# Video 16: Importing / Exporting Data

Stats 102A

Miles Chen, PhD

# Input from a file

## Basic commands

```
readline()  # for getting input from the user via stdin (the termir
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.

```
1 # install.packages("downloader")
2 library(downloader)
3 url <- "https://raw.githubusercontent.com/smileschen/playground/mas
4 download(url, file = "iris.csv")  # files get saved to your working
5 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
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 nam
```

# 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.

Recommended Reading: https://lubridate.tidyverse.org/

Lubridate Cheat Sheet: https://rawgit.com/rstudio/cheatsheets/main/lubridate.pdf

# Without Lubridate

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

```
[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

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

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

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

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

```
1 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 format
```

```
[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.

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

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

```
1 mdy(sdate3)
```

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

```
1 dmy(sdate3)
```

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

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

```
[1] NA
```