5.5 Data intake

> Every easy data format is alike. Every difficult data format is difficult in its own way. —inspired by Leo Tolstoy and Hadley Wickham

The tools that we develop in this book allow one to work with data in R. However, most data sets are not available in R to begin with—they are often stored in a different file format. While R has sophisticated abilities for reading data in a variety of formats, it is not without limits. For data that are not in a file, one common form of data intake is *Web scraping*, in which data from the Internet are processed as (structured) text and converted into data. Such data often have errors that stem from blunders in data entry or from deficiencies in the way data are stored or coded. Correcting such errors is called *data cleaning*.

The native file format for R is usually given the suffix `.Rda` (or sometimes, `.RData`). Any object in your R environment can be written to this file format using the `save()` command. Using the `compress` argument will make these files smaller.

```
save(hr_leaders, file = "hr_leaders.rda", compress = "xz")
```

This file format is usually an efficient means for storing data, but it is not the most portable. To load a stored object into your R environment, use the `load()` command.

```
load(file = "hr_leaders.rda")
```

Pro Tip: Maintaining the provenance of data from beginning to the end of an analysis is an important part of a reproducible workflow. This can be facilitated by creating one R Markdown file or notebook that undertakes the data wrangling and generates an analytic data set (using `save()`) that can be read (using `load()`) into a second R Markdown file.

5.5.1 Data-table friendly formats

Many formats for data are essentially equivalent to data tables. When you come across data in a format that you don't recognize, it is worth checking whether it is one of the data-table friendly formats. Sometimes the filename extension provides an indication. Here are several, each with a brief description:

CSV: a non-proprietary comma separated text format that is widely used for data exchange between different software packages. CSVs are easy to understand, but are not compressed, and therefore can take up more space on disk than other formats.

Pro Tip: Be careful with date and time variables in CSV format: these can sometimes be formatted in inconsistent ways that make it more challenging to ingest.

Software-package specific format some common examples include:

Octave (and through that, MATLAB): widely used in engineering and physics

Stata: commonly used for economic research

SPSS: commonly used for social science research

Minitab: often used in business applications

SAS: often used for large data sets

Epi: used by the Centers for Disease Control (CDC) for health and epidemiology data

Relational databases: the form that much of institutional, actively-updated data are stored in. This includes business transaction records, government records, Web logs, and so on. (See Chapter 12 for a discussion of relational database management systems.)

Excel: a set of proprietary spreadsheet formats heavily used in business. Watch out, though. Just because something is stored in an Excel format doesn't mean it is a data table. Excel is sometimes used as a kind of tablecloth for writing down data with no particular scheme in mind.

Web-related: For example:

- HTML (hypertext markup language): `<table>` format
- XML (extensible markup language) format, a tree-based document structure
- JSON (JavaScript Object Notation) is an increasingly common data format that breaks the "rows-and-columns" paradigm (see Section 17.2.4)
- Google spreadsheets: published as HTML
- Application programming interfaces (API)

The procedure for reading data in one of these formats varies depending on the format. For Excel or Google spreadsheet data, it is sometimes easiest to use the application software to export the data as a CSV file. There are also R packages for reading directly from either (`readxl` and `googlesheets`, respectively), which are useful if the spreadsheet is being updated frequently. For the technical software package formats, the `foreign` R package provides useful reading and writing functions. For relational databases, even if they are on a remote server, there are several useful R packages that allow you to connect to these databases directly, most notably `dplyr` and `DBI`. CSV and HTML `<table>` formats are frequently encountered sources for data scraping. The next subsections give a bit more detail about how to read them into R.

CSV (comma separated value) files

This text format can be read with a huge variety of software. It has a data table format, with the values of variables in each case separated by commas. Here is an example of the first several lines of a CSV file:

```
"year","sex","name","n","prop"
1880,"F","Mary",7065,0.0723835869064085
1880,"F","Anna",2604,0.0266789611187951
1880,"F","Emma",2003,0.0205214896777829
1880,"F","Elizabeth",1939,0.0198657855642641
1880,"F","Minnie",1746,0.0178884278469341
1880,"F","Margaret",1578,0.0161672045489473
```

The top row usually (but not always) contains the variable names. Quotation marks are often used at the start and end of character strings—these quotation marks are not part of the content of the string, but are useful if, say, you want to include a comma in the text of

a field. CSV files are often named with the `.csv` suffix; it is also common for them to be named with `.txt`, `.dat`, or other things. You will also see characters other than commas being used to delimit the fields: Tabs and vertical bars are particularly common.

Since reading from a CSV file is so common, several implementations are available. The `read.csv()` function in the `base` package is perhaps the most widely used, but the more recent `read_csv()` function in the `readr` package is noticeably faster for large CSVs. CSV files need not exist on your local hard drive. For example, here is a way to access a `.csv` file over the Internet using a URL (universal resource locator).

```
myURL <- "http://tiny.cc/dcf/houses-for-sale.csv"
Houses <- readr::read_csv(myURL)
head(Houses, 3)

# A tibble: 3  16
   price lot_size waterfront   age land_value construction air_cond  fuel
   <int>    <dbl>      <int> <int>      <int>        <int>    <int> <int>
1 132500     0.09          0    42      50000            0        0     3
2 181115     0.92          0     0      22300            0        0     2
3 109000     0.19          0   133       7300            0        0     2
# ... with 8 more variables: heat <int>, sewer <int>, living_area <int>,
#   pct_college <int>, bedrooms <int>, fireplaces <int>, bathrooms <dbl>,
#   rooms <int>
```

Just as reading a data file from the Internet uses a URL, reading a file on your computer uses a complete name, called a *path* to the file. Although many people are used to using a mouse-based selector to access their files, being specific about the full path to your files is important to ensure the reproducibility of your code (see Appendix D).

HTML tables

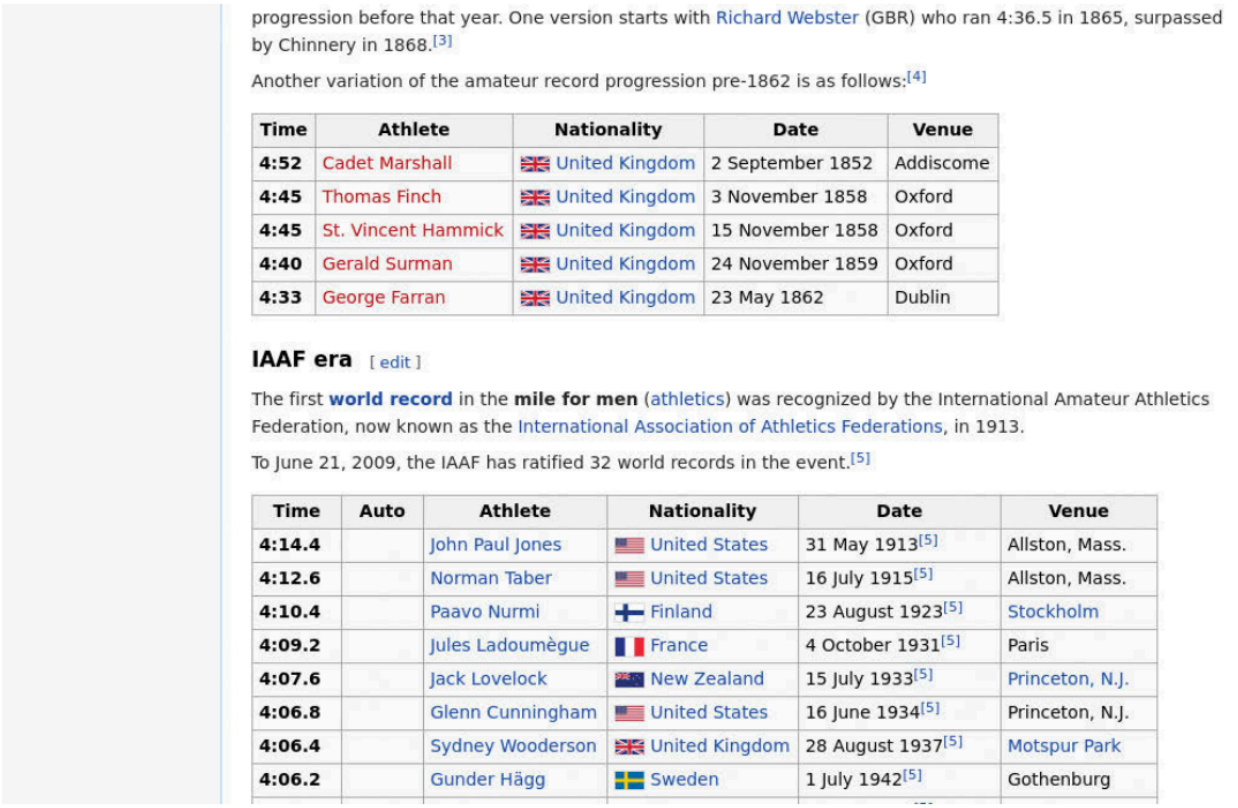
Web pages are HTML documents, which are then translated by a browser to the formatted content that users see. HTML includes facilities for presenting tabular content. The HTML `<table>` markup is often the way human-readable data is arranged.

When you have the URL of a page containing one or more tables, it is sometimes easy to read them into R as data tables. Since they are not CSVs, we can't use `read_csv()`. Instead, we use functionality in the `rvest` package to ingest the HTML as a data structure in R. Once you have the content of the Web page, you can translate any tables in the page from HTML to data table format.

In this brief example, we will investigate the progression of the world record time in the mile run, as detailed on the Wikipedia. This page (see Figure 5.7) contains several tables, each of which contains a list of new world records for a different class of athlete (e.g., men, women, amateur, professional, etc.).

```
library(rvest)
library(methods)
url <- "http://en.wikipedia.org/wiki/Mile_run_world_record_progression"
tables <- url %>%
  read_html() %>%
  html_nodes("table")
```

The result, `tables`, is not a data table. Instead, it is a *list* (see Appendix B) of the tables found in the Web page. Use `length()` to find how many items there are in the list of tables.



progression before that year. One version starts with Richard Webster (GBR) who ran 4:36.5 in 1865, surpassed by Chinnery in 1868.[3]

Another variation of the amateur record progression pre-1862 is as follows:[4]

Time	Athlete	Nationality	Date	Venue
4:52	Cadet Marshall	United Kingdom	2 September 1852	Addiscome
4:45	Thomas Finch	United Kingdom	3 November 1858	Oxford
4:45	St. Vincent Hammick	United Kingdom	15 November 1858	Oxford
4:40	Gerald Surman	United Kingdom	24 November 1859	Oxford
4:33	George Farran	United Kingdom	23 May 1862	Dublin

IAAF era [edit]

The first **world record** in the **mile for men** (athletics) was recognized by the International Amateur Athletics Federation, now known as the International Association of Athletics Federations, in 1913.

To June 21, 2009, the IAAF has ratified 32 world records in the event.[5]

Time	Auto	Athlete	Nationality	Date	Venue
4:14.4		John Paul Jones	United States	31 May 1913[5]	Allston, Mass.
4:12.6		Norman Taber	United States	16 July 1915[5]	Allston, Mass.
4:10.4		Paavo Nurmi	Finland	23 August 1923[5]	Stockholm
4:09.2		Jules Ladoumègue	France	4 October 1931[5]	Paris
4:07.6		Jack Lovelock	New Zealand	15 July 1933[5]	Princeton, N.J.
4:06.8		Glenn Cunningham	United States	16 June 1934[5]	Princeton, N.J.
4:06.4		Sydney Wooderson	United Kingdom	28 August 1937[5]	Motspur Park
4:06.2		Gunder Hägg	Sweden	1 July 1942[5]	Gothenburg

Figure 5.7: Part of a page on mile-run world records from Wikipedia. Two separate data tables are visible. You can't tell from this small part of the page, but there are seven tables altogether on the page. These two tables are the third and fourth in the page.

```
length(tables)

[1] 7
```

You can access any of those tables using the `[[()` operator. The first table is `tables[[1]]`, the second table is `tables[[2]]`, and so on. The third table—which corresponds to amateur men up until 1862—is shown in Table 5.10.

```
Table3 <- html_table(tables[[3]])
```

Time	Athlete	Nationality	Date	Venue
4:52	Cadet Marshall	United Kingdom	2 September 1852	Addiscome
4:45	Thomas Finch	United Kingdom	3 November 1858	Oxford
4:45	St. Vincent Hammick	United Kingdom	15 November 1858	Oxford
4:40	Gerald Surman	United Kingdom	24 November 1859	Oxford
4:33	George Farran	United Kingdom	23 May 1862	Dublin

Table 5.10: The third table embedded in the Wikipedia page on running records.

Likely of greater interest is the information in the fourth table, which corresponds to the current era of International Amateur Athletics Federation world records. The first few rows of that table are shown in Table 5.11. The last row of that table (now shown) contains the current world record of 3:43.13, which was set by Hicham El Guerrouj of Morocco in Rome on July 7th, 1999.

```
Table4 <- html_table(tables[[4]])
Table4 <- select(Table4, -Auto)  # remove unwanted column
```

Time	Athlete	Nationality	Date	Venue
4:14.4	John Paul Jones	United States	31 May 1913[5]	Allston, Mass.
4:12.6	Norman Taber	United States	16 July 1915[5]	Allston, Mass.
4:10.4	Paavo Nurmi	Finland	23 August 1923[5]	Stockholm
4:09.2	Jules Ladoumgue	France	4 October 1931[5]	Paris
4:07.6	Jack Lovelock	New Zealand	15 July 1933[5]	Princeton, N.J.
4:06.8	Glenn Cunningham	United States	16 June 1934[5]	Princeton, N.J.

Table 5.11: The fourth table embedded in the Wikipedia page on running records.

5.5.2 APIs

An *application programming interface* (API) is a protocol for interacting with a computer program that you can't control. It is a set of agreed-upon instructions for using a "black-box"—not unlike the manual for a television's remote control. APIs provide access to massive troves of public data on the Web, from a vast array of different sources. Not all APIs are the same, but by learning how to use them, you can dramatically increase your ability to pull data into R without having to "scrape" it.

If you want to obtain data from a public source, it is a good idea to check to see whether: a) the company has a public API; b) someone has already written an R package to said interface. These packages don't provide the actual data—they simply provide a series of R functions that allow you to access the actual data. The documentation for each package will explain how to use it to collect data from the original source.

5.5.3 Cleaning data

A person somewhat knowledgeable about running would have little trouble interpreting Tables 5.10 and 5.11 correctly. The `Time` is in minutes and seconds. The `Date` gives the day on which the record was set. When the data table is read into R, both `Time` and `Date` are stored as character strings. Before they can be used, they have to be converted into a format that the computer can process like a date and time. Among other things, this requires dealing with the footnote (listed as `[5]`) at the end of the date information.

Data cleaning refers to taking the information contained in a variable and transforming it to a form in which that information can be used.

Recoding

Table 5.12 displays a few variables from the `Houses` data table we downloaded earlier. It describes 1,728 houses for sale in Saratoga, NY.[1] The full table includes additional variables such as `living_area`, `price`, `bedrooms`, and `bathrooms`. The data on house systems such as `sewer_type` and `heat_type` have been stored as numbers, even though they are really categorical.

There is nothing fundamentally wrong with using integers to encode, say, fuel type, though it may be confusing to interpret results. What is worse is that the numbers imply a meaningful order to the categories when there is none.

[1] The example comes from Richard De Veaux at Williams College.