1

Базовые знания о данных

Для разных людей *данные* имеют разное значение: для биржевого маклера данные — это котировки акций в реальном времени, а для инженера NASA — сигналы, поступающие от марсохода. Однако когда дело касается обработки и анализа данных, к различным датасетам, независимо от их происхождения, могут применяться одинаковые или похожие подходы и методы, поскольку для этих целей важна лишь структура данных.

В этой главе объясняются основные понятия сферы обработки и анализа данных. Прежде всего, мы разберем основные категории данных, с которыми вам, скорее всего, придется иметь дело, а затем коснемся распространенных источников данных. Далее мы изучим этапы типичного пайплайна обработки данных (то есть фактического процесса получения, подготовки и анализа данных). И наконец, рассмотрим уникальные преимущества Python как инструмента работы с данными.

Категории данных

Программисты разделяют данные на три основные категории: неструктурированные, структурированные и слабоструктурированные. В пайплайне обработки исходные данные обычно являются неструктурированными; из них формируются структурированные или слабоструктурированные датасеты для дальнейшей обработки. Однако в некоторых пайплайнах сразу используются структурированные данные. К примеру, приложения, работающие с геолокацией,

могут получать структурированные данные непосредственно с датчиков GPS. В следующих разделах мы подробно рассмотрим эти три основные категории данных, а также данные временных рядов — особый тип данных, которые могут быть структурированными или слабоструктурированными.

Неструктурированные данные

Неструктурированные данные — это данные, не имеющие предопределенной организационной системы или схемы. Это наиболее распространенный вид данных, к нему относятся изображения, видео, аудио и текст на естественном языке. Для примера рассмотрим следующий финансовый отчет фармацевтической компании:

```
GoodComp shares soared as much as 8.2% on 2021-01-07 after the company announced
positive early-stage trial results for its vaccine.[1]
```

Этот текст относится к неструктурированному типу данных, поскольку содержащаяся в нем информация не организована по заранее определенной схеме. Вместо этого информация разбросана хаотично. Предложение можно переписать как угодно, при этом содержащаяся в нем информация останется прежней. Например:

```
Following the January 7, 2021, release of positive results from its vaccine
trial, which is still in its early stages, shares in GoodComp rose by 8.2%.
```

Несмотря на отсутствие структуры, неструктурированные данные могут содержать важную информацию, которую можно извлечь и преобразовать в структурированные или слабоструктурированные данные с помощью соответствующих методов обработки и анализа. Например, инструменты распознавания изображений сначала преобразуют набор пикселей изображения в датасет заранее определенного формата, а затем анализируют эти данные для идентификации содержимого. В следующем разделе показано несколько аналогичных способов структурирования данных, извлеченных из финансового отчета.

Структурированные данные

Структурированные данные организуются в заранее определенном формате. Такие данные обычно расположены в хранилище, например в реляционной базе данных или просто в файле *.csv* (comma-separated values — файл данных

[1] Акции GoodComp взлетели на 8.2% 07.01.2021 г. после того, как компания объявила о положительных результатах первого этапа испытаний своей вакцины.

с разделителями-запятыми). Данные, содержащиеся в таком хранилище, называются *записью*, а информация в записи организована в *полях*, которые должны поступать в последовательности, соответствующей ожидаемой структуре. В базе данных записи одинаковой структуры логически группируются в контейнер, называемый *таблицей*. База данных может содержать различные таблицы, и каждая из них имеет определенную структуру полей.

Существует два основных типа структурированных данных: числовые и категориальные. *Категориальные данные* — это данные, которые можно разделить по категориям на основе сходных характеристик; например, автомобили можно категоризировать по маркам и моделям. А *числовые данные* представляют информацию в числовой форме и позволяют выполнять над собой математические операции.

Следует отметить, что категориальные данные иногда могут принимать числовые значения. Возьмем, к примеру, почтовые индексы или телефонные номера. Хотя эти данные представлены в числовом выражении, выполнять над ними математические операции, например находить медианный почтовый индекс или среднее значение номеров телефонов, не имеет смысла.

Как преобразовать текст, представленный в примере из предыдущего раздела, в структурированные данные? Нас интересует конкретная информация из текста, например названия компаний, даты и цены на акции. Необходимо соответствующим образом форматировать эту информацию и разместить ее в полях, готовых для вставки в базу данных:

```
Компания:    ABC
Дата:        yyyy-mm-dd
Акция:       nnnnn
```

Используя методы *обработки естественного языка* (NLP, natural language processing), дисциплины, которая обучает машины понимать человекочитаемый текст, можно извлечь информацию для этих полей. Так, название компании можно найти, распознавая категориальную переменную данных, принимающую одно из заданных значений, например Google, Apple или GoodComp. Аналогично можно распознать дату, сопоставив последовательность ее символов с одним из возможных форматов даты, например `yyyy-mm-dd`. В нашем примере мы распознаем, извлекаем и представляем данные в заранее определенном формате следующим образом:

```
Компания:    GoodComp
Дата:        2021-01-07
Акция:       +8.2%
```

Чтобы сохранить эту запись в базе данных, лучше представить ее в виде строки, состоящей из последовательности полей. Таким образом ее можно реорганизовать в виде прямоугольного объекта данных, или двумерной матрицы:

```
Компания | Дата       | Акция
--------------------------
GoodComp |2021-01-07 | +8.2%
```

Из одного и того же источника неструктурированных данных при необходимости можно извлекать разную информацию. Наш пример не только содержит информацию об изменении стоимости акций GoodComp на определенную дату, но и указывает причину этого изменения, выраженную фразой «the company announced positive early-stage trial results for its vaccine[1]». Рассматривая предложение с этой точки зрения, можно создать запись с такими полями:

```
Компания:   GoodComp
Дата:       2021-01-07
Продукт:    vaccine
Стадия:     early-stage trial
```

Сравните с первой извлеченной записью:

```
Компания:   GoodComp
Дата:       2021-01-07
Акция:      +8.2%
```

Обратите внимание, что эти две записи содержат разные поля и потому имеют разные структуры. По этой причине они должны храниться в двух разных таблицах базы данных.

Слабоструктурированные данные

В случаях, когда структура информации не соответствует строгим требованиям форматирования, может понадобиться формат слабоструктурированных данных, позволяющий хранить записи различных структур в одном контейнере (таблице базы данных или документе). Как и неструктурированные данные, слабоструктурированные данные не привязаны к заданной схеме организации. Однако экземпляры слабоструктурированных данных, в отличие от неструктурированных,

[1] Компания объявила о положительных результатах первого этапа испытаний своей вакцины.

демонстрируют определенную структуру, которая обычно выражена в виде самоописываемых тегов или других маркеров.

Самый известный пример слабоструктурированного формата данных — XML и JSON. Вот как может выглядеть финансовый отчет в формате JSON:

```
{
    "Компания": "GoodComp",
    "Дата":     "2021-01-07",
    "Акция":    8.2,
    "Детали":   "the company announced positive early-stage trial results for
its
vaccine."
}
```

Здесь мы видим ту же ключевую информацию, которую извлекли ранее. Каждый фрагмент данных сопровождается описательным тегом, например `"Компания"` или `"Дата"`. Благодаря этим тегам информация организуется аналогично тому, как она была представлена в предыдущем разделе, однако теперь у нас есть четвертый тег, `"Детали"`, которым помечена неструктурированная часть исходного высказывания. Из примера видно, что в рамках одной записи в формате слабоструктурированных данных могут содержаться и структурированные, и неструктурированные фрагменты.

Более того, в один контейнер можно поместить несколько записей различной структуры. Например, две разные записи, полученные из финансового отчета, хранятся в одном документе JSON:

```
[
    {
        "Компания": "GoodComp",
        "Дата":     "2021-01-07",
        "Акция":    8.2
    },
    {
        "Компания": "GoodComp",
        "Дата":     "2021-01-07",
        "Продукт":  "vaccine",
        "Этап":     "early-stage trial"
    }
]
```

Помните, что в предыдущем разделе мы обсуждали реляционную базу данных, которая, будучи жестко структурированным хранилищем, не допускает размещения записей разной структуры в одной таблице.

Данные временных рядов

Временны́е ряды — это множество точек данных, индексированных или перечисленных в порядке от самой ранней до самой поздней. Многие датасеты с финансовыми данными хранятся в виде временных рядов, поскольку обычно включают наблюдения в конкретное время.

Данные временных рядов могут быть как структурированными, так и слабоструктурированными. Представьте, что через равные промежутки времени вы получаете данные о местоположении такси в виде записей от GPS-трекера. Такие данные могут иметь следующий формат:

```
[
    {
        "машина":     "cab_238",
        "координаты": (43.602508,39.715685),
        "время":      "14:47",
        "состояние":  "доступен"
    },
    {
        "машина":     "cab_238",
        "координаты": (43.613744,39.705718),
        "время":      "14:48",
        "состояние":  "доступен"
    }
    ...
]
```

Каждую минуту от машины `cab_238` поступает новая запись, содержащая последние координаты местоположения (широта/долгота). Каждая запись имеет одинаковую последовательность полей, и каждое поле имеет последовательную структуру от одной записи к следующей, что позволяет хранить временные ряды в таблице реляционной базы данных как обычные структурированные данные.

А теперь предположим, что данные поступают через разные промежутки времени (что часто бывает на практике) и за минуту вы получаете несколько наборов координат. Тогда структура может выглядеть так:

```
[
    {
        "машина": "cab_238",
        "координаты": [(43.602508,39.715685),(43.602402,39.709672)],
        "время": "14:47",
        "состояние": "доступен"
```

```
    },
    {
        "машина": "cab_238",
        "координаты": (43.613744,39.705718),
        "время": "14:48",
        "состояние": "доступен"
    }
]
```

Обратите внимание, что первое поле `"координаты"` состоит из двух пар значений и, таким образом, отличается от второго поля `"координаты"`. Такие данные являются слабоструктурированными.

Источники данных

Теперь, когда вы узнали об основных категориях данных, сможете ли вы назвать источники, из которых их можно получить? Вообще говоря, данные могут поступать из различных источников: текстов, видео, изображений, датчиков устройств и т. д. Однако с точки зрения Python-скриптов, которые вам предстоит писать, наиболее привычные — это:

- интерфейс прикладного программирования (API);
- веб-страница;
- база данных;
- файл.

Этот список не является ни исчерпывающим, ни строгим; существует множество других источников данных. В главе 9, к примеру, вы увидите, как использовать смартфон в качестве источника GPS-данных для пайплайна обработки, в частности, применяя бота как промежуточное звено между смартфоном и Python-скриптом.

Технически все перечисленные варианты требуют использования соответствующих библиотек Python. Так, например, прежде чем получить данные из API, нужно установить обертку Python для API или использовать библиотеку Python Requests для выполнения HTTP-запросов к API напрямую. Аналогично, чтобы получить доступ к данным из базы, необходимо установить коннектор из Python-кода, который позволит подключиться к базам данных конкретного типа.

Многие из этих библиотек необходимо загружать и устанавливать, однако некоторые из них поставляются с Python по умолчанию. Например, чтобы

загрузить данные из файла JSON, можно воспользоваться встроенным пакетом Python `json`.

В главах 4 и 5 мы более подробно рассмотрим процесс получения данных. В частности, вы узнаете, как преобразовывать конкретную информацию из различных источников в структуры данных Python-скрипта для дальнейшей обработки. А пока кратко рассмотрим каждый источник из списка выше.

API

API (программный посредник, позволяющий двум приложениям взаимодействовать друг с другом) на сегодняшний день, вероятно, самый известный способ получения данных. Как уже упоминалось, чтобы воспользоваться преимуществами API в Python, понадобится установить обертку в виде библиотеки Python. Чаще всего это делается с помощью команды `pip`.

Не все API имеют собственную обертку для Python, но это не значит, что не получится обратиться к ним из кода. Если API обслуживает HTTP-запросы, с ним можно взаимодействовать с помощью библиотеки Requests. Это открывает доступ к тысячам API, которые можно использовать в коде, запрашивая датасеты и затем обрабатывая их.

При выборе API для конкретной задачи следует обращать внимание на такие факторы:

Функциональность. Многие API предоставляют схожие функции, поэтому важно как можно точнее определиться с требованиями. Например, многие API позволяют осуществлять поиск в интернете из скрипта, но лишь в некоторых предусмотрена функция выбора результатов поиска по дате публикации.

Стоимость. Многие API позволяют использовать так называемый *ключ разработчика*. Обычно он предоставляется бесплатно, однако может иметь определенные ограничения, например, по количеству вызовов в день.

Стабильность. Благодаря репозиторию Python Package Index (PyPI)[1] любой желающий может упаковать API в пакет `pip` и сделать его общедоступным. По этой причине API (или даже несколько API) существует почти для любой задачи, которую только можно представить, но не все из этих API надежны. К счастью, репозиторий PyPI отслеживает производительность и использование пакетов.

[1] https://pypi.org

Документация. Популярные API обычно имеют соответствующий сайт с документацией, на котором можно изучить все команды и примеры использования. Хороший образец — страница с документацией для API Nasdaq Data Link (он же Quandl)[1], где доступны примеры выполнения различных запросов временных рядов.

Большинство API возвращают результаты в одном из следующих трех форматов: JSON, XML или CSV. Данные в любом из этих форматов легко преобразуются в структуры данных, которые либо встроены в Python, либо часто в нем используются. Yahoo Finance API, например, извлекает и анализирует данные фондового рынка, а затем возвращает информацию, уже переведенную в тип pandas DataFrame, повсеместно используемую структуру, которую мы обсудим в главе 3.

Веб-страницы

Веб-страницы могут быть статичными или создаваться «на лету» в ответ на действия пользователя, и в таком случае они, вероятно, будут содержать информацию из множества источников. В обоих случаях программа может читать веб-страницу и извлекать ее части. Это называется *веб-скрейпингом* (web scraping). Такая операция вполне законна, если страница находится в открытом доступе.

Типичный алгоритм скрейпинга в Python требует наличия двух библиотек: Requests и BeautifulSoup. Requests получает исходный код страницы, а BeautifulSoup создает *дерево разбора* (parse tree) для страницы. Такое дерево является иерархическим представлением содержимого страницы. По нему можно выполнять поиск и извлекать информацию, используя стандартный синтаксис Python. К примеру, следующий фрагмент дерева разбора:

```
[<td title="03/01/2020 00:00:00"><a href="Download.aspx?ID=630751"
id="lnkDownload630751"
 target="_blank">03/01/2020</a></td>,
<td title="03/01/2020 00:00:00"><a href="Download.aspx?ID=630753"
id="lnkDownload630753"
 target="_blank">03/01/2020</a></td>,
<td title="03/01/2020 00:00:00"><a href="Download.aspx?ID=630755"
id="lnkDownload630755"
 target="_blank">03/01/2020</a></td>]
```

[1] https://docs.data.nasdaq.com/docs/python-time-series

можно легко преобразовать в список элементов, используя в Python-скрипте цикл `for`:

```
[
    {'Document_Reference': '630751', 'Document_Date': '03/01/2020',
     'link': 'http://www.dummy.com/Download.aspx?ID=630751'}
    {'Document_Reference': '630753', 'Document_Date': '03/01/2020',
     'link': 'http://www.dummy.com/Download.aspx?ID=630753'}
    {'Document_Reference': '630755', 'Document_Date': '03/01/2020',
     'link': 'http://www.dummy.com/Download.aspx?ID=630755'}
]
```

Это пример преобразования слабоструктурированных данных в структурированные.

Базы данных

Еще один популярный источник данных — реляционная база данных (БД), обеспечивающая эффективное хранение и обработку структурированных данных, а также простой и удобный доступ к ним. Получение данных или отправка их в таблицы осуществляется с помощью запросов на языке SQL (Structured Query Language — язык структурированных запросов). Например, следующий запрос к таблице `employees` базы данных извлекает список только тех программистов, которые работают в IT-отделе. Таким образом, не нужно получать таблицу целиком:

```
SELECT first_name, last_name FROM employees WHERE department = 'IT' and title =
'programmer'
```

В Python есть встроенный движок базы данных, SQLite. В качестве альтернативы можно использовать любую другую БД. Для доступа к базе данных необходимо установить клиент БД в своей среде.

Помимо обычных жестко структурированных баз данных, в последние годы все чаще возникает потребность в хранении неоднородных и неструктурированных данных в контейнерах, подобных базам данных, что привело к появлению так называемых *NoSQL* (*non-SQL* или *not only SQL*) баз данных. БД NoSQL используют гибкие модели данных, позволяя хранить большие объемы неструктурированной информации с помощью метода *«ключ — значение»*, когда доступ к каждому фрагменту данных осуществляется с помощью связанного ключа. Вот как мог бы выглядеть наш финансовый отчет, если бы он хранился в базе данных NoSQL:

```
Key   value
---   -----

...

26   GoodComp shares soared as much as 8.2% on 2021-01-07 after the company
announced ...
```

Все высказывание сопряжено с ключом 26. Может показаться странным хранить отчет целиком в БД. Вспомним, однако, что из одного высказывания можно извлечь несколько записей. Хранение целого отчета в дальнейшем обеспечит гибкость извлечения частей информации.

Файлы

Файлы могут содержать структурированные, слабоструктурированные и неструктурированные данные. Встроенная в Python функция `open()` позволяет открыть файл и использовать его данные в скрипте. Однако в зависимости от формата данных (например, CSV, JSON или XML) может потребоваться импортировать соответствующую библиотеку, чтобы иметь возможность выполнять операции чтения, записи и/или добавления данных.

Файлам с обычным текстом не нужна библиотека для дальнейшей обработки, они просто рассматриваются в Python как последовательности строк. Возьмем в качестве примера следующее сообщение, которое маршрутизатор Cisco мог бы отправить в файл журнала:

```
dat= 'Jul 19 10:30:37'
host='sm1-prt-highw157'
syslogtag='%SYS-1-CPURISINGTHRESHOLD:'
msg=' Threshold: Total CPU Utilization(Total/Intr): 17%/1%,
          Top 3 processes(Pid/Util):   85/9%, 146/4%, 80/1%'
```

Можно читать сообщение построчно, вычленяя необходимую информацию. Так, если ваша задача — найти сообщения, содержащие информацию о загрузке процессора, и извлечь из них определенные показатели, ваш скрипт должен распознать последнюю строку фрагмента как сообщение, которое нужно выбрать.

В главе 2 мы рассмотрим пример извлечения конкретной информации из текстовых данных с помощью методов обработки текста.

Пайплайн обработки данных

В этом разделе мы рассмотрим этапы обработки данных, также известные как пайплайн обработки данных (data processing pipeline). Вот привычный алгоритм действий с данными:

1. Получение.
2. Очистка.
3. Преобразование.
4. Анализ.
5. Хранение.

Однако вы увидите, что он соблюдается не всегда. В некоторых приложениях можно объединять несколько этапов в один или вообще пропускать некоторые из них.

Получение

Прежде чем что-то делать с данными, их необходимо получить. Именно поэтому первый шаг в любом пайплайне обработки — сбор данных. В предыдущем разделе вы узнали о распространенных типах источников данных. Некоторые из них позволяют загружать конкретную часть данных в соответствии с запросом.

Например, запрос к API Yahoo Finance требует указать тикер компании и период времени, за который необходимо получить данные о ценах на акции. Аналогично News API, который позволяет извлекать новостные статьи, может обрабатывать ряд параметров для сужения списка запрашиваемых статей, включая источник и дату публикации. Однако несмотря на эти параметры, полученный список может потребовать дополнительной фильтрации. Это означает, что необходима очистка.

Очистка

Очистка данных — это процесс обнаружения и исправления поврежденных/неточных данных или удаления ненужных. В некоторых случаях этот шаг необязателен и получаемые данные сразу готовы к анализу. Например, библиотека `yfinance` (обертка Python для Yahoo Finance API) возвращает данные об акциях в виде удобного объекта pandas DataFrame, что позволяет пропустить этапы очистки и преобразования и сразу перейти к анализу данных. Однако если ваш инструмент сбора данных — веб-скрейпер (web scraper), данные, безусловно, будут нуждаться в очистке, поскольку среди полезных данных, скорее всего, будут присутствовать фрагменты HTML-разметки:

```
6.\t3астройщик должен удовлетворить следующие требования отдела водоотведения
DCC.\x80\x99 следующим образом\r\n\r\n\r\n
```

После очистки этот фрагмент текста будет выглядеть так:

```
6. Застройщик должен удовлетворить следующие требования отдела водоотведения DCC.
```

Помимо HTML-разметки, собранный скрейпером текст может содержать и другую ненужную информацию, как в следующем примере, где фраза `Просмотреть полный текст` является просто текстом гиперссылки. Чтобы получить доступ к тексту, может понадобиться открыть эту ссылку:

```
Разрешение на предлагаемые поправки к разрешению на строительство, полученное
30го Просмотреть полный текст
```

Этап очистки данных также можно использовать для выделения определенных сущностей. К примеру, после запроса выборки статей из News API может потребоваться отобрать из них только статьи за указанный период, в заголовках которых присутствует упоминание о денежной сумме или процентах. Такую фильтрацию можно считать операцией очистки, поскольку ее целью является удаление ненужных данных и подготовка к операциям преобразования и анализа данных.

Преобразование

Преобразование данных — это процесс изменения формата или структуры данных при подготовке к анализу. Например, чтобы извлечь информацию из неструктурированных текстовых данных о GoodComp, как мы делали в разделе «Структурированные данные», можно разбить текст на отдельные слова, или *лексемы*, чтобы инструмент распознавания именованных сущностей (NER, named entity recognition) мог искать нужную информацию. При извлечении информации *именованная сущность* обычно представляет собой объект реального мира, такой как человек, организация или продукт, то есть имя существительное. Существуют также именованные сущности, отражающие даты, проценты, финансовые термины и многое другое.

Большинство инструментов NLP может выполнять это преобразование автоматически. После такой «нарезки» данные GoodComp будут выглядеть так:

```
['GoodComp', 'shares', 'soared', 'as', 'much', 'as', '8.2%', 'on',
 '2021-01-07', 'after', 'the', 'company', 'announced', 'positive',
 'early-stage', 'trial', 'results', 'for', 'its', 'vaccine']
```

Еще одно, более глубокое преобразование — превращение текстовых данных в числовые. Например, коллекцию новостных статей можно преобразовать, выполнив *анализ тональности*, или *сентимент-анализ*, — метод обработки текста, который генерирует число, представляющее эмоции, выраженные в тексте.

Анализ тональности можно проводить с помощью такого инструмента, как `SentimentAnalyzer` из пакета `nltk.sentiment`. Типичный результат анализа может выглядеть следующим образом:

```
Тональность   URL
-----------   ------------------------------------------------------------
0.9313        https://mashable.com/uk/shopping/amazon-face-mask-store-july-28/
0.9387        https://skillet.lifehacker.com/save-those-crustacean-shells-to
                    -make-a-sauce-base-1844520024
```

Каждая запись в датасете теперь содержит число, например `0.9313`, представляющее эмоциональную окраску соответствующей статьи. Когда настроение каждой статьи выражено численно, можно рассчитать среднее значение тональности по всему датасету, что позволяет определить общую окраску интересующего объекта, например компании или продукта.

Анализ

Анализ — основной этап пайплайна обработки данных — это интерпретация исходных данных, позволяющая сделать выводы, которые очевидны не сразу.

В сценарии с сентимент-анализом, возможно, возникнет необходимость изучить, как менялось отношение к компании за определенный период времени, в зависимости от цены ее акций. Также можно сравнить показатели биржевого индекса, например S&P 500, с тональностью, выраженной в широкой подборке новостных статей за тот же период. Следующий фрагмент иллюстрирует, как может выглядеть датасет, где данные S&P 500 представлены вместе с общей тональностью новостей того же дня:

```
Дата          Тональность    S&P_500
-------------------------------------
2021-04-16    0.281074       4185.47
2021-04-19    0.284052       4163.26
2021-04-20    0.262421       4134.94
```

Поскольку и показатели тональности, и показатели акций выражены в числовом формате, можно построить два соответствующих графика на одном рисунке для визуального анализа, как показано на рис. 1.1.

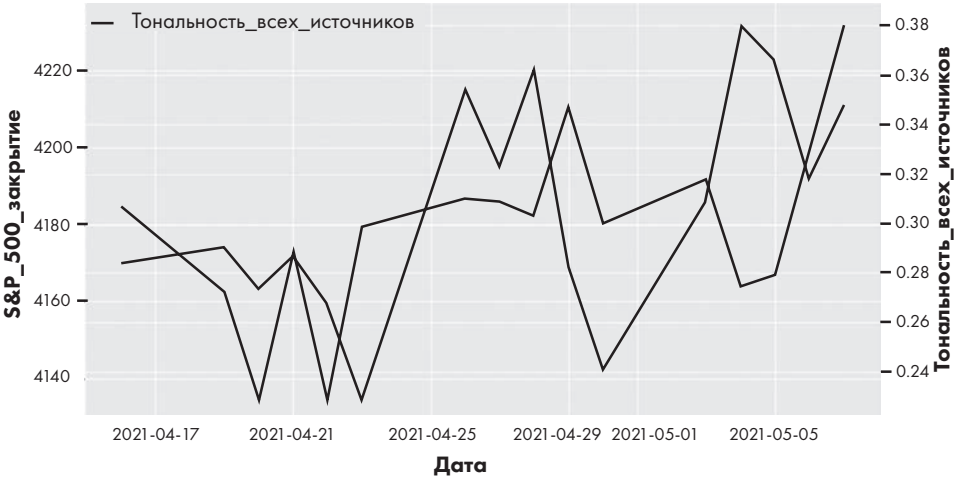

Рис. 1.1. Пример визуального анализа данных

Визуальный анализ — один из наиболее часто используемых и эффективных методов интерпретации данных. Мы подробнее разберем его в главе 8.

Хранение

В большинстве случаев результаты, полученные в процессе анализа данных, необходимо хранить для дальнейшего использования. Обычно есть два варианта хранения: файлы и базы данных. Последний предпочтителен, если предполагается, что данные будут использоваться часто.

Питонический стиль

Предполагается, что при работе с данными на Python вы будете писать код в *питоническом стиле*, то есть лаконично и продуктивно. Например, в питоническом коде часто используются *списковые включения* (list comprehensions) — метод реализации полезных функций обработки данных в одной строке кода.

Более подробно мы рассмотрим списковые включения в главе 2, а пока — краткий пример того, как концепция Python работает на практике. Допустим, необходимо обработать фрагмент текста, состоящий из нескольких предложений:

```
txt = ''' Eight dollars a week or a million a year - what is the difference?
A mathematician or a wit would give you the wrong answer. The magi brought
valuable gifts, but that was not among them. - The Gift of the Magi, O'Henry'''
```

В частности, необходимо разделить текст на предложения, создать список слов каждого из них и исключить знаки препинания. Благодаря функционалу списковых включений Python, все это можно реализовать одной строкой кода, так называемым однострочником (one-liner):

```
word_lists = [[w.replace(',','') ❶ for w in line.split() if w not in ['-']]
            ❷ for line in txt.replace('?','.').split('.')]
```

Цикл `for line in txt` ❷ разбивает текст на предложения и сохраняет их в список. Затем цикл `for w in line` ❶ разбивает каждое предложение на отдельные слова и сохраняет их в список внутри большого списка. В результате получается следующий список списков:

```
[['Eight', 'dollars', 'a', 'week', 'or', 'a', 'million', 'a', 'year', 'what',
  'is', 'the',      'difference'], ['A', 'mathematician', 'or', 'a', 'wit',
  'would', 'give', 'you', 'the', 'wrong', 'answer'], ['The', 'magi',
  'brought', 'valuable', 'gifts', 'but', 'that', 'was', 'not', 'among',
  'them'], ['The', 'Gift', 'of', 'the', 'Magi', "O'Henry"]]
```

Здесь в одной строке кода выполняется два этапа пайплайна обработки данных: очистка и преобразование. Мы очистили текст, удалив из него знаки препинания, и преобразовали, отделив слова друг от друга и сформировав список слов каждого предложения.

Если вы перешли на Python с другого языка программирования, попробуйте реализовать эту же задачу на другом языке. Сколько строк займет такой код?

Выводы

После прочтения этой главы вы должны понимать, какие основные категории данных существуют, откуда они берутся и как организован типичный пайплайн обработки данных.

Как вы увидели, существует три основные категории данных: неструктурированные, структурированные и слабоструктурированные. Исходным материалом в пайплайне обработки данных обычно являются неструктурированные данные, которые становятся готовыми к анализу, проходя этапы очистки и преобразования в структурированные или слабоструктурированные данные. Вы также узнали о пайплайнах обработки исходно структурированных или слабоструктурированных данных, полученных из API или реляционных БД.