A Short Introduction to Working With Data in R

EXTRAS

Jonathan Whiteley

2023-09-13

1. Downloading Data From the Internet
2. File Encoding & Microsoft Excel$^{\text{TM}}$
3. Reading Data in Other Formats
4. Exporting to other formats

Section 1

Downloading Data From the Internet

Downloading Data From the Internet

Section 2

File Encoding & Microsoft Excel$^{\mathrm{TM}}$

File Encoding & Microsoft Excel^{TM}

- Many OSes and applications encode text using "`UTF-8`" by default.
- Windows uses "`latin1`" encoding by default.
- Excel can save a `.csv` file using `UTF-8` encoding, but in doing so, it adds "**b**yte **o**rder **m**ark" ("BOM") to the file.
  - This is a special character that Excel also uses to recognize that the file is encoded using `UTF-8`.
  - Thus a BOM can make the file "easier" to use with Excel, by allowing it to automatically recognize the `UTF-8` encoding, but it can also **cause problems for other programs** (like `R`) that do not expect such a non-Unicode character.
- Without the BOM, Excel will assume the file is encoded in "`latin1`" if you double-click on the csv file to open it in Excel, even if it was actually encoded with `UTF-8`.
  - This can cause special characters to appear incorrectly.
  - You can still import a `.csv` file encoded in `UTF-8` into Excel correctly, but it requires opening the file within Excel, or importing it using commands in the "Data" ribbon / menu

Read a file with a BOM using read.csv()

- Reading a .csv file with a BOM using the usual method may cause the BOM to be included in the name of the first column (on Windows).

```
bom_bad <- read.csv("../data/data_example_bom.csv",
                    encoding = "UTF-8")
names(bom_bad)
```

- The solution with read.csv() is to use the argument ‘fileEncoding = "UTF-8-BOM"’ (instead of the ‘encoding’ argument)

```
bom <- read.csv("../data/data_example_bom.csv",
                fileEncoding = "UTF-8-BOM")
bom[4, 1:4]
```

```
#     Type Treatment PlantNum  X95
# 4 Québec   chilled        1 14.2
```

Read a file with a BOM using `read_csv()`

- The `readr` package uses "`UTF-8`" encoding by default, and **automatically ignores a BOM**, if present.

```
bom_readr <- readr::read_csv("../data/data_example_bom.csv")
bom_readr[4, 1:4] |> knitr::kable()
```

Type	Treatment	PlantNum	95
Québec	chilled	1	14.2

- `write_csv()` (in the `readr` package) automatically encodes output files using "UTF-8", for greater portability across systems.
  - *except* for older versions of base `R` (`read.csv()`) on Windows :(

!

Hopefully, these examples have demonstrated that the `readr` package makes it easy to work with "`UTF-8`" files by default, on any platform.

Add a BOM to an output file

- Some programs (e.g., Excel) expect a BOM in "`UTF-8`" encoded files.
  - Excel will automatically use `UTF-8` when opening a file with a BOM.
- It is possible to add a BOM to a csv file, but it must be done *manually*:
  - code adapted from this StackOverflow answer

```
writeChar(
  iconv("\ufeff", to = "UTF-8"),
  "output.csv",
  eos = NULL
)
write_csv(Data, "output.csv", append = TRUE, ... )
```

`R` **does not recommend doing this** (see `?file`), so use with caution.

Using other encodings with `readr`

- You can control the encoding used by `readr` functions with the `locale` **argument**.

```
write_csv(Data, "data.csv",
          locale = locale(encoding = "latin1")
)

read_csv(Data, "data.csv",
         locale = locale(encoding = "latin1")
)
```

- See `?readr::read_csv` and `?readr::locale` for details.

Section 3

Reading Data in Other Formats

Reading Data in Other Formats

- Parquet files: an efficient columnar format, popular with Big Data and cloud computing
  - Apache Arrow (i.e., the '`arrow`' package)
- The Data Import Chapter of R for Data Science (2e) describes these tidyverse packages for other types of data:
  - **`haven`** reads SPSS, Stata, and SAS files.
  - `DBI`, along with a database specific backend (e.g. `RMySQL`, `RSQLite`, `RPostgreSQL`, etc.) allows you to run SQL queries on a **database** and return a data frame.
- See the Import Section of R for Data Science (2nd edition) For more details on getting data into R from these and other sources.
- Other options are also described in the R Data Import/Export manual.

Section 4

Exporting to other formats

Writing to Microsoft Excel^{TM} files

Packages that can write to Excel files:

- `xlsx`: read, write, format Excel 2007 (`.xlsx`) and Excel 97/2000/XP/2003 (`.xls`) files.
  - Requires Java and the `rJava` package
- `XLConnect`: comprehensive and cross-platform R package for manipulating Microsoft Excel files (`.xlsx` & `.xls`) from within R.
  - Requires a Java Runtime Environment (JRE)
- `openxlsx`: simplified creation of Excel `.xlsx` files (**not** `.xls`).
  - *No dependency* on Java
- `writexl`: portable, light-weight data frame to **xlsx** exporter.
  - No Java or Excel required

!

I recommend *avoiding* exporting data to Excel files if possible. `csv` files are easier to read to & write from, and can be read by a wider variety of software (they are more portable).
Automated reports can be produced with R Markdown and output to a variety of more portable formats (pdf, HTML, etc.) instead.

References (Extras)

CANSIM / CODR data:

- An ecosystem of R packages to access and process Canadian data
- Analyzing Canadian Demographic and Housing Data