

Chapter 6, Part 2: Data Intake

- ▶ In many of the examples we've worked with in class, the data sets are already in R, as built-in data sets or as part of a package.
- ▶ Most data sets you will encounter outside a classroom are not already in R.
- ▶ They are typically stored on the web, or in a directory on a computer, or on a server or in the cloud, etc.
- ▶ They are likely in a different file format than R's default data format.
- ▶ Using such data in R requires such tasks as reading in a file (or *web scraping*) and *data cleaning* to fix problems in the data.

The Native R File Format

- ▶ The usual format for data files already in R is the `.rda` format, also suffixed `.RData`
- ▶ We can write an object to this file format using the `saveRDS` function.
- ▶ To load an `.rda` file into the R environment, we use the `readRDS` function.

Some Common Formats for Data Tables

- ▶ The format of a data file can also be inferred from the file extension.
- ▶ The comma-separated values (.csv) format is a widely used non-proprietary data format that just about any software package can recognize.
- ▶ Many software packages (like Minitab, SAS, SPSS, etc.) have data file formats specific to that package.
- ▶ Relational databases are common for storing institutional records that are actively updated. These include business transactions, government records, website logs, etc.
- ▶ The *SQL* language is ideal for querying such relational databases.

Other Common Formats for Data Tables

- ▶ *Excel* is proprietary software for spreadsheets very common in business settings.
- ▶ *Google Sheets* is a similar software for spreadsheets.
- ▶ Web-related data formats include *HTML*, *XML*, *JSON* – and *APIs*, which are protocols for interacting with external data.

Reading External Data Files into R

- ▶ The best way to actually read external data into R depends on the format.
- ▶ Often (not always), Excel spreadsheets and Google sheets can be saved as `.csv` files, which makes it easier to read data into software.
- ▶ Be careful – just because something is saved in a spreadsheet doesn't mean it's a tidy data table.
- ▶ The `readxl` package is designed to read data from Excel into R directly, and the `googlesheets4` package is designed to read data from Google Sheets into R directly.
- ▶ The packages `dbplyr` and `DBI` sometimes allow connection to relational databases on remote servers.
- ▶ The `readr` package is a powerful package for reading `.csv` files, and the `rvest` package is designed to read *HTML* tables.

Reading .csv Files into R

- ▶ “Comma-Separated Values” files have data values separated by commas.
- ▶ A .csv file in a spreadsheet usually appears like an ordinary spreadsheet of cells in rows and columns.
- ▶ If you view a .csv file in a text editor, you’ll see the commas, and character data entries will be surrounded by quotation marks (see simple baby names example).
- ▶ Other delimiters besides commas could be used to separate data values, and the `read.csv` and `readr::read_csv` functions allow you to specify another delimiter other than the default comma.
- ▶ The top row is usually a header with variable names, although if the header row is missing, this can be specified in `read.csv` and `read_csv` .

Specifying the Location of the External Data File

- ▶ The R functions for reading external data files typically have a `file =` argument where you can specify the location of the file.
- ▶ This could be the complete *path* name to a location on your local computer.
- ▶ It could also be a *universal resource locator* (URL), which is a website address that includes the file name, if the file is on the web.
- ▶ It's possible to use a mouse-based selector to select your files, especially in RStudio, or to use the menu options in R to navigate to the directory where a file is.
- ▶ However, your code will be more reproducible and usable for others if you are specific about the path to your files, e.g., by including the full path name in your code.

Reading *HTML* Tables on the Web

- ▶ Webpages are HTML documents, and they often have data displayed in a table using the `<table>` markup.
- ▶ We can use the `rvest` package in R to convert the HTML on the webpage to an R structure and then convert the tables on the webpage into R data tables.
- ▶ The `read_html` function produces a *list* containing tables taken from a webpage.
- ▶ The `purrr::pluck` function will extract any such tables from this list, and the table can be stored as an R tibble.
- ▶ See the example on the course website with the mile world record progression.

Application Programming Interface

- ▶ An *Application Programming Interface* (API) is a protocol for interacting with data that you cannot control.
- ▶ Using APIs, you can access many publicly available data sets from many sources.
- ▶ Sometimes accessing these require an ID name and access key, which for many public APIs you can obtain for free.
- ▶ For many popular APIs, people have written R packages to allow users to interface with the API.
- ▶ If there is not already a package for accessing data from an API, then you have to look into the documentation of the API to see how to call it yourself.
- ▶ See a brief example on the course web page with a Google Trends API.

Data Cleaning

- ▶ *Data cleaning* refers to transforming data that is read in into a format on which we can do statistical analysis.
- ▶ The times for the mile run in the world record progression table are easily understandable to human eyes.
- ▶ But R will store the time values in a character vector, and we will need to convert formats to be able to do statistical analysis.
- ▶ Recoding categories that are coded using numerical codes is a common type of data cleaning.
- ▶ The translation of how a code corresponds to a category may be stored in a *codebook*, and the translations can be joined to the original data table to allow the recoding.
- ▶ See the Saratoga houses example from the textbook.

Strings and Numbers

- ▶ Sometimes data whose meaning is numeric is stored as a character vector.
- ▶ Other times, we wish treat data that is stored in a numeric vector as a character string.
- ▶ The `parse_number` function takes a character string with numeric content and translates it into a numeric value.
- ▶ The `parse_character` function takes a numeric value/column and translates it into a character one.

Data Cleaning with Dates

- ▶ In raw data files, dates are usually stored as character strings, like 2/12/2026 or 29 Feb 2008.
- ▶ Dates have an order, which is not the same as their alphabetical order, that should be followed with doing a statistical analysis.
- ▶ This is important, for example, when doing a time series plot with dates on the x-axis.
- ▶ There are *many* custom functions in the `lubridate` package that convert dates stored as character strings into usable *date* (`Date`) or *date-time* (`dtm`) values.

Working with Dates

- ▶ Once properly stored, we can do sensible mathematical operations on dates and times, such as using the `interval` function to calculate differences in date or time values.
- ▶ We can also use functions like `hour` and `month` – and many others – to extract parts of variables that are stored as dates or times.
- ▶ See the Ordway birds example from the textbook and others on the course webpage.

Factors and Strings

- ▶ A *factor* in R is a special type of object meant to contain levels of a categorical variable.
- ▶ They allow customized ordering of the levels, which we have seen, using, e.g., `fct_relevel`.
- ▶ When reading data, it can be confusing whether character data entries will be read in as factors or as character strings.
- ▶ By default, `readr::read_csv` reads character strings as strings, not factors.

More on Factors and Strings

- ▶ Some older versions of base R's `read.csv` converts input character data to factors, which often requires converting them back for a character format with `parse_character`.
- ▶ The `forcats` package has useful tools for wrangling factor data.
- ▶ Be aware that some packages can be unpredictable in formatting character input data as either character strings or factors.
- ▶ To check the data types of the variables in your R data table, use the `summary`, `glimpse`, or `str` functions.

Extended Example: Japanese Nuclear Reactors

- ▶ Section 6.4.4 has an extended example, especially dealing with date and time data, on a Japanese nuclear reactor data set.

Japan [\[edit \]](#)

See also: [Nuclear power in Japan](#)

Power station reactors [\[edit \]](#)








Name	Reactor No.	Reactor		Status	Capacity in MW		Construction Start Date	Commercial Operation Date	Closure
		Type	Model		Net	Gross			
Fukushima Daiichi	1	BWR	BWR-3	 Inoperable	439	460	25 July 1967	26 March 1971	19 May 2011
Fukushima Daiichi	2	BWR	BWR-4	 Inoperable	760	784	9 June 1969	18 July 1974	19 May 2011
Fukushima Daiichi	3	BWR	BWR-4	 Inoperable	760	784	28 December 1970	27 March 1976	19 May 2011
Fukushima Daiichi	4	BWR	BWR-4	 Shut down/ Inoperable	760	784	12 February 1973	12 October 1978	19 May 2011
Fukushima Daiichi	5	BWR	BWR-4	 Shut down	760	784	22 May 1972	18 April 1978	17 December 2013
Fukushima Daiichi	6	BWR	BWR-5	 Shut down	1067	1100	26 October 1973	24 October 1979	17 December 2013
Fukushima Daini	1	BWR	BWR-5	 Operation suspended	1067	1100	16 March 1976	20 April 1982	

Figure: Figure 6.6 from MDSR textbook (the current version of this table in Wikipedia is slightly different)