Содержание

Часть III. Обработка и анализ финансовых данных

Часть V. Анализ деривативов

ГЛАВА 1

Python как инструмент финансовых расчетов

Банки — самые высокотехнологичные предприятия.
Хуго Банцигер

Язык программирования Python

Python — это универсальный язык программирования высокого уровня, который широко применяется в самых разных областях. На официальном сайте Python (`https://www.python.org/doc/essays/blurb`) он характеризуется следующим образом.

> Python — это интерпретируемый, объектно-ориентированный, высокоуровневый язык программирования с динамической семантикой. Высокоуровневые структуры данных в сочетании с динамической типизацией и динамическим связыванием делают его удобным инструментом быстрой разработки приложений, а также привлекательным средством написания сценариев и языком интеграции существующих компонентов. Простой в изучении и понятный синтаксис языка ориентирован на удобство чтения кода, что позволяет снизить затраты на сопровождение программ. В Python поддерживаются модули и пакеты, что облегчает повторное использования кода и способствует повышению модульности программ. Интерпретатор и богатая стандартная библиотека языка свободно доступны как в виде исходных кодов, так и в бинарном виде для большинства компьютерных платформ и могут свободно распространяться.

Приведенное описание как нельзя лучше показывает, почему Python стал одним из самых популярных языков программирования. В наши дни им пользуются как начинающие программисты, так и профессиональные разработчики. Он применяется везде: в школах, университетах, крупных компаниях и финансовых учреждениях, а также на веб-сайтах и в научной среде.

Среди преимуществ Python можно выделить следующее.

Открытый исходный код
: Большинство программных инструментов и библиотек Python находится в свободном доступе и распространяется на условиях открытой лицензии.

Интерпретируемость
: Эталонной реализацией считается интерпретатор CPython, преобразующий код Python в байтовый код, непосредственно выполняемый компьютером.

Мультипарадигменность
: Python поддерживает самые разные стили программирования, в частности объектно-ориентированный, императивный, функциональный и процедурный.

Универсальность
: Python подходит для написания как простых интерактивных приложений, так и сложных многофункциональных программ. При этом он прекрасно справляется с выполнением как низкоуровневых системных операций, так и высокоуровневых аналитических задач.

Многоплатформенность
: Python доступен для всех популярных операционных систем, таких как Windows, Linux и macOS. Он позволяет писать как настольные, так и веб-приложения, которые можно запускать как в огромных серверных кластерах, так и на небольших устройствах типа Raspberry Pi.

Динамическая типизация
: В Python типы данных определяются непосредственно на этапе выполнения, а не объявляются статически, как в большинстве компилируемых языков программирования.

Поддержка отступов в коде
: В отличие от других языков программирования в Python структура программы (группировка операторов) задается отступами, что позволяет не использовать фигурные скобки, точки с запятой и другие разделители.

Сбор мусора
: В Python реализован автоматический механизм сбора мусора, что избавляет программиста от необходимости заниматься управлением памятью.

Точнее всего философия языка Python изложена в следующих 20 положениях, известных как "The Zen of Python". Для ознакомления с ними достаточно выполнить команду `import this` в любой интерактивной оболочке Python.

```
In [1]: import this
        The Zen of Python, by Tim Peters

        Beautiful is better than ugly.
        Explicit is better than implicit.
        Simple is better than complex.
        Complex is better than complicated.
        Flat is better than nested.
        Sparse is better than dense.
        Readability counts.
        Special cases aren't special enough to break the rules.
        Although practicality beats purity.
        Errors should never pass silently.
        Unless explicitly silenced.
        In the face of ambiguity, refuse the temptation to guess.
        There should be one -- and preferably only one -- obvious
            way to do it.
        Although that way may not be obvious at first unless
            you're Dutch.
        Now is better than never.
        Although never is often better than *right* now.
        If the implementation is hard to explain, it's a bad idea.
        If the implementation is easy to explain, it may be
            a good idea.
        Namespaces are one honking great idea -- let's do more
            of those!
```

В переводе на русский язык они звучат так.

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.

- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и желательно только один — очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.
- Пространства имен — отличная вещь! Давайте будем делать их больше!

Краткая история Python

Некоторые до сих пор считают Python новым языком, хотя он существует уже достаточно давно. Его разработка началась в 80-х годах XX века программистом из Нидерландов Гвидо ван Россумом. Он длительное время участвовал в развитии языка на правах *великодушного пожизненного диктатора* (Benevolent Dictator For Life — BDFL) — почетный титул, который ему присвоило сообщество Python. Только в июле 2018 года, после нескольких десятилетий активной работы, Гвидо ван Россум принял решение оставить свой почетный пост BDFL и стать рядовым разработчиком.

Ниже перечислены основные вехи развития Python:

- Python 0.9.0, 1991 год (первый релиз);
- Python 1.0, 1994 год;
- Python 2.0, 2000 год;
- Python 2.6, 2008 год;
- Python 3.0, 2008 год;
- Python 3.1, 2009 год;
- Python 2.7, 2010 год;
- Python 3.2, 2011 год;
- Python 3.3, 2012 год;

- Python 3.4, 2014 год;
- Python 3.5, 2015 год;
- Python 3.6, 2016 год;
- Python 3.7, 2018 год;
- Python 3.8, 2019 год;

Новичков часто сбивает с толку тот факт, что начиная с 2008 года актуальными остаются сразу две версии Python, которые существуют параллельно. Это связано с наличием огромного количества программ, написанных на Python 2.6/2.7, которые продолжают активно использоваться. В то же время новые проекты пишутся на Python 3.6–3.8. Примеры, приводимые в книге, ориентированы на версию 3.7.

Экосистема Python

Python — это не только язык программирования, но и целая экосистема, включающая большое количество библиотек и различных программных инструментов. Их необходимо *импортировать* по мере необходимости (как, например, библиотеку построения диаграмм) или запускать в виде отдельного системного процесса (как, например, интерактивную среду разработки). Операция импорта делает пакет доступным в текущем пространстве имен и в текущем процессе интерпретатора.

В базовый дистрибутив Python уже входит большой набор пакетов и модулей, расширяющих функциональность интерпретатора. Этот набор называется *стандартная библиотека Python* (`https://docs.python.org/3/library/index.html`). Например, общие математические вычисления не требуют подключения каких-либо модулей, тогда как специализированные математические функции импортируются через модуль `math`.

```
In [2]: 100 * 2.5 + 50
Out[2]: 300.0

In [3]: log(1)  ❶

        ---------------------------------------------------------
        NameError                 Traceback (most recent call last)
        <ipython-input-3-74f22a2fd43b> in <module>
        ----> 1 log(1)  ❶

        NameError: name 'log' is not defined
```

```
In [4]: import math  ❷

In [5]: math.log(1)  ❷
Out[5]: 0.0
```

❶ Без импорта соответствующего модуля эта строка вызовет ошибку.

❷ После импорта модуля `math` математические вычисления выполняются корректно.

Модуль `math` — это стандартный компонент, доступный в любом дистрибутиве Python, но есть множество других пакетов, которые устанавливаются опционально, после чего применяются так же, как и стандартные модули. Такие пакеты доступны на специализированных сайтах в Интернете. Для работы с ними рекомендуется использовать менеджер пакетов (он будет рассмотрен в главе 2), который гарантирует совместимость всех библиотек.

Приведенные примеры ориентированы на две самые популярные среды: IPython (`http://ipython.org/`) и Jupyter (`https://jupyter.org/`). Проект IPython задумывался как улучшенная интерактивная оболочка Python, но со временем превратился в *интегрированную среду разработки* (Integrated Development Environment — IDE) с поддержкой таких средств, как профилирование и отладка. Расширенные инструменты редактирования кода доступны через внешние редакторы типа Vim (`http://vim.org`), которые можно интегрировать в IPython.

IPython существенно расширяет возможности стандартной интерактивной оболочки. Среди ключевых дополнений — улучшенные средства работы с историей командной строки и поддержка инспектирования объектов. Например, чтобы просмотреть справку по функции (`docstring`), достаточно предварить или завершить ее имя знаком вопроса (добавление двух знаков вопроса позволит получить расширенную справку).

Первоначально IPython поставлялся в двух вариантах: как оболочка и как браузерная версия (*Notebook*). Последняя приобрела настолько большую популярность, что была выделена в отдельный многоязыковой проект, получивший название Jupyter. Как следствие, среда Jupyter Notebook наследует все самое лучшее от IPython, дополняя ее множеством расширенных инструментов, особенно в части визуализации данных.

Круг пользователей Python

Неправильно полагать, будто Python интересен только профессиональным разработчикам. Это универсальный язык, применяемый в самых разных областях, в том числе для научных расчетов.

Профессиональные разработчики находят в Python все то, что необходимо для создания крупномасштабных приложений. В Python поддерживаются практически все основные парадигмы программирования и имеются средства для решения любых задач. Профессионалы часто пишут собственные библиотеки и классы и по максимуму задействуют весь стек научных пакетов.

Разработчики специализированных и научных приложений, как правило, активно пользуются конкретными пакетами и библиотеками, адаптируют экосистему под собственные задачи и постоянно дорабатывают и оптимизируют создаваемые программы. Они подолгу экспериментируют в рамках интерактивных сеансов, исследуя и визуализируя имеющиеся наборы данных.

Рядовые программисты обращаются к Python для решения конкретных задач, когда заранее известно, что имеются готовые наработки. Например, программист может посетить страницу с галереей визуализаций библиотеки `matplotlib`, скопировать оттуда готовый фрагмент кода и вставить его в собственный проект, адаптировав к условиям задачи. Это сильно ускоряет процесс разработки.

Есть еще одна многочисленная категория пользователей Python: *начинающие программисты*. На сегодняшний день Python стал одним из самых популярных языков, с которого начинают изучать программирование в университетах, колледжах и даже школах. Причиной служит простой и понятный синтаксис языка, доступный даже для неспециалистов, а также поддержка самых разных стилей программирования.

Стек научных пакетов

Существует целый набор специализированных пакетов, условно объединяемых в так называемый *научный стек*. Рассмотрим основные компоненты данного стека.

NumPy
: Библиотека, обеспечивающая поддержку многомерных массивов, предназначенных для хранения однородных и неоднородных данных. Включает оптимизированные функции/методы для работы с такими массивами.

SciPy
: Коллекция пакетов, реализующих функции для научных и финансовых расчетов. В частности, включает функции интерполяции кубических сплайнов и численного интегрирования.

`matplotlib`
: Один из самых популярных пакетов визуализации данных и построения как двухмерных, так и трехмерных диаграмм на Python.

`pandas`
: Основанная на NumPy библиотека, предназначенная для обработки и анализа временны*х* рядов и табличных данных. Тесно интегрирована с пакетами `matplotlib` (визуализация данных) и PyTables (хранение и загрузка данных).

Scikit-learn
: Популярный пакет машинного обучения, содержащий унифицированные реализации множества алгоритмов, таких как классификация и кластеризация.

PyTables
: Популярная надстройка для работы с файлами формата HDF5. Реализует оптимизированные операции дискового ввода-вывода.

В зависимости от конкретной задачи могут потребоваться и другие пакеты, которые зачастую реализуются поверх одного из вышеперечисленных базовых пакетов. Но в целом ключевыми структурными элементами выступают класс `ndarray` из пакета NumPy (глава 4) и класс `DataFrame` из пакета `pandas` (глава 5).

Если рассматривать Python только как язык программирования, то необходимо сказать, что есть и другие языки, которые могут составить ему конкуренцию в плане синтаксиса и эффективности, например Ruby. На официальном сайте (`www.ruby-lang.org`) он характеризуется следующим образом.

> Динамический язык программирования с открытым исходным кодом, для которого характерны простота и высокая производительность. Обладает элегантным синтаксисом, благодаря которому код легко читать и так же легко писать.

Те, кто программировали на Python, наверняка согласятся с тем, что аналогичным образом можно охарактеризовать и этот язык. Но что отличает Python от конкурентов наподобие Ruby, так это наличие научного стека, благодаря которому можно отказаться от специализированных языков и пакетов

типа Matlab и R. Кроме того, в языке по умолчанию имеется все необходимое, к примеру, для опытного веб-разработчика или системного администратора. К тому же Python отлично взаимодействует со специализированными языками типа R, поэтому вопрос обычно не в том, выбирать Python или нет, а в том, какой из языков разработки будет основным в конкретном проекте.

Технологии в финансовой отрасли

Получив общее представление о возможностях Python, поговорим о роли современных технологий в финансовом анализе. Нас в первую очередь будут интересовать то, какое влияние Python оказывает и, что самое важное, будет оказывать на развитие финансовой индустрии.

По сути, в применении передовых технологий в финансовой отрасли нет ничего необычного. Однако недавние инновации, а также законодательные изменения привели к тому, что банки и различные финансовые учреждения, например хедж-фонды, все больше стали превращаться из финансовых посредников в технологические компании. Технологическая база начала рассматриваться как один из ключевых активов для большинства финансовых компаний по всему миру, что дает как преимущества, так и недостатки. Следует разобраться, почему так происходит.

Инвестиции в технологии

Банки и другие финансовые учреждения образуют целую индустрию, которая ежегодно тратит огромные средства на передовые технологии. Следующая цитата показывает не только важность технологий для финансовой отрасли, но и важность самой этой отрасли для технологического сектора.

> Согласно аналитическим данным, собранным компанией IDC (International Data Corporation), к 2021 году общемировые затраты финансового сектора на информационные технологии вырастут до 500 млрд долл. по сравнению с 440 млрд долл. в 2018 году.
>
> IDC (июнь 2018 г.)

Банки и финансовые учреждения уже давно втянуты в гонку, направленную на переход к цифровой модели управления.

> Согласно прогнозам в 2017 году банковские инвестиции в передовые технологии в Северной Америке должны составить около 19,9 млрд долл.

> Банки разрабатывают современные информационные системы и новые технологические решения с целью получения конкурентных преимуществ на глобальном рынке и привлечения клиентов, ориентированных на использование онлайн-служб и мобильных приложений. Для глобальных финансово-технологических компаний это прекрасная возможность предложить новые идеи и программные решения банковской индустрии.
>
> Statista

Транснациональные банки задействуют тысячи разработчиков для обслуживания существующих систем и создания новых. Крупные инвестиционные компании, сталкиваясь с ростом технологических требований, зачастую имеют бюджет IT-направления в несколько миллиардов долларов.

Технологии как движущая сила

Развитие технологий способствует внедрению инноваций и повышению эффективности финансового сектора. Как правило, проекты в данной сфере следуют концепции полного перехода на цифровые модели.

> За последние несколько лет финансовая индустрия претерпела радикальные технологические изменения. Многие руководители требуют от своих IT-отделов повышать эффективность и внедрять качественно новые, инновационные услуги, при этом поддерживая существующие системы, но снижая затраты на их эксплуатацию. А тем временем на рынок все активнее выходят финтех-стартапы, предлагая пользовательские системы, разработанные с нуля, а не основанные на унаследованных решениях.
>
> PwC 19th Annual Global CEO Survey, 2016[1]

Побочным эффектом повышения эффективности становится то, что конкурентные преимущества приходится искать во все более сложных решениях. Это неизбежно влечет за собой рост рисков и затрат, связанных с соблюдением регуляторных требований. Финансовый кризис 2007–2008 годов показал, какую опасность могут нести новые технологии. С аналогичными технологическими рисками сталкивается и финансовый сектор, что нагляднее всего проявилось в обвале фондового рынка, который имел место в мае 2010 года, когда вследствие применения автоматизированных биржевых систем обрушение индексов достигло триллионного масштаба. (Вопросы, связанные с алгоритмической торговлей финансовыми инструментами, будут рассматриваться в части IV.)

[1] `https://pwc.to/10YTO2d`.

Технологии и кадры решают все

Как видим, с одной стороны, развитие технологий ведет к снижению затрат, а с другой — только постоянные инвестиции в разработку и внедрение новых решений позволяют финансовым компаниям добиваться конкурентных преимуществ и укреплять свои позиции на рынке. Во многих областях крупномасштабные инвестиции в технологии и квалифицированный персонал служат ключом к выживанию на рынке. В качестве примера рассмотрим рынок анализа деривативов.

> В среднем, если рассматривать весь цикл существования программного обеспечения, фирмы, внедряющие собственные стратегии внебиржевого управления деривативами, должны вложить от 25 до 36 млн долларов только на создание, поддержку и улучшение соответствующей программной библиотеки.
>
> Дин [1][2]

Построение полнофункциональной библиотеки анализа деривативов — не только трудоемкий и затратный процесс, для него еще и требуется наличие *достаточного числа экспертов*. К тому же сами эксперты должны располагать всеми необходимыми технологиями и программными средствами. В этом отношении современная экосистема Python выглядит очень привлекательно, снабжая разработчиков намного более эффективными и дешевыми инструментами, чем, к примеру, 10 лет назад. (Тема анализа деривативов рассматривается в части V, где мы напишем небольшую, но достаточно эффективную и гибкую библиотеку, задействуя только стандартные пакеты Python.)

Следующая цитата возвращает нас в эпоху LTCM — одного их крупнейших хедж-фондов, история которого завершилась крахом в конце 1990-х.

> Мериуэзер вложил 20 млн долл. в разработку передовой компьютерной системы и нанял команду первоклассных финансовых инженеров для управления компанией LTCM, офис которой находился в Гринуиче, штат Коннектикут. Это была система управления рисками промышленного масштаба.
>
> Паттерсон [3]

Те вычислительные мощности, на которые Мериуэзер потратил миллионы долларов, сегодня можно купить всего за несколько тысяч долларов или поступить еще проще, арендовав их в одной из облачных служб в рамках гибкого плана. (О развертывании облачной инфраструктуры для интерактивного финансового анализа и разработки приложений на Python будет рассказываться в главе 2.) Бюджеты подобной профессиональной инфраструктуры составляют

[2] См. раздел “Дополнительные ресурсы” в конце главы.

от нескольких долларов в месяц. В то же время сами системы биржевой торговли, финансового анализа и управления рисками стали настолько сложными, что современным компаниям приходится внедрять IT-инфраструктуры, насчитывающие десятки тысяч вычислительных ядер.

В погоне за скоростью, производительностью и объемами данных

Если говорить о тех аспектах финансовых вычислений, на которые технологические изменения оказывают наибольшее влияние, то это скорость и интенсивность финансовых транзакций. Льюис [2] описывает понятие *флеш-трейдинга* — процесса высокочастотной биржевой торговли, когда сделки совершаются с максимально возможной скоростью.

С одной стороны, повышение доступности биржевых данных требует от финансовых компаний принятия решений в реальном времени. А с другой — внедрение высокочастотной торговли приводит к еще большему росту объемов данных. Оба процесса идут рука об руку, непрерывно снижая временной масштаб финансовых транзакций. Впервые эта тенденция была выявлена еще десять лет назад.

> В 2008 году фонд Medallion, основанный компанией Renaissance Technologies, получил небывалую маржу в размере 80%, сумев воспользоваться крайней волатильностью рынка благодаря своим сверхбыстрым компьютерам. Крупнейшим бенефициарием в тот год стал Джим Симонс, который положил в карман солидные 2,5 млрд долл.
>
> Паттерсон [3]

Данные о биржевых котировках одного индекса на конец торгов, собранные за 30 лет, содержатся примерно в 7500 записях. Именно на этих данных базируются все современные финансовые теории, в частности *портфельная теория Марковица* (Mean-variance Portfolio Theory — MPT), *модель ценообразования капитальных активов* (Capital Asset Pricing Model — CAPM) и *стоимость под риском* (value-at-risk — VaR).

Для сравнения: в обычный торговый день курс акций компании Apple (AAPL) в среднем изменяется около 15 тыс. раз в час, что в два раза больше, чем количество котировок, собранных за последних 30 лет. Анализ таких объемов данных сопряжен с целым рядом трудностей.

Обработка данных

Сегодня уже недостаточно обрабатывать только сведения о котировках акций и других финансовых инструментов на момент закрытия торгов.

Слишком много событий происходит в течение каждого торгового дня, а для некоторых инструментов — 24 часа в сутки 7 дней в неделю.

Скорость анализа данных
: Биржевые решения должны приниматься за миллисекунды или даже быстрее, а это означает, что нужны соответствующие аналитические системы, способные обрабатывать огромные объемы данных в реальном времени.

Теоретическая база
: Несмотря на то что традиционные финансовые теории далеки от идеала, они прошли испытание временем. Но когда речь идет о принятии решений в милли- и микросекундных интервалах, надежных теорий пока еще недостаточно.

Указанные трудности можно преодолеть только с помощью современных технологий. Самое удивительное то, что даже отсутствие надежных финансовых теорий часто компенсируется технологическими преимуществами, поскольку алгоритмы высокочастотной торговли задействуют микроструктурные биржевые элементы (поток заявок, спреды и т.п.), а не опираются на теоретические суждения.

Анализ в реальном времени

За последние годы в финансовой сфере резко возросла важность *финансового анализа*. Это прямое следствие повышения скорости и производительности вычислений, а также увеличения объемов данных. По сути, реакцией рынка стало появление анализа в реальном времени.

Под финансовым анализом мы понимаем применение современных технологий, программных решений и алгоритмов для получения аналитических данных или принятия решений. В качестве примера можно привести оценку того, какое влияние на продажи окажет изменение стоимости банковского продукта, или крупномасштабную корректировку кредитной оценки (Credit Valuation Adjustment — CVA) для сложных портфелей деривативов крупного инвестиционного банка.

В этом контексте финансовые учреждения сталкиваются с двумя основными трудностями.

Большие данные
: Банкам и другим финансовым компаниям приходилось иметь дело с огромными массивами данных еще до начала эпохи “больших данных”.

Как бы там ни было, сегодня объемы анализируемых данных настолько велики, что требуют постоянного увеличения как вычислительной мощности компьютерных систем, так и размера хранилищ данных.

Расчеты в реальном времени
: В прошлом инвестиционные решения принимались на основе тщательно спланированных стратегий и процессов управления рисками. В нынешнюю эпоху решения приходится принимать в реальном времени. Задачи, которые раньше решались в офисе на протяжении рабочего дня, сегодня решаются прямо на бирже в процессе торгов.

Здесь снова прослеживается связь между развитием технологий и практикой ведения бизнеса. С одной стороны, существует постоянная потребность в повышении скорости и точности аналитических решений за счет внедрения современных технологий. А с другой стороны, появление новых технологий делает возможными новые аналитические подходы, которые всего несколько лет назад считались невозможными (или нереалистичными исходя из бюджетных соображений).

Одной из ключевых тенденций в сфере финансового анализа стало внедрение параллельных архитектур многоядерных вычислений на стороне центрального процессора и массово-параллельных архитектур на основе графических процессоров. Современные графические процессоры содержат тысячи вычислительных ядер, что заставляет переосмысливать алгоритмы параллельных вычислений. Но для того чтобы извлечь преимущества из новых аппаратных платформ, пользователям приходится осваивать новые языки и парадигмы программирования.

Python для финансовых расчетов

В предыдущем разделе рассматривались основные аспекты применения современных технологий в финансовой среде:

- стоимость технологий для финансовой индустрии;
- технологии как движущая сила бизнеса и инноваций;
- технологии и квалифицированные кадры как барьеры для выхода на финансовый рынок;
- увеличение скорости и интенсивности вычислений, а также объемов данных;
- необходимость анализа данных в реальном времени.

В этом разделе вы узнаете о том, как Python помогает преодолевать возникающие проблемы. Но для начала следует познакомиться с общеязыковыми инструментами и синтаксисом Python.

Синтаксис Python, применяемый в финансовых вычислениях

Большинство разработчиков, которые только начинают знакомиться с инструментами Python, применяемыми в финансовых расчетах, неизбежно сталкиваются с алгоритмическими проблемами. То же самое будет испытывать ученый, которому нужно решить дифференциальное уравнение, вычислить интеграл или просто визуализировать данные. На этом этапе еще никто не думает о формальном процессе разработки, тестировании, документации или развертывании готового приложения. И именно здесь люди начинают ценить Python как язык программирования. Причина заключается в том, что синтаксис Python в достаточной степени приближен к математическому синтаксису, применяемому для решения научных и финансовых задач.

В качестве иллюстрации можно взять любой финансовый алгоритм, например оценку европейского колл-опциона методом Монте-Карло. Предположим, применяется модель Блэка — Шоулза — Мертона (Black–Scholes–Merton — BSM), в которой рисковость опциона описывается через геометрическое броуновское движение.

Примем следующие значения параметров:

- начальный уровень индекса, $S_0 = 100$;
- страйк-цена опциона, $K = 105$;
- срок исполнения опциона, $T = 1$ год;
- безрисковая краткосрочная ставка, $r = 0{,}05$;
- волатильность доходности актива, $\sigma = 0{,}2$.

В модели Блэка — Шоулза — Мертона уровень индекса при экспирации опциона представляется случайной величиной, рассчитываемой по следующей формуле, в которой случайная переменная z имеет нормальное распределение вероятностей (уравнение 1.1).

Уравнение 1.1. Уровень индекса при экспирации опциона в модели Блэка — Шоулза — Мертона

$$S_T = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}z\right).$$

Ниже приведено алгоритмическое описание процедуры оценки опциона по методу Монте-Карло.

1. Образуйте множество I псевдослучайных чисел $z(i),\ i \in \{1,\ 2,\ \ldots,\ I\}$ из стандартного нормального распределения.
2. Вычислите уровень индекса при экспирации опциона $S_T(i)$ для всех заданных $z(i)$, воспользовавшись формулой из уравнения 1.1.
3. Вычислите внутренние стоимости по следующему уравнению:

$$h_T(i) = \max(S_T(i) - K,\ 0)\,.$$

4. Определите текущую стоимость опциона по методу Монте-Карло (уравнение 1.2).

Уравнение 1.2. Расчет стоимости европейского опциона по методу Монте-Карло

$$C_0 \approx e^{-rT}\,\frac{1}{I}\sum_I h_T(i).$$

Следующим этапом будет перевод алгоритма на язык Python. Решение выглядит следующим образом.

```
In [6]: import math
        import numpy as np  ❶

In [7]: S0 = 100.  ❷
        K = 105.  ❷
        T = 1.0  ❷
        r = 0.05  ❷
        sigma = 0.2  ❷

In [8]: I = 100000  ❷

In [9]: np.random.seed(1000)  ❸

In [10]: z = np.random.standard_normal(I)  ❹

In [11]: ST = S0 * np.exp((r - sigma ** 2 / 2) * T + sigma *
                          math.sqrt(T) * z)  ❺

In [12]: hT = np.maximum(ST - K, 0)  ❻
```

```
In [13]: C0 = math.exp(-r * T) * np.mean(hT)  ❼

In [14]: print('Стоимость европейского колл-опциона:
                {:5.3f}.'.format(C0))  ❽
         Стоимость европейского колл-опциона: 8.019.
```

❶ Финансовые расчеты в данном случае выполняются средствами пакета NumPy.

❷ Задаем параметры модели.

❸ Фиксируем затравочное значение для генератора случайных чисел.

❹ Формируем множество случайных чисел со стандартным распределением вероятности.

❺ Вычисляем цену актива при экспирации опциона.

❻ Вычисляем премию по опциону.

❼ Оценка опциона по методу Монте-Карло.

❽ Выводим результат оценки на экран.

Рассмотрим ключевые моменты.

Понятный синтаксис
: Синтаксис Python действительно схож с математическим языком, что особенно заметно при задании параметров модели.

Трансляция задачи
: Каждое математическое уравнение и каждый шаг алгоритма представляется в коде Python *одной* инструкцией.

Векторизация
: Одно из достоинств пакета NumPy — компактный векторизованный синтаксис, что позволяет выполнить, к примеру, 100 000 вычислений с помощью одной команды.

Приведенный выше код можно легко выполнить в интерактивной среде, такой как IPython или Jupyter Notebook. Но если код планируется выполнять на регулярной основе, его следует сохранить в виде *модуля* (сценария), представляющего собой отдельный файл Python (с технической точки зрения простой текстовый документ) с расширением *.py*. В данном случае модуль можно сохранить в файле *bsm_mcs_euro.py* (листинг 1.1).

Листинг 1.1. Оценка европейского колл-опциона по методу Монте-Карло

```
#
# Оценка европейского колл-опциона
# (модель Блэка – Шоулза – Мертона)
# bsm_mcs_euro.py
#
# Python for Finance, 2nd ed.
# (c) Dr. Yves J. Hilpisch
#
import math
import numpy as np

# Значения параметров
S0 = 100.     # начальный уровень индекса
K = 105.      # страйк-цена опциона
T = 1.0       # срок исполнения опциона
r = 0.05      # безрисковая краткосрочная ставка
sigma = 0.2   # волатильность

I = 100000    # количество этапов моделирования

# Алгоритм оценки
z = np.random.standard_normal(I)  # генерация псевдослучайных чисел

# Цена актива при экспирации опциона
ST = S0 * np.exp((r - 0.5 * sigma ** 2) * T + sigma *
     math.sqrt(T) * z)
hT = np.maximum(ST - K, 0)           # премия по опциону
C0 = math.exp(-r * T) * np.mean(hT)  # оценка по методу Монте-Карло

# Вывод результата
print('Стоимость европейского колл-опциона: %5.3f.' % C0)
```

Данный алгоритмический пример наглядно демонстрирует, насколько удобен синтаксис Python для выполнения научных расчетов. Он позволяет легко перейти от постановки задачи и ее математического описания к программной реализации. Порядок действий выглядит так.

- **Обычный язык.** Позволяет в *устном и письменном виде* выразить суть научной или финансовой задачи.
- **Математический язык.** Позволяет максимально *четко и лаконично описать и смоделировать* абстрактные аспекты задачи, алгоритм ее решения, числовые параметры и т.п.

- **Python.** Позволяет *технически смоделировать и реализовать* абстрактные аспекты задачи, алгоритм ее решения, числовые параметры и т.п.

Математические аспекты синтаксиса Python

Вряд ли вам удастся найти другой язык программирования, настолько близкий синтаксически к языку математики, как Python. В большинстве случаев математические выражения и формулы транслируются в Python почти в неизменном виде, что делает его невероятно эффективным языком разработки и написания приложений для финансовых расчетов.

Нередко в применяемый стек языков добавляется четвертый компонент: *псевдокод*. Его роль заключается в представлении математических формул, лежащих в основе финансовых расчетов, в наиболее понятном для программной реализации виде. Помимо самого алгоритма, псевдокод должен учитывать особенности обработки данных компьютером.

Практика применения псевдокода возникла вследствие огромного различия между формальным математическим представлением задачи и его описанием в большинстве компилируемых языков программирования. Очень часто написанный на таких языках код настолько усложняется добавлением обязательных технических элементов, что его затруднительно сопоставить с исходными математическими выкладками.

К счастью, при работе с Python отдельный этап составления псевдокода не нужен, поскольку синтаксис языка почти аналогичен языку математики, а чисто технические дополнения сведены к минимуму. Для этого в Python предусмотрено множество высокоуровневых конструкций. Конечно, у подобного подхода есть и недостатки, но в целом можно сказать, что всякий раз, когда возникает такая необходимость, в Python можно следовать строгим процедурам кодирования, как и в любых других языках программирования. В этом смысле Python берет все лучшее из двух миров: *высокоуровневую абстракцию* и *строгую реализацию*.

Эффективность и производительность кода Python

Если рассматривать ситуацию на высоком уровне, то преимущества Python можно оценить по следующим аспектам.

Эффективность
: Поможет ли Python получить результаты быстрее и дешевле?

Производительность
: Позволит ли Python достичь большего при тех же самых ресурсах (люди, средства и т.п.)?

Качество
: Что такого позволяет делать Python, чего не могут дать другие технологии?

Навряд ли на такие вопросы можно дать исчерпывающие ответы, но все же приведем ряд аргументов.

Высокая скорость вычислений

Область, в которой эффективность Python становится очевидной, — это интерактивный анализ данных. Именно здесь проявляются преимущества таких мощных инструментов, как IPython, Jupyter Notebook и программных пакетов наподобие `pandas`.

Рассмотрим студента экономического факультета, который пишет диплом, исследуя индекс S&P 500. Ему необходимо проанализировать значения индекса за несколько лет и попытаться найти доказательства того, что его волатильность, в противоположность принятым во многих финансовых моделях предположениям, изменяется со временем, а потому ее нельзя выражать константой. Результаты следует представить в графическом виде. Для решения такой задачи нужно выполнить следующие действия:

- загрузить данные о значениях индекса S&P 500 из Интернета;
- вычислить среднеквадратическое отклонение логарифмической доходности (волатильности) в годовом разрезе;
- построить графики изменения индекса S&P 500 и волатильности.

Еще совсем недавно такого рода задачи считались настолько сложными, что их решение было под силу только профессиональным финансовым аналитикам. Сегодня же с ней легко справится даже студент экономического факультета. Ниже приведен соответствующий код (на синтаксис можете пока не обращать внимания — мы все детально рассмотрим в последующих главах).

```
In [16]: import numpy as np  ❶
         import pandas as pd  ❶
         from pylab import plt, mpl  ❷

In [17]: plt.style.use('seaborn')  ❷
         mpl.rcParams['font.family'] = 'serif'  ❷
         %matplotlib inline
```

```
In [18]: data = pd.read_csv('../../source/tr_eikon_eod_data.csv',
                            index_col=0, parse_dates=True)  ❸
         data = pd.DataFrame(data['.SPX'])  ❹
         data.dropna(inplace=True)  ❹
         data.info()  ❺
         <class 'pandas.core.frame.DataFrame'>
         DatetimeIndex: 2138 entries, 2010-01-04 to 2018-06-29
         Data columns (total 1 columns):
         .SPX    2138 non-null float64
         dtypes: float64(1)
         memory usage: 33.4 KB

In [19]: data['Доходность'] = np.log(data / data.shift(1))  ❻
         data['Волатильность'] =
             data['Доходность'].rolling(252).std() * np.sqrt(252)  ❼

In [20]: data[['.SPX', 'Волатильность']].plot(subplots=True,
                                              figsize=(10, 6));  ❽
```

❶ Импорт пакетов NumPy и `pandas`.

❷ Импорт библиотеки `matplotlib` и настройка стиля диаграмм, отображаемых в среде Jupyter.

❸ Метод `pd.read_csv()` позволяет загрузить данные из CSV-файла, хранящегося локально или на сервере.

❹ Формирование подмножества данных и исключение нечисловых значений (`NaN`).

❺ Вывод метаинформации о выбранном наборе данных.

❻ Вычисление логарифмической доходности в векторизованной форме (без применения циклов).

❼ Определение среднегодовой волатильности.

❽ Визуализация двух временны*х* рядов.

Результат данного интерактивного сеанса представлен на рис. 1.1. Просто поразительно, как с помощью всего нескольких строк кода нам удалось решить три сложные задачи финансового анализа: сбор данных, выполнение повторяющихся математических вычислений и визуализация результатов. Обратите внимание на то, что благодаря использованию пакета `pandas` с

временны*ми* рядами можно работать почти так же легко, как и с вещественными числами.

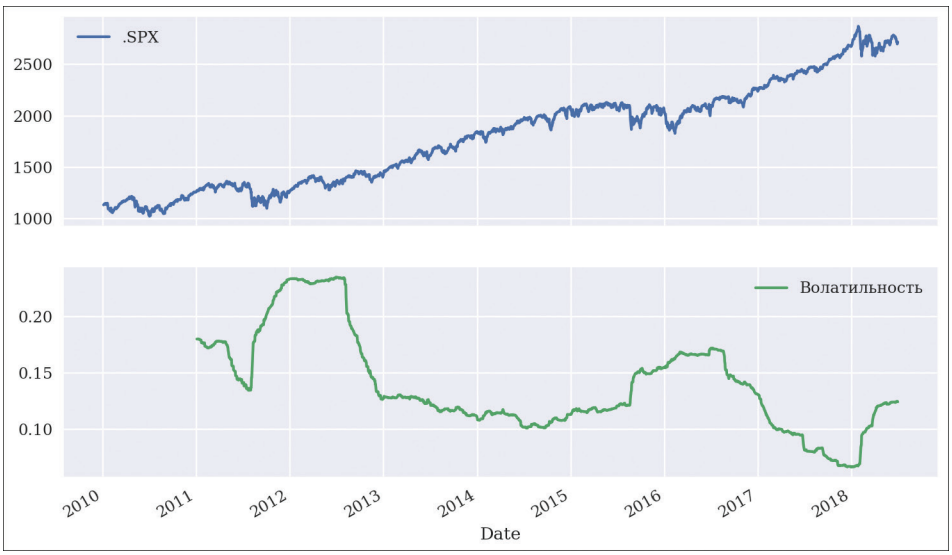

Рис. 1.1. *Уровни индекса S&P 500 и среднегодовая волатильность*

Как показывает данный пример, при правильном выборе инструментов и пакетов Python, предлагающих соответствующие высокоуровневые абстракции, финансовый аналитик получает возможность сконцентрироваться на решении конкретной профессиональной задачи, а не на технических аспектах программной реализации. Это позволяет аналитику быстрее реагировать на изменения рыночных условий, делая оценки практически в режиме реального времени и тем самым опережая своих конкурентов. Таким образом, *увеличение эффективности* может легко давать коммерческий эффект.

Повышение производительности

Как правило, критики признают, что Python обладает понятным синтаксисом, позволяющим писать эффективный код. Однако по своей природе это интерпретируемый язык, а потому существует предубеждение, будто бы код Python слишком медленный для выполнения трудоемких финансовых расчетов. В определенных ситуациях это действительно так, но в реальности все зависит от выбранного подхода. Никто не мешает вам писать эффективные приложения! Можно выделить три стратегии повышения производительности.

Идиомы и парадигмы программирования

Python позволяет решать одни и те же задачи разными способами, которые характеризуются разной производительностью. Выбрав правильный способ реализации (т.е. разумно используя структуры данных, избегая циклов в рамках векторизации и применяя пакеты наподобие `pandas`), можно существенно повысить скорость любых финансовых вычислений.

Компиляция кода

Существуют специальные пакеты повышения производительности, содержащие скомпилированные версии важных функций или выполняющие статическую либо динамическую компиляцию кода Python, что на порядок ускоряет выполнение важных функций в сравнении с чистым кодом Python. Самые популярные пакеты такого рода — Cython и Numba.

Параллельные вычисления

Многие вычислительные процессы, особенно в финансовой области, можно существенно ускорить за счет параллельного выполнения. В Python это можно сделать очень легко.

Высокопроизводительные вычисления на Python

Сам по себе Python нельзя назвать высокопроизводительным языком. Но он превратился в идеальную платформу для доступа к производительным технологиям, что делает его своего рода связующим языком.

Рассмотрим простой, но реалистичный пример, в котором задействуются все три вышеуказанные стратегии (подробнее они рассматриваются в последующих главах). В финансовом анализе часто возникает задача оценки сложных математических выражений применительно к большим числовым массивам. В Python для этого есть собственные инструменты.

```
In [21]: import math
         loops = 2500000
         a = range(1, loops)
         def f(x):
             return 3 * math.log(x) + math.cos(x) ** 2
         %timeit r = [f(x) for x in a]
         1.59 s ± 41.2 ms per loop (mean ± std. dev. of 7 runs,
                                    1 loop each)
```

Как видите, на выполнение функции `f()` 2,5 млн раз интерпретатору Python потребовалось примерно 1,6 с. Ту же самую задачу можно решить с помощью *оптимизированных* (т.е. предварительно откомпилированных) функций пакета NumPy.

```
In [22]: import numpy as np
         a = np.arange(1, loops)
         %timeit r = 3 * np.log(a) + np.cos(a) ** 2
         87.9 ms ± 1.73 ms per loop (mean ± std. dev. of 7 runs,
                                     10 loops each)
```

В результате время выполнения кода сократилось до 88 мс. Впрочем, в Python есть специальный программный пакет для решения таких задач: `numexpr`. Он *компилирует* все выражение, чтобы добиться еще большей, чем позволяет NumPy, производительности. Это, в частности, достигается за счет предотвращения хранения в памяти множественных копий объектов `ndarray`.

```
In [23]: import numexpr as ne
         ne.set_num_threads(1)
         f = '3 * log(a) + cos(a) ** 2'
         %timeit r = ne.evaluate(f)
         50.6 ms ± 4.2 ms per loop (mean ± std. dev. of 7 runs,
                                    10 loops each)
```

Данный подход позволяет сократить время обработки данных еще сильнее — до 50 мс. Наряду с этим пакет `numexpr` располагает средствами распараллеливания вычислений, что позволяет задействовать множественные потоки центрального процессора.

```
In [24]: ne.set_num_threads(4)
         %timeit r = ne.evaluate(f)
         22.8 ms ± 1.76 ms per loop (mean ± std. dev. of 7 runs,
                                     10 loops each)
```

Параллелизация дополнительно сокращает время выполнения кода до 23 мс за счет применения четырех потоков. Таким образом, проведенная нами оптимизация позволила повысить производительность более чем в 90 раз. Причем это стало возможным без изменения базового алгоритма и без знания алгоритмов компиляции или параллельных вычислений. Такого рода программные средства доступны на высоком уровне даже для пользователей, не являющихся экспертами в Python. Но, разумеется, необходимо знать, какие средства есть в вашем распоряжении.

Рассмотренный пример показывает, что Python располагает целым рядом средств, позволяющих извлечь максимум из имеющихся ресурсов. Тем самым

мы получаем *повышение производительности*. При параллельном подходе, в отличие от последовательного, за то же самое время можно выполнить в 6 раз больше вычислений. Для этого достаточно указать интерпретатору на необходимость использования всех потоков центрального процессора, а не только одного.

От прототипа к готовому приложению

Эффективность интерактивного анализа и высокая скорость вычислений — два ключевых преимущества Python. Но есть и еще одно, не очевидное на первый взгляд преимущество, которое может иметь стратегическое значение с точки зрения финансовых вычислений. Речь идет о возможности использовать Python на протяжении *всего производственного цикла*, от создания прототипа до выпуска готового приложения.

Практика современных финансовых компаний по всему миру, когда речь идет о разработке финансовых приложений, зачастую подразумевает двухэтапный процесс. На первом этапе к работе подключаются специалисты по *финансовой математике* ("кванты"), отвечающие за разработку модели и технического прототипа. Они предпочитают инструменты типа Matlab и R, удобные для быстрой интерактивной разработки приложений. На данной стадии вопросы производительности, стабильности, развертывания инфраструктуры, контроля доступа и управления версиями не имеют принципиального значения. Речь идет лишь о получении принципиально работоспособной модели и прототипа, соответствующего ключевым требованиям алгоритма или итогового приложения.

Как только прототип готов, в игру вступают *разработчики* из IT-отдела, которые отвечают за перевод кода прототипа в надежный, управляемый и производительный *производственный код*. Как правило, на этом этапе происходит сдвиг парадигмы, поскольку требованиям развертывания программной инфраструктуры соответствуют компилируемые языки, такие как C++ или Java. Кроме того, применяется формальный процесс разработки с задействованием профессиональных инструментов, систем управления версиями и пр.

Описанный двухэтапный процесс имеет ряд принципиальных недостатков.

Низкая эффективность

Код прототипа не подлежит повторному использованию, и алгоритмы приходится реализовывать дважды, что приводит к дополнительным тратам времени и ресурсов. Кроме того, при переписывании кода прототипа могут возникать ошибки.

Квалификация исполнителей
: Специалисты из разных отделов обладают разными навыками и применяют разные языки для разработки одного и того же. Более того, они даже выражают свои мысли по-разному.

Унаследованный код
: Полученный код приходится обслуживать на разных языках и платформах.

Python позволяет *упростить* весь цикл разработки, поскольку один набор инструментов применяется от первых интерактивных попыток получить рабочий прототип и до финального выпуска надежного и эффективного производственного кода. Упрощается не только взаимодействие специалистов из разных департаментов, но и обучение персонала, поскольку на всех этапах разработки финансового приложения задействуется один язык программирования. Устраняется также неизбежная избыточность и неэффективность, связанная с применением разных технологий на разных этапах процесса разработки. В целом Python предлагает *согласованную технологическую среду*, пригодную для решения любых задач финансового анализа, разработки финансовых приложений и реализации алгоритмов.

Финансовые расчеты на основе данных и искусственного интеллекта

Все те соображения касательно применения технологий в финансовой отрасли, которые были сформулированы в первом издании книги, вышедшем в 2014 году, справедливы и сейчас. Но в последние годы в финансовой сфере проявились две важнейшие тенденции, которые несут фундаментальные изменения.

Финансовые системы, управляемые данными

Некоторые фундаментальные финансовые теории, например *портфельная теория Марковица* (Modern Portfolio Theory — MPT) и *модель ценообразования капитальных активов* (Capital Asset Pricing Model — CAPM), разрабатывались еще в середине XX века. Они по-прежнему остаются краеугольным камнем образовательных программ, по которым студенты изучают экономику, финансы, финансовую инженерию и деловое администрирование. И это в определенной степени удивительно, так как их доказательная база достаточно

скудная, если не сказать больше, а имеющиеся наблюдения часто вступают в противоречие с тем, что утверждает та или иная теория. С другой стороны, они обладают очевидной популярностью, поскольку близки к нашему пониманию того, как должны работать финансовые рынки. Плюс это все-таки красивые математические теории, построенные на достаточно логичных, хоть и слишком упрощенных, рассуждениях.

Научный метод, применяемый, к примеру, в физике, базируется на *данных*, собираемых в ходе наблюдений и экспериментов. Только после этого строятся *гипотезы и теории*, которые затем снова проверяются на данных. Если научный тест подтверждает предположения ученых, то гипотеза или теория должным образом формализуется в виде диссертации либо академической публикации. Если же тест не дал результатов, то гипотеза или теория отвергается, и ученые начинают искать новые объяснения, которые не противоречили бы имеющимся данным. Поскольку законы физики фундаментальны и не меняются, открытие того или иного закона приводит к тому, что он становится частью наших знаний об окружающем мире на столетия вперед.

Финансовые теории исторически шли вразрез с научным методом. Зачастую их разрабатывали "от печки" на основе упрощенных математических моделей с целью поиска элегантных ответов на ключевые вопросы финансового анализа. Одно из популярных допущений — предположение о нормальном распределении доходности финансовых инструментов и линейной зависимости между процентными ставками. Но поскольку такого рода вещи редко наблюдаются на финансовых рынках, не удивительно, что эмпирических доказательств красивых теорий зачастую нет. Многие финансовые теории и модели сначала были сформулированы и описаны в научных изданиях и только впоследствии были проверены на реальных данных. В определенной степени это связано с тем, что в 1950–1970-е годы (и даже в следующие десятилетия) исследователи не располагали тем объемом финансовых данных, который сегодня есть даже у студентов-бакалавров.

Ситуация с доступностью финансовых данных радикально изменилась в середине 1990-х годов, и сегодня любой исследователь может получить доступ к огромным объемам исторических данных, а также к биржевым данным в реальном времени через потоковые службы. Это дает нам возможность вернуться к научному методу и начать с исследования данных, что всегда должно предшествовать появлению идей, гипотез, моделей и стратегий.

Следующий простой пример показывает, насколько легко в наши дни получить любые финансовые данные за интересующий период времени, используя только Python и подписку на программный интерфейс Eikon (`https://`

developers.refinitiv.com/eikon-apis/eikon-data-apis[]). Приведенный ниже код получает сведения о колебаниях курса акций компании Apple в течение одного часа дневных торгов. Всего загружается 15 000 записей, включая данные об объемах сделок. Стандартный биржевой тикер Apple — AAPL, но в классификации RIC (Reuters Instrument Code) он обозначается как AAPL.O.

```
In [26]: import eikon as ek  ❶

In [27]: data = ek.get_timeseries('AAPL.O', fields='*',
                                  start_date='2018-10-18 16:00:00',
                                  end_date='2018-10-18 17:00:00',
                                  interval='tick')  ❷

In [28]: data.info()  ❷
         <class 'pandas.core.frame.DataFrame'>
         DatetimeIndex: 35350 entries, 2018-10-18 16:00:00.002000 to
                                       2018-10-18 16:59:59.888000
         Data columns (total 2 columns):
         VALUE     35285 non-null float64
         VOLUME    35350 non-null float64
         dtypes: float64(2)
         memory usage: 828.5 KB

In [29]: data.tail()  ❸
Out[29]:                      AAPL.O    VALUE   VOLUME
                                Date
         2018-10-18 16:59:59.433   217.13     10.0
         2018-10-18 16:59:59.433   217.13     12.0
         2018-10-18 16:59:59.439   217.13    231.0
         2018-10-18 16:59:59.754   217.14    100.0
         2018-10-18 16:59:59.888   217.13    100.0
```

❶ Для работы с программным интерфейсом Eikon требуется подписка на службу и наличие подключения к серверу.

❷ Получение котировок акций Apple (AAPL.O).

❸ Вывод последних пяти записей из таблицы котировок.

Программный интерфейс Eikon позволяет получить доступ не только к структурированным финансовым данным, например к исторической информации о стоимости акций, но и к неструктурированным данным, таким как *новости*. В следующем примере показано, как получить метаданные

небольшой подборки новостей и вывести на экран начальный фрагмент одной из статей.

```
In [30]: news = ek.get_news_headlines('R:AAPL.O Language:LEN',
                                      date_from='2018-05-01',
                                      date_to='2018-06-29',
                                      count=7)  ❶

In [31]: news  ❶
Out[31]:
                                      versionCreated \
2018-06-28  23:00:00.000  2018-06-28  23:00:00.000
2018-06-28  21:23:26.526  2018-06-28  21:23:26.526
2018-06-28  19:48:32.627  2018-06-28  19:48:32.627
2018-06-28  17:33:10.306  2018-06-28  17:33:10.306
2018-06-28  17:33:07.033  2018-06-28  17:33:07.033
2018-06-28  17:31:44.960  2018-06-28  17:31:44.960
2018-06-28  17:00:00.000  2018-06-28  17:00:00.000

                                                                text \
2018-06-28 23:00:00.000 RPT-FOCUS-AI ambulances and robot doc...
2018-06-28 21:23:26.526 Why Investors Should Love Apple's (AA...
2018-06-28 19:48:32.627 Reuters Insider - Trump: We're reclai...
2018-06-28 17:33:10.306 Apple v. Samsung ends not with a whim...
2018-06-28 17:33:07.033 Apple's trade-war discount extended f...
2018-06-28 17:31:44.960 Other Products: Apple's fast-growing ...
2018-06-28 17:00:00.000 Pokemon Go creator plans to sell the ...

                                                             storyId \
2018-06-28 23:00:00.000 urn:newsml:reuters.com:20180628:nL4N1TU4F8:6
2018-06-28 21:23:26.526 urn:newsml:reuters.com:20180628:nNRA6e2vft:1
2018-06-28 19:48:32.627 urn:newsml:reuters.com:20180628:nRTV1vNw1p:1
2018-06-28 17:33:10.306 urn:newsml:reuters.com:20180628:nNRA6e1oza:1
2018-06-28 17:33:07.033 urn:newsml:reuters.com:20180628:nNRA6e1pmv:1
2018-06-28 17:31:44.960 urn:newsml:reuters.com:20180628:nNRA6e1m3n:1
2018-06-28 17:00:00.000 urn:newsml:reuters.com:20180628:nL1N1TU0PC:3

                          sourceCode
2018-06-28 23:00:00.000      NS:RTRS
2018-06-28 21:23:26.526    NS:ZACKSC
2018-06-28 19:48:32.627      NS:CNBC
2018-06-28 17:33:10.306    NS:WALLST
2018-06-28 17:33:07.033    NS:WALLST
```

```
    2018-06-28 17:31:44.960   NS:WALLST
    2018-06-28 17:00:00.000     NS:RTRS

In [32]: story_html = ek.get_news_story(news.iloc[1, 2])  ❷

In [33]: from bs4 import BeautifulSoup  ❸

In [34]: story = BeautifulSoup(story_html, 'html5lib').get_text()  ❹

In [35]: print(story[83:958])  ❺
         Jun 28, 2018 For years, investors and Apple AAPL have been
          beholden to the iPhone, which is hardly a negative since its
          flagship product is largely responsible for turning Apple
          into one of the world's biggest companies. But Apple has
          slowly pushed into new growth areas, with streaming
          television its newest frontier. So let's take a look at what
          Apple has planned as it readies itself to compete against the
          likes of Netflix NFLX and Amazon AMZN in the battle for the
          new age of entertainment. Apple's second-quarter revenues
          jumped by 16% to reach $61.14 billion, with iPhone revenues
          up 14%. However, iPhone unit sales climbed only 3% and iPhone
          revenues accounted for over 62% of total Q2 sales. Apple
          knows this is not a sustainable business model, because rare
          is the consumer product that can remain in vogue for decades.
          This is why Apple has made a big push into news,
```

❶ Получение метаданных небольшой подборки новостей.

❷ Получение полного текста отдельной статьи в виде HTML-документа.

❸ Импорт библиотеки BeautifulSoup, отвечающей за синтаксический анализ HTML-файлов.

❹ Извлечение содержимого статьи в текстовом виде.

❺ Вывод начала статьи.

Несмотря на простоту, эти примеры демонстрируют, насколько легко с помощью пакетов Python и служб подписки можно получить доступ к структурированным и неструктурированным историческим финансовым данным. Иногда такого рода информация бывает доступна даже бесплатно через торговые площадки наподобие FXCM (будет рассматриваться в главах 14 и 16). Откуда бы ни поступали наборы данных, после успешного импорта к ним можно применять весь спектр аналитических инструментов Python.

Финансовые системы, управляемые данными

Современные финансовые системы управляются данными. Даже крупнейшие и самые прибыльные хедж-фонды делают акцент на данных, а не собственно на финансах. Финансовые данные становятся доступными во все более крупных масштабах, причем как для коммерческих клиентов, так и для рядовых пользователей. В свою очередь, Python является языком выбора для взаимодействия с соответствующими программными интерфейсами и анализа получаемых данных.

Финансовые системы на основе искусственного интеллекта

Благодаря доступности больших объемов финансовых данных через программные интерфейсы стало возможным применять методы *искусственного интеллекта*, а также *машинного и глубокого обучения* для решения финансовых задач, таких как, например, алгоритмическая торговля.

Python — идеальный язык для задач искусственного интеллекта, к которому в первую очередь обращаются исследователи и разработчики. В этом смысле финансовая отрасль получает дополнительные преимущества, пользуясь имеющимися наработками в самых разных областях, иногда совершенно не связанных с финансами. В качестве примера можно взять открытую нейросетевую библиотеку TensorFlow (`www.tensorflow.org`), разработанную и сопровождаемую компанией Google. Эта же библиотека применяется компанией Alphabet (владельцем Google) для разработки беспилотных автомобилей.

Несмотря на то что цели TensorFlow даже косвенно не связаны с задачами автоматизированной алгоритмической биржевой торговли, с помощью TensorFlow можно, например, предсказывать волатильность финансовых рынков (соответствующие примеры будут приведены в главе 15).

Один из самых популярных пакетов для машинного обучения — Scikit-learn. В следующем примере упрощенно показано, как применять алгоритмы классификации для предсказания того, в каком направлении будут меняться рыночные цены, и как на основе этих предсказаний построить стратегию алгоритмической торговли. Все детали будут объясняться в главе 15, поэтому пример дан в максимально компактном виде. Для начала необходимо импортировать данные и подготовить массив признаков (накопительные показатели логарифмической доходности с векторами смещений).

```
In [36]: import numpy as np
         import pandas as pd
```

```
In [37]: data = pd.read_csv('../../source/tr_eikon_eod_data.csv',
                            index_col=0, parse_dates=True)
         data = pd.DataFrame(data['AAPL.O'])  ❶
         data['Доходность'] = np.log(data / data.shift())  ❷
         data.dropna(inplace=True)

In [38]: lags = 6

In [39]: cols = []
         for lag in range(1, lags + 1):
             col = 'lag_{}'.format(lag)
             data[col] = np.sign(data['Доходность'].shift(lag))  ❸
             cols.append(col)
         data.dropna(inplace=True)
```

❶ Получение исторических котировок акций компании Apple (`AAPL.0`).

❷ Вычисление логарифмической доходности за весь период.

❸ Генерирование столбцов объекта `DataFrame` и заполнение их направленными показателями логарифмической доходности (`+1` или `-1`).

Далее создается объект модели для *метода опорных векторов* (Support Vector Machine — SVM), после чего выполняется обучение модели и строится прогноз. На рис. 1.2 можно увидеть, что торговая стратегия, основанная на полученном прогнозе, применительно к акциям компании Apple дает лучший результат, чем инвестиция на основе простой доходности.

```
In [40]: from sklearn.svm import SVC

In [41]: model = SVC(gamma='scale')  ❶

In [42]: model.fit(data[cols], np.sign(data['Доходность']))  ❷
Out[42]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='scale',
             kernel='rbf', max_iter=-1, probability=False,
             random_state=None, shrinking=True, tol=0.001,
             verbose=False)

In [43]: data['Прогноз'] = model.predict(data[cols])  ❸

In [44]: data['Стратегия'] = data['Прогноз'] * data['Доходность']  ❹
```

```
In [45]: data[['Доходность', 'Стратегия']].cumsum().apply(np.exp).plot(
                    figsize=(10, 6));  ❺
```

❶ Создание объекта модели.

❷ Обучение модели на основе имеющихся признаков и размеченных данных (с учетом векторов направлений).

❸ Применение обученной модели для создания прогнозов (в пределах выборки), которые становятся позициями торговой стратегии.

❹ Вычисление логарифмической доходности по торговой стратегии с учетом прогнозных значений и эталонных значений доходности.

❺ Построение графика доходности торговой стратегии, основанной на алгоритмах машинного обучения, в сравнении с инвестицией на основе простой доходности.

Рис. 1.2. *Доходность торговой стратегии, полученной с помощью алгоритмов машинного обучения, в сравнении с инвестицией в акции компании Apple на основе простой доходности*

Конечно, в этом упрощенном примере не учитываются трансакционные издержки, а исходный набор данных не разделяется на обучающий и тестовый наборы. Тем не менее он наглядно показывает, насколько легко применять алгоритмы машинного обучения к финансовым данным, по крайней мере, с технической точки зрения (на практике придется решить еще ряд важных задач).

Финансовые системы на основе искусственного интеллекта

Технологии искусственного интеллекта вызывают такие же изменения в финансовой отрасли, как и в других областях. Доступность огромных объемов финансовых данных через программные интерфейсы служит катализатором изменений. Базовые методы искусственного интеллекта, а также машинного и глубокого обучения будут рассмотрены в главе 13, а в главах 15 и 16 мы научимся применять их в алгоритмической торговле. Но учитывайте, что применение искусственного интеллекта в финансовых расчетах — тема отдельной книги.

Финансовые системы на основе искусственного интеллекта как естественное продолжение финансовых систем, управляемых данными, — привлекательное направление как для исследователей, так и для разработчиков. В книге описывается применение целого ряда методик искусственного интеллекта и машинного обучения в различных контекстах, но все же основная тема книги, как следует из ее названия, — фундаментальные методы работы с финансовыми данными с помощью базовых инструментов Python. Эти методы имеют не меньшее значение и в системах искусственного интеллекта.

Резюме

Язык программирования Python — в особенности его экосистема — служит идеальной технологической базой для разработки финансовых приложений, причем это касается как индустрии в целом, так и отдельных проектов. Язык обладает целым рядом преимуществ, среди которых понятный синтаксис, эффективные подходы к разработке приложений, а также пригодность для разработки как прототипов, так и окончательных версий приложений в рамках единого производственного цикла. Наличие большого количества свободно доступных пакетов, библиотек и функциональных средств позволяет решать большинство задач, возникающих в современной финансовой среде. Язык соответствует всем требованиям, выдвигаемым с точки зрения анализа данных, работы с большими данными, производительности вычислений, соответствия регуляторным ограничениям и технологическим стандартам. Экосистема Python образует целостную, полнофункциональную среду разработки полного цикла, соответствующую целям даже крупных финансовых организаций.

Кроме того, Python стал языком выбора в системах искусственного интеллекта, как в целом, так и в частных проектах машинного и глубокого обучения.

Таким образом, это правильный язык для создания финансовых приложений, управляемых данными, а также финансовых систем на основе искусственного интеллекта — два ключевых направления, которые будут определять перспективы финансовой отрасли в обозримом будущем.

Дополнительные ресурсы

В следующих книгах более глубоко освещаются темы, поверхностно затронутые в данной главе (в частности, инструменты Python, анализ деривативов, машинное обучение и его применение в финансовом анализе).

- Hilpisch, Yves. *Derivatives Analytics with Python* (2015, Wiley).
- Lopez de Prado, Marcos. *Advances in Financial Machine Learning* (2018, Wiley).
- VanderPlas, Jake. *Python Data Science Handbook* (2016, O'Reilly).

Что касается алгоритмической торговли, то на сайте автора книги предлагается ряд обучающих курсов, посвященных применению Python и ряда других программных инструментов в этой быстро развивающейся сфере:

- `http://pyalgo.tpq.io`
- `http://certificate.tpq.io`

В главе цитировались следующие источники.

1. Ding, Cubillas. *Optimizing the OTC Pricing and Valuation Infrastructure* (2010, Celent).
2. Lewis, Michael. *Flash Boys* (2014, W. W. Norton & Company).
3. Patterson, Scott. *The Quants* (2010, Crown Business).