple Syndication (RSS)` feeds every 10 minutes that lists structured disclosure submissions
- There are `public index files` for the retrieval of all filings via FTP for automated processing
- The Financial Statement (and Notes) datasets contain parsed XBRL data from all financial statements and the accompanying notes.

The SEC also publishes `log files` containing the internet search traffic for EDGAR filings through SEC.gov, albeit with a six months delay.

Building a Fundamental Data Time Series

The scope of the data in the `financial statement and notes data sets` consists of numeric data extracted from the primary financial statements (Balance Sheet, Income Statement, Cash Flows, Changes in Equity, and Comprehensive Income) and footnotes on those statements. The data is available since 2009.

Extracting the Financial Statements & Notes Data Set

The following code downloads and extracts all historical filings contained in the Financial Statement and Notes (FSN) data sets:

```
from io import BytesIO
from zipfile import ZipFile
import requests
```

```
SEC_URL =
'https://www.sec.gov/files/dera/data/financial-statement-and-notes-data-set
s/'

today = pd.Timestamp(date.today())
this_year = today.year
this_quarter = today.quarter

past_years = range(2009, this_year)
filing_periods = [(y, q) for y in past_years for q in range(1, 5)]
filing_periods.extend([(this_year, q) for q in range(1, this_quarter + 1)])
for yr, qtr in filing_periods:
    filing = f'{yr}q{qtr}_notes.zip'
    path = Path('edgar', f'{yr}_{qtr}')
    if not path.exists():
        path.mkdir(exist_ok=True, parents=True)

    resp = requests.get(SEC_URL + filing).content
    with ZipFile(BytesIO(resp)) as zip_file:
        for file in zip_file.namelist():
            local_file = path / file
            with local_file.open('wb') as output:
                for line in zip_file.open(file).readlines():
                    output.write(line)
```

The data is fairly large and in order to enable faster access than the original text files permit, it is better to convert the text files to binary, columnar parquet format:

```
for f in notes_dir.glob('**/*.tsv'):
    file_name = f.stem + '.parquet'
    path = Path(f.parent) / 'parquet'
    if (path / file_name).exists():
        continue
    if not path.exists():
        path.mkdir(exist_ok=True)
    df = pd.read_csv(f, sep='\t', encoding='latin1', low_memory=False)
    df.to_parquet(path / file_name)
```

For each quarter, the FSN data is organized into eight files sets containing information about submissions, numbers, taxonomy tags, presentation, and more. Each dataset consists of rows and fields and is provided as a tab-delimited TXT format file.

File	Dataset	Description
SUB	Submission	Identifies each XBRL submission by company, form, date, etc
TAG	Tag	Defines and explains each taxonomy tag
DIM	Dimension	Adds detail to numeric and plain text data
NUM	Numeric	One row for each distinct data point in filing
TXT	Plain Text	Contains all non-numeric XBRL fields
REN	Rendering	Information for rendering on SEC website

PRE	Presentation	Detail on the tag and number presentation in primary statements
CAL	Calculation	Shows arithmetic relationships among tags

Retrieve all Quarterly Apple Filings

The submission dataset contains the unique identifiers required to retrieve the filings: the Central Index Key (**CIK**) and the Accession Number (code: **adsh**). The following shows some of the information about Apple's 2018Q1 10-Q filing.

```
name = 'APPLE INC'
apple = sub[sub.name == name].T.dropna().squeeze()
info_cols = ['name', 'adsh', 'cik', 'name', 'sic', 'countryba', 'stprba',
             'cityba', 'zipba', 'bas1', 'form', 'period', 'fy', 'fp',
'filed']
apple.loc[info_cols]

name                    APPLE INC
adsh         0000320193-18-000007
cik                        320193
name                    APPLE INC
sic                          3571
countryba                      US
stprba                         CA
cityba                  CUPERTINO
zipba                       95014
bas1            ONE INFINITE LOOP
form                         10-Q
period                   20171231
fy                           2018
fp                             Q1
filed                    20180202
```

Using the Central Index Key, we can identify all historical quarterly filings available for Apple, and combine this information to obtain 26 Forms 10-Q and 9 annual Forms 10-K:

```
aapl_subs = pd.DataFrame()
for sub in notes_dir.glob('**/sub.parquet'):
    sub = pd.read_parquet(sub)
    aapl_sub = sub[(sub.cik.astype(int) == apple.cik) &
(sub.form.isin(['10-Q', '10-K']))]
    aapl_subs = pd.concat([aapl_subs, aapl_sub])

aapl_subs.form.value_counts()
10-Q    26
10-K     9
```

With the Accession Number for each filing, we can now rely on the taxonomies to select the appropriate XBRL tags (listed in the TAG file) from the NUM and TXT files to obtain the numerical or textual/footnote data points of interest.

First, let's extract all numerical data available from the 27 Apple filings:

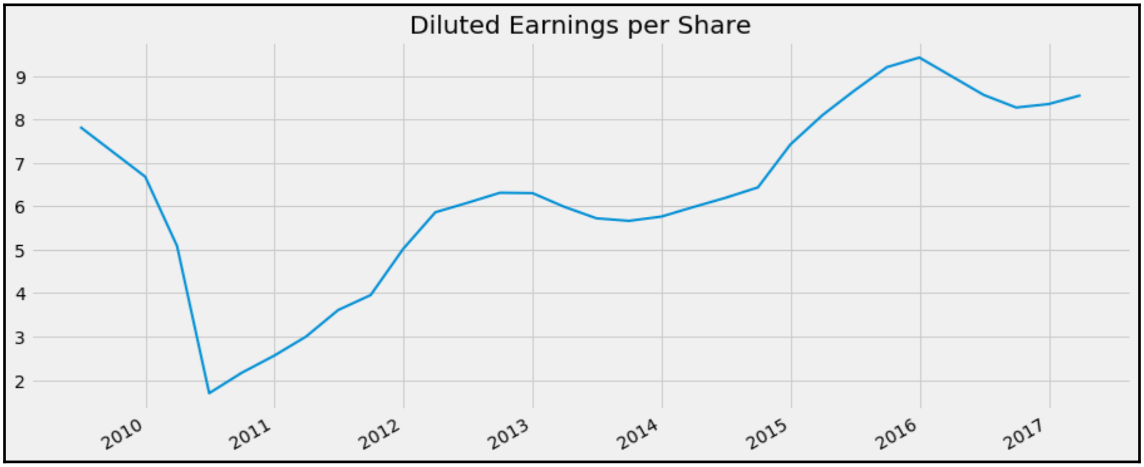
```
aapl_nums = pd.DataFrame()
for num in notes_dir.glob('**/num.parquet'):
    num = pd.read_parquet(num)
    aapl_num = num[num.adsh.isin(aapl_subs.adsh)]
    aapl_nums = pd.concat([aapl_nums, aapl_num])

aapl_nums.ddate = pd.to_datetime(aapl_nums.ddate, format='%Y%m%d')
aapl_nums.shape
(28281, 16)
```

Build a Price/Earnings time series

In total, we obtain over 28,000 numerical values from the slightly over nine years of filing history. We can select a useful field like Earnings per Diluted Share (EPS) that we can combine with market data to calculate the popular Price/Earnings valuation ratio. We do need to take into account, however, that Apple split its stock 7:1 on June 4, 2014, and adjust Earnings per Share prior to the split to make earnings comparable.

```
field = 'EarningsPerShareDiluted'
stock_split = 7
split_date = pd.to_datetime('20140604')

# Filter by tag; keep only values measuring 1 quarter
eps = aapl_nums[(aapl_nums.tag == 'EarningsPerShareDiluted')
                & (aapl_nums.qtrs == 1)].drop('tag', axis=1)

# Keep only most recent data point from each filing
eps = eps.groupby('adsh').apply(lambda x: x.nlargest(n=1,
columns=['ddate']))

# Adjust earnings prior to stock split downward
eps.loc[eps.ddate < split_date,'value'] = eps.loc[eps.ddate < split_date,
'value'].div(7)
eps = eps[['ddate', 'value']].set_index('ddate').squeeze()
eps = eps.rolling(4).sum().dropna() # create trailing 12-months eps from
quarterly data
eps.plot(lw=2, figsize=(14, 6), title='Diluted Earnings per Share')
```

We get the following plot for the preceding code:



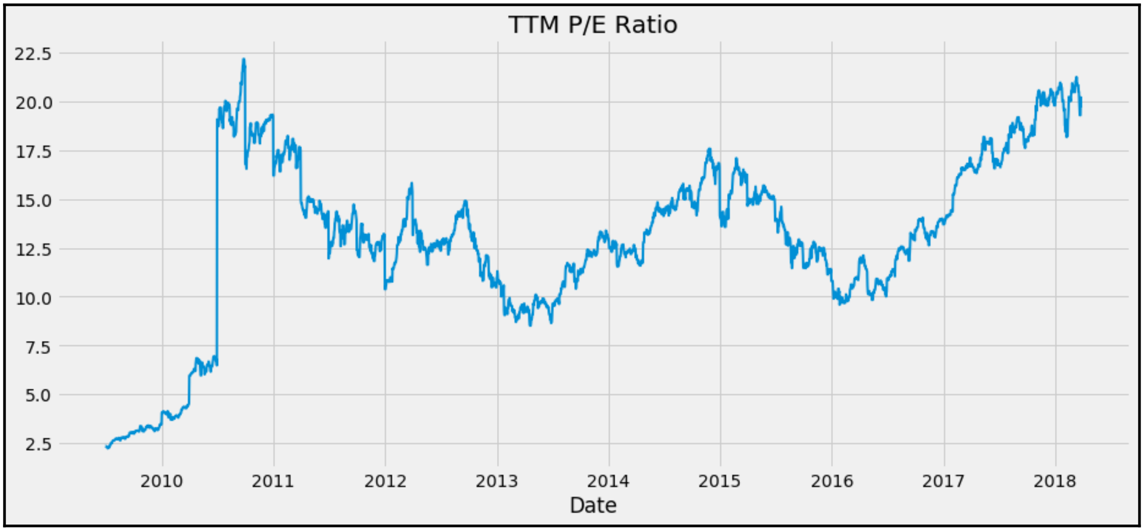
We can use (for example) Quandl to obtain Apple stock price data since 2009:

```
import pandas_datareader.data as web
symbol = 'AAPL.US'
aapl_stock = web.DataReader(symbol, 'quandl', '2009-06-01')
aapl_stock = aapl_stock.resample('D').last() # ensure dates align with eps
data
```

Now we have the data to compute the trailing 12-months P/E ratio for the entire period:

```
pe = aapl_stock.AdjClose.to_frame('price').join(eps.to_frame('eps'))
pe = pe.fillna(method='ffill').dropna()
pe['P/E Ratio'] = pe.price.div(pe.eps)
pe['P/E Ratio'].plot(lw=2, figsize=(14, 6), title='TTM P/E Ratio');
```

We get the following plot for the preceding code:



Other Fundamental Data Sources

There are numerous other sources for fundamental data. Many are accessible via the pandas_datareader module introduce earlier. Other data is available from certain organizations directly.

pandas_datareader: Macro & Industry Data

The `pandas_datareader` library facilitates access according to the conventions introduced above to APIs with numerous global fundamental macro a and industry data sources, including:

- Kenneth French's data library: market data on portfolios capturing size, value and momentum factors, disaggregated y industry
- St.Louis FED (FRED): Federal Reserve data on the US economy and financial markets
- World Bank: Global database on long-term, lower-frequency economic and social development and demographics
- OECD: similar for OECD countries

- Enigma: various datasets, including alternative sources
- Eurostat: EU-focused economics, social and demographic data

Summary

This chapter has introduced market and fundamental data sources that form the backbone of most trading strategies. You have learned about numerous ways to access this data, and how to preprocess the raw information so that you can begin extracting trading signals using the machine learning techniques that we will be introducing shortly.

Before we move on the design and evaluation of trading strategies and the use of machine learning models, we need to cover alternative datasets that have emerged in recent years and have been a significant driver of the popularity of machine learning for algorithmic trading.

3
Working with Alternative Data

Propelled by the explosive growth of the Internet and mobile networks, digital data continues to grow exponentially amid advances in the technology to process, store and analyze new data sources. The exponential growth in the availability of and ability to manage more diverse digital data, in turn, has been a key force behind the dramatic performance improvements of machine learning that are driving innovation across industries, including the investment industry.

The scale of the data revolution is extraordinary: the past two years alone have witnessed the creation of 90% of all data that exists in the world today, and by 2020, each of the 7.7 billion people worldwide is expected to produce 1.7 MB of new information every second of every day. On the other hand, back in 2012 only 0.5 percent of all data was ever analyzed and used, whereas 33% is deemed to have value by 2020. The gap between data availability and usage is likely to narrow quickly as global investments in analytics are set to rise beyond $210 billion by 2020, while the value creation potential is a multiple higher (IDC, MIT Tech Review 2017, McKinsey 2016).

This chapter explains how individuals, business processes, and sensors produce alternative data. It also provides a framework to navigate and evaluate the proliferating supply of alternative data for investment purposes. It demonstrates the workflow from acquisition to preprocessing and storage using python for data obtained through web scraping to set the stage for the application of machine learning. It concludes by providing examples of sources, providers, and applications.

This chapter will cover the following topics:

- How the Alternative Data Revolution has unleashed new sources of information
- How individuals, business processes, and sensors generate alternative data
- How to evaluate the proliferating supply of alternative data used for algorithmic trading
- How to work with alternative data in python, e.g. by scraping the internet
- Important categories and providers of alternative data

The alternative data revolution

The data deluge driven by digitization, networking, and plummeting storage costs has led to profound qualitative changes in the nature of information available for predictive analytics often summarized by the **5V**:

- **Volume**: the amount of data generated, collected and stored is orders of magnitude larger as the byproduct of online and offline activity, transactions, records, and other sources and volumes continue to grow with the capacity for analysis and storage
- **Velocity**: data is produced, transferred and processed close to or at real-time speed
- **Variety**: data is organized in formats beyond structured, tabular form (e.g. CSV files or relational database tables), diversifying into semi-structured formats like JSON or HTML and unstructured content such as raw text, image, audio or video data, creating new pre-processing challenges to render it suitable for machine learning algorithms
- **Veracity**: the diversity of sources and formats makes it much more difficult to validate the reliability of the data's information content
- **Value**: as a consequence, determining the value of new datasets can be much more time- and resource-consuming as well as uncertain than before

For algorithmic trading, new sources of data provide **informational advantages** if they uncover new information that is not available from traditional sources, or uncover this information sooner. Following global trends, the investment industry is rapidly expanding beyond conventional market and fundamental to alternative sources of data to reap alpha through an informational edge. Annual spending on data, technological capabilities, and related talent are expected to increase from currently $3bn by 12.8% annually through 2020 (JPM 2017).

Today, investors can access **real-time macro or company-specific data** that historically has only been available at a much lower frequency. For instance, the online prices of a representative set of goods and services can be used to gauge inflation. the number of customers visiting a store and making purchases can give real-time estimates of sales or economic activity, while satellite imaging can reveal agricultural yields or activity of mines or oil rigs before this information is available elsewhere. As standardization and adoption of big data sets advances, the information in conventional data will likely lose most predictive value.

Access to new sources of data, combined with the capability to capture, process and integrate diverse datasets quickly allows for complex insights based on machine learning. In the past, quantitative approaches relied on simpler rules to rank companies based on historical data for metrics like the price-to-book ratio, whereas machine learning techniques allow algorithms to synthesize new metrics and learn and adopt such rules using constantly changing data. These insights create new ways to capture investment themes such as value, momentum, quality, or sentiment. For instance:

- **Momentum**: machine learning can identify data-driven connections among companies using industry sentiment, market price movements, or exposure to and correlations of economic factors

- **Value**: algorithms can analyze large amounts of economic and industry-specific structured and unstructured data beyond financial statements to estimate or predict a company's intrinsic value

- **Quality**: evaluate customer or employee reviews as well as web traffic to identify gains in market or e-commerce share

In practice, however, useful data is often not freely available and alternative datasets instead require thorough evaluation, costly acquisition, careful management, and sophisticated analysis in order to extract tradeable signals.

Sources of alternative data

Alternative datasets originate from many sources but can be classified at a high level as predominantly generated by:

- **Individuals** through activities like social media posts, product reviews or internet searches
- **Business** processes through commercial transactions, credit card data, and so on
- **Sensors** such as satellite imagery of activity around raw material production or foot and car traffic, geo-location data, and so on

The definition of alternative data continues to evolve as new data sources become available. At the same time, sources previously dubbed alternative become part of the financial mainstream like the Baltic Dry Index that assembles data from ~600 shipping companies to approximate the demand/supply of dry bulk carriers.

Alternative data includes data in its raw form, as well as data that is aggregated or processed in some form. For instance, some providers aim to extract tradeable signals such as sentiment scores. We will address the various types of providers in the next chapter.

Sources of alternative data differ in key respects that determine their value or signal content for algorithmic trading strategies. We will address these aspects in the subsequent chapter.

Individuals

Individuals generate date through their online activity, or through offline activity that is captured electronically. Many sources originated by individuals are unstructured in the form of text, images, and video and is disseminated over multiple platforms. Data generated by individuals include:

- Social media posts, both general purpose through sites like Twitter, Facebook or LinkedIn or through specialized sites that review businesses like Glassdoor or Yelp
- Generic E-commerce sites like Amazon or Wayfair as well as proprietary e-commerce sites that include product reviews,
- Web searches through search engines like Google Search
- Mobile app activity and downloads
- Personal data such as personal inboxes

Social media sentiment analysis is a very popular type of alternative data because it can be applied to individual stocks, baskets or broader indices. The most common source is Twitter, followed by various news vendors and blog sites. Sentiment data is competitive and cheaper to acquire because it is often obtained through web scraping. Reliable datasets of social media activity including blogs, tweets or videos are usually available with less than ~5 years of history.

Search history, in contrast, is available since 2004 and can be used to extract signals with a significantly longer time history.

Business processes

Another important source of alternative data is produced or collected by businesses and public entities. In contrast to human-generated data, data generated by business processes is often highly structured and is popular as a leading indicator for corporate activity that otherwise is reported at a significantly lower frequency.

Data generated by business processes include:

- Credit and debit card transaction data made available by payments processors and financial institutions
- Company exhaust data that results from record-keeping in the course of business like banking records, supermarket scanner data, or supply chain data
- Federal and state-level agencies generate large amounts of data on economic and industry-level activity often with longer histories but lower frequencies
- Trade flow and market microstructure data (such as L-2 and L-3 order-book tick data as illustrated in the previous chapter)
- Agencies or financial institutions that monitor or manage company payments to assess liquidity and creditworthiness

The most reliable datasets generated by businesses are credit card transactions and company exhaust data like POS data. Credit card data is available with approximately 10 years of history and can be obtained with a six-day lag, while corporate earnings occur quarterly with a two and a half week lag. The time horizon for company exhaust data varies widely with the source.

Market microstructure datasets are available with over 15 years of history, while sell-side flow data is typically available with less than 5 years of consistent history.

Public agencies often provide datasets with very long histories like the large federal data pools posted online by the US government in the past few years, but at a much-reduced frequency.

Sensors

Data collected through networked sensors that are embedded in various devices are among the most rapidly growing categories of data sources. The proliferation of sensors in smartphones (with >1.5 billion units shipped in 2018) and the cost reductions in satellite technologies continue to be key drivers.

This category of alternative data is typically **highly unstructured** and often **significantly larger in volume** than either data generated by human activity or business processes. Alternative data sources in this category include:

- Satellite imaging to monitor economic activity like construction, shipping, or commodity supply

- Geolocation data to track traffic in retail stores, e.g. using volunteered smartphone data, or on transport routes like ships or trucks
- Cameras positioned at a location of interest, or
- Weather and pollution sensors.

The **Internet of Things** (**IoT**) will further accelerate the large-scale collection of alternative data by embedding microprocessors and networking technology into personal and commercial electronic devices like home appliances, public spaces, industrial production processes, etc.

Datasets covering sensor-based alternative data like satellite imagery, mobile app data, and cellular location tracking are typically available with a history of ~3-4 years.

Satellites

The costs and timelines to launch a traditional geospatial imaging satellite have fallen dramatically. Instead of tens of millions of dollars and years of preparation, it now costs only approx. $100,000 to place shoe-box sized nano-satellites as secondary payloads into low-earth orbits. As a result, companies can now use fleets of satellites for much higher-frequency (currently about daily) aerial coverage of specific locations.

These can be used to monitor numerous aspects of economic and commercial activity like agricultural and mineral production and shipments, construction of real estates or ships, industrial incidents like a fire, or car and foot traffic at locations of interest.

Qualitative challenges for the use of satellite imaging data in machine learning models include the treatment of cloud cover, seasonalities around holidays, and irregular coverage of certain locations that may affect the quality of the predictive signals.

Geolocation data

Geolocation data is another important and rapidly growing category of data generated by sensors. It is typically sourced from personal geographic information volunteered by individuals. The location of smartphones can be tracked using either GPS, CDMA or WiFi signals to measure, for instance, foot traffic around store locations.

In addition, a rising number of malls and retail stores are installing sensors to track the numbers and movements of customers. Originally developed to measure the effectiveness of marketing activity, the resulting data can also be used to estimate business activity metrics. These sensors include 3D stereo video and thermal imaging. The latter reduces privacy concerns but works well with moving objects. Irisys, a leading provider, has installed over 300,000 thermal imaging sensors worldwide. Other technologies include sensors fixed to ceilings or pressure-sensitive mats, while some providers combine multiple sensors - including vision, audio and cell-phone tracking for a more detailed account of shopper count, dwell time, conversion rate and frequency of visits.

Another source of sensor data originates from drones used in agriculture that monitor crop health using infra-red light.

Evaluating alternative datasets

The ultimate objective of alternative datasets is to provide an informational advantage in the competitive search for trading signals that produce alpha: positive, uncorrelated investment returns. In practice, the signals extracted from alternative datasets can be used on a standalone basis if the Sharpe ratio (see next chapter for detail) is sufficiently high, or combined with other signals as part of a quantitative strategy.

In fact, quant firms are building libraries of **alpha factors** that may be weak signals individually but can produce attractive returns in combination. As highlighted in the first chapter, investment factors should be based on a fundamental and economic rationale because otherwise, they are more likely the result of overfitting to the historical data than to persist and generate alpha on new data.

Signal decay due to competition is a serious concern, and as the alternative data ecosystem evolves, it is unlikely that many datasets will maintain high Sharpe ratio signals. Strategies to prolong the half-life of the alpha content of an alternative dataset include exclusivity agreements or a focus on datasets that are challenging to deal with in order to raise the barriers to entry.

Evaluation criteria

An alternative dataset can be evaluated based on the quality of its signal content, qualitative aspects of the data, and various technical aspects.

Quality of the signal content

The signal content can be evaluated with respect to the target asset class, the suitable investment style, the relation to conventional risk premia, and most importantly, its alpha content.

Asset Class

The majority of alternative datasets focus on equities and commodities. There are also interesting datasets on real estate investments. The availability of alternative data on fixed income securities or interest rates, as well as currencies is small, but growing as more datasets on corporate payments are being offered.

Investment style

Most datasets are sector and stock-specific, and most relevant for investors in equities that take long and short positions. There is also a significant amount of data relevant for macro investors (for example, consumer credit, emerging market economic activity, shipping, and so on).

Certain alternative datasets can be used as substitutes for traditional metrics of market risk, and some signals are only relevant for high-frequency quant traders with a very short time horizon.

Risk premia

There is evidence that the correlation of signals extracted from alternative datasets with traditional equity risk premia like momentum, value, quality or volatility is low (less than 5%). Hence, integrating signals based on alternative datasets like social media sentiment scores into an algorithmic trading strategy can be important building blocks of a diversified risk premia portfolio.

Alpha content and quality

The signal strength required from a dataset is related to its costs, and the prices of alternative datasets vary widely: sentiment score data is available for a few thousand dollars or less, while comprehensive credit card data can cost a few million USD a year. Trading strategies based on alternative data are tested to estimate the alpha content from backtests. These tests can determine whether a dataset contains sufficient alpha signal to make it viable as a standalone driver of a strategy, but these situations are (increasingly) rare.

Most datasets allow for the extraction of weak signals that translate into a small positive Sharpe ratio that would no justify capital allocation n a standalone basis. However, these datasets add value when their signals are integrated with other signals to deliver a viable portfolio-level strategy. However, there are also plenty of alternative datasets without any meaningful alpha content.

In addition to pure alpha, it is also necessary to assess whether the signal provided by the dataset is incremental or orthogonal, i.e. unique to a dataset, or already captured by other data, and how the costs compare.

Furthermore, it is important to gauge the potential capacity of a strategy based on a dataset - capacity limitations make it much more difficult to recoup the investment in the dataset.

Quality of the data

Quality of data is another important feature and include aspects like the data frequency and length of available history, the reliability of the information it contains, the degree of compliance with current or likely future regulations, and the exclusivity of access.

Legal and reputational risks

The usage of many alternative data sets may carry legal or reputational risk due to the following potential items included in a dataset, assuming data and terms of use are available:

- Material Non-Public Information (MNPI) to avoid infringement of insider trading regulation, and
- Personally Identifiable Information (PII), in particular since the European General Data Protection Regulation has entered into force

One should ensure that all legal and compliance requirements are satisfied. There could also be conflicts of interest when the data provider is actively trading using the dataset.

Exclusivity

The probability that a dataset contains a signal strong enough to drive a stand-alone strategy with a high Sharpe ratio is inversely related to its availability and ease of processing. The more exclusive, and the harder to process, the more likely it is that a dataset can drive a strategy based on signals that decay more slowly.

Public fundamental datasets containing financial ratios (P/E, P/B, etc.) have fairly low alpha content and are not attractive as a standalone strategy, but may be useful in the context of a diversified risk factor portfolio. Large, complex datasets will take more time to be widely used and fully explored, while new datasets continue to emerge on a frequent basis. It is important to assess how widely a dataset is known, and the data provider is the best source for this type of information.

There are additional benefits to being an exclusive or at least early client, e.g. when a business begins to sell exhaust data: it may be possible to influence the scope of data collection and curation or to request exclusive or limited-sales deals to limit access to a pre-defined number of clients.

Time Horizon

Data with a longer history is more desirable for the purpose of testing. The available history varies between several months and several decades with important implications for the type of trading strategy that can be developed and tested. For satellite imagery, time horizon is typically > 3 years, sentiment data > 5 years, and credit card data > 7 years.

Frequency

The frequency can be intra-day, daily, weekly, or even lower frequency.

Reliability

Lastly, it is necessary to be able to verify the accuracy of the information represented by the data and carry out an audit if the scale of the purchase justifies the effort.

Technical aspects

Technical aspects include latency and format of the data

Latency

Data providers often provide data in batches, and a delay is possible either due to the collection, operational or legal constraints.

Format

The data must be extracted in a suitable format, preferably CSV or JSON for static data. The API (Application Programming Interface) should be robust. It should not fail or result in additional latency, and it should be flexible to accommodate different programming languages.

The market for alternative data

The investment management industry is estimated to spend $2-3bn on big data, and this number is expected to grow at double digits annually in line with Big Data spending growth in other industries. This expenditure includes the acquisition of alternative datasets, investments into Big Data technology, and hiring appropriate talent.

The market for alternative data providers is quite fragmented. J.P. Morgan lists over 500 specialized data firms, while alternativedata.org lists over 300.

Providers play numerous roles, including intermediaries like consultants, aggregators, and tech solutions; sell-side supports deliver data in various formats, ranging from raw to semi-processed data or some form of a signal extracted from one or more sources.

Challenge: short history; microstructure & macro 15+, many others rather short-term; signal extraction more challenging

- Gnip (acquired by Twitter)

Data providers and use cases

Alternativedata.org (`https://alternativedata.org/`) lists several categories that can serve as a rough proxy for activity in various data provider segments. Social sentiment analysis is by far the largest category, while satellite and geo-location data have been growing rapidly in recent years.

Product Category	# **Providers**
Social Sentiment	48
Satellite	26
Geo-Location	22
Web Data & Traffic	22
Infrastructure & Interfaces	20
Consultants	18
Credit & Debit Card Usage	14
Data Brokers	10
Public Data	10
App Usage	7
Email & Consumer Receipts	6
Sell Side	6
Weather	4
Other	87

Social Sentiment Data

Gnip (acquired by Twitter)

DataMinr

StockTwits

RavenPack

Satellite Data

RS Metrics

Geolocation data

Advan

Email receipt data

Eagle Alpha

Working with alternative data in Python

We will first learn Scraping OpenTable data and then move on to earning call transcripts.

Scraping OpenTable Data

Typical sources of alternative data are review websites like Glassdoor or Yelp that convey insider insights via employee comments or guest reviews. This data provides valuable inputs for machine learning models that aim to predict changes in the prospects for a business or directly in market values to obtain trading signals.

To obtain the data, it needs to be extracted from the html source, barring any legal obstacles. To illustrate the web scraping tools that python offers, we'll retrieve information on restaurant bookings from OpenTable. Data of this nature could be used to forecast changes in geographic economic activity, real estate prices, or restaurant chain revenues.

Extracting data from HTML using requests and BeautifulSoup

Request and parse HTML source code: We will be using the libraries `requests` for Hyper Text Transfer Protocol (HTTP) requests to retrieve the html source code, and `BeautifulSoup` to parse and extract the text content. We will, however, encounter a common obstacle: websites may request certain information from the server only after initial page load using javascript. As a result, a direct HTTP request will not be successful. To sidestep this type of protection, we will use a headless browser that retrieves the website content like a browser would.

```
from bs4 import BeautifulSoup
import requests

# set and request url; extract source code
url = "https://www.opentable.com/new-york-restaurant-listings"
html = requests.get(url)
html.text[:500]

' <!DOCTYPE html><html lang="en"><head><meta charset="utf-8"/><meta http-
equiv="X-UA-Compatible" content="IE=9; IE=8; IE=7; IE=EDGE"/>
<title>Restaurant Reservation Availability</title> <meta name="robots"
content="noindex" > </meta> <link rel="shortcut icon"
href="//components.otstatic.com/components/favicon/1.0.4/favicon/favicon.ic
o" type="image/x-icon"/><link rel="icon"
href="//components.otstatic.com/components/favicon/1.0.4/favicon/favicon-16
.png" sizes="16x16"/><link rel='
```

Now we can use BeautifulSoup to parse the html content, and then look for all `span` tags with the class associated with the restaurant names that we obtain by `inspecting the source code`: `rest-row-name-text`

```
# parse raw html => soup object
soup = BeautifulSoup(html.text, 'html.parser')

# for each span tag, print out text => restaurant name
for entry in soup.find_all(name='span', attrs={'class':'rest-row-name-
text'}):
    print(entry.text)

Wade Coves
Alley
Dolorem Maggio
Islands
...
```

Once you have identified the page elements of interest, `BeautifulSoup` makes it easy to retrieve the contained text. If you want to get the price category for each restaurant, you can use:

```
# get the number of dollars signs for each restaurant
for entry in soup.find_all('div', {'class':'rest-row-pricing'}):
    price = entry.find('i').text
```

When you try to get the number of bookings, however, you just get an empty list because the site uses javascript code to request this information after the initial loading is complete.

```
soup.find_all('div', {'class':'booking'})
[]
```

Introducing Selenium: Using browser automation

We will use the browser automation tool `Selenium` to operate a headless Firefox browser that will parse the html content for us.

The following code opens the FireFox browser:

```
from selenium import webdriver

# create a driver called Firefox
driver = webdriver.Firefox()
```

Let's close the browser again:

```
# close it
driver.close()
```

To retrieve the html source code using selenium and Firefox:

```
import time, re

# visit the opentable listing page
driver = webdriver.Firefox()
driver.get(url)

time.sleep(1) # wait 1 second

# retrieve the html source
html = driver.page_source
html = BeautifulSoup(html, "lxml")

for booking in html.find_all('div', {'class': 'booking'}):
    match = re.search(r'\d+', booking.text)
```

```
        if match:
            print(match.group())
```

Build a dataset of restaurant bookings

Now you only need to combine all interesting elements from the website to create a feature that you could use in a model to predict economic activity in geographic regions or foot traffic in specific neighborhoods.

With Selenium, you can follow the links to the next pages and quickly build a dataset of over 10,000 restaurants in NYC that you could then update periodically to track a time series:

```
from bs4 import BeautifulSoup
from selenium import webdriver
from time import sleep
import pandas as pd
import re

url = "https://www.opentable.com/new-york-restaurant-listings"

def parse_html(html):
    data, item = pd.DataFrame(), {}
    soup = BeautifulSoup(html, 'lxml')
    for i, resto in enumerate(soup.find_all('div', class_='rest-row-
info')):
        item['name'] = resto.find('span', class_='rest-row-name-text').text

        booking = resto.find('div', class_='booking')
        item['bookings'] = re.search('\d+', booking.text).group() if
booking else 'NA'

        rating = resto.select('div.all-stars.filled')
        item['rating'] = int(re.search('\d+',
rating[0].get('style')).group()) if rating else 'NA'

        reviews = resto.find('span', class_='star-rating-text--review-
text')
        item['reviews'] = int(re.search('\d+', reviews.text).group()) if
reviews else 'NA'

        item['price'] = int(resto.find('div', class_='rest-row-
pricing').find('i').text.count('$'))
        item['cuisine'] = resto.find('span', class_='rest-row-meta--
cuisine').text
        item['location'] = resto.find('span', class_='rest-row-meta--
location').text
        data[i] = pd.Series(item)
    return data.T


restaurants = pd.DataFrame()
driver = webdriver.Firefox()
driver.get(url)
while True:
    sleep(1)
    new_data = parse_html(driver.page_source)
    if new_data.empty:
        break
    restaurants = pd.concat([restaurants, new_data], ignore_index=True)
    driver.find_element_by_link_text('Next').click()

driver.close()
```

One Step Further: Scrapy and Splash

Scrapy is a powerful library to build bots that follow links, retrieve the content and store the parsed result in a structured way. In combination with the headless browser splash it can also interpret javascript and becomes an efficient alternative to Selenium:

```
from opentable.items import OpentableItem
from scrapy import Spider
from scrapy_splash import SplashRequest


class OpenTableSpider(Spider):
    name = 'opentable'
    start_urls = ['https://www.opentable.com/new-york-restaurant-listings']

    def start_requests(self):
        for url in self.start_urls:
            yield SplashRequest(url=url,
                                callback=self.parse,
                                endpoint='render.html',
                                args={'wait': 1},
                                )

    def parse(self, response):
        item = OpentableItem()
        for resto in response.css('div.rest-row-info'):
            item['name'] = resto.css('span.rest-row-name-
text::text').extract()
            item['bookings'] = resto.css('div.booking::text').re(r'\d+')
            item['rating'] = resto.css('div.all-
stars::attr(style)').re_first('\d+')
            item['reviews'] = resto.css('span.star-rating-text--review-
text::text').re_first(r'\d+')
            item['price'] = len(resto.css('div.rest-row-pricing >
i::text').re('\$'))
            item['cuisine'] = resto.css('span.rest-row-meta--
cuisine::text').extract()
            item['location'] = resto.css('span.rest-row-meta--
location::text').extract()
            yield item
```

There are numerous ways to extract information from this data beyond the reviews and bookings of individual restaurants or chains.

We could further collect and geoencode the restaurants' addresses, for instance, to link the restaurants' physical location to other areas of interest, says important retail spots or neighborhoods to gain insights into particular aspects of economic activity. As mentioned before, such data will be most valuable in combination with other information.

Earning Call Transcripts

Textual data is an important alternative data source. One example of textual data id transcripts of earnings calls where executives do not only present the latest financial results, but also respond to questions by financial analysts. Investors utilize transcripts to evaluate changes in sentiment, emphasis on particular topics, or style of communication.

We will illustrate the scraping and parsing of earnings call transcripts from the popular trading website `www.seekingalpha.com`.

```
import re
from pathlib import Path
from time import sleep
from urllib.parse import urljoin
from bs4 import BeautifulSoup
from furl import furl
from selenium import webdriver

transcript_path = Path('transcripts')

SA_URL = 'https://seekingalpha.com/'
TRANSCRIPT = re.compile('Earnings Call Transcript')

next_page = True
page = 1
```

```
driver = webdriver.Firefox()
while next_page:
    url = f'{SA_URL}/earnings/earnings-call-transcripts/{page}'
    driver.get(urljoin(SA_URL, url))
    response = driver.page_source
    page += 1
    soup = BeautifulSoup(response, 'lxml')
    links = soup.find_all(name='a', string=TRANSCRIPT)
    if len(links) == 0:
        next_page = False
    else:
        for link in links:
            transcript_url = link.attrs.get('href')
            article_url = furl(urljoin(SA_URL,
transcript_url)).add({'part': 'single'})
            driver.get(article_url.url)
            html = driver.page_source
            meta, participants, content = parse_html(html)
            meta['link'] = link

driver.close()
```

Parse HTML using regular expressions

In order to collect structured data from the unstructured transcripts, we can use regular expressions in addition to BeautifulSoup.

This allows us to collect detailed information not only about the Earnings Call company and timing, but also capture who was present and attribute the statements to analysts and company representatives.

```
def parse_html(html):
    date_pattern = re.compile(r'(\d{2})-(\d{2})-(\d{2})')
    quarter_pattern = re.compile(r'(\bQ\d\b)')
    soup = BeautifulSoup(html, 'lxml')

    meta, participants, content = {}, [], []
    h1 = soup.find('h1', itemprop='headline').text
    meta['company'] = h1[:h1.find('(')].strip()
    meta['symbol'] = h1[h1.find('(') + 1:h1.find(')')]

    title = soup.find('div', class_='title').text
    match = date_pattern.search(title)
    if match:
        m, d, y = match.groups()
        meta['month'] = int(m)
```

```
            meta['day'] = int(d)
            meta['year'] = int(y)

        match = quarter_pattern.search(title)
        if match:
            meta['quarter'] = match.group(0)

        qa = 0
        speaker_types = ['Executives', 'Analysts']
        for header in [p.parent for p in soup.find_all('strong')]:
            text = header.text.strip()
            if text.lower().startswith('copyright'):
                continue
            elif text.lower().startswith('question-and'):
                qa = 1
                continue
            elif any([type in text for type in speaker_types]):
                for participant in header.find_next_siblings('p'):
                    if participant.find('strong'):
                        break
                    else:
                        participants.append([text, participant.text])
            else:
                p = []
                for participant in header.find_next_siblings('p'):
                    if participant.find('strong'):
                        break
                    else:
                        p.append(participant.text)
                content.append([header.text, qa, '\n'.join(p)])
        return meta, participants, content
```

We store the result in several csv files for easy access later when we use machine learning to process natural language.

```
    def store_result(meta, participants, content):
        path = transcript_path / 'parsed' / meta['symbol']
        if not path.exists():
            path.mkdir(parents=True, exist_ok=True)
        pd.DataFrame(content, columns=['speaker', 'q&a',
    'content']).to_csv(path / 'content.csv', index=False)
        pd.DataFrame(participants, columns=['type', 'name']).to_csv(path /
    'participants.csv', index=False)
        pd.Series(meta).to_csv(path / 'earnings.csv')
```

Resources & References

Open Source:

- `Awesome Quant`: a curated collection of useful libraries, data source and other resources for quantitative finance, incl. beyond python
- https://datahub.io/
- Kaggle

Summary

In this chapter, we have introduced the new sources of alternative data made available as a result of the big data revolution, including individuals, business processes and sensors like satellites or GPS location devices. We have presented a framework to evaluate alternative datasets from an investment perspective and laid out key categories and providers to help you navigate this vast and quickly expanding area that provides the key inputs for algorithmic trading strategies that use machine learning.

We have explored powerful python tools to collect your own datasets at scale so that you can potentially work on getting your own information edge as an algorithmic trader using web scraping.

We will now proceed to the design and evaluation of alpha factors that produce trading signals, and how to combine them in a portfolio context.

4

Alpha Factors: Research and Evaluation

Coming soon...

5
Strategy Evaluation and Portfolio Management

Coming soon...

6
The Machine Learning Process

Coming soon...

7 Generalized Linear Models

Coming soon...

8 Linear Time Series Models

Coming soon...

9
Decision Trees and Random Forests

Coming soon...

10
Gradient Boosting Machines

Coming soon...

11
Bayesian Machine Learning for Algorithmic Trading using PyMC3

Coming soon...

12 From text to numbers: the document-text matrix and text classification

Coming soon...

13

Making sense of very large language data: Topic Modeling using gensim

Coming soon...

14 State-of-the-art text features: word2vec models

Coming soon...

15 Introduction to Deep Neural Networks

Coming soon...

16 Recurrent Neural Networks

Coming soon...

17
Convolutional Neural Networks: Image Recognition for Satellite Data

Coming soon...

18
Reinforcement Learning: Key Principles, Q-Learning and the Dyna Architecture

Coming soon...

19

Transfer Learning: Leapfrogging based on Pre-Trained Models using keras

Coming soon...

20 Next Steps

Coming soon...

21

Unsupervised Learning: Filters, Clustering and Dimensionality Reduction for Financial Data

Coming soon...