ОНЛАЙН-ИССЛЕДОВАНИЯ

С.В. Жучкова, А.Н. Ротмистров
(Москва)

АВТОМАТИЧЕСКОЕ ИЗВЛЕЧЕНИЕ ТЕКСТОВЫХ И ЧИСЛОВЫХ ВЕБ-ДАННЫХ ДЛЯ ЦЕЛЕЙ СОЦИАЛЬНЫХ НАУК

Статья посвящена процедуре автоматического извлечения данных с веб-страниц, т.е. скрапингу веб-данных. Рассмотрены виды веб-данных (цифровые следы и прочие веб-данные числовой, текстовой и других модальностей), возможности их использования (скорость сбора данных и, как следствие, сплошной охват, оперативность и др.) и ограничения (ограниченная репрезентативность, трудности организации хранения большого объема данных, отклонения от традиционной последовательности постановки исследования и др.) по сравнению с традиционными методами сбора информации. Описаны пути извлечения веб-данных со статических и динамических веб-страниц посредством интерфейса API, пакета requests, фреймворка selenium. Разобраны минимально необходимые для извлечения веб-данных компетенции, в том числе в программировании на языке Python и ориентировании в коде веб-страниц. Также дана подробная иллюстрация на основе фрагмента сбора данных исследования конкурсов для фрилансеров.

Светлана Васильевна Жучкова – младший научный сотрудник, Национальный исследовательский университет «Высшая школа экономики», Москва, Россия. E-mail: szhuchkova@hse.ru.

Алексей Николаевич Ротмистров – кандидат социологических наук, старший научный сотрудник, Национальный исследовательский университет «Высшая школа экономики», Москва, Россия. E-mail: arotmistrov@hse.ru.

Исследование выполнено при финансовой поддержке Российского научного фонда по гранту 19-78-10080.

Ключевые слова: автоматическое извлечение данных, большие данные, веб-данные, веб-скрапинг, вычислительные социальные науки, текстовые данные, API, requests, selenium

Введение

2019 год ознаменовался 10-летним юбилеем официального признания вычислительных социальных наук (computational social science), предмет которых – использование больших данных для изучения, объяснения и прогнозирования индивидуального и коллективного поведения [1]. И хотя потенциал таких данных обсуждался и ранее [2; 3], после 2009 г. прирост работ на их основе существенно ускорился. В этих работах содержится множество примеров успешного использования подобных данных, а также констатация их обилия и разнообразия. Примеры изучения политических предпочтений [4; 5; 6; 7; 8], психологических характеристик [9; 10; 11], академических достижений [12] показывают, что большие данные могут быть крайне полезны для социальных наук. Они воспроизводят естественные паттерны поведения информантов, поскольку собраны без участия последних в исследовательском процессе и позволяют с высокой точностью это поведение прогнозировать – построенные на основе них предсказательные модели достигают точности 80–90% [10]. В качестве данных при этом выступают публикации и комментарии в социальных сетях, информация о банковских транзакциях и о геопозиции, записи с камер видеонаблюдения и т.п. **Данные, полученные из источников в сети Интернет (т.е. веб-ресурсов) без участия информантов, мы называем веб-данными.** В России проникновение веб-данных в социальные науки происходит медленнее: за 10 лет появилось лишь несколько десятков подобных исследований, а на русском языке – еще меньше; основная их часть оперирует данными из социальных сетей («ВКонтакте», Facebook, Twitter) и лишь некоторые – из иных источников (см., например: [13; 14]).

Одним из главных тормозов в использовании социальными учеными веб-данных считается трудность овладения инструментами их извлечения, т.е. программированием и веб-технологиями, хотя бы на базовом уровне[1] [15]. В связи с этим веб-данные иногда называют псевдодоступными – они легко наблюдаемы, но требуют кропотливой работы по их получению [13: 46]. При этом структурированные и доступные для представителей социальных наук материалы для овладения соответствующими навыками на русском языке все еще немногочисленны. Цель настоящей статьи – рассмотреть процедуры **веб-скрапинга – автоматического извлечения веб-данных** [16] с помощью языка программирования Python (далее мы будем использовать более привычное русскоязычному читателю название – Питон) и необходимые для этого компетенции, опираясь на опыт нашего научного коллектива. В частности, в статье рассматриваются необходимый минимум владения языком[2], устройство веб-страниц, особенности хранения данных на них и возможные трудности веб-скрапинга – все это также соотнесено с

[1] Ряд приложений для получения веб-данных не требуют наличия таких компетенций (например, Octoparse и Portia), но многие из них платные и функционал многих ограничен.

[2] Вступительный материал, покрывающий базовые концепции программирования на Питоне, вынесен в отдельное онлайн-приложение к настоящей статье: https://github.com/LanaLob/Sociology_4M. Приложение оформлено как файл программы Jupyter Notebook – наиболее популярной и дружественной пользователю оболочки для работы с этим языком. Она содержится в пакетном менеджере Anaconda, который устанавливается с официального сайта Anaconda: (URL: https://www.anaconda.com/distribution/ (дата обращения: 02.11.2020)) с учетом операционной системы. После установки на компьютере появится программа Anaconda Navigator, через которую можно войти в Jupyter Notebook. Файл Jupyter Notebook – это файл формата .ipynb, для работы с которым используется вкладка браузера (который является браузером по умолчанию). При запуске Jupyter Notebook в браузере автоматически открывается вкладка со списком папок и файлов компьютера, в этой вкладке можно создавать новые или открывать уже существующие файлы.

подробной иллюстрацией из недавнего релевантного российского исследования. Статья поможет представителям социальных наук, в том числе не сталкивавшимся с программированием и извлечением веб-данных, понять логику этого процесса и овладеть базовыми навыками, покрывающими приличную часть возможных задач. Материал изложен в следующем порядке:

– краткое обсуждение видов веб-данных, примеров, возможностей и ограничений их использования;

– описание способов хранения и путей извлечения веб-данных;

– иллюстрация, как с помощью Питона можно собрать веб-данные для исследования с помощью одного из описываемых способов.

Виды веб-данных, примеры, возможности и ограничения их использования

Наиболее многообещающими для социальных наук представляются веб-данные в виде так называемых цифровых следов (digital traces, или trace data [17]). Они возникают как естественный результат взаимодействия пользователя с веб-ресурсами [17] – сайтами, в первую очередь[1]. Дело в том, что веб-ресурсы, как правило, фиксируют в своих базах данных действия пользователей в виде записей («логов»). Другое дело, что цифровые следы различаются по степени открытости: от общедоступных (скажем, «лайки» в открытых аккаунтах социальных сетей) до закрытых для всех, кроме администраторов веб-ресурса (например, история активности покупателей в интернет-магазине), используемых преимущественно для оптимизации (персонализации) взаимодействия ресурса с пользователями в дальнейшем.

[1] Существуют также цифровые следы, являющиеся результатом взаимодействия пользователя с различными цифровыми девайсами (телефонами, Smart TV, фитнес-трекерами и т.д.), которые находятся вне фокуса этой статьи.

Встречаются веб-данные, не являющиеся непосредственно «следами» пользователей; обычно это информация, ради распространения которой веб-ресурсы и созданы – скажем, информация из Википедии, с новостных сайтов, сайтов, агрегирующих фильмы или музыку, и т.д.; даже если какая-то часть такой информации добавлена пользователями, а не администраторами, то она – не вполне цифровой след, поскольку она не характеризует пользователей непосредственно и добавлена ими намеренно (и в этом смысле возникает место для умышленного искажения информации).

Веб-данные бывают разных модальностей: числа, тексты, изображения, видео, географические координаты и т.п., а также так называемые сетевые данные (выражают отношения в общем смысле между пользователями или другими объектами анализа: «дружбу», степень совпадения интересов и т.п. [15]). Хотя сетевые данные тоже иногда представлены числами (скажем, дихотомиями, выражающими наличие или отсутствие отношений), к ним применяются специальные методы анализа: построение графов, прочие методы сетевого анализа (network analysis).

Искомые данные на веб-странице могут выглядеть уже готовыми для анализа: например, быть представленными в виде таблицы с числами. Возникает впечатление, что такая организация данных похожа на формат обычных статистических данных, но оно зачастую обманчиво: таблица в коде сайта бывает записана не как таблица, а числа в ней не как числа. Поэтому выгрузку веб-данных и их анализ опосредует важный этап – предобработка; для веб-данных разных модальностей требуется разная предобработка: например, для текстовых веб-данных может потребоваться токенизация (удаление вспомогательных символов), лемматизация (приведение каждого слова к начальной форме), удаление стоп-слов, иногда – исправление опечаток [18].

Веб-данные набирают популярность в исследовательской практике; эта тенденция находит как сторонников, так и противников. На наш взгляд, основные противоречия внутри академического

сообщества по поводу веб-скрапинга и веб-данных вращаются вокруг репрезентативности веб-данных, их объема и валидности. Рассмотрим эти противоречия.

Сбор веб-данных, по сравнению со сбором традиционных данных, обычно требует гораздо меньших временны́х и материальных затрат на единицу информации [17], что открывает возможность собрать всю информацию из релевантного веб-источника – назовем это сплошным веб-скрапингом. Сплошной веб-скрапинг ассоциируется с переходом от выборочной совокупности к генеральной [19]. Это оправданно, если эмпирический объект исследования исходно конструируется как участники некоторой социальной сети или отдельного онлайн-сообщества, пользователи некоторой онлайн-платформы; такие исследования встречаются нечасто, хотя их доля уверенно растет по мере интернетизации человечества. Если же эмпирический объект конструируется более традиционно – в привязке к геополитическим, административным, социально-демографическим рамкам, то корректность использования веб-данных для его изучения не столь очевидна. С одной стороны, многие социальные сети позволяют найти именно тех участников, которые отвечают характеристикам изучаемого эмпирического объекта. С другой стороны, такая совокупность участников социальной сети, скорее всего, имеет смещения по содержательно важным для исследования характеристикам, поскольку не является вероятностной выборкой из генеральной совокупности; степень этого смещения оценить и, следовательно, компенсировать затруднительно. Так, выбор пользователями конкурирующих веб-ресурсов зависит от характеристик как первых, так и вторых [20], в связи с чем пользователи любого веб-ресурса (скажем, все пользователи «ВКонтакте» или все пользователи «Одноклассников») не являются вероятностной выборкой из соответствующей генеральной совокупности (например, взрослого населения России, имеющего доступ к Интернету).

Веб-скрапинг открывает широкие возможности для оперативного изучения социального влияния специфических событий

(скажем, некоторого политического решения, экологической проблемы и т.п.), изучать которые традиционными методами неэффективно в силу соотношения масштаба события и требуемых временны́х и материальных издержек. В рамках веб-скрапинга такое влияние можно изучать, опираясь на активности пользователей социальных сетей (например, «лайки», количество и тональность комментариев). Ложкой дегтя в этих практиках снова выступает «НЕнейтральность» веб-ресурсов: действия пользователей в большей или меньшей мере обусловлены рекомендательными алгоритмами интерфейса этих ресурсов [21; 22]; скажем, публикация получает широкий отклик в виде «лайков» и комментариев в том числе из-за того, что алгоритмы веб-ресурса показали ее бóльшему количеству пользователей.

Получается, что изучение поведения людей посредством веб-скрапинга оказывается изучением одновременно и людей, и используемых ими веб-ресурсов, причем эксплицитные целевые характеристики людей оказываются в латентном взаимодействии с имплицитными характеристиками веб-ресурсов. Другими словами, к известным эффектам исследователя и инструментария [23] добавляется эффект среды, в которой изучается объект; этот эффект сам по себе требует изучения, чтобы в дальнейшем эксплицитно учитывать его влияние на репрезентативность веб-данных и, в идеале, компенсировать. Тем не менее теоретическое обоснование нерепрезентативности веб-данных в какой-то мере уравновешивается эмпирическими свидетельствами статистического совпадения результатов массовых опросов и анализа данных, полученных путем веб-скрапинга [19]. Вероятно, изучение влияния веб-ресурсов как социальной среды на репрезентативность веб-данных могло бы происходить посредством такого рода экспериментов, в которых сопоставляются результаты традиционных методов сбора информации и веб-скрапинга при максимально возможном перечне контролируемых переменных [19].

Сплошной веб-скрапинг часто приводит к выявлению объема данных, существенно превышающему объем самых масштабных

межстрановых социальных исследований; хранение, содержимое и обработка такого объема имеют особенности по сравнению с работой с традиционными статистическими данными. Хранение больших веб-данных организуется не как единая двумерная таблица, а как набор связанных баз, все содержимое которых нельзя увидеть одновременно и извлечение данных из которых требует знания специального языка SQL-запросов [16]. Объем веб-данных сочетается с высокой долей пропусков (50% и выше – распространенная ситуация): если опросные методы предполагают меры по предотвращению неответов респондентов, то веб-скрапинг собирает данные «как есть», поэтому веб-данные сильно «страдают» от нежелания пользователей указывать какую-либо информацию о себе, от наличия в сети «заброшенных» страниц и т.п. Объем веб-данных также накладывает ограничения на применимость к ним традиционных статистических методов. Дело в том, что при проверке статистических гипотез увеличение количества наблюдений, как правило, увеличивает вероятность отвержения нулевой гипотезы (а для большинства методов эта ситуация содержательно интерпретируется как наличие связи между изучаемыми признаками), даже если отклонение выборочной статистики от нуля совсем маленькое и содержательно ничтожное [24].

Наконец, критики оспаривают валидность веб-данных, т.е. их способность отразить измеряемый конструкт [15]. Действительно, поскольку веб-скрапинг собирает данные «как есть», то исследователь ограничен теми характеристиками людей, которые фиксируются релевантным веб-ресурсом. Поэтому возникает ощущение нарушения традиционной последовательности постановки исследования: от теории к эмпирике. В таком случае операционализировать концепты как будто бы приходится уже на имеющихся данных – такое явление иногда называют «обратной операционализацией» [14]. Нам ближе иное понимание ситуации: последовательность постановки исследования скорее не нарушается, а приобретает итеративность: исследователь ищет реле-

вантные веб-ресурсы под интересующую тему, предварительно и в очень общем виде рассмотренную сквозь призму некоторой теории; если находит такие веб-ресурсы, то соотносит фиксируемые ими характеристики пользователей с применяемой теорией и на получившемся «стыке» формулирует операциональные гипотезы. Впрочем, параллельно веб-данные чрезвычайно усилили предложенную еще Джоном Тьюки парадигму data-driven [25], отчасти противоречащую доминирующей в социологии многие десятилетия парадигме theory-driven.

Обзор путей извлечения веб-данных

Легальный и этичный веб-скрапинг имеет дело только с открытыми данными и следует указаниям, содержащимся на странице [доменное имя ресурса]/robots.txt. Физически веб-данные любого веб-ресурса – это набор связанных баз данных, хранимый на собственных или арендуемых серверах. Открытые веб-данные – это разделы соответствующих баз данных, к которым открыт доступ третьих лиц посредством сети Интернет и которые чаще всего представлены уже в понятном внешнему пользователю виде. Веб-скрапинг – это установление компьютером исследователя соединения с сервером, на котором расположены базы данных интересующего веб-ресурса, и копирование оттуда на компьютер исследователя той информации, которая в явном или неявном виде доступна по релевантным URL-адресам. Пути установления соединения и копирования различаются в зависимости от функциональных и визуальных особенностей интересующего веб-ресурса.

Бывает, что администраторы веб-ресурса не только открывают для третьих лиц какие-то разделы своих веб-данных, но и готовят инструменты, пригодные для веб-скрапинга этих данных. Такие инструменты называются API (application programming interface) и создаются не для исследователей, а для инициативных разработчиков дополнений к соответствующему веб-ресурсу (например,

для Instagram создано множество приложений-дополнений для автоматизации работы с подписчиками).

Использование API для нужд как разработки приложений, так и науки включает следующие действия:

– изучение страницы (раздела) «Разработчикам» интересующего веб-ресурса, из которой можно узнать о процедуре получения разрешения (авторизации) на использование API и о командах API (называемых методами);

– получение у администраторов веб-ресурса разрешения на извлечение данных; чаще всего это происходит путем получения автоматически сгенерированного персонального токена – набора символов;

– выбор релевантных задаче команд API и запуск их в предпочитаемой программной среде (скажем, в Питоне).

Методы API, предполагающие выгрузку некоторых данных, обычно передают эти данные в формате JSON. JSON – JavaScript Object Notation – написанный на языке программирования JavaScript порядок организации данных (преимущественно текстовых), автоматически «переводимый» на большинство других языков программирования. Так, в Питоне такой формат воспринимается как словарь со вложенными в него списками (словарь и список – классы объектов в Питоне; подробнее см. онлайн-приложение к статье) и обрабатывается функциями, применимыми к словарям и спискам.

API есть у всех крупных международных и российских социальных сетей («ВКонтакте», Facebook, Twitter, Instagram), у видеохостингов (YouTube), у почтовых сервисов (Gmail), у образовательных порталов (Coursera) и множества других сайтов. Яндекс предоставляет API для большинства своих сервисов. API каждого веб-ресурса уникальны с точки зрения авторизации, предоставляемых методов и накладываемых ограничений. Поэтому, освоив API одного веб-ресурса и написав коды для применения его методов, нельзя применить тот же код для других веб-ресурсов – в этом смысле API не универсальный путь веб-скрапинга.

Большинство веб-ресурсов не имеет API, по крайней мере доступного третьим лицам; веб-скрапинг таких ресурсов происходит через HTML-код их страниц. Веб-страницы бывают статическими и динамическими. Статическая – это такая страница, вся информация с которой доступна для обозрения и скрапинга сразу после ее загрузки; после нажатия на ссылки/кнопки на такой странице меняется адрес в адресной строке и загружается новая страница. Другими словами, в случае статической страницы ее «техническая начинка» максимально прозрачна для пользователя, поэтому доступна для «простого» пакета `requests`, пример применения которого подробно рассмотрен в следующих разделах статьи.

Динамическая – это такая страница, при загрузке которой загружается только первая порция информации; следующая порция информации загрузится только после специальной активности пользователя: прокрутки страницы, наведения мыши на нижнюю границу страницы и т.п. – при этом адрес в адресной строке не изменится; более того, при нажатии ссылок/кнопок на такой странице может поменяться некоторая ее область, а адрес в адресной строке опять же не изменится. Пример: страница Интернет-магазина, в которой при прокручивании страницы вниз отображаются новые товарные позиции, адрес остается неизменным и порой возникает ощущение, что страница бесконечна. Такие страницы выглядят более современными и продвинутыми по сравнению со статическими, но их «техническая начинка» не столь прозрачна, поэтому не полностью доступна для пакета requests; если применять его к динамической странице, то придется вручную «показывать» ему все новые области страницы – это неэффективный и не универсальный путь. Поэтому в случае динамической страницы рекомендуется перейти от пакета `requests` к более сложному для освоения и применения, но универсальному пакету `selenium`[1].

[1] Есть неофициальная традиция называть разветвленные пакеты библиотеками, а разветвленные библиотеки фреймворками; в рамках этой традиции selenium – это фреймворк.

Исходно созданный для тестирования сайтов их разработчиками, он позволяет имитировать типичные активности пользователя: открывать страницы, прокручивать их, кликать на кнопки, закрывать всплывающие окна, вбивать поисковые запросы и т.д.

Методы пакетов `requests` и `selenium`, в отличие от работы с API, передают на компьютер исследователя всю информацию со страницы: и содержательно полезную, и ее техническое обрамление (элементы кода), причем вперемешку; приходится самостоятельно структурировать эту «мешанину» и извлекать из нее содержательно полезную информацию. Пакет `selenium`, в отличие от `requests`, предполагает запись в коде имитирования активности пользователя. Получается, что путь веб-скрапинга бывает попроще, но не универсальный, или посложнее, но более универсальный. Работа с API, пакеты `requests` и `selenium` выступают тремя маркерами на континууме соотношения универсальности и сложности. Эти три наименования мы используем именно как маркеры, отдавая себе отчет в разнородности обозначаемых ими объектов. Разнородность заключается, во-первых, в том, что работа с API может быть осуществлена с помощью языка, отличного от Питона. Во-вторых, работа с API зачастую может осуществляться с помощью пакета `requests`.

Бывают и промежуточные ситуации: скажем, веб-ресурс не предоставляет API, но некоторым неочевидным образом дает доступ к своему JSON. Чтобы обнаружить это, мы рекомендуем изучить «поведение» кода страницы при ее прокрутке и нажимании кнопок вручную. Дело в том, что за изменениями визуальной оболочки страницы скрывается множество «технических» процессов. Пример: при загрузке страницы с некоторым изображением браузер отправляет запрос не только по адресу самой страницы (который затем окажется в адресной строке), но и на страницу, где отдельно размещено нужное изображение. Увидеть «технические процессы» можно через вкладку Network веб-инспектора (инструмента для знакомства с кодом страницы в браузере, работа

с которым более подробно описана в следующих разделах статьи). Среди этих процессов можно обнаружить и запросы на страницы, содержащие JSON, а обнаружив, обратиться к такой странице самостоятельно посредством пакета `requests`. На странице онлайн-приложения к статье размещена иллюстрация такого случая.

Из нашего опыта, подавляющее большинство сайтов доступны для скрапинга хотя бы одним из описанных путей, но не все (есть сайты без открытого API, фрагменты кода страниц которых обновляются при каждом перезапуске страницы). Если выбирать из трех инструментов скрапинга, маркированных нами как API, `requests` и `selenium`, то, пожалуй, requests выступает «золотой серединой», поскольку он достаточно универсален, чтобы решать большинство задач скрапинга, и при этом довольно прост в использовании; на его основе и построен наш эмпирический пример, детально рассмотренный в следующих разделах статьи.

Извлечение веб-данных через обработку HTML-кода

Итак, большинство страниц в Интернете существуют как код, написанный на языке разметки HTML (далее – HTML-код) – пример страницы и соответствующего ей фрагмента HTML-кода представлены на *рис*. 1–2. Этот код обусловливает, что мы видим на странице, как эта страница оформлена и как она взаимодействует с остальным Интернетом.

Традиционная процедура веб-скрапинга состоит в том, чтобы с помощью какого-либо языка программирования «попросить» компьютер связаться с другим компьютером (сервером), на котором хранится код интересующей веб-страницы, и загрузить этот код с того компьютера себе целиком (бывает, что не целиком, но не бывает, что сразу скопируются только нужные части кода). Затем в рамках оперативной памяти своего компьютера отбираются (и сохраняются на жесткий диск) нужные фрагменты кода. Нужные –

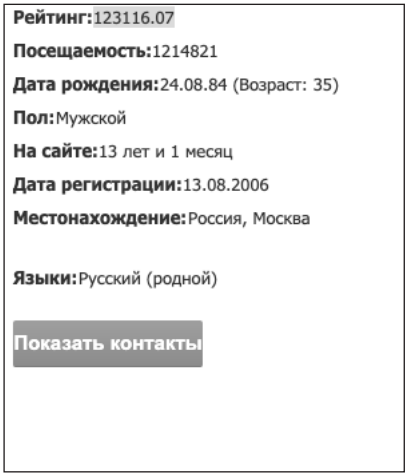

Рис. 1. Фрагмент веб-страницы с нужными данными

Рис. 2. Фрагмент HTML-кода с нужными данными

это, как правило, уникальные числа, тексты, изображения, видео и т.п., а ненужные – это как раз обрамляющие их HTML-коды. Проблема в том, что ориентирами автоматического отбора могут служить только типовые элементы кода страницы. К счастью, обычно уникальные числа и тексты встроены в типовые HTML-коды, образующие специфичные комбинации. Эти комбинации придется сначала найти «глазами», указать на них компьютеру посредством используемого языка программирования, после чего процесс продолжится автоматически.

Типовые элементы HTML-кода страницы называются тегами. Визуально на *рис*. 2 это все слова, стоящие после знака < или </. Как раз они в основном обеспечивают техническую и эстетическую стороны функционирования веб-страницы. Их очень много, но знать их названия и функции не обязательно. Главное видеть, что:

– разные теги – скажем, `<tr>` и `<td>` – разные (просто потому, что называются по-разному),

– большинство из них формируют пары («открывающий» начинается со знака <, а «закрывающий» называется точно так же, но начинается со знака </),

– они иерархически структурированы (теги, расположенные левее, выше уровнем и включают в себя теги, расположенные правее),

– некоторые из них имеют атрибуты (слова, располагающиеся после имени тега и заканчивающиеся знаком =) и значения атрибутов (слова в кавычках, располагающиеся после знака =).

Дальнейшие действия мы иллюстрируем веб-скрапингом сайта FL.ru – онлайн-биржи фриланса, облегчающей поиск удаленной работы. С него брали данные в исследовании конкурсов для фрилансеров авторы статьи «Социальные факторы выбора контрагентов на бирже удаленной работы: исследование конкурсов с помощью “больших данных”» [14]; мы частично воспроизвели их действия, а именно – сбор данных из аккаунтов пользователей сайта и оформление собранных данных в виде, пригодном для дальнейшего анализа. Для веб-скрапинга использовались реальные URL-адреса, но в тексте статьи они заменены вымышленными, чтобы сохранить конфиденциальность пользователей сайта.

Увидеть теги можно и нужно до копирования кода веб-страницы на компьютер. Как уже упоминалось, для этого в любом браузере есть инструмент, называемый веб-инспектором. Чтобы запустить его, нужно кликнуть правой кнопкой мыши на интересующем месте веб-страницы и выбрать одну из опций: «Исследовать элемент», «Проверить объект», «Проверить элемент», «Просмотреть код» и т.п. (в разных браузерах формулировки разные).

Запускаем веб-инспектор для страницы пользователя и ищем комбинацию тегов, внутри которой находится значение рейтинга пользователя. На *рис*. 1–2 слева приведен нужный фрагмент этой страницы в привычном виде, справа – его отображение в веб-инспекторе. Теги расположены иерархически: весь выбранный фрагмент кода расположен внутри открывающего тега `<table>`

и закрывающего </table>. Это первый уровень. На втором уровне (т.е. с отступом правее) расположены теги <colgroup> и <tbody> с их закрывающими парами. Рейтинг пользователя находится внутри тега <tbody>, поэтому на втором и последующем уровнях продолжаем исследовать содержимое именно этого тега. Он содержит в себе несколько тегов <tr> с закрывающими парами (третий уровень), каждый из которых содержит теги <th> и <td> с закрывающими парами (четвертый уровень). Получается, что в приведенном фрагменте к искомому рейтингу ведет комбинация тегов (называемая «путем»): <table> <tbody> <tr> <td>; порядок важен, так как структура кода иерархична. Можно ли сделать комбинацию еще более специфичной, оставаясь в пределах этих тегов? Видим, что первый нужный нам тег <table> имеет атрибут class со значением user-info-tbl и третий нужный нам тег <tr> имеет атрибут class со значением first. Поэтому уточняем комбинацию тегов: <table class = 'user-info-tbl'> <tbody> <tr class = 'first'> <td>. Из нашего опыта, это вполне специфичная комбинация, приводящая именно к искомому значению рейтинга на всех однотипных страницах пользователей сайта FL.ru. Поэтому переходим к написанию кода на Питоне для выгрузки значений рейтинга.

Для получения и структурирования HTML-кода применяются два пакета: requests и beautifulsoup4. Они не предустановлены; их, как и практически любой другой пакет, можно установить в самом Jupyter Notebook с помощью пакетного менеджера pip, по умолчанию встроенного в Питон. Например, два упомянутых пакета устанавливаются запуском следующих кодов в Jupyter Notebook (здесь и далее коды пронумерованы сверху справа – согласно порядку их описания в тексте):

```
                                                    1
!pip install requests
!pip install beautifulsoup4
```

Аналогично устанавливаются большинство широко используемых пакетов. Конкретное название для установки (например, именно `beautifulsoup4`, а не `BeautifulSoup`) обычно указано на сайте разработчика пакета[1]. Пакеты устанавливаются на жесткий диск компьютера, но для их запуска в рамках каждого файла Jupyter Notebook их «импортируют» – целиком или частично (только те функции и классы, которые необходимы для работы). В коде ниже `requests` запускается целиком, а `beautifulsoup4` – не целиком, причем в коде запуска пакет `beautifulsoup4` называется `bs4` – по прихоти разработчика:

```
                                                    2
import requests
from bs4 import BeautifulSoup
```

Если пакет запущен целиком, то для вызова из него конкретной функции вначале прописывается название самого пакета, а затем – через точку – название требуемой функции (например, `requests.get()`; функции пакета тоже указаны на сайте его разработчика). Если же пакет импортирован не целиком, то прописывается только название функции (например, `BeautifulSoup()`, а не `bs4.BeautifulSoup()`).

Как было упомянуто, существуют более сложные инструменты веб-скрапинга, например, Scrapy и Selenium, требующиеся, когда `requests` и `beautifulsoup4` не справляются (скажем, источник данных имеет слишком сложную структуру или требуется повышенная скорость выгрузки данных). Но и в их основе лежат базовые алгоритмы отправки запросов к веб-страницам и структурирования информации на них, рассматриваемые ниже.

Планируемый код Питона логически включает четыре части: 1) отправить запрос на сайт и убедиться, что запрос обрабатывается корректно, 2) извлечь HTML-код в неструктурированном

[1] Название пакета обычно и появляется первым в списке выдачи поисковика.

(«сыром») виде, 3) восстановить его в иерархическую структуру, 4) извлечь нужные данные. Такой алгоритм типичен для задачи извлечения данных из HTML-кода.

Запускаем пакет `requests` и применяем содержащуюся в нем функцию `get()`[1]:

```
                                                    3
import requests
response = requests.get('https://www.fl.ru/users/
tklopf/info/') #в скобках URL-адрес интересующей
веб-страницы
response
Out: <Response [200]>
```

Получаем ответ: «200», т.е. запрос отправлен успешно («ОК»). Будь код ответа другим, мы искали бы в Интернете причину проблемы по номеру этого кода. Коды, начинающиеся с цифр 4–5, свидетельствуют об ошибке (например, широко известный код «404» – «Страница не найдена»).

В синтаксисе во второй строке мы сохранили в оперативной памяти «сырой» код нужной веб-страницы. Для того чтобы его увидеть, достаем из переменной ее атрибут `.text`:

```
                                                    4
Out: '<!DOCTYPE HTML>\n<HTML xmlns="http://
www.w3.org/1999/xHTML">\n <head>\n   <meta
charset="utf-8" ... </HTML>\n'
```

Для экономии места приведены только начало и конец вывода. Пока Питон видит выведенный HTML-код как сплошной текст, а не как иерархическую структуру целых тегов и их содержимого. Можно было бы самим написать в Питоне код для разбиения этого текста и его структурирования, но это очень долго и трудно.

[1] Она отвечает за get-запрос. Существуют также post-, delete- и put-запросы, но они редки в практике веб-скрапинга.

Поэтому прибегаем к готовому пакету `beautifulsoup4`, а именно – к его части `BeautifulSoup`, и помещаем в нее «строку» с выгруженным HTML-кодом:

```
                                                              5
from bs4 import BeautifulSoup
soup = BeautifulSoup(response.text, 'lxml')
soup
Out: <!DOCTYPE HTML>
<HTML xmlns="http://www.w3.org/1999/xHTML">
<head>
<meta charset="utf-8"/>
...
</HTML>
```

Теперь в переменной soup хранится структурированный HTML-код. Ищем внутри него составленную ранее комбинацию тегов. Но сразу найти комбинацию нельзя, если на каком-то из уровней иерархической структуры расположено несколько одноименных тегов (независимо от наличия у них атрибутов), – как теги `<tr>` на *рис*. 2. Приходится по очереди искать каждый тег из комбинации. Для поиска первого попавшегося тега с искомым названием и всех его тезок на том же уровне иерархической структуры HTML-кода в Питоне есть функция `.find_all()`; она записывает найденные теги со всем их содержимым (теги более низких уровней и уникальные тексты и числа) в список:

```
                                                              6
table = soup.find_all('table')
table
Out: [<table class="b-layout_table b-layouttable_
width_full b-layout_table_margbot_30">
...
</table>,
...,
```

```
<table cellpadding="19" cellspacing="0" class="b-
information-clause-content" width="100%">
...
 </table>]
```

Оказалось, что в HTML-коде страницы пользователя с *рис.* 1–2 за пределами приведенного фрагмента HTML-кода встречается много тезок нужного тега `<table>`, поэтому используем знание того, что у нужного тега есть атрибут. Вносим в предыдущий код уточнение (`attrs` – аргумент функции `.find_all()`; в него помещаются необходимые атрибуты и их значения в виде словаря, ключом и значением в котором выступают название атрибута и его значение соответственно; значение атрибута не обязательно указывать целиком, если это не повредит специфичности поискового запроса):

```
                                                    7
table = soup.find_all('table', attrs={'class':
'user-info-tbl'})
table

Out: [<table class="user-info-tbl">
 <colgroup>
 <col width="170"/>
 <col/>
 <col width="20"/>
 </colgroup>
 <tbody>
 <tr class="first">
 <th>Рейтинг:</th>
 <td>123144.07</td>
...
</table>,
<table class="user-info-tbl"> #второй из тегов
<table> не содержит уникальных текстов и чисел
 <colgroup>
 <col width="170"/>
```

```
 <col/>
 <col width="20"/>
 </colgroup>
 <tbody id="contacts_info_block"></tbody>
 </table>]
```

В полученном списке всего два тега <table> с их содержимым, причем «наш» – первый[1]. Для того чтобы «добраться» до него, индексируем список:

```
                                                    8
table = table[0]
table
Out: <table class="user-info-tbl">
 <colgroup>
 <col width="170"/>
 <col/>
 <col width="20"/>
 </colgroup>
 <tbody>
 <tr class="first">
 <th>Рейтинг:</th>
 <td>123144.07</td>
...
</table>
```

В переменную table сохранен фрагмент структурированного HTML-кода, ограниченный нужным тегом <table> и его закрывающей парой.

Продолжаем поиск уже внутри этого фрагмента. Причем можно сразу перейти к поиску тега <tr>, поскольку такие теги в этом фрагменте содержатся только внутри тега <tbody> (см. *рис*. 2). Памятуя, что у тега <tr> есть тезки и что он первый среди них, используем индексирование после поиска:

[1] В этой ситуации можно было бы применить функцию .find() – более простой, но менее универсальный аналог функции .find_all().

```
                                                    9
tr_tags_raiting = table.find_all('tr')[0]
tr_tags_raiting
Out: [<tr class="first">
<th>Рейтинг:</th>
<td>123116.07</td>
<td></td>
</tr>,
...]
```

Аналогично можно добраться до последнего тега из комбинации: `<td>`. Но нам нужен не он сам, а содержащееся в нем значение рейтинга пользователя. Для извлечения данных из тегов используются две функции:

– `.get()`, если извлекаемые данные находятся внутри угловых скобок, обрамляющих тег, причем внутри одного из атрибутов этого тега; часто применяется для извлечения гиперссылки, поскольку они обычно находятся в атрибуте `<href>` тегов `<a>`[1];

– `.get_text()`, если извлекаемые данные находятся между тегом и его закрывающей парой, как в нашем случае:

```
                                                   10
rating = tr_tags_raiting.find_all('td')[0].get_
text()
rating
Out: '123116.07'
```

Выдача этой функции принадлежит к классу `str` (об этом свидетельствуют кавычки), то есть воспринимается как текст. Если это не исправить, затем к таким данным не будут применимы привычные математические операции – например, собрав данные о рейтинге всех пользователей, вычислить средний рейтинг не получится, поэтому необходимо привести выдачу к более подходящему классу:

[1] Тогда код выглядит так: `a.get('href')`.

```
                                                          11
rating = float(rating)
rating
Out: 123116.07
```

Задача извлечь рейтинг пользователя с его страницы довольно проста, если сравнивать ее с задачами, встречавшимися в нашей практике. Попробуем задачу посложнее: извлечем данные из поля «Дата рождения» (см. *рис*. 1) и разделим их на саму дату и возраст (без чего анализ извлеченных данных затруднителен).

Если снова вызвать веб-инспектор, кликнув на дате рождения пользователя на его странице, то видно, что эти данные содержатся в теге `<tr>`, расположенном несколько ниже уже извлеченного тега `<tr>` со значением рейтинга внутри. Фрагмент структурированного HTML-кода, в котором находятся только эти теги с их содержимым, был записан в переменную `tr_tags`. Если с тех пор файл Jupyter Notebook не закрывался, то этот фрагмент остается доступен. Находим в нем данные о дате рождения:

```
                                                          12
birthday = tr_tags[2].find('td').get_text()
birthday
Out: '\nДата рождения: 24.08.84 (Возраст:
35)\r\n\r\n\t\t\t\t\n\n'
```

Обнаружилось множество технических (пробельных) символов по краям нужных данных – избавляемся от них функцией `.strip()`:

```
                                                          13
birthday = birthday.strip()
birthday
Out: 'Дата рождения: 24.08.84 (Возраст: 35)'
```

Далее достаем из полученного текста возраст с помощью функций для «строк», представленных нами в приложении. Алгоритм такой: делим полученный текст по пробелам, берем по-

следний элемент получившегося списка, заменяем в нем символ «) » на пустоту и меняем получившийся результат на класс `int`:

```
                                                    14
birthday = int(birthday.split()[-1].replace(')',
'’))
birthday
Out: 35
```

Аналогично можно достать саму дату. А если «замахнуться» на все данные с *рис*. 1? В веб-инспекторе можно убедиться, что они находятся в тегах `<tr>`, следовательно, есть в переменной `tr_tags` (в ней фрагмент структурированного HTML-кода в виде списка). Для решения задачи «пройдемся» по каждому тегу из этого списка с помощью цикла, обратимся к нужному вложенному тегу `<td>` и извлечем информацию, при необходимости обработав текст по аналогии с возрастом. При этом в некоторых тегах `<tr>` нет никакой информации о пользователях, а в некоторых – лишняя для нас информация (скажем, нам неинтересны языки, которыми владеет пользователь). Чтобы отличить нужные теги `<tr>` от ненужных, нельзя ориентироваться ни на их атрибуты (за их отсутствием), ни на их порядковые номера (индексы). Дело в том, что если один пользователь заполнил, условно, 3 поля своего профиля, а другой – 4 поля, то в кодах их страниц будет разное число тегов `<tr>`, и теги, соответствующие одному и тому же полю, могут иметь разные порядковые номера. К счастью, в нашем случае есть тег `<th>`, вложенный в тег `<tr>` параллельно тегу `<td>` и содержащий названия полей. Поэтому, «проходясь» циклом по каждому тегу `<tr>`, «заходим» во вложенный в него тег `<th>` и проверяем название поля; если это поле нам интересно (например, «Рейтинг»), то «выходим» из тега `<th>` и «заходим» в параллельно вложенный в тег `<tr>` тег `<td>`, из которого уже извлекаем интересующие данные. Чтобы выполнение кода не остановилось при «заходе» в тег `<tr>`, в котором нет вообще

никакой информации о пользователе, «оборачиваем» весь цикл в конструкцию `try-except`. Наконец, чтобы в случае каких-то незаполненных полей у пользователя на экран выводилось пустое значение, перед циклом вписываем во все переменные значение `None` (напоминаем: порядок и отступы, знаки препинания и математические символы – крайне важны):

```
                                                    15
rating = None
age = None
date = None
sex = None
location = None

for i in range(len(tr_tags)):
  current_tag = tr_tags[i]
  try:
    if current_tag.find('th').get_text() ==
'Рейтинг:':
      rating = float(current_tag.find('td').get_
text())
    elif current_tag.find('th').get_text() ==
'Дата рождения:':
      age = int(current_tag.find('td').get_text().
strip().split()[-1].replace(')', ''))
    elif current_tag.find('th').get_text() ==
'Дата регистрации:':
      date = current_tag.find('td').get_text().
strip()
    elif current_tag.find('th').get_text() ==
'Пол:':
      sex = current_tag.find('td').get_text().
strip()
    elif current_tag.find('th').get_text() ==
'Местонахождение:':
```

```
      location = current_tag.find('td').get_
text().strip().split(',')[0]
  except:
    continue

print('Рейтинг пользователя:', rating)
print('Возраст пользователя:', age)
print('Дата регистрации пользователя:', date)
print('Пол пользователя:', sex)
print('Страна пользователя:', location)

Out: Рейтинг пользователя: 123116.07
Возраст пользователя: 35
Дата регистрации пользователя: 13.08.2006
Пол пользователя: Мужской
Страна пользователя: Россия
```

Последняя задача – самая сложная из рассмотренных. На страницах пользователей под полями, содержимое которых мы выгрузили, находится поле «Резюме» с текстовым описанием предлагаемых пользователем услуг (см. *рис*. 3–4). Хотя такие данные нетипичны для исследований, их использование уже набирает популярность и как источник дополнительной информации в количественном исследовании, и как предмет нового направления исследований – интеллектуального анализа текстов (text mining). Так, из описаний деятельности фрилансера можно математически вывести детерминанты успешности их авторов на онлайн-бирже. Исследовательский вопрос тогда звучал бы следующим образом: как фрилансеру позиционировать себя на бирже, чтобы увеличить вероятность поступления заказов?

Сложность выгрузки резюме пользователей сайта FL.ru обусловлена двумя особенностями его HTML-кода:

– текст находится в «трудноуловимом» теге: он неспецифичен и его атрибуты либо тоже неспецифичны (встречаются без привязки к искомым текстам: `class = "b_information_clause_`

```
▼<table width="100%" cellspacing="0" cellpadding="19" class="b-information-clause-content"> == $0
  ▼<tbody>
    ▼<tr>
      ▼<td style="padding:19px">
          <b>Профессионально разрабатываю сайты более 15 лет.</b>
          <br>
          <br>
          "✪ "
          <b>Преимущества работы со мной</b>
          <br>
          "• я не "кидаю" – за 18 лет работы я наработал хорошую репутацию в интернете "
          <br>
          "• работаю на совесть – выполняю работу качественно и ответственно "
          <br>
          "• по мимо дизайна делаю сайты под ключ и отдельно программирование "
```

Рис. 3. Фрагмент HTML-кода с «Резюме» фрилансера

Резюме

Профессионально разрабатываю сайты более 15 лет.

✪ Преимущества работы со мной
- я не "кидаю" – за 18 лет работы я наработал хорошую репутацию в интернете
- работаю на совесть – выполняю работу качественно и ответственно
- по мимо дизайна делаю сайты под ключ и отдельно программирование
- программирую на многих известных движках (WordPress, ModX, Bitrix, Shop-Script, Drupal, OpenCart, Joomla)
- программирую крупные проекты на фреймворках (Yii 2, Django, Zend, Laravel)
- берусь за доработки на движках: WordPress, ModX, Bitrix, Drupal, OpenCart, Yii 2, Django, Zend, Laravel
- Ios и Android приложения

Рис. 4. Фрагмент веб-страницы с «Резюме» фрилансера

content"), либо технические, по которым не рекомендуется осуществлять поиск тегов (width = "100%" и др.);

– фрагменты единого по смыслу текста находятся во множестве тегов: <b> и
. Это довольно частая ситуация в HTML-коде.

Впрочем, и эти трудности преодолимы:

– необходимо ориентироваться и на атрибуты тега, и на порядковый номер тега с нужными атрибутами. При этом следует на примере хотя бы нескольких страниц убедиться, что нужный тег всегда имеет один и тот же порядковый номер среди тезок. В нашем случае мы обнаружили, что нужный тег – всегда первый на странице тег <table> с атрибутом "class"="b-information-clause-content";

– функция .get_text(), примененная к тегу со вложенными в него тегами с фрагментами текста, автоматически извлекает все фрагменты текста. В нашем случае применяем эту функцию не к каждому отдельному тегу <b> или
, а к тегу уровнем выше, например к <table>.

Таким образом, получаем код для извлечения текста резюме (приводится только фрагмент вывода):

```
                                                    16
soup.find_all('table', attrs={'class':'b-
information-clause-content'})[0].get_text().
strip()
Out: 'Профессионально разрабатываю сайты более 15
лет. ... Оренбургжский Государственный Вуз и др.
'
```

Переход по страницам

Каким образом масштабировать готовый алгоритм, чтобы данные автоматически собирались со всех страниц пользователей, а не только с одной? Сначала автоматически собрать URL-адреса

страниц пользователей в, например, список, затем циклом «пройтись» по каждой странице и извлечь из нее данные с помощью уже готового алгоритма. Собрать требуемые URL-адреса – это также задача веб-скрапинга.

На сайте FL.ru есть раздел с информацией о пользователях (https://www.fl.ru/freelancers/). Он включает неизвестное заранее число страниц с 40 пользователями на каждой; на каждой странице есть ссылка на следующую страницу. Поэтому мы используем основную (первую) страницу раздела как точку входа, собираем в список все ссылки на пользователей с этой страницы, извлекаем ссылку на следующую страницу и переходим на нее[1], пополняем список ссылками с этой страницы, извлекаем ссылку на следующую страницу и т.д. до тех пор, пока существует ссылка на следующую страницу.

На *рис*. 5–6 приведены фрагмент страницы списка фрилансеров со ссылкой для перехода на страницу одного из них и соответствующий этому фрагменту HTML-код в веб-инспекторе (все персональные данные пользователя изменены на вымышленные).

На *рис*. 6 видно, что ссылка на страницу пользователя находится в атрибуте `href` тега `<a class="freelancer-name...">`, вложенного, пропуская уровень, в тег `<td class="cf-user">`. Причем эта ссылка сокращенная, т.е. без указания доменного имени сайта; при извлечении его следует добавить к ней спереди.

«Проходимся» циклом по всем тегам `<td class="cf-user">` на странице, извлекаем сокращенные ссылки на страницы пользователей, добавляем к ним доменные имена. В извлеченных ссылках после последнего слеша находятся технические символы; они не нужны, поэтому заменяем их на `/info` (именно по такому адресу располагается информация о пользователе на его странице):

[1] Такой процесс называется кроулингом (crawling).

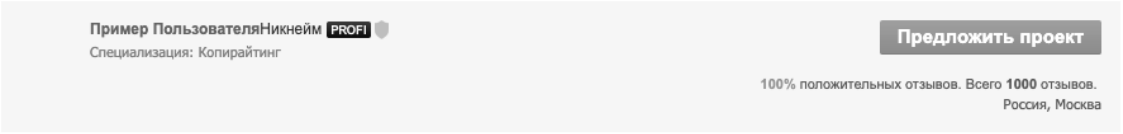

Рис. 5. Фрагмент страницы со ссылкой на пользователя

```
▼<td class="cf-user"> == $0
  <img src="https://st.fl.ru/users/User_Nickname/foto/sm_f_123456789.jpg" alt="User_Nickname" width="50" height="50" class="cf-
  avatar cf-avatar_margbot_none" border="0">
 ▼<div class="cf-user-in">
  ▶<a class="freelancer-name freelancer-catalog" onclick="yaCounter6051055reachGoal('frl_cat_ref');" data-ga-event="{ec:
  "freelance_profile", ea: "catalog_freelancer-click"}" href="/users/User_Nickname/?f=2&stamp=84985#0" title="Пример Пользователя"
  data-ga-installed="true">…</a>
  ▶<a class="b-layout__link" target="_blank" href="/profi/">…</a>
  ▶<a class="b-layout__link" href="/promo/bezopasnaya-sdelka/" title="Пользователь работал через Безопасную сделку" target=
  "_blank">…</a>
  ▶<span class="cf-spec">…</span>
  </div>
```

Рис. 6. Фрагмент HTML-кода со ссылкой на пользователя

```
                                                    17
response = requests.get('https://www.fl.ru/
freelancers/')
soup = BeautifulSoup(response.text, 'lxml')
td_tags = soup.find_all('td', attrs={'class': 'cf-
user'})

users = []
for tag in td_tags:
  link = 'https://www.fl.ru' + tag.find('a',
attrs={'class': 'freelancer-name'}).get('href')
  link = link.split('/')
  link[-1] = 'info'
  link = '/'.join(link)
  users.append(link)
print(users)

Out: ['https://www.fl.ru/users/User_Nickname_1/
info,
 'https://www.fl.ru/users/User_Nickname_2/info,
 'https://www.fl.ru/users/User_Nickname_3/info,
...,
'https://www.fl.ru/users/User_Nickname_40/info']
```

На рис. 7–8 приведены фрагменты страницы и HTML-кода соответственно – со ссылкой для перехода на следующую страницу списка фрилансеров.

Можно придумать несколько способов, как извлечь ссылку на следующую страницу. Один из них – внутри тега `<div>` с атрибутом `class="b-pager"` найти тег `<a>` с атрибутом `id="PrevLink"` и извлечь из него значение атрибута `href`, т.е. сделать ориентиром именно атрибут `id`. И хотя нужный тег `<a>` можно искать разными способами, годится только универсальный для всех дальнейших страниц, учитывая, что на них есть также кнопка для перехода на предыдущую страницу (она нас не инте-

Рис. 7. Фрагмент веб-страницы с пагинацией

Рис. 8. Фрагмент HTML-кода с пагинацией

ресует, но может находиться в похожих тегах). Описанный способ как раз универсален.

Объединяем а) формирование списка ссылок на страницы пользователей и б) извлечение ссылки для перехода на следующую страницу:

```
                                                    18
url = 'https://www.fl.ru/freelancers/' #URL-адрес
точки входа
all_users = []

while url is not None:
  response = requests.get(url)
  soup = BeautifulSoup(response.text, 'lxml')
  td_tags = soup.find_all('td', attrs={'class':
'cf-user'})

  users = []
  for tag in td_tags:
    link = 'https://www.fl.ru' + tag.find('a',
attrs={'class': 'freelancer-name'}).get('href')
    users.append(link)

  all_users.extend(users)

  url = soup.find('div', attrs={'class':
'b-pager'}).find('a', attrs={'id': 'PrevLink'}).
get('href')

Out: ['https://www.fl.ru/users/User_Nickname_1/
info,
 'https://www.fl.ru/users/User_Nickname_2/info,
 'https://www.fl.ru/users/User_Nickname_3/info,
...']
```

Итерации в коде повторяются, пока существует значение переменной `url`. Оно перестанет существовать, когда ссылка на

следующую страницу не обнаружится (т.е. когда последняя страница достигнута). Главный результат работы кода – переменная `all_users` (список ссылок на страницы всех пользователей сайта). Затем этот список можно использовать для «прохода» по всем страницам пользователей и сбора данных с них с помощью цикла `for`.

Организация данных

Если собрать данные всех пользователей сайта, получится огромная последовательность текстов и чисел. Если не привнести в нее организацию, то ее затруднительно анализировать. Нужна, например, таблица, в которой по строкам располагались бы пользователи, а по столбцам – выгруженные поля с персональной информацией. Для организации данных в виде таблицы существует мощный[1] инструмент – модуль `pandas`. Формировать таблицы в `pandas` можно поячеечно, по целым строкам или столбцам, на основе базовых структур данных – например, списка списков или словаря. Мы рассмотрим последний вариант.

Сначала необходимо получить словарь (до этого выгружаемые данные лишь выводились на экран, но никуда не сохранялись). Достраиваем написанный ранее код, чтобы он оформлял выгруженные данные о каждом пользователе в словарь, ключами которого являются названия полей персональной информации, а значениями ключей – содержимое этих полей. Число пар ключ–значение в словарях для всех пользователей должно быть одинаковым. Следовательно, в случае незаполненного поля к списку с соответствующим ключом добавляется значение `None`. Объединяем этот код с предыдущими фрагментами:

[1] Только официальная документация этого модуля содержит более 100 страниц.

```
                                                        19
data = {'rating': [], 'age': [], 'date': [],
    'sex': [], 'location': [], 'resume': []}

for user in all_users:
  response = requests.get(user)
  soup = BeautifulSoup(response.text, 'lxml')
  table = soup.find_all('table', attrs={'class':
'user-info-tbl'})[0]
  tr_tags = table.find_all('tr')

  rating = None
  age = None
  date = None
  sex = None
  location = None

  for i in range(len(tr_tags)):
    current_tag = tr_tags[i]
    try:
      if current_tag.find('th').get_text() ==
'Рейтинг:':
        rating = float(current_tag.find('td').get_
text())
      elif current_tag.find('th').get_text() ==
'Дата рождения:':
        age = int(current_tag.find('td').get_
text().strip().split()[-1].replace(')', ''))
      elif current_tag.find('th').get_text() ==
'Дата регистрации:':
        date = current_tag.find('td').get_text().
strip()
      elif current_tag.find('th').get_text() ==
'Пол:':
        sex = current_tag.find('td').get_text().
strip()
```

```
      elif current_tag.find('th').get_text() ==
'Местонахождение:':
        location = current_tag.find('td').get_
text().strip().split(',')[0]
    except:
      continue
  try:
    resume = soup.find_all('table',
attrs={'class':'b-information-clause-content'})
[0].get_text().strip()
  except:
    resume = None

  data['sex'].append(sex)
  data['date'].append(date)
  data['age'].append(age)
  data['rating'].append(rating)
  data['location'].append(location)
  data['resume'].append(resume)

print(data)

Out: {'age': [30,
33,
31,
…,
'date': ['03.08.2006',
 '10.02.2009',
 '12.02.2010',
…}
```

На выходе получаем словарь `data` с информацией обо всех пользователях сайта в разрезе пяти персональных характеристик и резюме. Теперь импортируем модуль `pandas`; он превращает ключи словаря в названия столбцов таблицы, значения ключей словаря в содержимое соответствующих столбцов, пользователей располагает

по строкам. В коде это выглядит просто как замена переменной `data` исходного класса (словарь) на класс `DataFrame`.

```
                                                                    20
import pandas as pd
data_table = pd.DataFrame(data)
data_table
```

Фрагмент вывода кода представлен на рис. 9.

	rating	age	date	sex	location		resume
0	374974.32	33.0	10.02.2009	Мужской	Россия	! ВНИМАНИЕ	НИКОГДА не платите деньги, не нап...
1	277443.21	31.0	12.02.2010	Мужской	Россия		None
2	244266.85	37.0	27.02.2008	Мужской	Россия		Работаю с 2008 года. Основные услуги:
3	211519.59	35.0	24.05.2005	Женский	Россия	БЕЗОПАСНОСТЬ	Внимание! Участились случаи мошен...

Рис. 9 Фрагмент вывода кода по преобразования словаря в таблицу

Наконец, экспортируем данные в привычный excel-файл. Файл появляется под именем, заданном в скобках, в той же папке, что и файл с кодом:

```
                                                                    21
data_table.to_excel('filename.xlsx')
```

Заключение

Статья ознакомила читателя с основами автоматического извлечения данных с веб-страниц с помощью языка программирования Python. Она содержит подробное и иллюстрированное изложение принципов работы с веб-инспектором (инструментом для поиска тегов с нужными данными) и этапов извлечения данных с отдельной страницы: отправка запроса на сайт, извлечение «сырого» HTML-кода, структурирование этого кода и навигация по тегам для извлечения нужных данных.

Все операции были рассмотрены пошагово на реальном примере; сначала «пробный» сбор данных с одной страницы

пользователя, затем сбор ссылок на все страницы пользователей и организация перехода по страницам; наконец, сбор данных со страниц всех пользователей. Результатом стала база с информацией обо всех пользователях онлайн-биржи удаленной работы в разрезе их пяти персональных характеристик и резюме.

Реальный пример включал несколько задач возрастающей сложности. В их контексте были рассмотрены типичные трудности, с которыми сталкиваются исследователи при получении веб-данных. Безусловно, не все потенциальные трудности рассмотрены: например, блокировка IP-адреса из-за большого числа запросов, необходимость регистрации для получения веб-данных с интересующего сайта, несоответствие выгруженного HTML-кода коду в веб-инспекторе. В фокусе этой статьи находится скорее подробное изложение логики извлечения данных и написания кода, а также соответствующей терминологии. Однако мы убеждены, что представленный материал составит крепкую базу тем исследователям, которые попытаются освоить веб-скрапинг, – такую, что они затем самостоятельно смогут найти ответы на вопросы, не освещенные в этой статье.

ЛИТЕРАТУРА

1. Computational Social Science / D. Lazer, A. Pentland, L. Adamic [et al.] // Science. 2009. Vol. 323. No. 5915. P. 721–723. DOI: 10.1016/j.jocs.2010.12.007

2. *Bainbridge W.S.* The Scientific Research Potential of Virtual Worlds // Science. 2007. Vol. 317. No. 5837. P. 472–476. DOI: 10.1126/science.1146930

3. *Watts D.J.* A Twenty-first Century Science // Nature. 2007. Vol. 445. No. 7127. P. 489. DOI: 10.1038/445489a

4. More Tweets, More Votes: Social Media as a Quantitative Indicator of Political Behavior / J. DiGrazia, K. McKelvey, J. Bollen, F. Rojas // PLoS ONE. 2013. Vol. 8. No. 11. Art. no. e79449. DOI: 10.1371/journal.pone.0079449

5. *Gayo-Avello D.* A Meta-Analysis of State-of-the-Art Electoral Prediction From Twitter Data // Social Science Computer Review. 2013. Vol. 31. No. 6. P. 649–679. DOI: 10.1177/0894439313493979

6. *Jungherr A.* Tweets and Votes, a Special Relationship: the 2009 Federal Election in Germany // Proceedings of the 2nd Workshop on Politics, Elections and Data – PLEAD'13. New York, 2013. P. 5–14. DOI: 10.1145/2508436.2508437

7. Beyond Binary Labels: Political Ideology Prediction of Twitter Users / D. Preoţiuc-Pietro, Y. Liu, D. Hopkins, L. Ungar // Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. 2017. No. 1. P. 729–740. DOI: 10.18653/v1/P17-1068

8. Cross-platform and Cross-interaction Study of User Personality Based on Images on Twitter and Flickr / Z.R. Samani, S.C. Guntuku, M.E. Moghaddam [et al.] // PLoS ONE. 2018. Vol. 13. No. 7. Art. no. e0198660. DOI: 10.1371/journal.pone.0198660

9. *Chittaranjan G.* Who's Who with Big-Five: Analyzing and Classifying Personality Traits with Smartphones / G. Chittaranjan, J. Blom, D. Gatica-Perez // 2011 15th Annual International Symposium on Wearable Computers. 2011. P. 29–36. DOI: 10.1109/iswc.2011.29

10. *Kosinski M.* Private Traits and Attributes are Predictable from Digital Records of Human Behavior / M. Kosinski, D. Stillwell, T. Graepel // Proceedings of the National Academy of Sciences. 2013. Vol. 110. No. 15. P. 5802–5805. DOI: 10.1073/pnas.1218772110

11. Automatic Ppersonality Aassessment through Social Media Language / G. Park, H. Schwartz, J. Eichstaedt [et al.] // Journal of Personality and Social Psychology. 2015. Vol. 108. No. 6. P. 934–952. DOI: 10.1037/pspp0000020

12. *Smirnov I.* Schools are Segregated by Educational Outcomes in the Digital Space // PLoS ONE. 2019. Vol. 14. No. 5. P. 1–9. DOI: 10.1371/journal.pone.0217142

13. *Смирнов В.* Новые компетенции социолога в эпоху больших данных // Мониторинг общественного мнения: экономические и социальные перемены. 2015. № 2. С. 44–54. DOI: 10.14515/monitoring.2015.2.04

14. Социальные факторы выбора контрагентов на бирже удаленной работы: исследование конкурсов с помощью «больших данных» / Д.О. Стребков, А.В. Шевчук, А.А. Лукина [и др.] // Экономическая социология. 2019. Т. 20. № 3. С. 25–65. DOI: 10.17323/1726-3247-2019-3-25-65

15. *Golder S.A.* Digital Footprints: Opportunities and Challenges for Online Social Research / S.A. Golder, M.W. Macy // Annual Review of Sociology. 2014. Vol. 40. No. 1. P. 129–152. DOI: 10.1146/annurev-soc-071913-043145

16. Distilling Digital Traces: Computational Social Science Approaches to Studying the Internet / H. Wesler, M. Smith, D. Fisher, E. Gleave // The Sage Handbook of Online Research Methods. London: SAGE, 2008. P. 116–40.

17. *Hampton K.N.* Studying the Digital: Directions and Challenges for Digital Methods // Annual Review of Sociology. 2017. Vol. 43. No. 1. P. 167–188. DOI: 10.1146/annurev-soc-060116-053505

18. *Grimmer J.* Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts / J. Grimmer, B.M. Stewart // Political Analysis. 2013. Vol. 21. No. 3. P. 267–297. DOI: 10.1093/pan/mps028

19. Social Media Analyses for Social Measurement / M.F. Schober, J. Pasek, L. Guggenheim [et al.] // Public Opinion Quarterly. 2016. Vol. 80. No. 1. P. 180–211. DOI: 10.1093/poq/nfv048

20. Social Networking Sites and Our Lives: How People's Trust, Personal Relationships, and Civic and Political Involvement are Connected to Their Use of Social Networking Sites and Other Technologies / K.N. Hampton, L.S. Goulet, L. Rainie, K. Purcell. Washington, DC: Pew Research Cent, 2011.

21. A Path to Understanding the Effects of Algorithm Awareness / K. Hamilton, K. Karahalios, C. Sandvig, M. Eslami // Proceedings of the Extended Abstracts of the 32nd Annual ACM Conference on Human Factors in Computing Systems – CHI EA'14. April 26 – May 1, 2014. Toronto, Ontario, Canada. 2014. P. 631–642. DOI: 10.1145/2559206.2578883

22. *Rader E.* Examining User Surprise as a Symptom of Algorithmic Filtering // International Journal of Human-Computer Studies. 2017. No. 98. P. 72–88. DOI: 10.1016/j.ijhcs.2016.10.005

23. *Tourangeau R.* The Psychology of Survey Response / R. Tourangeau, L.J. Rips, K. Rasinski. Cambridge: Cambridge Univ. Press, 2000.

24. *Lin M.* Research Commentary – Too Big to Fail: Large Samples and the *p*-Value Problem / M. Lin, H.C. Lucas, G. Shmueli // Information Systems Research. 2013. Vol. 24. No. 4. P. 906–917. DOI: 10.1287/isre.2013.0480

25. *Tukey J.* Exploratory Data Analysis. London: Pearson, 1977.

Zhuchkova Svetlana,
National Research University Higher School of Economics (NRU HSE), Moscow, szhuchkova@hse.ru

Rotmistrov Alexey,
National Research University Higher School of Economics (NRU HSE), Moscow, arotmistrov@hse.ru

Automatic extraction of text and numeric web data for social science purposes

The paper is devoted to the procedures of automatic data extraction from web pages, i.e., web scraping of web data. We consider different types of web data such as digital traces and other numeric and text web data as well as its advantages (the speed of data collection and, as a consequence, the continuous coverage, efficiency, etc.) and limitations (the limited representativeness, difficulties in organizing storage of a large amount of data, deviation from the traditional procedure for setting up a study, etc.) in comparison with traditional methods of data collection. Various tools of web data extraction (API, requests, and selenium) are described to illustrate principles of handling static and dynamic web pages. The paper also gives an overview of the basic minimum of competencies for web scraping: in particular, programming using Python and navigating though the web pages' code. A detailed illustration is given based on a fragment of the data collection process from a recent relevant Russian study.
Keywords: automatic data extraction, big data, web data, text data, web scraping, computational social science, API, requests, selenium

References

1. Lazer D. et al. Computational social science, *Science*, 2009, 323, 721–723.
2. Bainbridge W. S. (2007) The Scientific Research Potential of Virtual Worlds, *Science*, 317 (5837), 472–476.
3. Watts D. J. A twenty-first century science, *Nature*, 2007, 445 (7127). P. 489.
4. DiGrazia J. et al. More Tweets, More Votes: Social Media as a Quantitative Indicator of Political Behavior, *PLoS ONE*, 2013, 8 (11), e79449
5. Gayo-Avello D. A Meta-Analysis of State-of-the-Art Electoral Prediction From Twitter Data, *Social Science Computer Review*, 2013, 31 (6), 649–679.

6. Jungherr A. Tweets and votes, a special relationship. *Proceedings of the 2nd Workshop on Politics, Elections and Data - PLEAD'13,* 2013. P. 5–14.
7. Preoţiuc-Pietro D. et al. Beyond binary labels: political ideology prediction of Twitter users. *Proceedings of the 55th annual meeting of the Association for Computational Linguistics*, 2017, 1, 729–740.
8. Samani Z. R. et al. Cross-platform and cross-interaction study of user personality based on images on Twitter and Flickr, *PLoS ONE*, 2018, 13 (7), e0198660.
9. Chittaranjan G., Blom J., Gatica-Perez D. Who's Who with Big-Five: Analyzing and Classifying Personality Traits with Smartphones, *15th Annual International Symposium on Wearable Computers*, 2001. P. 29–36.
10. Kosinski M., Stillwell D., Graepel T. Private traits and attributes are predictable from digital records of human behavior, *Proceedings of the National Academy of Sciences*, 2013, 110(15), 5802–5805.
11. Park G. et al. Automatic personality assessment through social media language, *Journal of Personality and Social Psychology*, 2015, 108 (6), 934–952.
12. Smirnov I. Schools are segregated by educational outcomes in the digital space, PLoS ONE, 2019, 14 (5), 1–9.
13. Smirnov V. New Competencies of a Sociologist in the Era of "Big Data" (in Russian). *Monitoring obshchestvennogo mneniya: ekonomicheskie i sotsial`nye peremeny / Monitoring of Public Opinion: Economic and Social Changes Journal,* 2015, 2, 44–54.
14. Strebkov D. et al. Social Factors of Contractor Selection on Freelance Online Marketplace: Study of Contests Using "Big Data" (in Russian), *Ekonomicheskaya sotsiologiya / Journal of economic sociology*, 2019, 20 (3), 25–66.
15. Golder S. A., Macy M. W. Digital Footprints: Opportunities and Challenges for Online Social Research, *Annual Review of Sociology*, 2014, 40 (1), 129–152.
16. Wesler H., Smith M., Fisher D., Gleav, E. Distilling digital traces: computational social science approaches to studying the Internet. *The Sage Handbook of Online Research Methods.* London: SAGE, 2008. P. 116–40.
17. Hampton K. N. Studying the Digital: Directions and Challenges for Digital Methods, *Annual Review of Sociology,* 2017, 43 (1), 167–188.

18. Grimmer J., Stewart B. M. Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts, Political Analysis, 2013, 21 (03), 267–297.
19. Schober M. F. et al. Social Media Analyses for Social Measurement, *Public Opinion Quarterly*, 2016, 80 (1), 180–211.
20. Hampton K.N. et al. *Social Networking Sites and Our Lives: How People's Trust, Personal Relationships, and Civic and Political Involvement are Connected to Their Use of Social Networking Sites and Other Technologies*. Washington, DC: Pew Research Cent, 2011.
21. Hamilton K. et al. A path to understanding the effects of algorithm awareness. *Proceedings of the Extended Abstracts of the 32nd Annual ACM Conference on Human Factors in Computing Systems*. 2014. P. 631–642.
22. Rader E. Examining user surprise as a symptom of algorithmic filtering, *International Journal of Human-Computer Studies*, 2017, 98, 72–88.
23. Tourangeau R., Rips L.J., Rasinski K. *The Psychology of Survey Response*. Cambridge: Cambridge University Press, 2000.
24. Lin M. at al. Research Commentary—Too Big to Fail: Large Samples and the p-Value Problem, *Information Systems Research*, 2013, 24 (4), 906–917.
25. Tukey J. *Exploratory data analysis*. London: Pearson, 1977.