Дмитрий Храмов

Сбор данных в Интернете

на языке R

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

УДК 004.738.5:004.438R
ББК 32.971.353

Х89

Храмов Д. А.
Х89 Сбор данных в Интернете на языке R. – М.: ДМК Пресс, 2017. – 280 с.: ил.

ISBN 978-5-97060-459-5

Всё, что регистрирует человек и созданные им машины, может считаться данными. Фиксируя новое и переводя архивы в цифровую форму, мы с каждым днём производим всё больше данных. Но гораздо чаще случается так, что данные разбросаны по всемирной сети на многочисленных страницах онлайновых магазинов, заметках в социальных сетях, логах серверов и т. п. Прежде чем начать работать с такими данными, их необходимо собрать и сохранить в пригодном для анализа виде. Решению этих вопросов и посвящена данная книга.

Основной материал книги разделён на две части. В первой части дано краткое введение в R – описание среды разработки, языка и основных пакетов-расширений. Вторая часть посвящена непосредственно сбору данных: работе с открытыми данными, извлечению данных из веб-страниц и из социальных сетей. Также рассмотрены необходимые технические вопросы: протокол HTTP, функции импорта данных различных форматов и регулярные выражения. Завершается рассказ созданием карт на основе собранных данных.

Издание предназначено специалистам по анализу данных, а также программистам, интересующихся сбором данных в Интернете.

УДК 004.738.5:004.438R
ББК 32.971.353

ISBN 978-5-97060-459-5

Содержание

Глава 1

Знакомство с R

В этой главе мы установим R и научимся пользоваться средой разработки RGui. По пути решим простую, но весьма распространённую задачу – построим график функции.

Установка

Рассмотрим способ установки R, который подойдёт для любого типа операционных систем, поддерживаемых пакетом: Linux, Mac или Windows. В случае Windows этот путь – единственный, для Linux и Mac проще воспользоваться менеджером пакетов соответствующей системы.

Чтобы установить R, зайдём на его официальный сайт и выберем интересующую версию программы. Сам R, документация к нему и дополнительные пакеты распространяются через сеть ftp- и веб-серверов, называемую CRAN (Comprehensive R Archive Network). Поэтому следующим шагом будет выбор одного из более чем шести десятков зеркал CRAN. После этого, указав тип операционной системы, скачиваем дистрибутив R.

Запустим инсталляционный файл и будем следовать указаниям программы-установщика. Единственный момент, который потребует вашего внимания, – выбор разрядности операционной системы (рис. 1.1).

После установки запустим программу. В Linux и Mac, по умолчанию, R работает в консоли. Кроме того, вместе с пакетом поставляется графический интерфейс. В Linux он основан на связке Tcl/Tk и запускается командой `R --gui=Tk`. В Mac встроенный графический интерфейс для R называется R.app.

Версия R для Windows поставляется с графической оболочкой RGui (рис. A.5), которую мы и рассмотрим в дальнейшем.

Заметим, что работа со всеми указанными выше графическими оболочками выглядит примерно одинаково.

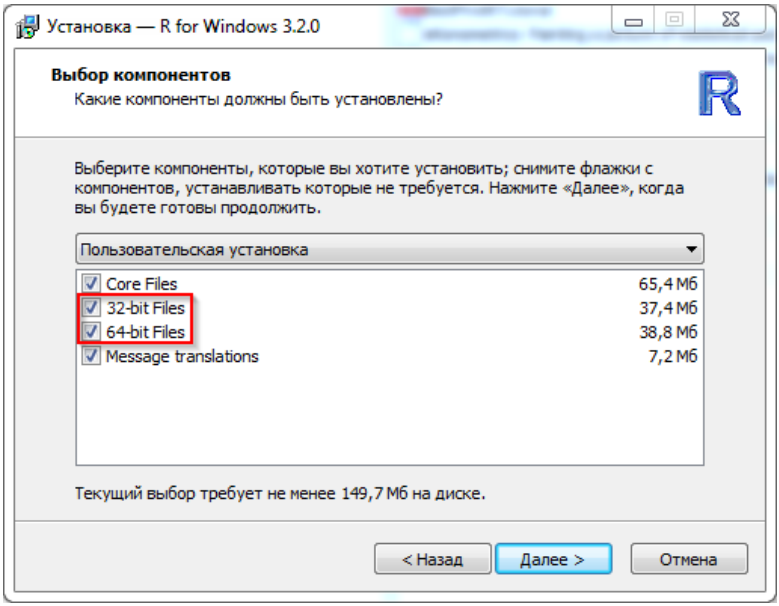

Рис. 1.1 ❖ Выбор компонентов в ходе установки R под Windows

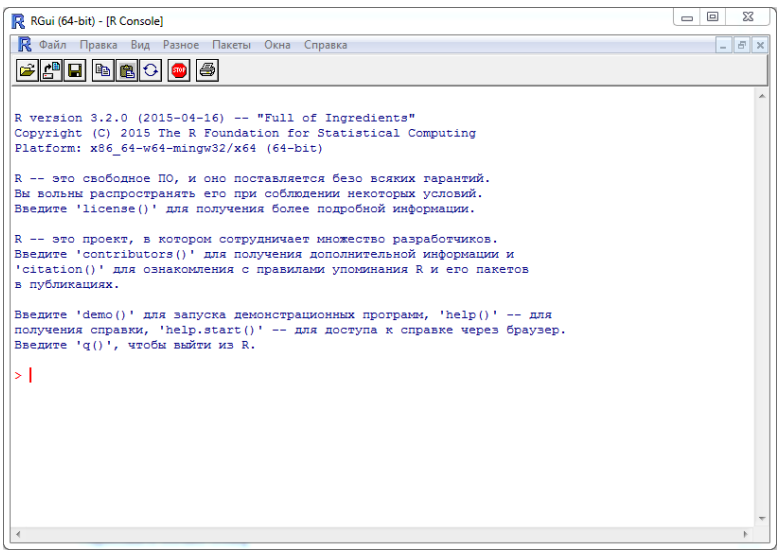

Рис. 1.2 ❖ Консоль R в графической оболочке RGui (Windows)

Работа в среде RGui

RGui – это стандартная графическая оболочка R под Windows, являющаяся простейшей средой разработки. Она быстро загружается и достаточно удобна в использовании. В RGui есть три вида окон:

- консоль;
- редактор скриптов;
- окно графического устройства.

Команды R вводятся в консоли (рис. А.5) после приглашения пользователя (значка `'>'`) и отправляются на выполнение нажатием *Enter*.

Управление консолью:

- дополнение команды – *Tab*;
- перемещение по истории команд – клавиши со стрелками;
- прекращение выполнения команды – *Esc*;
- переключение в другое окно – *Ctrl+Tab*;
- очистка консоли – *Ctrl+L*.

Для создания собственных программ (скриптов)[1] удобнее использовать не консоль, а редактор (рис. 1.3).

Открыть его можно в меню Файл/Новый скрипт. Первое окно открывается с помощью меню, последующие – так же или комбинацией клавиш *Ctrl+N*.

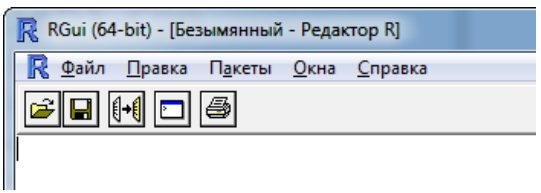

Рис. 1.3 ❖ Интерфейс редактора кода в RGui

Обратите внимание на то, как изменилась панель инструментов по сравнению с консолью (рис. А.5).

В качестве примера построим график синусоиды:

```
x <- seq(-pi,pi,.1)
y <- sin(x)
plot(x,y)
```

В первой строке формируется одномерный массив (вектор) x-координат, значения которых изменяются от $-\pi$ до π с шагом 0,1. Этот массив сохраняется в переменной `x`. Комбинация символов `<-` обозначает операцию присваивания.

[1] Мы используем термины «скрипт» и «программа» как синонимы.

Во второй строке создаётся вектор y, состоящий из синусов элементов вектора x.

Наконец, в третьей строке функция `plot` строит требуемый график.

В редакторе можно набирать команды и выполнять их как по одной, так и целыми блоками, с помощью комбинации *Ctrl+R* (на панели инструментов также есть соответствующая кнопка). Например, можно выделить весь скрипт – *Ctrl+A* и отправить его на выполнение *Ctrl+R* (рис. 1.4).

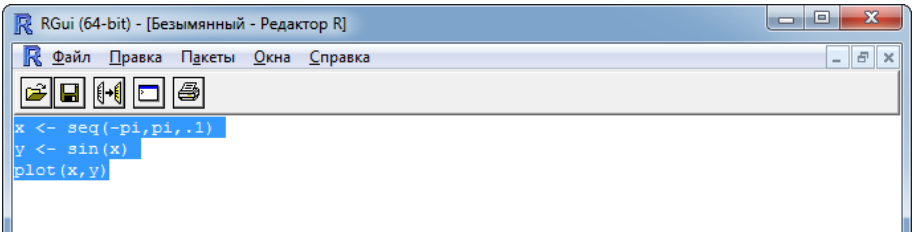

Рис. 1.4 ❖ Выделенную часть кода можно отправить на выполнение комбинацией клавиш *Ctrl+R*

В результате получим графическое окно с построенным в нём графиком синусоиды (рис. 1.5). По терминологии R окна, в которых строятся графики, и файлы, в которых эти графики сохраняются, вместе называются *графическими устройствами*.

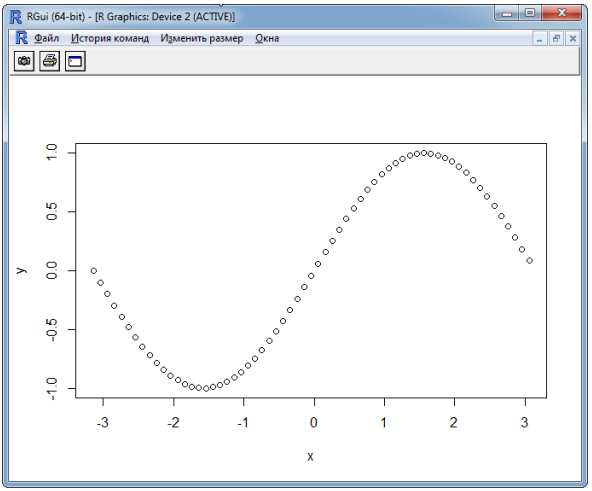

Рис. 1.5 ❖ Графическое устройство в RGui

Вернуться к консоли можно нажатием кнопки на панели инструментов (рис. 1.6) или при помощи *Ctrl+Tab*.

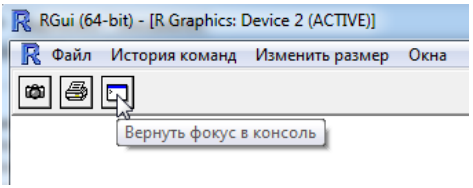

Рис. 1.6 ❖ Кнопка **Вернуть фокус в консоль** на панели инструментов графического устройства

Управление окнами при помощи меню Окна показано на рис. 1.7.

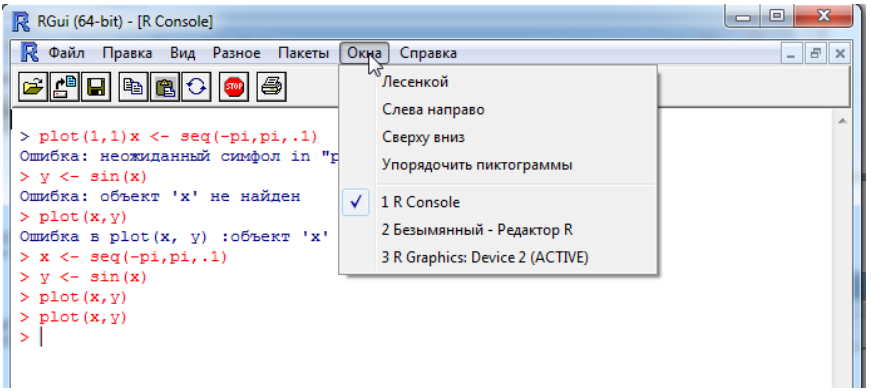

Рис. 1.7 ❖ Управление окнами с помощью меню Окна

Сохранить скрипт можно при помощи команд Сохранить или Сохранить как... меню Файл редактора или соответствующей кнопки на его панели управления.

Заметим, что скрипты R, объединённые общей темой, удобно держать в отдельном каталоге.

Выйти из R можно, вызвав в консоли `q()` или воспользовавшись меню Файл/Выйти.

При выходе из RGui среда предложит сохранить рабочее пространство (рис. 1.8).

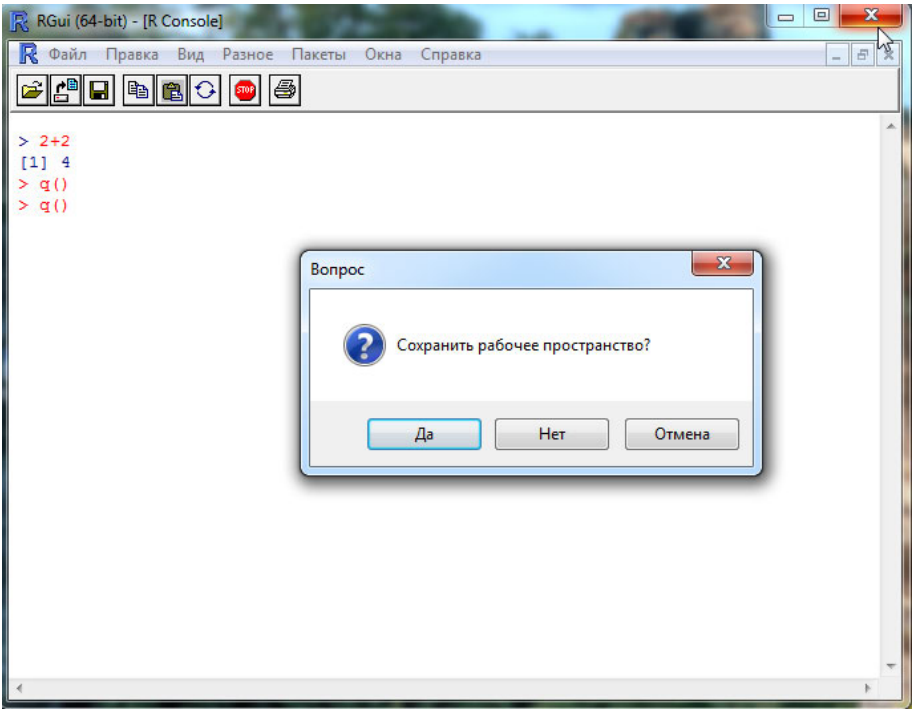

Рис. 1.8 ❖ Сохранение рабочего пространства по выходе из RGui

Рабочее пространство (workspace) – это область оперативной памяти, в которой хранятся все созданные пользователем объекты (векторы, матрицы, таблицы, списки, функции и т. п.). Сохранение рабочего пространства в файле и его последующая загрузка в новом сеансе работы с R (Файл/Загрузить рабочее пространство...) позволяют продолжить работу с того места, где она была прервана. При этом сохраняются значения всех переменных, вычисленные на прошлом сеансе работы.

Просмотреть список объектов в рабочем пространстве можно при помощи функции `ls`. Вызовем её в консоли

```
> ls()
```

и получим в результате

```
## [1] "x" "y"
```

Действительно, в памяти сейчас хранятся векторы координат синусоиды.

Два символа решётки (##) указывают на то, что далее идёт результат выполнения программы (в консоли они не отображаются, см. рис. 1.9). Смысл единицы ([1]) станет ясен в следующей главе.

Другие команды управления рабочим пространством:

```
# Сохранить рабочее пространство в файл .RData в текущем рабочем каталоге
save.image()
# Сохранить заданные объекты в файле
save(object_list, file="myfile.RData")
# Загрузить образ рабочего пространства
load("myfile.RData")
# Очистить рабочее пространство
rm(list=ls())
```

Символ # открывает строку комментария.

Сохранить или загрузить образ рабочего пространства можно при помощи меню **Файл**. Следует иметь в виду, что это меню, как и любое другое в RGui, просто запускает на выполнение соответствующие функции R.

Например, команда меню **Файл/Сохранить рабочее пространство...** вызывает функцию `save.image`, и результат этого вызова можно увидеть в консоли (рис. 1.9).

```
> x <- seq(-pi, pi, .1)
> y <- sin(x)
> plot(x,y)
> ls()
[1] "x" "y"
> save.image("C:\\Users\\Admin\\Documents\\myimage")
>
```

Рис. 1.9 ❖ Команда меню Файл/Сохранить рабочее пространство... представляет собой вызов функции `save.image`

Клавиши со стрелками вверх и вниз позволяют перемещаться по списку выполненных ранее команд – *истории команд*. Историю команд также можно сохранить в файл на диске.

```
# Вывод истории команд
history()                 # выводит список 25 последних выполненных команд
history(max.show=Inf)     # выводит все выполненные в сессии команды
# Сохранить историю команд
savehistory(file="myfile") # по умолчанию ".Rhistory"
# Загрузить историю команд
loadhistory(file="myfile") # по умолчанию ".Rhistory"
```

Помимо RGui, для R существуют другие, более продвинутые среды разработки, например R Commander или RStudio. Так же, как и сам R, они являются кросс-платформенными и распространяются свободно. Заметим, что хотя возможностей у этих сред гораздо больше, они дополняют возможности RGui, но не перечёркивают их. Так что навыки работы с RGui пригодятся и при работе в более совершенной среде.

Справка

Верхний пункт меню Справка (в консоли это Консоль, в редакторе скриптов, соответственно, Редактор) даёт короткую подсказку по работе с соответствующим окном. Справка для редактора показана на рис. 1.10.

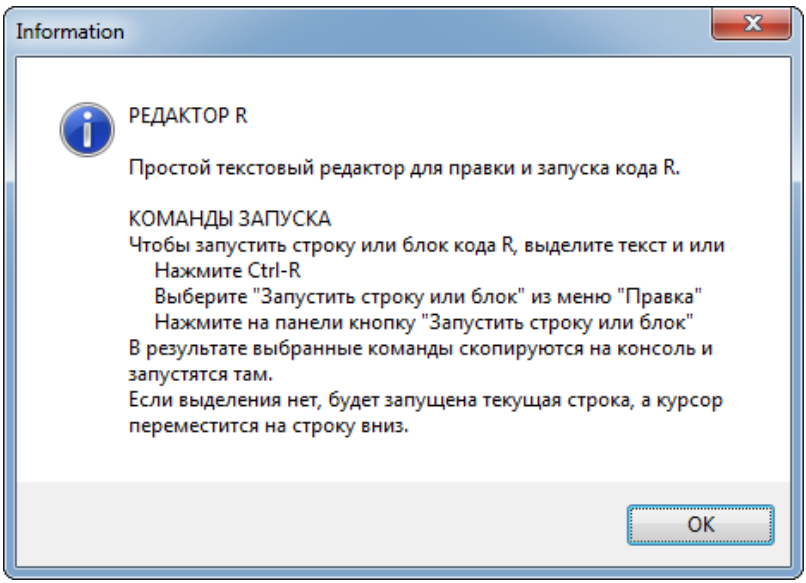

Рис. 1.10 ❖ Справка по работе в редакторе скриптов RGui

Справку по функции с именем `fun` можно получить, набрав в консоли `?fun`. Например, для получения справки по функции создания последовательностей `seq` наберём:

```
?seq
# или
help(seq)
```

В ответ на выполнение этой команды откроется браузер со справочной информацией (рис. 1.11).

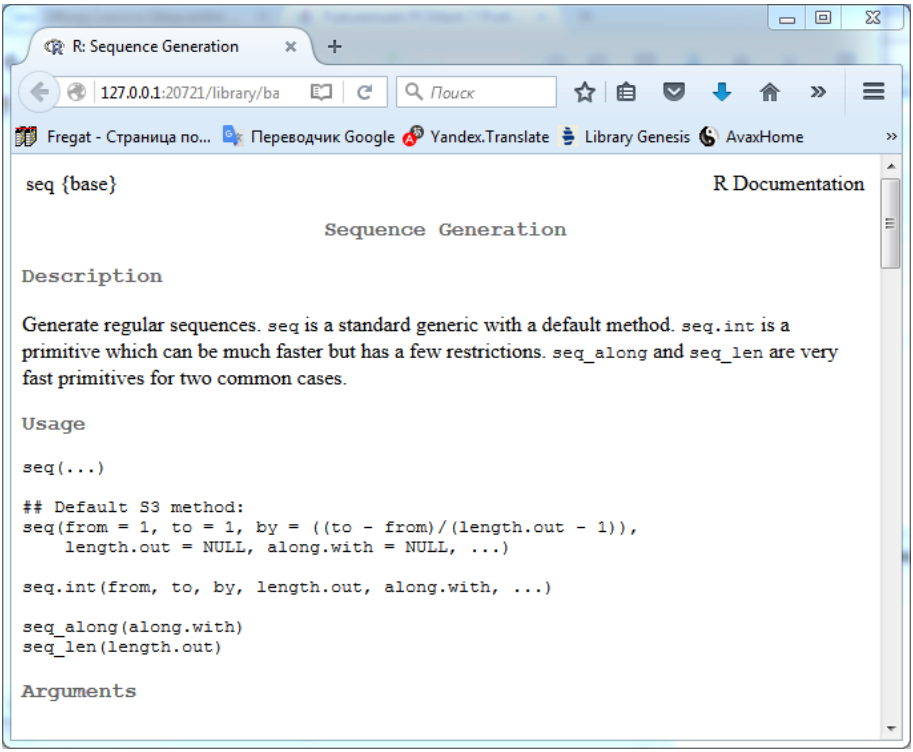

Рис. 1.11 ❖ Справка по функции seq

Глава 2

Скаляры, векторы и матрицы

В этой главе мы научимся создавать переменные и выполнять над ними арифметические и логические операции. Познакомимся с основными типами данных: числовыми, символьными и логическими, а затем узнаем, что все они представляют собой векторы, которые в R вовсе не составной тип данных, а наоборот – простейший. Научимся создавать из векторов матрицы и в несколько строк кода вычислять интегралы.

Арифметические операции и присваивание

Арифметические операции в R выглядят так же, как и во многих других языках программирования:

```
3+2
2*(4-1)/6
3^2; sqrt(4) # ';' служит для разделения выражений в строке
```

```
## [1] 5
## [1] 1
## [1] 9
## [1] 2
```

А вот запись операции присваивания имеет особенности. Оно может обозначаться знаком равенства `'='` или «стрелками», позволяющими присваивать как влево (`a <- b`), так и вправо (`b -> a`).

```
a <- 2   # присвоим значение a - традиционная форма записи для R
3 -> b   # присвоим значение b
c = a+b  # и это - то же присваивание
c        # напечатаем значение c
```

```
## [1] 5
```

Мы будем использовать обозначение `'<-'` как из соображений совместимости со старыми версиями пакета, так и чтобы отличать присваивание от других операций, например от задания имён столбцов в таблицах – в последнем случае всегда используется `'='`.

Имена

При назначении имён переменных и функций придерживаются следующих правил:

1. Использовать для имён только латинские буквы, символ подчёркивания `'_'`, цифры и точку `'.'`; при этом имя не должно начинаться с цифры или точки: `plot_new2` и `plot.new` – это правильно, а `.plot` и `2plot` – нет.
2. Помнить, что R чувствителен к регистру: `Var`, `var` и `VAR` – это разные имена.
3. Не давать объектам имена, занятые распространёнными функциями (например, не следует создавать функцию с именем `c()`) или ключевыми словами (особенно это касается `T`, `F`, `NA`, `NaN`, `Inf`).

Одна из особенностей R состоит в том, что в именах объектов допустимо использовать точку. Её нередко применяют для разделения смысловых частей имени, подобно тому как в других языках используют символ подчёркивания (который в ранних версиях R использовать было нельзя). Так, функция `install.packages()` служит для установки новых пакетов расширений, но это именно функция, а не метод `packages()` объекта `install`.

Проверить, существует ли переменная или функция с заданным именем, можно при помощи функции `exists`:

```
exists('a')    # переменная 'a' уже определена,
```

```
## [1] TRUE
```

```
exists('d')    # ...a 'd' - ещё нет
```

```
## [1] FALSE
```

```
exists('all')  # существует функция с именем 'all'
```

```
## [1] TRUE
```

Параметр `mode` позволяет указать вид объектов, среди которых выполняется поиск. Так, переменная `a` существует, а вот функция `a` – нет:

```
exists("a", mode="function")
```

```
## [1] FALSE
```