Лабораторная работа №1. Загрузка данных

С.А.Суязова (Аксюк), sa_aksyuk@guu.ru
02 сен, 2021

Table of Contents

Ключевые слова: R[1], r-project, RStudio

Примеры выполнены R версии 4.1.1, «Kick Things».

Версия RStudio: 1.4.1717.

Все ссылки действительны на 8 февраля 2021 г.

Репозиторий с материалами к курсу: github.com/aksyuk/R_data_glimpse

Файл с макетом кода для этой практики: `.Labs/lab-01_before.R`

[1] R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

Лабораторная работа №1: загрузка данных из .csv и с помощью API сайтов

Несколько полезных привычек

В любой работе есть важные вещи, которые быстро становятся рутиной. В аналитике это работа с данными на начальном этапе, ещё до того, как нам представится случай проявить своё творческое начало в визуализации переменных и в интерпретации того, что мы видим. Часто загрузка данных – это рутина, но если к этому этапу отнестить неаккуратно, в ходе анализа можно создать себе неприятные проблемы. Поэтому специалисту по работе с данными лучше сразу формировать у себя полезные привычки, которые, к слову, пригодятся не только в R.

Одно из преимуществ R как языка обработки данных – воспроизводимость результатов. Технически, если код описывает всё, что происходит на разных стадиях исследования, от их загрузки до генерации отчёта, то любой пользователь может выполнить его на своём компьютере и получить такой же отчёт. Разумеется, для этого необходимо выполнение ряда условий, в частности, инструкции должны быть универсальными, а источники данных и пакеты, использованные для их обработки, – открытыми. В этой практике мы рассмотрим несколько вариантов загрузки данных из открытых источников. При этом будем придерживаться нескольких простых правил, которые помогают избежать многих проблем.

Не задавайте явно рабочую директорию. Пока код не является самодостаточным пакетом или отдельным приложением, мы считаем, что он адресован пользователям, знакомым с азами R. Рабочую директорию адресат задаёт без нашего участия, и её имя и расположение, очевидно, не совпадут с нашими.

Однако, *сохраняйте данные в отдельную директорию внутри рабочей*. Это поможет отделить «сырые» данные и сохранить их на случай, если не будет возможности перезагрузить файл. Далее во всех примерах данные загружаются в папку «data» внутри рабочей директории.

Сохраняйте время и дату загрузки. Это облегчает контроль версий и просто даёт представление о том, как давно всё произошло. Далее в примерах эта информация сохраняется в текстовом файле внутри директории «data».

Снабжайте данные описанием. Обычно после предварительной обработки, которая может включать переименование столбцов, заполнение пропусков, изменение макета таблицы, файл данных отличается от исходного. В этом случае хорошим тоном будет составить короткий справочник с описанием проделанных трансформаций и с итоговым списком переменных (столбцов), с обязательным указанием их единиц измерения. В англоязычных источниках такой справочник носит название «code book», дословно – кодовая книга. Его назначение в том, чтобы составленная вами таблица данных не превращалась для стороннего человека в шифровку. Подобными кодовыми книгами, или справочниками, как они будут называться ниже, снабжены все встроенные в R наборы данных. Чтобы убедиться в этом, достаточно вызвать справку по файлу данных, например: `?mtcars` или `?iris`.

Наработав собственный опыт, читатель, разумеется, прибавит к этому минимальному списку свои правила, которые, при должной практике, превратятся в полезные привычки. Главная цель здесь, с одной стороны, настроить рабочее окружение под себя, а с другой – в случае совместной работы над проектом свести необходимость дополнительных разъяснений к минимуму.

Загрузка файла .csv из сети

Пример №1. Первый и самый простой способ получить данные – загрузить их в виде файла с известного адреса. Загрузим таблицу со статистикой импорта сливочного масла в РФ за 2010-2018 гг. Источник данных – база UN COMTRADE (http://comtrade.un.org/data/). Данные сохранены в репозитории на github.com и доступны по ссылке: https://raw.githubusercontent.com/aksyuk/R-data/master/COMTRADE/040510-Imp-RF-comtrade.csv.

Создадим директорию `data` внутри рабочей директории с помощью функции `dir.create()` и файл для записи лога загрузок с помощью функции `file.create()`. Чтобы не перезаписывать их при повторных прогонах кода, добавим проверку условия. При загрузке и чтении данных полезны следующие функции R:

- `file.exists('`*путь_к_файлу*`')` возвращает `TRUE`, если указанный файл существует, и `FALSE` в противном случае;
- `exists('`*имя_объекта*`')` возвращает `TRUE`, если указанный объект существует в рабочем пространстве R, и `FALSE` в противном случае.

Проверка условия существования файла (объекта) перед загрузкой (чтением) существенно экономит время при работе с таблицами большой размерности.

```
# создаём директорию для данных, если она ещё не существует:
data.dir <- './data'
if (!file.exists(data.dir)) dir.create(data.dir)
# создаём файл с логом загрузок, если он ещё не существует:
log.filename <- './data/download.log'
if (!file.exists(log.filename)) file.create(log.filename)
```

На этапе непосредственной загрузки файла используем функции:

- `download.file(*URL_файла*, *имя_файла_для_сохранения*)` загружает файл и сохраняет под указанным именем. Второй аргумент может быть именем файла, тогда он будет сохранён в рабочую директорию, а также абсолютным или относительным путём к файлу. Мы используем относительную ссылку на директорию с данными: `./data/*имя_файла*`, где точка означает «в текущей (рабочей) директории».
- `write(*текст_для_записи*, file = *имя_файла*, append = TRUE)` записывает текст в указанный файл. Аргумент `append = TRUE` означает, что новая строка будет добавлена в конец файла.

```
# адрес файла
fileURL <- 'https://raw.githubusercontent.com/aksyuk/R-
data/master/COMTRADE/040510-Imp-RF-comtrade.csv'
dest.file <- './data/040510-Imp-RF-comtrade.csv'
```

```
# загружаем файл, если он ещё не существует, и делаем запись о загрузке в лог:
if (!file.exists(dest.file)) {
    # загрузить файл
    download.file(fileURL, dest.file)
    # сделать запись в лог
    write(paste('Файл', dest.file, 'загружен', Sys.time()),
          file = log.filename, append = T)
}
```

Наконец, чтение данных из загруженного файла во фрейм и просмотр содержимого. Помните, что в случае если таблица содержит текстовые переменные, R автоматически сделает их факторами, присвоив каждому уникальному текстовому значению порядковый номер. Если требуется прочесть заранее неизвестную таблицу, нужно запретить такое преобразование, указав в функции `read.csv()` аргумент `stringsAsFactors = FALSE`. Буквально это будет означать: не делать из символьных столбцов факторы.

```
# читаем данные из загруженного .csv во фрейм,
#  если он ещё не существует
if (!exists('DF.import')) {
    DF.import <- read.csv(dest.file, stringsAsFactors = F)
}
# предварительный просмотр
dim(DF.import)     # размерность таблицы
str(DF.import)     # структура (характеристики столбцов)
head(DF.import)    # первые несколько строк таблицы
tail(DF.import)    # последние несколько строк таблицы
```

Это простой пример, поскольку загруженная таблица уже очищена от пустых столбцов и приведена к аккуратному виду, то есть:

- каждая строка содержит одно наблюдение;
- каждому столбцу соответствует одна переменная;
- каждый тип наблюдений (объектов) хранится в отдельной таблице.

Это список требований к тому, что принято называть «tidy data», или дословно – «аккуратные данные». Термин предложен Хэдли Уикхэмом в одноимённой статье в 2014 году[2]. В дополнение к этому, по заголовкам столбцов этого файла можно понять, что за переменные в нём содержатся. Для данных уже написан справочник: https://github.com/aksyuk/R-data/blob/master/COMTRADE/CodeBook_040510-Imp-RF-comtrade.md.

Подчеркнём, что данные примера №1 уже подверглись предварительной обработке. Мы вернёмся к этому примеру в разделе «Очистка и предобработка данных», чтобы подробно поговорить об этом.

---

[2] R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

Загрузка данных с помощью API

Сайт статистики международной торговли comtrade.un.org/

Некоторые сайты предоставляют разработчикам API (application programming interface) – интерфейсы программирования приложений. У R есть средства для работы с API: twitter.com (пакет «twitteR» [3]), facebook.com («Rfacebook» [4]), quandl.com («Quandl» [5]), finance.yahoo.com («quantmod» [6]) и многих других. Некоторые сайты сами выкладывают API для R.

Пример №2. Данные из примера №1 по импорту масла в РФ можно получить двумя способами:

- Сделав запрос к базе UN COMTRADE через форму на веб-странице: http://comtrade.un.org/data/.
- Воспользовавшись API для R, описание которого приводится по адресу: http://comtrade.un.org/data/Doc/api/ex/r.

На странице API для R сайта UN COMTRADE приводятся коды пользовательских функций для формирования запроса и примеры использования этих функций. Как и всякий API, этот имеет свои ограничения:

- Не более одного запроса в секунду с одного IP адреса.
- Не более 100 запросов в час с одного IP адреса.
- API находится в режиме разработки и может быть изменён.

Используем API UN COMTRADE чтобы извлечь данные, которые лежат в основе первого примера. База данных по умолчанию выдаёт результаты в формате JSON[7], для работы с которым из R нужна библиотека `rjson` [8]. Для начала найдём код Российской Федерации в справочнике UN COMTRADE.

---

[3] Jeff Gentry (2015). twitteR: R Based Twitter Client. R package version 1.1.9. https://CRAN.R-project.org/package=twitteR.

[4] Pablo Barbera and Michael Piccirilli (2015). Rfacebook: Access to Facebook API via R. R package version 0.6. https://CRAN.R-project.org/package=Rfacebook.

[5] Raymond McTaggart, Gergely Daroczi and Clement Leung (2015). Quandl: API Wrapper for Quandl.com. R package version 2.7.0. https://CRAN.R-project.org/package=Quandl.

[6] Jeffrey A. Ryan (2015). quantmod: Quantitative Financial Modelling Framework. R package version 0.4-5. https://CRAN.R-project.org/package=quantmod.

[7] JSON (JavaScript Object Notation) – простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером.

[8] Введение в JSON. URL: http://www.json.org/json-ru.html.

```
# библиотека для работы с JSON
library('rjson')
# адрес справочника по странам UN COMTRADE
fileURL <- "http://comtrade.un.org/data/cache/partnerAreas.json"
# загружаем данные из формата JSON
reporters <- fromJSON(file = fileURL)
is.list(reporters)
#> [1] TRUE

# соединяем элементы списка построчно
reporters <- t(sapply(reporters$results, rbind))
dim(reporters)
#> [1] 294   2

# превращаем во фрейм
reporters <- as.data.frame(reporters)
head(reporters)
#>    V1                         V2
#> 1 all                        All
#> 2   0                      World
#> 3   4                Afghanistan
#> 4 472 Africa CAMEU region, nes
#> 5   8                    Albania
#> 6  12                    Algeria

# даём столбцам имена
names(reporters) <- c('State.Code', 'State.Name.En')
# находим РФ
reporters[reporters$State.Name.En == 'Russian Federation', ]
#>     State.Code      State.Name.En
#> 219        643 Russian Federation
```

Итак, код РФ в базе: 643. У базы UN COMTRADE есть ещё одно ограничение: при выборе ежемесячных данных один запрос может охватывать максимум год. Поэтому чтобы собрать данные по месяцам с 2010 по 2018 гг., нужно сделать пять запросов. Для формирования запроса за 2010 год воспользуемся функцией get.Comtrade(), сохранённой по адресу: https://raw.githubusercontent.com/aksyuk/R-data/master/API/comtrade_API.R.

```
# функция, реализующая API (источник: UN COMTRADE)
source("https://raw.githubusercontent.com/aksyuk/R-
data/master/API/comtrade_API.R")
# ежемесячные данные по импорту масла в РФ за 2010 год
# 040510 – код сливочного масла по классификации HS
s1 <- get.Comtrade(r = 'all', p = "643",
                   ps = as.character(2010), freq = "M",
                   rg = '1', cc = '040510',
                   fmt = "csv")
dim(s1$data)
#> [1] 22 35
is.data.frame(s1$data)
#> [1] TRUE
```

Код выше загружает в рабочее пространство R функцию get.Comtrade() с известного URL, а затем вызывает её с аргументами: * `r` – страны, подавшие отчёт о поставке, 'all' означает выбор всех поставщиков товара в РФ;

- `p` – партнёр, в данном случае Российская Федерация;
- `ps` – период времени, здесь 2010 год в символьком формате;
- `freq` – частота, в данном случае 'M' означает ежемесячные данные;
- `rg` – код торгового потока, здесь '1' – это импорт;
- `cc` – код товара по гармонизированной классификации, '040510' – сливочное масло;
- `fmt` – формат, в котором выдаются данные, здесь – «csv».

Объект `s1` хранит данные (`s1$fdata`) и результаты проверки запроса (`s1$validation`). Данные представляют собой объект типа `data.frame` с 10 строками и 35 столбцами. Стоит сразу сохранить результаты запроса на диск, чтобы обращаться к ним в любое время без ограничений, которыми обладает API.

```
# записываем выборку за 2010 год в файл
write.csv(s1$data, './data/comtrade_2010.csv', row.names = F)
```

Загрузку данных за все годы можно сделать в цикле. Перебор файлов или URL – задачи, в которых применение циклов в R оправдано.

```
# загрузка данных в цикле
for (i in 2011:2019) {
    # таймер для ограничения API: не более запроса в секунду
    Sys.sleep(5)
    s1 <- get.Comtrade(r = 'all', p = "643",
                       ps = as.character(i), freq = "M",
                       rg = '1', cc = '040510',
                       fmt="csv")
    # имя файла для сохранения
    file.name <- paste('./data/comtrade_', i, '.csv',
                       sep = '')
    # записать данные в файл
    write.csv(s1$data, file.name, row.names = F)
    # вывести сообщение в консоль
    print(paste('Данные за', i, 'год сохранены в файл',
                file.name))
    # сделать запись в лог
    write(paste('Файл',
                paste('comtrade_', i, '.csv', sep = ''),
                'загружен', Sys.time()),
          file = './data/download.log', append = T)
}
```

Итак, с помощью API базы данных международной торговли UN COMTRADE мы загрузили ежемесячные данные по импорту сливочного масла в РФ и сохранили их в папку «data» внутри рабочей директории в отдельных файлах для каждого года, с 2010 по 2019.

Сайт Всемирного банка, база данных показателей развития стран мира (World Development Indicators)

https://databank.worldbank.org/source/world-development-indicators

Для загрузки данных через API базы Всемирного банка в R существует пакет `WDI`. **Пример №3**: воспользуемся пакетом `WDI`, чтобы получить статистику по индексу лёгкости ведения бизнеса и ряду показателей, которые с ним связаны. В базе данных, с которой мы будем работать, у каждого показателя есть свой буквенный код. Полный список доступных показателей по странам можно посмотреть по ссылке: https://data.worldbank.org/indicator. При нажатии на показатель в списке открывается его страница, в адресе которой содержится код. Например, ссылка на страницу с показателем "Agricultural irrigated land (% of total agricultural land)" (сельскохозяйственные ирригируемые земли, процент от площади возделываемых земель) выглядит так: https://data.worldbank.org/indicator/AG.LND.IRIG.AG.ZS?view=chart. Код показателя стоит между последней косой чертой и знаком вопроса: `AG.LND.IRIG.AG.ZS`.

Загрузим следующие индексы:

- `NY.GDP.PCAP.CD` – валовой внутренний продукт на душу населения в текущих ценах, долларов США (GDP per capita, current US$);
- `IC.REG.COST.PC.ZS` – затраты на создание бизнеса, % от валового национального дохода на душу населения (Cost of business start-up procedures, % of GNI per capita);
- `IC.REG.DURS` – время на создание бизнеса, дней (Time required to start a business, days);
- `IC.TAX.TOTL.CP.ZS` – налоговая нагрузка на бизнес, % от прибыли (Total tax and contribution rate, % of profit);
- `IC.TAX.DURS` – время на выплату налогов, часов (Time to prepare and pay taxes, hours);
- `IC.BUS.EASE.XQ` – рейтинг стран по лёгкости ведения бизнеса (Ease of doing business index).

Загрузим значения показателей за 2019 год.

```
# загрузка пакета
library('WDI')

# коды и названия показателей по странам
ind.names <- c('NY.GDP.PCAP.CD', 'IC.REG.COST.PC.ZS',
               'IC.REG.DURS', 'IC.TAX.TOTL.CP.ZS',
               'IC.TAX.DURS', 'IC.BUS.EASE.XQ')
ind.labels <- c('ВВП на душу, текущие цены, USD',
                'Затраты на создание бизнеса, % от ВНД на душу',
                'Время на создание бизнеса, дней',
                'Налоговая нагрузка на бизнес, % от прибыли',
                'Время на выплату налогов, часов',
                'Рейтинг лёгкости ведения бизнеса')
```

```
# скачиваем данные за 2019 год
df.wdi.2019 <- WDI(country = 'all', indicator = ind.names,
                   start = 2019, end = 2019)
# смотрим первые строки таблицы
head(df.wdi.2019)
#>   iso2c                                      country year NY.GDP.PCAP.CD
#> 1    1A                                 Arab World 2019       6570.174
#> 2    1W                                      World 2019      11417.155
#> 3    4E   East Asia & Pacific (excluding high income) 2019       8194.265
#> 4    7E Europe & Central Asia (excluding high income) 2019       8337.857
#> 5    8S                                 South Asia 2019       1959.342
#> 6    AD                                    Andorra 2019      40897.331
#>   IC.REG.COST.PC.ZS IC.REG.DURS IC.TAX.TOTL.CP.ZS IC.TAX.DURS IC.BUS.EASE.XQ
#> 1          27.79545    19.65455          42.51429    202.1905             NA
#> 2          19.77539    19.59738          40.38368    232.8908             NA
#> 3          21.40500    29.73000          33.76000    195.1000             NA
#> 4           3.56000    11.78000          32.54000    226.2250             NA
#> 5           8.26250    14.56250          43.86250    273.5475             NA
#> 6                NA          NA                NA          NA             NA
```

Все данные относятся к одному и тому же году, поэтому можно убрать столбец `year`.

```
# выбрасываем столбец с годом
df.wdi.2019 <- df.wdi.2019[, colnames(df.wdi.2019) != 'year']
```

Как видно по столбцу `country`, в таблице присутствуют не только страны, но и группы стран по уровню дохода и географическому соседству. Чтобы сделать статистику сопоставимой, уберём эти укрупнённые регионы. Пакет `WDI` содержит метаданные: например, все коды стран в базе можно найти в элементе списка `WDI_data` под названием `country`. Посмотрим его содержимое.

```
# смотрим список всех стран, чтобы убрать строки макрорегионов
meta.data <- as.data.frame(WDI_data$country)
head(meta.data)
#>   iso3c iso2c     country                   region          capital longitude
#> 1   ABW    AW       Aruba Latin America & Caribbean       Oranjestad  -70.0167
#> 2   AFG    AF Afghanistan               South Asia            Kabul   69.1761
#> 3   AFR    A9      Africa               Aggregates                           
#> 4   AGO    AO      Angola        Sub-Saharan Africa           Luanda    13.242
#> 5   ALB    AL     Albania     Europe & Central Asia           Tirane   19.8172
#> 6   AND    AD     Andorra     Europe & Central Asia Andorra la Vella    1.5218
#>   latitude              income        lending
#> 1  12.5167         High income Not classified
#> 2  34.5228          Low income            IDA
#> 3                   Aggregates     Aggregates
#> 4 -8.81155 Lower middle income           IBRD
#> 5  41.3317 Upper middle income           IBRD
#> 6  42.5075         High income Not classified

# смотрим значения классов стран
table(meta.data$region)     # по географическому расположению
#>
#>                 Aggregates        East Asia & Pacific
```

```
#>                          86                          38
#>      Europe & Central Asia  Latin America & Caribbean
#>                          58                          42
#> Middle East & North Africa              North America
#>                          21                           3
#>                  South Asia         Sub-Saharan Africa
#>                           8                          48
table(meta.data$income)     # по уровню дохода
#>
#>          Aggregates         High income          Low income Lower middle income
#>                  86                  80                  31                  47
#> Upper middle income
#>                  60
```

Строки со статистикой по макрорегионам закодированы в столбце `region` как `'Aggregates'`. Уберём их из таблицы с данными, используя фильтр по кодам из столбца `iso2c`.

```
# оставляем в таблице с показателями только страны
all.countries <- meta.data[meta.data$region != 'Aggregates',
                           ]$iso2c
df.wdi.2019 <- df.wdi.2019[df.wdi.2019$iso2c %in% all.countries, ]
```

Чтобы строить графики, искать корреляции и строить модели по собранным данным, нужно избавиться от пропущенных значений. Подсчитаем количество `NA` в каждом столбце.

```
# считаем пропуски в столбцах
sapply(df.wdi.2019, function(x){sum(is.na(x))})
#>             iso2c           country    NY.GDP.PCAP.CD IC.REG.COST.PC.ZS
#>                 0                 0                18                27
#>       IC.REG.DURS IC.TAX.TOTL.CP.ZS       IC.TAX.DURS    IC.BUS.EASE.XQ
#>                27                28                28                28
# то же как доля от всех стран
round(sapply(df.wdi.2019, function(x){sum(is.na(x))}) /
          nrow(df.wdi.2019), 2)
#>             iso2c           country    NY.GDP.PCAP.CD IC.REG.COST.PC.ZS
#>              0.00              0.00              0.08              0.12
#>       IC.REG.DURS IC.TAX.TOTL.CP.ZS       IC.TAX.DURS    IC.BUS.EASE.XQ
#>              0.12              0.13              0.13              0.13
```

Как видно, число пропусков не превышает 13%. Однако это пропуски по отдельным столбцам, а нас интересуют страны, по которым присутствуют все показатели. Посчитаем, сколько строк выпадает из-за того, что в них есть `NA` хотя бы по одному столбцу.

```
# считаем строки, в которых есть пропуск хотя бы в одном столбце
sum(apply(df.wdi.2019, 1, function(x){any(is.na(x))}))
#> [1] 34
# то же как доля от всех стран
round(sum(apply(df.wdi.2019, 1, function(x){any(is.na(x))})) /
          nrow(df.wdi.2019), 2)
#> [1] 0.16
```

Таких стран 16%. Уберём их из таблицы с данными.

```
# убираем строки с пропусками
nrow(df.wdi.2019)    # сколько было строк до удаления NA
#> [1] 217
df.wdi.2019 <- na.omit(df.wdi.2019)
nrow(df.wdi.2019)    # сколько осталось после
#> [1] 183
```

Все страны мира – это крайне неоднородная выборка, поскольку уровень развития сильно отличается, и, как следствие, разброс значений каждого показателя будет велик. Вернём в данные сведения о классах стран для отбора более мелких и однородных выборок, но уже не в строки, а в качестве столбцов `income` и `region`.

```
# добавляем столбцы с классами стран по доходам и географии
df.wdi.2019 <- merge(df.wdi.2019,
                     meta.data[, c('iso2c', 'income', 'region')],
                     by = 'iso2c')
head(df.wdi.2019)
#>   iso2c              country NY.GDP.PCAP.CD IC.REG.COST.PC.ZS IC.REG.DURS
#> 1    AE United Arab Emirates     43103.3363              17.2         3.8
#> 2    AF          Afghanistan       507.1034               6.8         8.5
#> 3    AG  Antigua and Barbuda     17113.3498               8.0        19.0
#> 4    AL              Albania      5355.8478              10.8         4.5
#> 5    AM              Armenia      4622.7382               0.8         4.0
#> 6    AO               Angola      2809.6261              11.1        36.0
#>   IC.TAX.TOTL.CP.ZS IC.TAX.DURS IC.BUS.EASE.XQ             income
#> 1              15.9         116             16        High income
#> 2              71.4         270            173         Low income
#> 3              43.0         177            113        High income
#> 4              36.6         252             82 Upper middle income
#> 5              22.6         264             47 Upper middle income
#> 6              49.1         287            177 Lower middle income
#>                      region
#> 1 Middle East & North Africa
#> 2                 South Asia
#> 3  Latin America & Caribbean
#> 4      Europe & Central Asia
#> 5      Europe & Central Asia
#> 6         Sub-Saharan Africa
```

Сохраним результаты в файл `wdi_2019.csv`.

```
# сохраняем данные
file.name <- './data/wdi_2019.csv'
write.csv(df.wdi.2019, file.name, row.names = F)
# сделать запись в лог
write(paste('Файл', file.name, 'загружен', Sys.time()),
      file = log.filename, append = T)
```

Индивидуальные задания на импорт данных

1. Загрузить данные по странам мира с помощью пакета `WDI`: показатели из примера 4, 2019 год. Оставить только страны из класса, указанного в вашем варианте.

2. Загрузить из базы данных международной торговли статистику импорта за 2019 год (данные ежемесячные) по стране и кодам, указанным в варианте.

3. Результат: скрипт с расширением .R и два набора данных с расширениями .csv отправить преподавателю сообщением в личный кабинет, либо в чате Teams.

Варианты

Вариант	Класс стран для задания 1	Код товара для задания 2	Страна для задания 2
1	`income`: High income	86	Любая страна из класса High income
2	`income`: Low income	87	Любая страна из класса Low income
3	`income`: Lower middle income	88	Любая страна из класса Lower middle income
4	`income`: Upper middle income	89	Любая страна из класса Upper middle income
5	`region`: East Asia & Pacific, South Asia	02	Любая страна из класса East Asia & Pacific или South Asia
6	`region`: Europe & Central Asia	03	Любая страна из класса Europe & Central Asia
7	`region`: Latin America & Caribbean	07	Любая страна из класса Latin America & Caribbean
8	`region`: Middle East & North Africa	08	Любая страна из класса Middle East & North Africa
9	`region`: Sub-Saharan Africa	10	Любая страна из класса Sub-Saharan Africa
10	`region`: Latin America & Caribbean, North America	09	Любая страна из класса Latin America & Caribbean или North America
11	`income`: High income	16	Любая страна из класса High income
12	`income`: Low income	17	Любая страна из класса Low income
13	`income`: Lower middle income	22	Любая страна из класса Lower middle income
14	`income`: Upper middle income	25	Любая страна из класса Upper middle income
15	`region`: East Asia & Pacific,	26	Любая страна из класса East Asia

Вариант	Класс стран для задания 1	Код товара для задания 2	Страна для задания 2
	South Asia		& Pacific или South Asia
16	`region`: Europe & Central Asia	27	Любая страна из класса Europe & Central Asia
17	`region`: Latin America & Caribbean	28	Любая страна из класса Latin America & Caribbean
18	`region`: Middle East & North Africa	29	Любая страна из класса Middle East & North Africa
19	`region`: Sub-Saharan Africa	30	Любая страна из класса Sub-Saharan Africa
20	`region`: Latin America & Caribbean, North America	31	Любая страна из класса Latin America & Caribbean или North America

Дополнительная информация: парсинг данных с сайтов средствами R

Парсинг XML

Значительная часть открытых данных в интернете содержится не в файлах .xls или .csv, а в виде таблиц на веб-страницах. Анализ текста веб-страницы с целью извлечь нужную информацию, ориентируясь по тегам разметки, носит название парсинга (от англ. parse – разбор, структурный анализ). Легче всего понять технологию парсинга веб-страниц в R на примере разбора XML, хотя на практике сайты, написанные на чистом XML, в настоящее время редки.

Пример №4. Рассмотрим учебную XML-страницу: http://www.w3schools.com/xml/simple.xml [9]. На Рис. 1 показана структура этого файла.

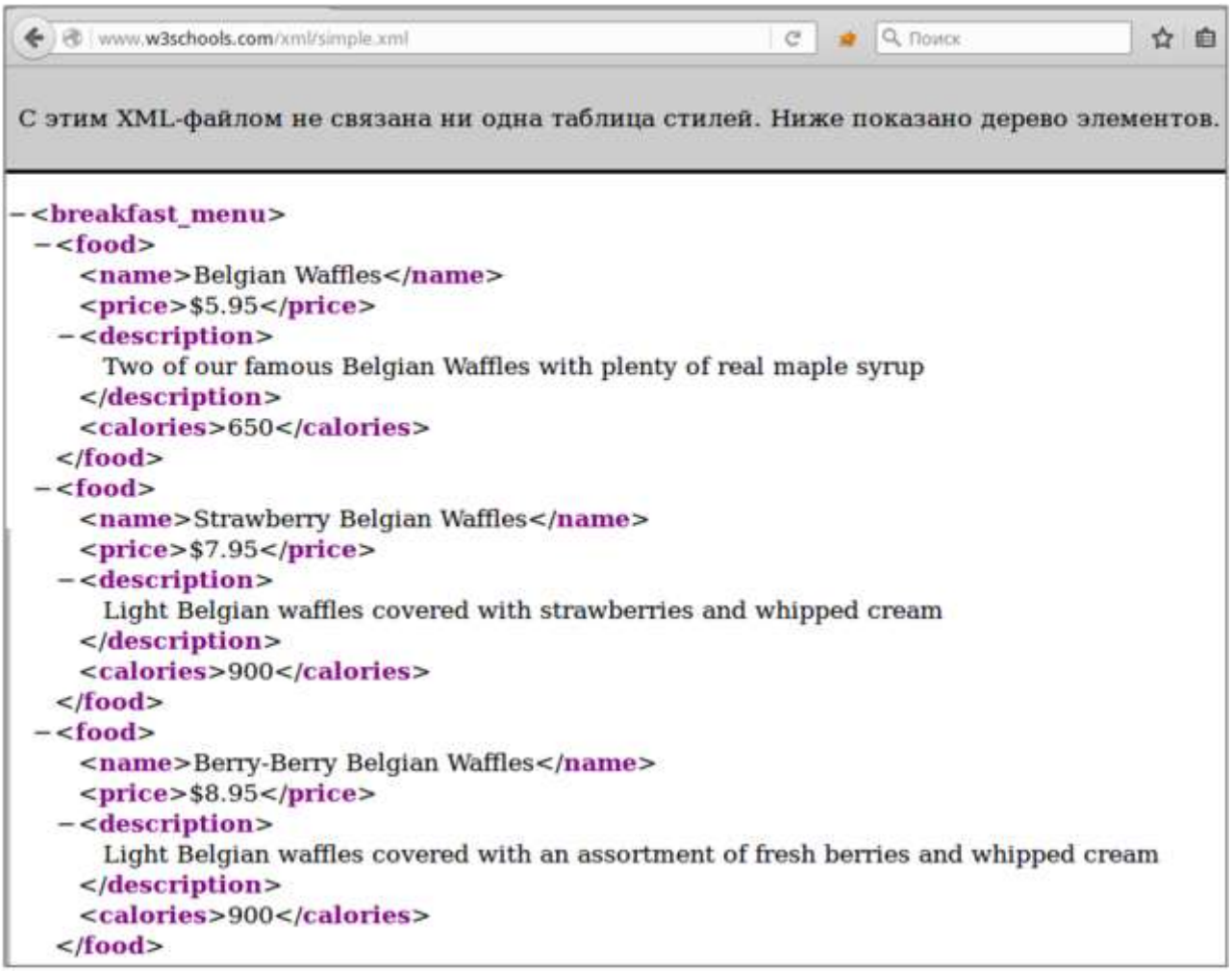

Рис.1. Структура XML-файла из примера 2

---

[9] Jeffrey Leek. Материалы курса «Getting and Cleaning Data» Университета Джонса Хопкинса на портале coursera.org, доступные в репозитории на github.com: https://github.com/jtleek/modules/tree/master/03_GettingData.

Для разбора XML-страниц в R служит пакет «XML»[10].

```
# Загрузка пакетов
library('httr')                 # работа с URL по https
library('RCurl')                # работа с URL по http
library('XML')                  # разбор XML-файлов

# адрес XML-страницы
fileURL <- 'https://www.w3schools.com/xml/simple.xml'

# Define certificate file
cafile <- system.file("CurlSSL", "cacert.pem", package = "RCurl")

# Read page
doc <- GET(fileURL, config(cainfo = cafile))

# разбираем объект как XML
parsedXML <- xmlTreeParse(doc, useInternalNodes = T)
#> No encoding supplied: defaulting to UTF-8.
```

Итак, в файле содержится информация о меню завтрака, о чём говорит название корневого тега «breakfast menu». По каждой позиции меню, описанной в теге «food», есть название блюда («name»), его цена («price»), описание («description») и количество калорий («calories»). Просмотрев объект `parsedXML`, можно убедиться, что он полностью повторяет эту структуру.

```
# просмотр загруженного документа
# ВНИМАНИЕ: не повторять для больших страниц!
parsedXML
#> <?xml version="1.0" encoding="UTF-8"?>
#> <breakfast_menu>
#>   <food>
#>     <name>Belgian Waffles</name>
#>     <price>$5.95</price>
#>     <description>Two of our famous Belgian Waffles with plenty of real maple
syrup</description>
#>     <calories>650</calories>
#>   </food>
#>   <food>
#>     <name>Strawberry Belgian Waffles</name>
#>     <price>$7.95</price>
#>     <description>Light Belgian waffles covered with strawberries and whipped
cream</description>
#>     <calories>900</calories>
#>   </food>
#>   <food>
#>     <name>Berry-Berry Belgian Waffles</name>
#>     <price>$8.95</price>
```

[10] Duncan Temple Lang and the CRAN Team (2015). XML: Tools for Parsing and Generating XML Within R and S-Plus. R package version 3.98-1.3. https://CRAN.R-project.org/package=XML.

```
#>     <description>Light Belgian waffles covered with an assortment of fresh
berries and whipped cream</description>
#>     <calories>900</calories>
#>   </food>
#>   <food>
#>     <name>French Toast</name>
#>     <price>$4.50</price>
#>     <description>Thick slices made from our homemade sourdough
bread</description>
#>     <calories>600</calories>
#>   </food>
#>   <food>
#>     <name>Homestyle Breakfast</name>
#>     <price>$6.95</price>
#>     <description>Two eggs, bacon or sausage, toast, and our ever-popular hash
browns</description>
#>     <calories>950</calories>
#>   </food>
#> </breakfast_menu>
#>
```

Пакет «XML» содержит функции, которые позволяют перемещаться по дереву документа и извлекать текст из тегов с определённым именем и/или атрибутами. Рассмотрим некоторые из них:

- `xmlTreeParse(`*URL_страницы*`, useInternalNodes = T)` читает структуру XML-страницы, используя её внутреннюю разметку (`useInternalNodes = T`).
- `xmlRoot(`*документ_XML*`)` возвращает корневую запись (тег) документа. Здесь и далее под документом понимается объект в R, который содержит структуру XML-страницы, прочитанной с помощью функции `xmlTreeParse()`.
- `xmlName(`*элемент_дерева_XML*`)` возвращает имя тега. Аргумент – объект типа XMLNode (XML запись).
- `xmlValue(`*элемент_дерева_XML*`)` возвращает содержимое тега.

```
# корневой элемент XML-документа
rootNode <- xmlRoot(parsedXML)

# имя корневого тега
xmlName(rootNode)
#> [1] "breakfast_menu"

# объект rootNode относится к специальному типу «XML запись»
class(rootNode)
#> [1] "XMLInternalElementNode" "XMLInternalNode"        "XMLAbstractNode"

# имена тегов, дочерних к корню (именованный вектор)
names(rootNode)
#>   food   food   food   food   food
#> "food" "food" "food" "food" "food"
```

```
# первый элемент дерева (обращаемся как к элементу списка)
rootNode[[1]]
#> <food>
#>   <name>Belgian Waffles</name>
#>   <price>$5.95</price>
#>   <description>Two of our famous Belgian Waffles with plenty of real maple
syrup</description>
#>   <calories>650</calories>
#> </food>

# первый потомок первого потомка корневого тега...
rootNode[[1]][[1]]
#> <name>Belgian Waffles</name>

# ...и его содержимое
xmlValue(rootNode[[1]][[1]])
#> [1] "Belgian Waffles"
```

Код выше реализует своего рода «слепую навигацию» по дереву, когда мы не знаем имён нужных тегов и просто движемся от одного потомка к другому. Для поиска конкретных элементов дерева используем функции:

- `xmlSApply(`*XML_запись*`,`*имя_функции*`)` применяет («apply») функцию ко всем элементам XML записи.
- `xpathSApply(`*XML_запись*`, "`*условие*`",`*имя_функции*`)` применяет функцию ко всем элементам, удовлетворяющим условию.

```
# извлечь все значения из потомков в XML-записи
values.all <- xmlSApply(rootNode, xmlValue)

# просмотреть первые два элемента
values.all[1:2]
#>
food
#>             "Belgian Waffles$5.95Two of our famous Belgian Waffles with plenty
of real maple syrup650"
#>
food
#> "Strawberry Belgian Waffles$7.95Light Belgian waffles covered with strawberries
and whipped cream900"
```

Функция `xmlSApply()` применила функцию `xmlValue()` ко всем потомкам корневого тега XML-записи и соединила значения входящих в них тегов в одну длинную строку. Затем результат был объединён в вектор, в котором столько же элементов, сколько было тегов «food» под корнем. Определённо, это неудовлетворительный результат, поскольку значения нескольких переменных образовали одно значение. Пакет «XML» содержит функции, как `xpathSApply()`, поддерживающие XPath (XML Path Language) – язык запросов к элементам XML-документа. Аргумент, задающий условие на отбор тегов, может использовать, в частности, такие конструкции:

- `nodename` – выбрать все записи (теги) с именем «nodename».
- `/` – выбрать запись на верхнем уровне иерархии.
- `//` – выбрать запись на любом уровне иерархии.
- `.` – выбрать текущую запись.
- `..` – выбрать родителя текущей записи.
- `@` – выбрать атрибуты.
- `[]` – в квадратных скобках после имени записи записываются предикаты, т.е. условия на значения.
- * – выбрать все записи документа.
- *@ – выбрать все записи документа с атрибутами.
- `node()` – выбрать все записи всех видов[11].
- `text()` – извлечь значение тега (работает при наличии нескольких пространств имён).
- `node1/parent::node2` – выбрать тег с именем «node2», который является родительским по отношению к тегу «node1».
- `node1/child::node2` – выбрать тег с именем «node2», который является потомком по отношению к тегу «node1».
- `node1/following-sibling::node2` – выбрать тег с именем «node2», который находится на том же уровне иерархии, что и «node1», и следует за ним.
- `node1/preceding-sibling::node2` – выбрать тег с именем «node2», который находится на том же уровне иерархии, что и «node1», и следует за ним [12].

```
# вытащить содержимое тегов "name" на любом уровне
xpathSApply(rootNode, "//name", xmlValue)
#> [1] "Belgian Waffles"             "Strawberry Belgian Waffles"
#> [3] "Berry-Berry Belgian Waffles" "French Toast"
#> [5] "Homestyle Breakfast"

# вытащить содержимое тегов "price" на любом уровне
xpathSApply(rootNode, "//price", xmlValue)
#> [1] "$5.95" "$7.95" "$8.95" "$4.50" "$6.95"
```

Пойти дальше, то есть превратить хорошо структурированный файл в объект `data.frame`, поможет функция `xmlToDataFrame(`*XML_запись*`)`:

---

[11] XPath Syntax. URL: http://www.w3schools.com/xsl/xpath_syntax.asp.

[12] XSLT: Применение осей. URL: https://xsltdev.ru/xslt/recipes/primenenie-osey/

```
# разобрать XML-страницу и собрать данные в таблицу
DF.food <- xmlToDataFrame(rootNode, stringsAsFactors = F)
# предварительный просмотр
dim(DF.food)      # размерность таблицы
#> [1] 5 4

str(DF.food)      # структура (характеристики столбцов)
#> 'data.frame':    5 obs. of  4 variables:
#>  $ name       : chr  "Belgian Waffles" "Strawberry Belgian Waffles" "Berry-
Berry Belgian Waffles" "French Toast" ...
#>  $ price      : chr  "$5.95" "$7.95" "$8.95" "$4.50" ...
#>  $ description: chr  "Two of our famous Belgian Waffles with plenty of real
maple syrup" "Light Belgian waffles covered with strawberries and whipped cream"
"Light Belgian waffles covered with an assortment of fresh berries and whipped
cream" "Thick slices made from our homemade sourdough bread" ...
#>  $ calories   : chr  "650" "900" "900" "600" ...
```

Напомним, что аргумент `stringsAsFactors = F` запрещает кодировать символьные переменные в факторы. Структура документа из этого примера слишком проста, чтобы проверить работу более сложных запросов.

Пример №5. Приведём пример использования синтаксиса XPath. Используем курсы обмена евро на другие валюты, устанавливаемые на дату, которые публикуются Европейским центральным банком на сайте https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml. Статистика доступна в формате XML. На Рис. 2 показана структура страницы с курсами на последнюю установленную дату.

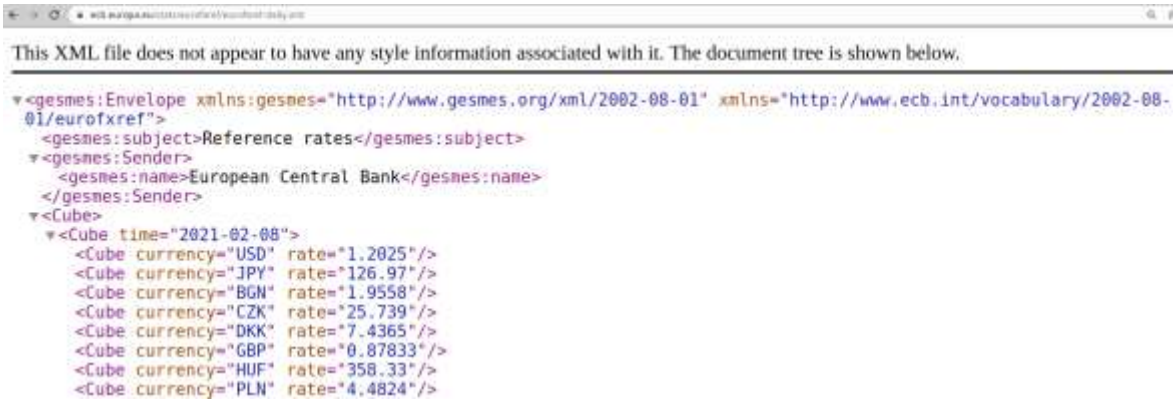

Рис.2. Структура XML-файла из примера 3

```
# обменный курс евро по отношению к иностранным
#  валютам, на текущую дату
fileURL <- 'https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml'
# xmlParse() не работает с https, поэтому сначала читаем страницу как текст
xmlData <- getURL(fileURL)
# и разбираем содержимое в объект doc
parsedXML <- xmlParse(xmlData, useInternalNodes = T)

# корневой элемент
rootNode <- xmlRoot(parsedXML)
# класс объекта rootNode
```

```
class(rootNode)
#> [1] "XMLInternalElementNode" "XMLInternalNode"        "XMLAbstractNode"
# имя корневого элемента
xmlName(rootNode)
#> [1] "Envelope"
```

Структура этого дерева XML проста, но содержит пространство имён, что может доставить сложности при разборе. Запись корневого тега как `<gesmes:Envelope>` означает, что тег называется “Envelope”, но имя задано на пространстве имён “gesmes”, ссылка на которое дана в шапке файла. Сами курсы записаны тегами “Cube”, причём и название валюты, и обменный курс записаны в атрибутах (“currency” и “rate” соответственно). Для начала попробуем извлечь все уникальные имена тегов документа, используя XPath-запрос.

```
# вытаскиваем имена всех тегов документа (*)
#  на любом уровне иерархии (//)

tag <- xpathSApply(rootNode, "//*", xmlName)
# оставляем только уникальные
tag <- unique(tag)

# считаем их количество
length(tag)
#> [1] 5

# смотрим названия
tag
#> [1] "Envelope" "subject"  "Sender"   "name"     "Cube"
```

Посмотрим, как обращаться к тегу с явно заданным пространством имён.

```
# в документе есть теги с явно объявленным пространством имён (namespace) gesmes
try.tag <- xpathSApply(rootNode, "//name", xmlValue)         # пусто
try.tag
#> list()
try.tag <- xpathSApply(rootNode, "//gesmes:name", xmlValue)  # тег найден
try.tag
#> [1] "European Central Bank"
```

Пространство имён в файле может быть не одно. Посмотрим все пространства документа.

```
# посмотреть пространство имён xml-документа
xmlNamespace(rootNode)
#>                                gesmes
#> "http://www.gesmes.org/xml/2002-08-01"
#> attr(,"class")
#> [1] "XMLNamespace"
```

Теперь извлечём наименования валют и обменные курсы из атрибутов тегов “Cube”, которые содержат атрибут “currency”.

```
# информация о курсах записана в тегах Cube без явного namespace
#  поэтому используем функцию name() для поиска тега
```

```
# source: https://stackoverflow.com/questions/45634155/parse-nested-xml-with-
namespaces-in-r
tag <- xpathSApply(rootNode, "//*[name()='Cube'][@currency]", xmlGetAttr,
'currency')
tag
#>  [1] "USD" "JPY" "BGN" "CZK" "DKK" "GBP" "HUF" "PLN" "RON" "SEK" "CHF" "ISK"
#> [13] "NOK" "HRK" "RUB" "TRY" "AUD" "BRL" "CAD" "CNY" "HKD" "IDR" "ILS" "INR"
#> [25] "KRW" "MXN" "MYR" "NZD" "PHP" "SGD" "THB" "ZAR"
curr.names <- unlist(tag)

# курсы обмена
tag <- xpathSApply(rootNode, "//*[name()='Cube'][@currency]", xmlGetAttr, 'rate')
tag
#>  [1] "1.1846"   "130.31"   "1.9558"   "25.382"   "7.4360"   "0.85915"
#>  [7] "347.85"   "4.5069"   "4.9393"   "10.1939"  "1.0848"   "150.60"
#> [13] "10.2938"  "7.5010"   "86.2301"  "9.7942"   "1.6038"   "6.1429"
#> [19] "1.4944"   "7.6503"   "9.2075"   "16874.01" "3.8029"   "86.5205"
#> [25] "1372.66"  "23.6529"  "4.9256"   "1.6702"   "59.039"   "1.5921"
#> [31] "38.446"   "16.9882"
curr.rate <- unlist(tag)
```

Обязательно нужно зафиксировать дату, она записана в теге «time» тега «Cube».

```
# дата (обращаемся только к тегу Cube, в котором есть атрибут time)
tag <- xpathSApply(rootNode, "//*[name()='Cube'][@time]", xmlGetAttr, 'time')
tag
#> [1] "2021-09-02"
```

Записываем всё во фрейм.

```
# превращаем XML во фрейм
DF.EUR <- cbind(curr.names, curr.rate, tag)
colnames(DF.EUR) <- c('Валюта', 'Курс к евро', 'Дата')
# предварительный просмотр
dim(DF.EUR)     # размерность таблицы
#> [1] 32  3
str(DF.EUR)     # структура (характеристики столбцов)
#>  chr [1:32, 1:3] "USD" "JPY" "BGN" "CZK" "DKK" "GBP" "HUF" "PLN" "RON" ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr [1:3] "Валюта" "Курс к евро" "Дата"
```

Парсинг HTML

Пример №6. Технически разбор HTML-страниц мало чем отличается от разбора XML, поскольку здесь используются те же функции пакета XML и язык XPath. Однако извлекать данные из HTML обычно труднее из-за большого количества не относящейся к данным информации как в содержимом страницы, так и в разметке. Разберём страницу с топ-200 книг по версии BBC, размещённый на Википедии.

В коде ниже есть закомментированные строки кода, которые сработают в Linux, но вызывают ошибку в Windows. Это связано с тем, что пакет `RCurl` содержит ошибки,

которые на момент написания этого руководства не были исправлены. Под Windows для загрузки данных по протоколу https рекомендуется использовать пакет `httr`.

Ещё одна проблема связана с распознаванием кириллицы. Кодировка не всегда интерпретируется верно, поэтому надёжнее строить запросы на отбор тегов, по возможности без использования кириллицы.

```
# пакет, который позволяет загружать данные по протоколу https под Windows
library('httr')

# URL страницы топ-200 книг по версии BBC на Википедии
fileURL <-
"https://ru.wikipedia.org/wiki/200_%D0%BB%D1%83%D1%87%D1%88%D0%B8%D1%85_%D0%BA%D0%
BD%D0%B8%D0%B3_%D0%BF%D0%BE_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B8_%D0%91%D0%B8-
%D0%B1%D0%B8-%D1%81%D0%B8"

# загружаем текст html-страницы, явно указывая кодировку (помогает от проблем с
кириллицей)
# html <- getURL(fileURL, .encoding = 'UTF-8')  # работает под Linux
html <- GET(fileURL)     # работает под Windows

# класс объекта с загруженным содержимым
class(html)
#> [1] "response"

# дальше только для функции GET()
html <- content(html, 'text', encoding = 'UTF-8')
class(html)
#> [1] "character"

# разбираем как html
# parsedHTML <- htmlParse(html)                       # для getURL()
parsedHTML <- htmlParse(html, useInternalNodes = T)  # для GET()

# корневой элемент
rootNode <- xmlRoot(parsedHTML)
```

Далее, чтобы определить, в каких тегах содержится нужная информация, необходимо изучить исходный код страницы. Браузеры «Mozilla Firefox» и «Chrome» позволяют просматривать отдельно исходный код выделенного фрагмента. Для этого нужно выделить интересующую часть текста, кликнуть на ней правой кнопкой мыши и выбрать пункт контекстного меню «Исходный код выделенного фрагмента» или «Просмотреть код». Просмотр кода страницы позволил определить, что названия книг – это значения атрибутов «title» тегов «a», которые находятся внутри тегов «li» после заголовка (`<h2...><span...>Список</span></div>`) с текстом «Список» (Рис. 3). У нужного нам тега «span» есть атрибут «class», равный “mw-headline” – используем это, чтобы не применять кириллицу в xpath-запросе. Чтобы вытащить нужные теги «a», необходимо найти «span» с атрибутом «class», равным “mw-headline”, подняться от него к тегу «h2» на уровень вверх, затем выбрать следующий тег «div» на том же уровне, и найти всех его потомков «li/a».

Причём для каждой книги есть две гиперссылки, первая для названия (`a[1]`), вторая для автора (`a[2]`).

Выберем все названия и авторов, а также ссылки на статьи с описанием каждой книги, кроме тех случаев, когда такой страницы нет.

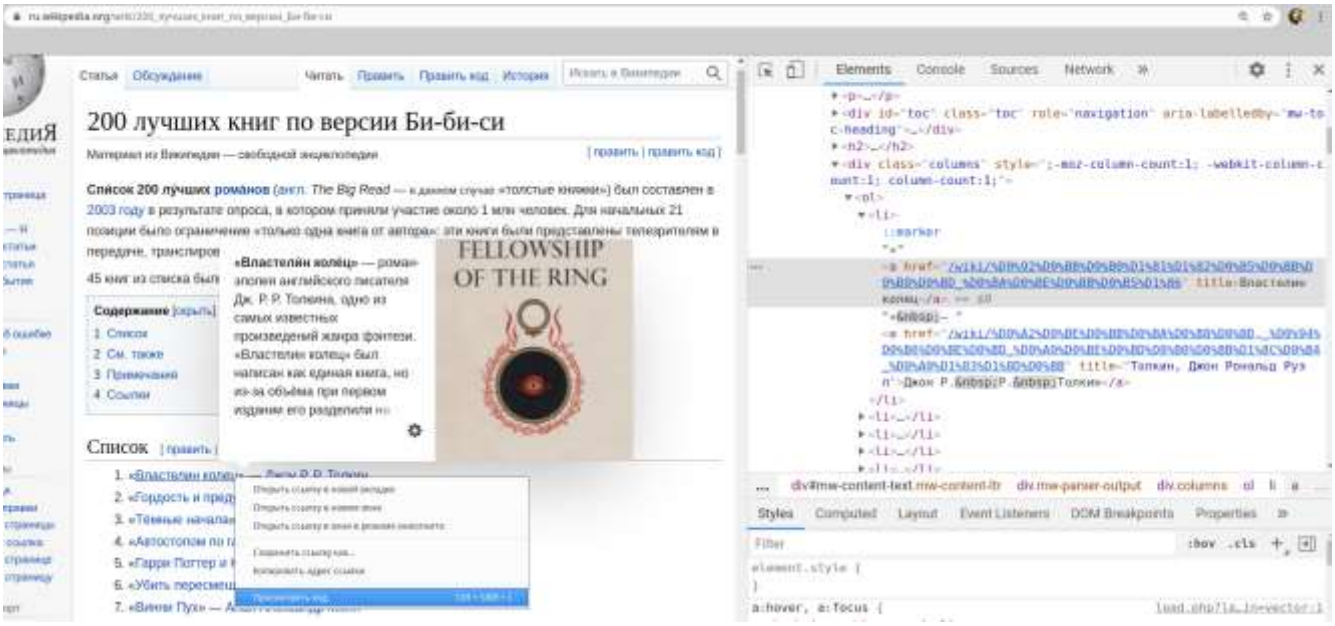

Рис.3. Структура html-страницы из примера 4

```
# выбираем все наименования книг
wiki.title <- xpathSApply(rootNode, '//span[@class="mw-
headline"][1]/parent::h2/following-sibling::div//li/*[1]',
                          xmlGetAttr, 'title')
# проверяем длину
length(wiki.title)
#> [1] 201

# превращаем в вектор
wiki.title[sapply(wiki.title, is.null)] <- NA
wiki.title <- unlist(wiki.title)

# просмотр первых трёх элементов вектора
wiki.title[1:3]
#> [1] "Р’Р»Р°СЃС‚РµР»РёРЅ Р¢РѕР»РєРёРЅ"
#> [2] "Р“РѕСЂРґРѕСЃС‚СЊ Рё РїСЂРµРґСѓР±РµР¶РґРµРЅРёРµ"
#> [3] "РўС‘РјРЅС‹Рµ РЅР°С‡Р°Р»Р°"

# исправляем кодировку
Encoding(wiki.title) <- 'UTF-8'
wiki.title[1:3]
#> [1] "Властелин колец"          "Гордость и предубеждение"
#> [3] "Тёмные начала"

# выбираем всех авторов книг
wiki.author <- xpathSApply(rootNode, '//span[@class="mw-
```

```
headline"][1]/parent::h2/following-sibling::div//li/*[2]',
                            xmlGetAttr, 'title')
# проверяем длину
length(wiki.author)
#> [1] 201

# превращаем в вектор
wiki.author[sapply(wiki.author, is.null)] <- NA
wiki.author <- unlist(wiki.author)

# исправляем кодировку
Encoding(wiki.author) <- 'UTF-8'
wiki.author[1:3]
#> [1] "Толкин, Джон Рональд Руэл" "Остин, Джейн"
#> [3] "Пулман, Филип"

# выбираем все ссылки на книги
wiki.link <- xpathSApply(rootNode, '//span[@class="mw-
headline"][1]/parent::h2/following-sibling::div//li/*[1]',
                         xmlGetAttr, 'href')
# проверяем длину
length(wiki.link)
#> [1] 201
# просмотр первых трёх элементов вектора
wiki.link[1:3]
#> [[1]]
#> [1]
"/wiki/%D0%92%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D0%BB%D0%B8%D0%BD_%D0%BA%D0%BE%D0%BB%D
0%B5%D1%86"
#>
#> [[2]]
#> [1]
"/wiki/%D0%93%D0%BE%D1%80%D0%B4%D0%BE%D1%81%D1%82%D1%8C_%D0%B8_%D0%BF%D1%80%D0%B5%
D0%B4%D1%83%D0%B1%D0%B5%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5"
#>
#> [[3]]
#> [1]
"/wiki/%D0%A2%D1%91%D0%BC%D0%BD%D1%8B%D0%B5_%D0%BD%D0%B0%D1%87%D0%B0%D0%BB%D0%B0"

# превращаем в вектор
wiki.link[sapply(wiki.link, is.null)] <- NA
wiki.link <- unlist(wiki.link)

# добавляем адрес сайта во внутренние ссылки
wiki.link[!is.na(wiki.link)] <- paste0('https://ru.wikipedia.org',
                                       wiki.link[!is.na(wiki.link)])

# объединяем во фрейм
DF.wiki <- data.frame(Книга = wiki.title, Автор = wiki.author, Ссылка = wiki.link)

# отбрасываем полностью пустые строки
DF.wiki <- DF.wiki[rowSums(is.na(DF.wiki)) != ncol(DF.wiki), ]
```

```
# записываем в файл .csv
write.csv(DF.wiki, file = './data/DF_wiki.csv', row.names = F)
# сделать запись в лог
write(paste('Файл "DF_wiki.csv" записан', Sys.time()),
      file = log.filename, append = T)
```

В этом примере мы столкнулись с неверным разбором символов кириллицы. Пакет `rvest`, предназначенный для более удобного сбора данных с html-страниц, позволяет решить проблему кодировок в большинстве случаев.

Веб-скраппинг с пакетом "rvest"

Термин "веб-скраппинг" (Web Scraping) понемногу входит в обиход специалистов по данным, заменяя понятие "парсинг веб-страниц". Скраппинг означает именно анализ сайтов с целью сбора статистики, в то время как парсинг – более общий процесс анализа структуры текста. Сбор данных с сайтов можно производить самыми разными способами:

- **Ручная копипаста** – способ медленный, но устойчивый к различным вариациям структуры сайтов. Человеку, с одной стороны, проще понять, какие сведения со страницы требуется собрать. С другой стороны, при больших объёмах работы неизбежны случайные ошибки, а производительность самая здесь низкая.
- **Поиск по текстовым шаблонам** – другой простойи мощный подход к извлечению информации из интернета. Применяются регулярные выражения языков программирования. Методы XPath, рассмотренные выше, как раз из этой серии.
- **Использование API** возможно для большинства крупных платформ: Facebook, Twitter, LinkedIn и других. Минус в том, что бесплатные версии API обычно сильно ограничивают объём данных, доступных для скачивания в единицу времени.
- **Парсинг DOM** (Document Object Model – «объектная модель документа»). Используя браузеры, программы могут извлекать динамический контент, генерируемый на стороне клиента. Также можно анализировать веб-страницы с помощью дерева объектов DOM [13]. В R такие возиожности реализованы в пакете `rvest`.

Пример №7. Мы будем собирать данные о самых популярных фильмах 2016 года выпуска по версии IMDB, и для поиска нужных объектов на сайте нам понадобится свободная программа "Selector Gadget" (https://selectorgadget.com/), доступная в виде расширения к Chrome.

```
# загружаем пакеты
library('rvest')     # работа с DOM сайта
library('dplyr')     # инструменты трансформирования данных
#>
#> Присоединяю пакет: 'dplyr'
```

---

[13] Saurav Kaushik (MARCH 27, 2017). Beginner's Guide on Web Scraping in R (using rvest) with hands-on example. URL: https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/.

```
#> Следующие объекты скрыты от 'package:stats':
#>
#>     filter, lag
#> Следующие объекты скрыты от 'package:base':
#>
#>     intersect, setdiff, setequal, union

# URL страницы для скраппинга
url <-
'http://www.imdb.com/search/title?count=100&release_date=2016,2016&title_type=feat
ure'

# читаем HTML страницы
webpage <- read_html(url)
```

Теперь соберём со страницы следующие данные:

- `Rank` – Ранг фильма от 1 до 100 в списке самых популярных фильмов 2016 года выпуска.
- `Title` – название фильма;
- `Description` – описание фильма;
- `Runtime` – длительность фильма;
- `Genre` – жанр фильма; в случае если их несколько, берём первый;
- `Metascore` – метаоценка сайта IMDB фильма.

На Рис.4 показан скриншот страницы.

Рис.4. Вид веб-страницы из примера 6

Начнём с поля `Rank`. Используем расширение “Selector Gadget”, чтобы получить CSS-селектор для тега, в котором записан ранг фильма. Для этого нажмите сначала на иконку расширения справа от адресной строки браузера, а затем на ранг фильма (Рис.5). Убедитесь, что все ранги подсвечены жёлтым. В рамке внизу страницы появится искомый селектор.

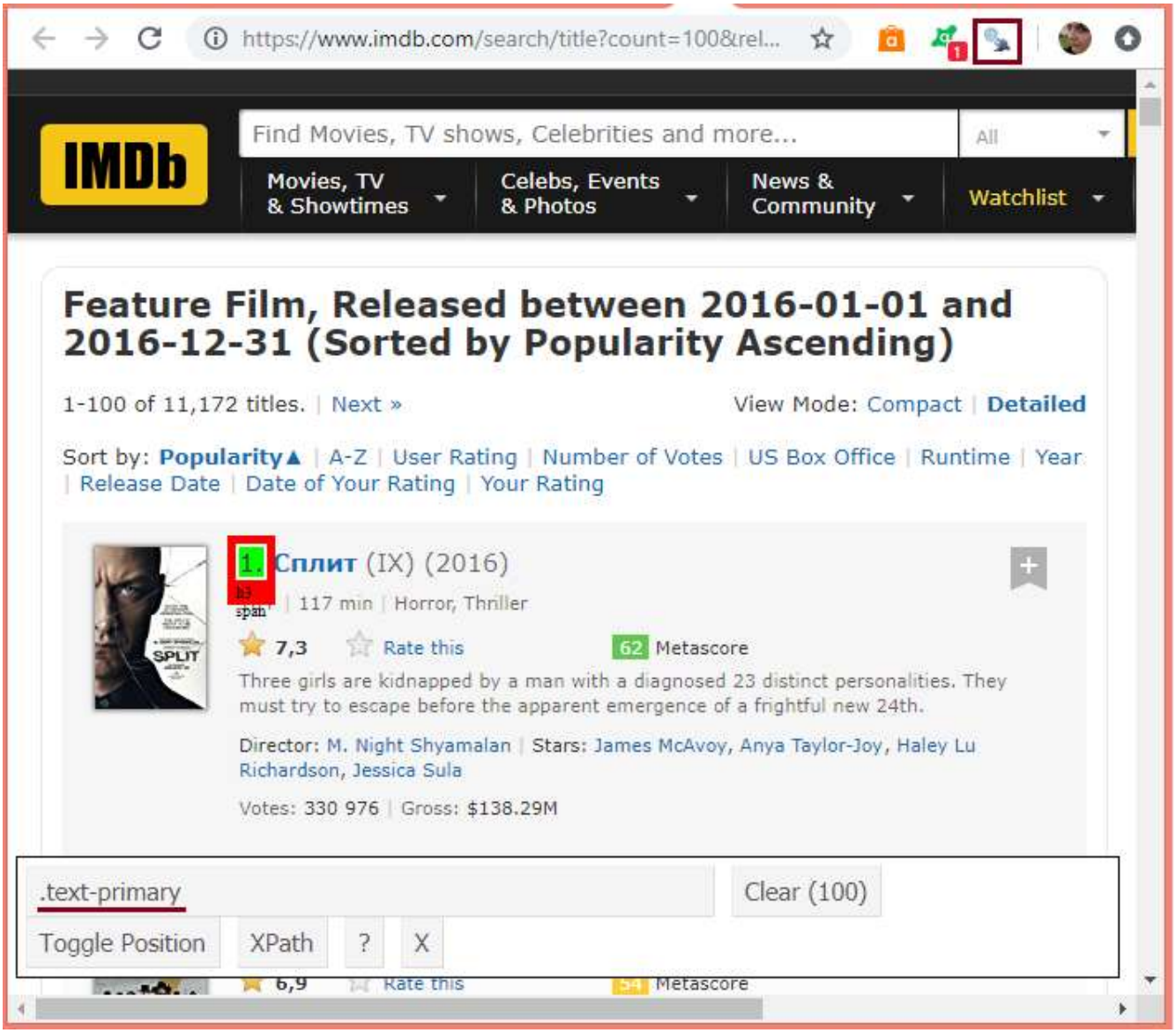

Рис.5. Поиск селектора для ранга фильма с помощью Selector Gadget

Теперь используем этот селектор для выбора всех объектов, содержащих ранги, на странице. Обратите внимание, как мы объединяем две функции в одну строку с помощью пайплайна %>%. Объект, который возвращает функция `html_nodes()`, подаётся на вход функции `html_text()`, а результат записывается в переменную `rank_data`.

```
# скраппим страницу по селектору и преобразуем в текст
rank_data <- webpage %>% html_nodes('.text-primary') %>% html_text

# размер вектора
length(rank_data)
#> [1] 100

# первые шесть рангов
head(rank_data)
#> [1] "1." "2." "3." "4." "5." "6."
```

Для рангов предпочтителен числовой формат, поэтому преобразуем полученные данные с помощью функции `as.numeric()`.

```
# конвертируем ранги в числовые данные
rank_data <- as.numeric(rank_data)

# результат
head(rank_data)
#> [1] 1 2 3 4 5 6
```

Теперь точно так же найдём селектор для названий фильмов (Рис.6). Чтобы выбрать только названия фильмов (они подсвечены жёлтым), здесь требуется творческий подход: необходимо выбрать первый заголовок, а затем кликнуть по сортировке “A-Z”, чтобы отсечь все лишние ссылки. Применим селектор `'.lister-item-header a'` для загрузки названий.

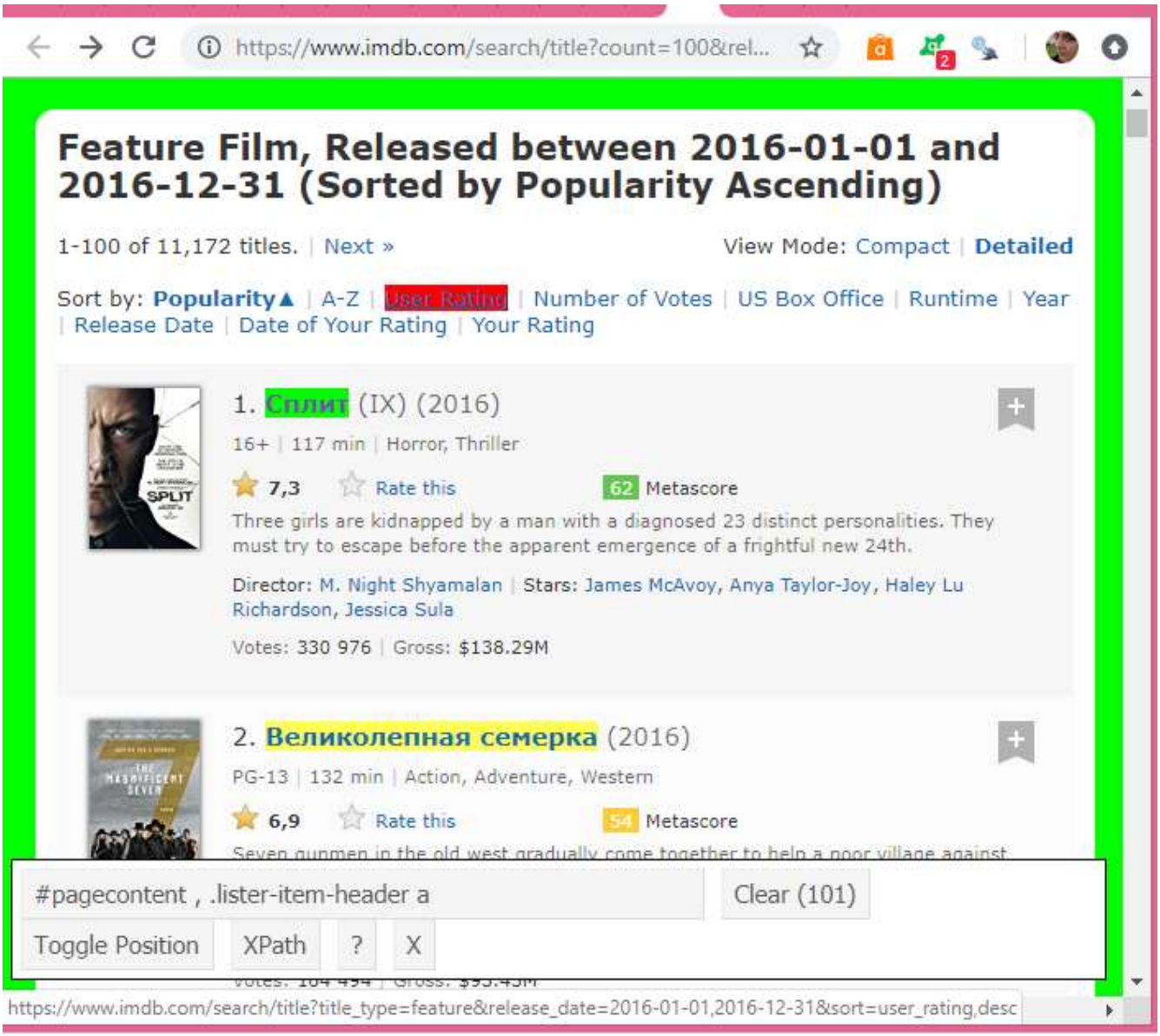

Рис.6. Поиск селектора для названия фильма

```
# отбор названий фильмов по селектору
title_data <- webpage %>% html_nodes('.lister-item-header a') %>% html_text

# результаты
length(title_data)
#> [1] 100
head(title_data)
#> [1] "Отряд самоубийц"      "Не дыши"              "Дэдпул"
#> [4] "Доктор Стрэндж"       "Прибытие"             "Великолепная семерка"
```

Код ниже отвечает за скраппинг остальных полей, кроме сводного рейтинга (Metascore).

```
# описания фильмов
description_data <- webpage %>% html_nodes('.ratings-bar+ .text-muted') %>%
  html_text()
length(description_data)
#> [1] 100
head(description_data)
#> [1] "\nA secret government agency recruits some of the most dangerous
incarcerated super-villains to form a defensive task force. Their first mission:
save the world from the apocalypse."
#> [2] "\nHoping to walk away with a massive fortune, a trio of thieves break into
the house of a blind man who isn't as helpless as he seems."
#> [3] "\nA wisecracking mercenary gets experimented on and becomes immortal but
ugly, and sets out to track down the man who ruined his looks."
#> [4] "\nWhile on a journey of physical and spiritual healing, a brilliant
neurosurgeon is drawn into the world of the mystic arts."
#> [5] "\nA linguist works with the military to communicate with alien lifeforms
after twelve mysterious spacecraft appear around the world."
#> [6] "\nSeven gunmen from a variety of backgrounds are brought together by a
vengeful young widow to protect her town from the private army of a destructive
industrialist."

# длительности фильмов
runtime_data <- webpage %>% html_nodes('.text-muted .runtime') %>% html_text
length(runtime_data)
#> [1] 100
head(runtime_data)
#> [1] "123 min" "88 min"  "108 min" "115 min" "116 min" "132 min"

# жанры фильмов
genre_data <- webpage %>% html_nodes('.genre') %>% html_text
length(genre_data)
#> [1] 100
head(genre_data)
#> [1] "\nAction, Adventure, Fantasy           "
#> [2] "\nCrime, Horror, Thriller           "
#> [3] "\nAction, Adventure, Comedy           "
#> [4] "\nAction, Adventure, Fantasy           "
#> [5] "\nDrama, Sci-Fi           "
#> [6] "\nAction, Adventure, Western           "
```

С полем Metascore возникает проблема, поскольку оно есть только у 97 фильмов из 100.

```
# селектор для общего рейтинга (метарейтинга)
metascore_data <- webpage %>% html_nodes('.ratings-metascore') %>% html_text
# предварительный результат
length(metascore_data)
#> [1] 97
```

Проблему решает пользовательская функция, которая работает по принципу перебора всех тегов, в которые вложен .ratings-metascore; анализ страницы показал, что это теги .lister-item-content. Функция html_nodes() возвращает строку нулевой длины, когда не находит искомый тег внутри заданного, и нам нужно только отловить все эти случаи и заменить их на NA.

```
# функция перебора тегов внутри тегов более высокого уровня
get_tags <- function(node){
  # найти все теги с метарейтингом
  raw_data <- html_nodes(node, selector) %>% html_text
  # значения нулевой длины (для фильма нет такого тега) меняем на пропуски
  data_NAs <- ifelse(length(raw_data) == 0, NA, raw_data)
}

# это глобальная переменная будет неявно передана функции get_tags()
selector <- '.ratings-metascore'
# находим все ноды (теги) верхнего уровня, с информацией о каждом фильме
doc <- html_nodes(webpage, '.lister-item-content')
# применяем к этим тегам поиск метарейтинга и ставим NA там, где тега нет
metascore_data <- sapply(doc, get_tags)
# предварительный результат
length(metascore_data)
#> [1] 100
head(metascore_data)
#> [1] "\n40        \n        Metascore\n            "
#> [2] "\n71        \n        Metascore\n            "
#> [3] "\n65        \n        Metascore\n            "
#> [4] "\n72        \n        Metascore\n            "
#> [5] "\n81        \n        Metascore\n            "
#> [6] "\n54        \n        Metascore\n            "
```

Совместим данные в один фрейм и запишем его в файл .csv. В следующей практике мы вернёмся к этой таблице, чтобы почистить столбцы от лишних символов.

```
# совмещаем данные в один фрейм
DF_movies_short <- data.frame(Rank = rank_data, Title = title_data,
                              Description = description_data,
                              Runtime = runtime_data,
                              Genre = genre_data,  Metascore = metascore_data)
# результат
dim(DF_movies_short)
#> [1] 100   6
str(DF_movies_short)
#> 'data.frame':    100 obs. of  6 variables:
#>  $ Rank       : num  1 2 3 4 5 6 7 8 9 10 ...
```

```
#>  $ Title      : chr  "Отряд самоубийц" "Не дыши" "Дэдпул" "Доктор Стрэндж" ...
#>  $ Description: chr  "\nA secret government agency recruits some of the most
dangerous incarcerated super-villains to form a defensiv"| __truncated__ "\nHoping
to walk away with a massive fortune, a trio of thieves break into the house of a
blind man who isn't a"| __truncated__ "\nA wisecracking mercenary gets
experimented on and becomes immortal but ugly, and sets out to track down the m"|
__truncated__ "\nWhile on a journey of physical and spiritual healing, a brilliant
neurosurgeon is drawn into the world of the mystic arts." ...
#>  $ Runtime    : chr  "123 min" "88 min" "108 min" "115 min" ...
#>  $ Genre      : chr  "\nAction, Adventure, Fantasy            " "\nCrime,
Horror, Thriller            " "\nAction, Adventure, Comedy            " "\nAction,
Adventure, Fantasy            " ...
#>  $ Metascore  : chr  "\n40        \n        Metascore\n            " "\n71
\n        Metascore\n            " "\n65        \n        Metascore\n            "
"\n72        \n        Metascore\n            " ...

# записываем в .csv
write.csv(DF_movies_short, file = './data/DF_movies_short.csv', row.names = F)
# сделать запись в лог
write(paste('Файл "DF_movies_short.csv" записан', Sys.time()),
      file = log.filename, append = T)
```

Загрузка данных из других форматов

Не будет преувеличением сказать, что в настоящее время в R реализовано чтение данных из всех популярных форматов. Мы ограничимся рассмотренными выше примерами, однако перечислим ещё несколько форматов и названия пакетов R для их обработки.

- Книги Ms Excel .xls, .xlsx – пакет «xlsx» [14].
- Электронные таблицы Open Office Calc в формате .ods – пакет «readODS» [15].
- Файлы данных SPSS, SAS, Octave, Minitab, Stata, Weka – пакет «foreign» [16].
- Базы данных MySQL – пакет «RMySQL» [17].
- Базы данных PostgreQL, Microsoft Access, MySQL, SQLite – пакет «RODBC» [18].

---

[14] Adrian A. Dragulescu (2014). xlsx: Read, write, format Excel 2007 and Excel 97/2000/XP/2003 files. R package version 0.5.7. https://CRAN.R-project.org/package=xlsx.

[15] Gerrit-Jan Schutten (2014). readODS: Read ODS files and puts them into data frames. R package version 1.4. https://CRAN.R-project.org/package=readODS.

[16] R Core Team (2015). foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, ... R package version 0.8-66. https://CRAN.R-project.org/package=foreign.

[17] Jeroen Ooms, David James, Saikat DebRoy, Hadley Wickham and Jeffrey Horner (2015). RMySQL: Database Interface and 'MySQL' Driver for R. R package version 0.10.7. https://CRAN.R-project.org/package=RMySQL.

- База данных документов MongoDB – пакет «RMongo» [19].
- Shapefiles и другие форматы представления геоданных – пакеты «shapefiles» [20], «rdgal» [21], «rgeos» [22], «raster» [23].
- Изображения в форматах .jpeg, .png, .bmp – пакеты «jpeg» [24], «png» [25], «readbitmap» [26].
- Чтение и визуализация аудио – пакеты «tuneR» [27], «seewave» [28].

---

[18] Brian Ripley and Michael Lapsley (2015). RODBC: ODBC Database Access. R package version 1.3-12. https://CRAN.R-project.org/package=RODBC.

[19] Tommy Chheng (2013). RMongo: MongoDB Client for R. R package version 0.0.25. https://CRAN.R-project.org/package=RMongo.

[20] Ben Stabler (2013). shapefiles: Read and Write ESRI Shapefiles. R package version 0.7. https://CRAN.R-project.org/package=shapefiles.

[21] Roger Bivand, Tim Keitt and Barry Rowlingson (2015). rgdal: Bindings for the Geospatial Data Abstraction Library. R package version 1.1-3. https://CRAN.R-project.org/package=rgdal.

[22] Roger Bivand and Colin Rundel (2015). rgeos: Interface to Geometry Engine – Open Source (GEOS). R package version 0.3-15. https://CRAN.R-project.org/package=rgeos.

[23] Robert J. Hijmans (2015). raster: Geographic Data Analysis and Modeling. R package version 2.5-2. https://CRAN.R-project.org/package=raster.

[24] Simon Urbanek (2014). jpeg: Read and write JPEG images. R package version 0.1-8. https://CRAN.R-project.org/package=jpeg.

[25] Simon Urbanek (2013). png: Read and write PNG images. R package version 0.1-7. https://CRAN.R-project.org/package=png.

[26] Gregory Jefferis (2014). readbitmap: Simple Unified Interface to Read Bitmap Images (BMP,JPEG,PNG). R package version 0.1-4. https://CRAN.R-project.org/package=readbitmap.

[27] Uwe Ligges, Sebastian Krey, Olaf Mersmann, and Sarah Schnackenberg (2013). tuneR: Analysis of music. URL: http://r-forge.r-project.org/projects/tuner/.

[28] Sueur J., Aubin T., Simonis C. (2008). Seewave: a free modular tool for sound analysis and synthesis. Bioacoustics, 18: 213-226. URL: https://CRAN.R-project.org/package=seewave.