Importing and Exporting Data

Arjun Mannem

Friday, January 26, 2024

library(data.table)  
library(tidyverse)  
library(arrow)  
library(bench)  
library(flextable)  
library(ggbeeswarm)

### Importing data from outside R

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 <- read.csv("https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/possum.csv")  
  
possums|>  
 dplyr:: select(1:6) |>  
 slice\_head(n = 5) |>  
 flextable() |>  
 autofit()

| 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 |

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.

*read as possums*

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 flextable() function.

library(arrow)  
  
# Downloading file from GitHub  
download.file(url = "https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/possum.parquet", destfile = "possum.parquet", mode = "wb")  
  
# Import after manually downloading file  
possums\_parquet\_data <- read\_parquet("possum.parquet")

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.

possum\_fread <- fread("https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/possum.csv")

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.

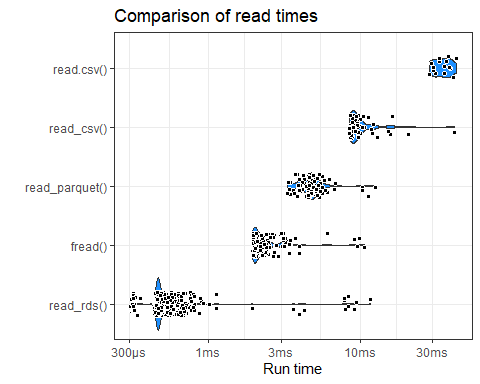
# Generating 'big data'  
set.seed(1994)  
x <- runif(5e4)  
y <- runif(5e4)  
x[sample(5e4, 5e3)] <- NA  
y[sample(5e4, 5e3)] <- NA  
bigData <- as.data.frame(x = x, y = y)  
  
# Saving as CSV file w/ data.table  
fwrite(bigData, "bigData.csv")  
  
# Saving as parquet file  
write\_parquet(bigData, "bigData.parquet")  
  
# Saving as RDS file  
write\_rds(bigData, "bigData.rds")

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

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)) |>   
 dplyr::select(expression:mem\_alloc) |>   
 distinct() |>   
 flextable() |>   
 autofit()

| expression | min | median | itr/sec | mem\_alloc |
| --- | --- | --- | --- | --- |
| read.csv() | 28.33ms | 35.69ms | 28.58151 | 1.7MB |
| read\_csv() | 8.41ms | 9.41ms | 100.20318 | 1.98MB |
| read\_parquet() | 3.28ms | 4.74ms | 212.24997 | 9.01KB |
| fread() | 1.95ms | 2.11ms | 433.65378 | 795.97KB |
| read\_rds() | 306.3µs | 475.2µs | 1,924.05738 | 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()



### Exporting data from R

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

library(bench)  
  
# Comparing run times  
writeBmResult <- mark(write.csv(bigData,"bigData.csv"), write\_csv(bigData,"bigData.csv"),   
 fwrite(bigData,"bigData.csv"), write\_rds(bigData,"bigData.rds"),  
 write\_parquet(bigData,"bigData.parquet"),  
 check = FALSE, min\_iterations = 5)   
  
ggObj <- plot(writeBmResult)  
  
exportTimes <- ggObj$data |> mutate(expression = paste0(map\_chr(str\_split(expression, pattern = "[(]"), 1), "()"))  
  
# Printing table  
exportTimes |>   
 arrange(desc(median)) |>   
 dplyr::select(expression:mem\_alloc) |>   
 distinct() |>   
 flextable() |>   
 autofit()

| expression | min | median | itr/sec | mem\_alloc |
| --- | --- | --- | --- | --- |
| write\_csv() | 212.28ms | 221.2ms | 4.449842 | 103.1KB |
| write.csv() | 164.25ms | 167.91ms | 5.417803 | 520.4KB |
| write\_rds() | 915.7µs | 35.5ms | 29.594041 | 8.63KB |
| write\_parquet() | 12.64ms | 21.04ms | 48.156436 | 14.94KB |
| fwrite() | 3.58ms | 3.93ms | 218.422075 | 0B |

# Creating violin plots  
exportTimes |> 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 write times", y = "", x = "Run time") + theme\_bw()

