Importing & Exporting Data with R

STA 418/518 - Statistical Computing and Graphics with R

Andrew DiLernia

Complete the following activity using R Markdown and then submit your .Rmd file and output Word, PDF, or HTML document on Blackboard.

Learning Objectives

1. Importing CSV files into R
2. Importing special data files into R
3. Exporting data from R

➡️ First, create a new R Markdown document with *File > New File > R Markdown…* Knit it by clicking the Knit button (top left).

- Knit it by using the appropriate keyboard shortcut (**Mac**: *Command + Shift + K*, **Windows**: *Ctrl + Shift + K*).
- Load packages necessary for this activity using the code below. Note: you may need to install packages beforehand by using the `install.packages()` function and the package name in quotes.

```
library(data.table)
library(tidyverse)
library(knitr)
library(arrow)
library(bench)

library(ggbeeswarm)
library(haven)
```

Importing data from outside R

CSV files

- **C**omma **S**eparated **V**alue files, such as the file here, contain values separated by commas
- CSV files are one of the most common types of data files

```
case,site,sex,age,head_length_mm,skull_width_mm,total_length_cm,tail_length_cm
1,Cambarville,Male,8,94.1,60.4,89,36,74.5,54.5,15.2,28,36,Victoria
2,Cambarville,Female,6,92.5,57.6,91.5,36.5,72.5,51.2,16,28.5,33,Victoria
3,Cambarville,Female,6,94,60,95.5,39,75.4,51.9,15.5,30,34,Victoria
4,Cambarville,Female,6,93.2,57.1,92,38,76.1,52.2,15.2,28,34,Victoria
5,Cambarville,Female,2,91.5,56.3,85.5,36,71,53.2,15.1,28.5,33,Victoria
6,Cambarville,Female,1,93.1,54.8,90.5,35.5,73.2,53.6,14.2,30,32,Victoria
7,Cambarville,Male,2,95.3,58.2,89.5,36,71.5,52,14.2,30,34.5,Victoria
8,Cambarville,Female,6,94.8,57.6,91,37,72.7,53.9,14.5,29,34,Victoria
9,Cambarville,Female,9,93.4,56.3,91.5,37,72.4,52.9,15.5,28,33,Victoria
10,Cambarville,Female,6,91.8,58,89.5,37.5,70.9,53.4,14.4,27.5,32,Victoria
11,Cambarville,Female,9,93.3,57.2,89.5,39,77.2,51.3,14.9,31,34,Victoria
12,Cambarville,Female,5,94.9,55.6,92,35.5,71.7,51,15.3,28,33,Victoria
```

- The `read.csv()` base R function is the most common way people import CSV files into RStudio.

➡️ Import the possum.csv data directly from the URL https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/possum.csv into R using the `read.csv()` function creating an object called `possums`. Print the first 5 rows of the data using the `slice_head()` function and the first 6 columns using the `select()` function as in the code below.

```
possums %>% dplyr::select(1:6) %>%
  slice_head(n = 5) %>% kable()
```

case	site	sex	age	head_length_mm	skull_width_mm
1	Cambarville	Male	8	94.1	60.4
2	Cambarville	Female	6	92.5	57.6
3	Cambarville	Female	6	94.0	60.0
4	Cambarville	Female	6	93.2	57.1
5	Cambarville	Female	2	91.5	56.3

- Instead of manually writing code, there is an import wizard available in RStudio as well.
  - To use the import wizard, click *File* -> *Import Dataset* and then select the appropriate option based on the file type of the external data set.

- The *From Text* options are commonly for .csv and .txt files, *From Excel* is commonly for .xlsx or .xls files from Microsoft Excel, and then there are options for SPSS, SAS, and Stata files as well.

➡️ Download the csv file we just imported, but then use the import wizard to import the possum.csv data using the `read.csv()` function as well.

Importing large data sets

Several packages have been introduced that allow reading (and writing) of large data files with R efficiently:

- `readr`
- `data.table`
- `arrow`

`read_csv` from the `readr` package

- The `readr` package is part of the `tidyverse` and is useful for importing data into RStudio.
- `read_csv()` imports csv files and creates a tidy table, called a "tibble"

➡️ Import the possum.csv data directly from the URL https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/possum.csv into R

using `read_csv()`, and again display a subset of the data set using the previously provided code.

`read_parquet()` from the `arrow` package

Parquet is an open source, column-based data file format, as opposed to row-based storage like CSV files.

observation 1 variable 1	observation 1 variable 2	⋯	observation 1 variable *p*
observation 2 variable 1	observation 2 variable 2	⋯	observation 2 variable *p*
⋮	⋮	⋱	⋮
observation *N* variable 1	observation *N* variable 2	⋯	observation *N* variable *p*

Row-based storage used in CSV files for tables with N observations of p variables.

variable 1 observation 1	variable 1 observation 2	⋯	variable 1 observation *N*
variable 2 observation 1	variable 2 observation 2	⋯	variable 2 observation *N*
⋮	⋮	⋱	⋮
variable *p* observation 1	variable *p* observation 2	⋯	variable *p* observation *N*

Column-based storage used in parquet files for tables with N observations of p variables.

For example, consider a data set on islands with land masses exceeding 10,000 square miles containing 48 observations of 2 variables*.

- *Island*: name of island
- *Area*: area in thousands of square miles

Row-based storage of islands data set.

Island	*Area*
Africa	11506
Antarctica	5500
⋮	⋮
Victoria	82

Column-based storage of islands data set.

Island	Africa	Antarctica	…	Victoria
Area	11506	5500	…	82

**Data from McNeil, D. R. (1977) Interactive Data Analysis. Wiley.*

This columnar storage format can improve efficiency in reading and writing files.

➡️ Download the parquet file from the URL https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/possum.parquet and then import the data into R using the `read_parquet()` function from the `arrow` package using the code below. Display a subset of the data, only the first 5 rows of the data and the first 8 columns, and print the subset using the `kable()` function.

```
library(arrow)

# Import after manually downloading file
possums <- read_parquet("possum.parquet")
```

fread() from the data.table package

- `readr()` is not particularly fast (though `read_csv()` **is** often faster than `read.csv()`):
- Enter the `data.table` package and the much faster `fread()` function, which returns a `data.table` object.

➡️ Import the possum.csv data directly from the URL https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/possum.csv into R using the `fread()` function from the `data.table` package.

Comparing read & write speeds

Let's compare how quickly these functions import and export slightly larger data from R. First, we simulate data with 50,000 observations, 2 variables, and roughly 10% of its values being missing. This is not truly big data, but is large enough to demonstrate differences in speeds across various functions.

```
# Generating 'big data'
set.seed(1994)
x <- runif(5e4)
y <- runif(5e4)
x[sample(5e4, 5e3)] <- NA
y[sample(5e4, 5e3)] <- NA
bigData <- as.data.frame(x = x, y = y)
```

```r
# Saving as CSV file w/ data.table
fwrite(bigData, "bigData.csv")

# Saving as parquet file
write_parquet(bigData, "bigData.parquet")

# Saving as RDS file
write_rds(bigData, "bigData.rds")
```

Importing data into R

Using our artificial data, we will consider 5 different functions for importing data into R:

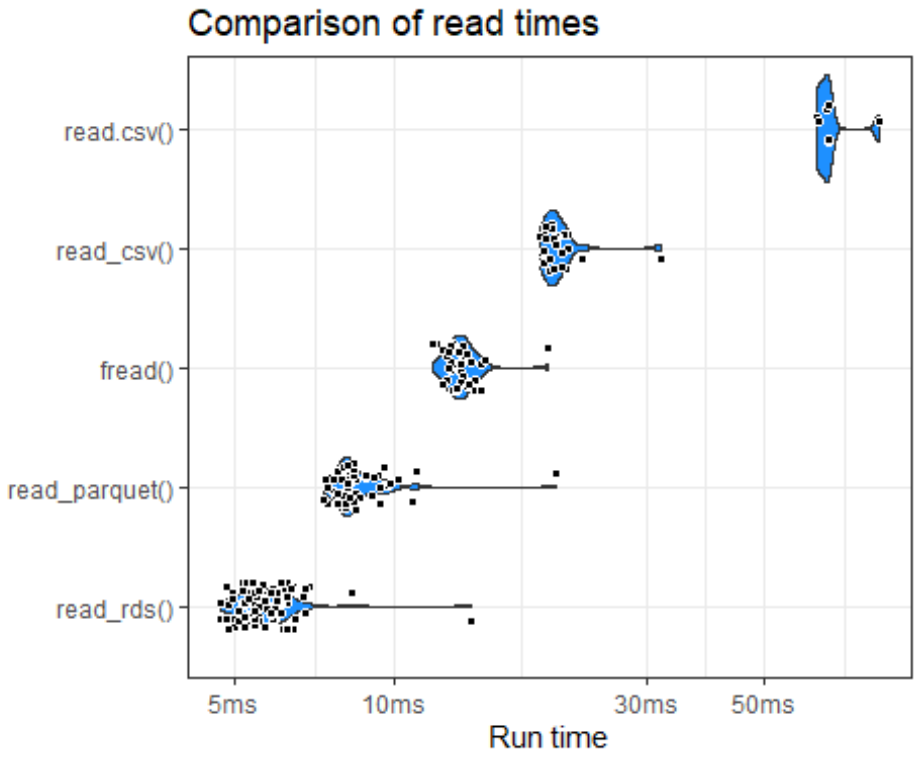
1. `read.csv()`
2. `read_csv()`
3. `fread()`
4. `read_parquet()`
5. `read_rds()`

➡️ Reproduce the table and violin plots below comparing the differences in import speeds between the various functions using the code below.

```r
library(bench)

# Comparing run times
readBmResult <- mark(read.csv("bigData.csv"), read_csv("bigData.csv",
show_col_types = FALSE),
                fread("bigData.csv"), read_rds("bigData.rds"),
     read_parquet("bigData.parquet", as_tibble = TRUE),
     check = FALSE, min_iterations = 5)

ggObj <- plot(readBmResult)

importTimes <- ggObj$data %>% mutate(expression =
paste0(map_chr(str_split(expression, pattern = "[(]"), 1), "()"))

# Printing table
importTimes %>% arrange(desc(median)) %>%
  select(expression:mem_alloc) %>% distinct() %>% knitr::kable()
```

expression	min	median	itr/sec	mem_alloc
read.csv()	62.83ms	65.59ms	14.94561	1.7MB
read_csv()	18.84ms	19.95ms	49.81647	2.06MB
fread()	11.76ms	13.29ms	75.39459	1.17MB
read_parquet()	7.34ms	8.18ms	118.70892	3.46MB
read_rds()	4.7ms	5.67ms	175.59747	395.83KB

```
# Creating violin plots
importTimes %>% ggplot(aes(x = time, y = fct_reorder(expression, time))) +
  geom_violin(fill = "dodgerblue") +
  geom_jitter(height = 0.2, pch=21, fill = "black", color = "white") +
  labs(title = "Comparison of read times", y = "", x = "Run time") +
theme_bw()
```



Parquet files are typically the fastest to be read into R using `read_parquet()` from the `arrow` package, while the base R function `read.csv()` is the slowest. Often using the `read_csv` function or other functions from `readr` imports data in a small amount of time though, so the difference in performance is negligible unless the data is quite large.

Moreover, often the type of data file you have is not under your control, so unless it is a CSV file there is not always a choice in which function to use for importing data sets.

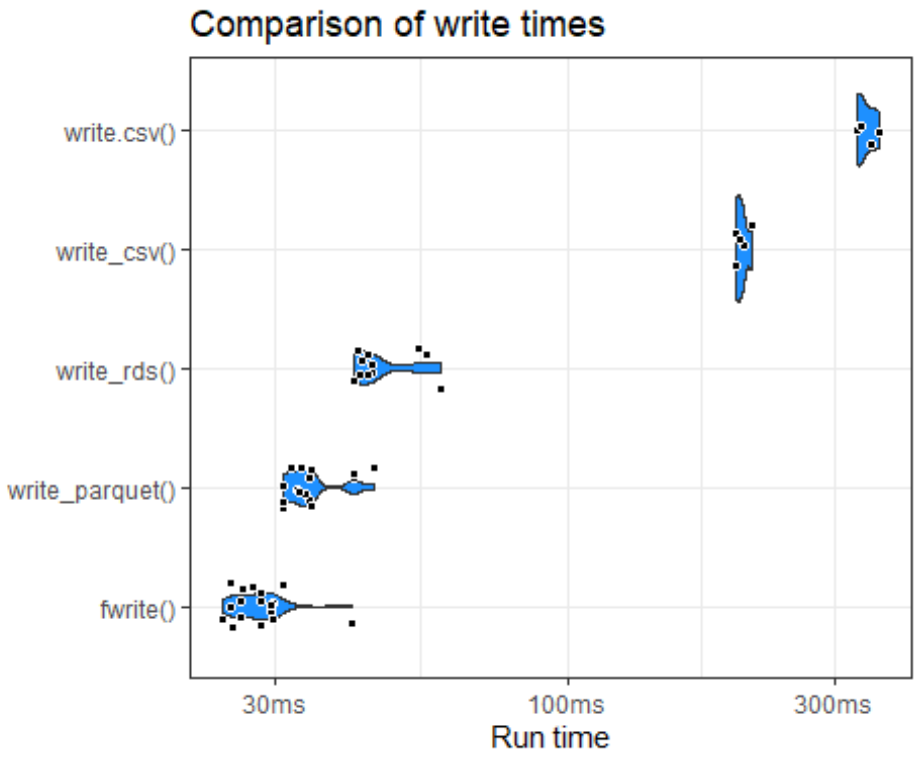
Exporting data from R

We will consider 5 different functions for exporting data from R:

1. `write.csv()`
2. `write_csv()`
3. `write_rds()`
4. `write_parquet()`
5. `fwrite()`

➡️ Reproduce the table and violin plots below comparing the differences in export speeds between the various functions.

expression	min	median	itr/sec	mem_alloc
write.csv()	327.2ms	332.3ms	2.943812	1.51MB
write_csv()	198.9ms	203.5ms	4.893857	101.72KB
write_rds()	41.7ms	44.2ms	21.303458	8.63KB
write_parquet()	31.1ms	33.8ms	28.834446	15.08KB
fwrite()	24.3ms	28.3ms	35.197421	0B



We can see that exporting a data set of this size to a CSV file is fastest using `fwrite()` from the `data.table` package, while the base R function `write.csv()` is again the slowest. Exporting data generally takes longer than importing data, and typically one is constrained in the file formats that are needed.

There are a number of considerations aside from speed when considering which function to use for exporting data:

- Who is the data being exported for?
- Is a particular file type required?
- Are you exporting tabular data, or a more complex R object?

If you want to make a data file as widely accessible to others as possible, CSV files are a great option for tabular data. Parquet files are fast to work with, can take up less hard drive space, and are accessible for most programming languages, but they are not widely used like CSV files. RDS files are specific to R, so are fast and can handle R objects outside of just tabular data (e.g., lists, arrays, model fit objects, etc.), but are limited in their utility outside of R users.

Reading "foreign" data

Although it is hard to believe, not everyone uses R 😱. You will likely work with people who use other statistical software programs:

- 
- 
- 
- 
- 

Working with these types of data files, we have a few options:

Option 1: Export the data from those programs into a format R can read

- This may be easy to do, or not
- Likely some information loss when exporting

Option 2: Use specialized packages in R that can read "foreign" data formats

Several options are available:

- `foreign`
- `readstata13`, `sas7bdat`, `R.matlab`
- `haven`

The `foreign` package

The `foreign` package provides a suite a functions for reading a variety of data formats:

- Stata: `read.dta()`
- Epi Info: `read.epiinfo()`
- Minitab: `read.mtb()`
- Octave: `read.octave()`
- SPSS: `read.spss()`
- Systat: `read.syd()`

- SAS: `read.xport()`

The `foreign` package is useful for importing data from more obscure formats (Epi Info?), but more recent packages like `haven` are a better choice for SAS and Stata files.

The haven package

Hadley Wickham strikes again! The `haven` package is part of the tidyverse, and offers some functions for importing data from various file formats:

- SAS: `read_sas()` and `read_xpt()`
- SPSS: `read_sav()` and `read_por()`
- Stata: `read_dta()`

These are generally much faster and more flexible than other options, and tend to preserve more information embedded in the files. There are both `read_*()` functions for importing data and `write_*()` functions for exporting data as well.

`read_excel()` from the `readxl` package

- Also a part of the tidyverse, the `readxl` package provides means of reading Excel files (.xlsx or .xls) into R.
- The `read_excel()` function reads both .xls and .xlsx files and detects the format from the file extension.