Importing / Exporting / Webscraping

Stats 102A

Miles Chen

Department of Statistics

Week 3 Wednesday

UCLA

Section 1

Importing / Exporting Data

Input from a file

Basic commands

```
readline()  # for getting input from the user via stdin (the terminal)
read.table()
read.csv()
```

If you have text data, remember to use `stringsAsFactors = FALSE`

Great Import Packages in the tidyverse

```
# install.packages("readr")
# install.packages("readxl")
# install.packages("haven")
# install.packages("data.table")
library(readr)  # general file reader (for csv, txt, etc.)
library(readxl) # for importing excel files
library(haven)  # for importing SAS, SPSS, STATA
library(data.table)  # imports large tables quickly
```

Learn more about these packages at

- https://readr.tidyverse.org/
- https://readxl.tidyverse.org/
- https://haven.tidyverse.org/
- https://rdatatable.gitlab.io/data.table/

Package `readr`

`readr` supports seven file formats, each with its own read_ function:

- `read_csv()`: comma separated (CSV) files
- `read_tsv()`: tab separated files
- `read_delim()`: general delimited files
- `read_fwf()`: fixed width files
- `read_table()`: tabular files where columns are separated by white-space.
- `read_log()`: web log files

Package `data.table`

The package `data.table` has a function `fread()` which acts very much like `read_csv()`

It is designed for fastest performance. If you have a very massive data file (like $>$ 1GB), I recommend using `fread()`

The best source of information on how to use `data.table` is the official documentation:

- https://rdatatable.gitlab.io/data.table/

readr::read_csv() vs data.table::fread()

(Taken directly from https://readr.tidyverse.org/)

`data.table` has a function similar to `read_csv()` called `fread`. Compared to `fread`, `readr` functions:

- Are slower (currently ~1.2-2x slower). If you want absolutely the best performance, use data.table::fread().
- Use a slightly more sophisticated parser, recognising both doubled ("""") and backslash escapes ("""), and can produce factors and date/times directly.
- Forces you to supply all parameters, where `fread()` saves you work by automatically guessing the delimiter, whether or not the file has a header, and how many lines to skip.
- Are built on a different underlying infrastructure. Readr functions are designed to be quite general, which makes it easier to add support for new rectangular data formats. `fread()` is designed to be as fast as possible

Package readxl

```
library(readxl)
read_excel("datasets.xls") # will read in the first worksheet
# you can specify a different worksheet by name or number
read_excel("datasets.xls", sheet = "mtcars")
read_excel("datasets.xls", sheet = 2)
```

Package downloader

Occasionally, you'll want to download a data file from the internet. R's native `download.file()` function can achieve this, but often runs into problems, especially with secure HTTPS sites (almost all sites). Function `downloader::download()` resolves a lot of issues with downloading files over HTTPS on Windows and Mac OS.

```
# install.packages("downloader")
library(downloader)
url <- "https://raw.githubusercontent.com/smileschen/playground/master/iris.csv"
download(url, file = "iris.csv")  # files will get saved to your working directory
iris <- read_csv("iris.csv")
```

Saving and Exporting Data

You can save objects to files for reuse.

```
save(object1, object2, ... , file = "object.RData") # native .RData format
write(x, "file.txt", ncol = 1) # saves atomic vector as plain text
write.csv(df, file = "df.csv") # saves a data.frame to csv file
write.csv(df, file = "df.csv", row.names = FALSE) # removes row names
```

Package lubridate

Handling Date and Time info in R can be tedious, frustrating, and painful

The `lubridate` package make getting date info into R much easier.

Documentation: https://lubridate.tidyverse.org/

Without Lubridate

```
sdate1 <- "January 15, 1999"
as.Date(sdate1, "%B %d, %Y") # formatting match perfectly or it will fail
```

```
## [1] "1999-01-15"
```

```
as.Date(sdate1, "%B %d %Y") # comma missing leads to NA
```

```
## [1] NA
```

```
sdate2 <- "12-15-2001"
as.Date(sdate2, "%m-%d-%Y")
```

```
## [1] "2001-12-15"
```

```
as.Date(sdate2, "%m %d %Y") # no dashes leads to NA
```

```
## [1] NA
```

Without Lubridate

```
sdates <- c("January 15, 1999", "12-15-2001", "03/18/2002")
as.Date(sdates,"%B %d, %Y") # only one format at a time
```

```
## [1] "1999-01-15" NA           NA
```

```
as.Date(sdates,"%m-%d-%Y")
```

```
## [1] NA           "2001-12-15" NA
```

```
as.Date(sdates,"%m/%d/%Y")
```

```
## [1] NA           NA           "2002-03-18"
```

Lubridate

```
# install.packages("lubridate")
library(lubridate)
sdates <- c("January 15, 1999", "12-15-2001", "03/18/2002")
mdy(sdates) # Can parse a vector of dates with for different formats
```

```
## [1] "1999-01-15" "2001-12-15" "2002-03-18"
```

Does require that all dates are written in same mdy order

Lubridate

Lubridate requires you to specify the order of the fields.

```
sdate3 <- "03/04/05" # ambiguous date
ymd(sdate3)
```

```
## [1] "2003-04-05"
```

```
mdy(sdate3)
```

```
## [1] "2005-03-04"
```

```
dmy(sdate3)
```

```
## [1] "2005-04-03"
```

```
mdy("25-12-99") # Will return NA if it can't parse it
```

```
## Warning: All formats failed to parse. No formats found.
```

```
## [1] NA
```

Package rvest

```
# install.packages("rvest")
library(rvest)
```

```
## Loading required package: xml2
```

"harvesting" from the web, aka. web scraping

Use the Selector Gadget with rvest: http://selectorgadget.com/

```
vignette("selectorgadget")
```

rvest

rvest is great for fairly static webpages with well defined html tags.

rvest does not work as well for sites that are generated via javascript, e.g. sites with infinite scroll.

rvest does not work well for sites where you must be logged in to view content.

HTML and CSS

A very brief understanding of HTML can help you understand how rvest works.

Webpages are written in a markup language called HTML. Formatting is applied by using tags. To keep a page's format consistent, the designer will often use CSS classes and in a separate file specify how those classes should be displayed.

HTML and CSS

Here's a simplified version of what those tags may look like in the underlying HTML.

```
<ul class="flat-list buttons">
    <li class="first">
        <a href="#" class="comments">4612 comments</a>
    </li>
    <li class="share-button">
        <a href="#">share</a>
    </li>
    <li class="save-button">
        <a href="#">save</a>
    </li>
    <li class="hide-button">
        <a href="#">hide</a>
    </li>
    <li class="report-button">
        <a href="#">report</a>
    </li>
</ul>
```

Selector Gadget

The selector gadget works by looking at the CSS tags and selecting all tags on the webpage that has a specific tag.

I go to the site and click text that I want to capture. The selector gadget highlights everything it will capture in yellow. If there is stuff I do not want, I click that and selector gadget will adjust the tag to unselect.

So if I use the tag `.comments`, the selector gadget will identify all text that is surrounded by a tag with the class `.comments`. [Don't worry about the fact that there's a dot, the selector gadget will determine if there should or should not be a dot included in the tag to use.]

With the tag, we can go back to rvest and select the nodes.

```
old_reddit <- read_html("https://old.reddit.com/")
comment_counts <- old_reddit %>%
  html_nodes(".comments") %>%
  html_text()
comment_counts
```

```
##  [1] "3394 comments"  "497 comments"   "345 comments"
##  [4] "3825 comments"  "1323 comments"  "264 comments"
##  [7] "152 comments"   "908 comments"   "1943 comments"
## [10] "1723 comments"  "2354 comments"  "2321 comments"
## [13] "1777 comments"  "2006 comments"  "25369 comments"
## [16] "112 comments"   "530 comments"   "303 comments"
## [19] "621 comments"   "578 comments"   "2030 comments"
## [22] "2115 comments"  "719 comments"   "1599 comments"
## [25] "2096 comments"
```

Using your scraped text

We can now extract these values with regular expressions (covered later).

```
library(stringr)
as.numeric(str_extract(comment_counts, "\\d+"))
```

```
##  [1]  3394   497   345  3825  1323   264   152   908  1943  1723  2354
## [12]  2321  1777  2006 25369   112   530   303   621   578  2030  2115
## [23]   719  1599  2096
```

Hard for rvest - this site features infinite scroll

You'll notice the css selector is incomprehensible as it was likely generated by javascript. This selector is unlikely to work in the future.

```
reddit <- read_html("http://www.reddit.com")
reddit %>%
  html_nodes("._3wqmjmv3tb_k-PROt7qFZe ._eYtD2XCVieq6emjKBH3m") %>%
  html_text()
```

```
## [1] "Biden revokes presidential permit for Keystone XL pipeline expansion
## [2] "What book series did you love as a kid?"
## [3] "Couple who stormed black child's birthday party with a gun and confed
## [4] "Blden sworn in as U.S. president"
## [5] "Biden signs federal mask mandate, repeals Muslim Ban, and rejoins Par
## [6] "His first photo in the Oval Office"
```

Easier for rvest

The css selector

```
old_reddit <- read_html("https://old.reddit.com/")
old_reddit %>%
  html_nodes(".title.may-blank") %>%
  html_text()
```

```
##  [1] "Biden signs federal mask mandate, repeals Muslim Ban, and rejoins Pa
##  [2] "National treasures"
##  [3] "The squirrel photographer"
##  [4] "His first photo in the Oval Office"
##  [5] "Biden revokes presidential permit for Keystone XL pipeline expansion
##  [6] "Historically accurate"
##  [7] "pre-bath vs post-bath"
##  [8] "Joe Biden fist-bumps Barack Obama"
##  [9] "Incoming first daughter Ashley Biden says she won't be working in he
## [10] "White House Website Recognizes Climate Change Is Real Again"
```

Other Webscraping Options

To work with javascript generated pages, you can use `RSelenium`, which is powerful but has a steeper learning curve.

Other language options:

- Python Beautiful Soup - very easy to learn
- Python Scrapy - more complex, can crawl across many pages
- Python Selenium - supports javascript generated pages

Section 2

rvest demo