Homework2: Reading & Writing Data with R

Evans_G02504972

2023-10-20

1.

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

2(a)In the setup chunk, update the knitr::opts_chunk$set(echo = TRUE)$**codetoknitr** :: $opts_chunk$**set(echo = TRUE, error = TRUE). The error global chunk option being set to TRUE still renders / compiles the document even if an error occurs.**

2.

2(a)Import data on Taylor Swift songs directly from the URL https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/swiftSongs.csv into R using the read_csv() function creating an object called swiftSongs.

```
library(data.table)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()     masks data.table::between()
## x dplyr::filter()      masks stats::filter()
## x dplyr::first()       masks data.table::first()
## x lubridate::hour()    masks data.table::hour()
## x lubridate::isoweek() masks data.table::isoweek()
## x dplyr::lag()         masks stats::lag()
## x dplyr::last()        masks data.table::last()
## x lubridate::mday()    masks data.table::mday()
## x lubridate::minute()  masks data.table::minute()
## x lubridate::month()   masks data.table::month()
## x lubridate::quarter() masks data.table::quarter()
## x lubridate::second()  masks data.table::second()
```

```
## x purrr::transpose()   masks data.table::transpose()
## x lubridate::wday()    masks data.table::wday()
## x lubridate::week()    masks data.table::week()
## x lubridate::yday()    masks data.table::yday()
## x lubridate::year()    masks data.table::year()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(knitr)
library(arrow)
```

```
##
## Attaching package: 'arrow'
##
## The following object is masked from 'package:lubridate':
##
##     duration
##
## The following object is masked from 'package:utils':
##
##     timestamp
```

```
library(bench)
library(ggbeeswarm)
library(haven)
```

```
url <- "https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/swiftSongs.csv"
swift_songs <- read.csv(url)
```

2(b)Print a single table of the swiftSongs data containing the last 13 rows of the data using the slice_tail() function and the 30th through 34th columns using the select() function from the dplyr package.

```
last_13_rows <- swift_songs %>%
  slice_tail(n = 13)

# Use the select() function to choose columns 30 through 34
selected_columns <- last_13_rows %>%
  select(30:34) %>% kable

# Print the resulting table
selected_columns
```

time_signature	duration_ms	explicit	track_name	track_number
4	250093	FALSE	This Love	11
4	248106	FALSE	Tied Together with a Smile	7
4	232106	FALSE	Tim McGraw	1
4	240773	FALSE	Treacherous	3
4	164801	TRUE	Vigilante Shit	8

time_signature	duration_ms	explicit	track_name	track_number
4	191880	FALSE	We Are Never Ever Getting Back Together	8
4	212600	FALSE	Welcome To New York	1
4	234906	FALSE	White Horse	5
4	220440	FALSE	Wildest Dreams	9
4	231453	FALSE	You Belong With Me	6
4	171360	FALSE	You Need To Calm Down	14
4	262173	FALSE	You're Not Sorry	9
4	194206	FALSE	You're On Your Own, Kid	5

2(c)Import data on Taylor Swift songs directly from the URL using the read.csv(), read_csv(), and the fread() functions, comparing the read times using the mark function from the bench package, storing the results of the mark function in an object called readTimes. Specify a minimum of 5 iterations in the mark function.

```
url <- "https://raw.githubusercontent.com/dilernia/STA418-518/main/Data/swiftSongs.csv"
swift_songs <- read_csv(url)
```

```
## Rows: 151 Columns: 34
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (10): youtube_title, youtube_description, youtube_duration, youtube_url...
## dbl  (22): youtube_view_count, youtube_like_count, youtube_favorite_count, y...
## lgl   (1): explicit
## dttm  (1): youtube_publish_date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
last_13_rows <- swift_songs %>%
  slice_tail(n = 13)

# Use the select() function to choose columns 30 through 34
selected_columns <- last_13_rows %>%
  select(30:34) %>% kable

# Print the resulting table
selected_columns
```

time_signature	duration_ms	explicit	track_name	track_number
4	250093	FALSE	This Love	11
4	248106	FALSE	Tied Together with a Smile	7
4	232106	FALSE	Tim McGraw	1
4	240773	FALSE	Treacherous	3
4	164801	TRUE	Vigilante Shit	8
4	191880	FALSE	We Are Never Ever Getting Back Together	8
4	212600	FALSE	Welcome To New York	1

time_signature	duration_ms	explicit	track_name	track_number
4	234906	FALSE	White Horse	5
4	220440	FALSE	Wildest Dreams	9
4	231453	FALSE	You Belong With Me	6
4	171360	FALSE	You Need To Calm Down	14
4	262173	FALSE	You're Not Sorry	9
4	194206	FALSE	You're On Your Own, Kid	5

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

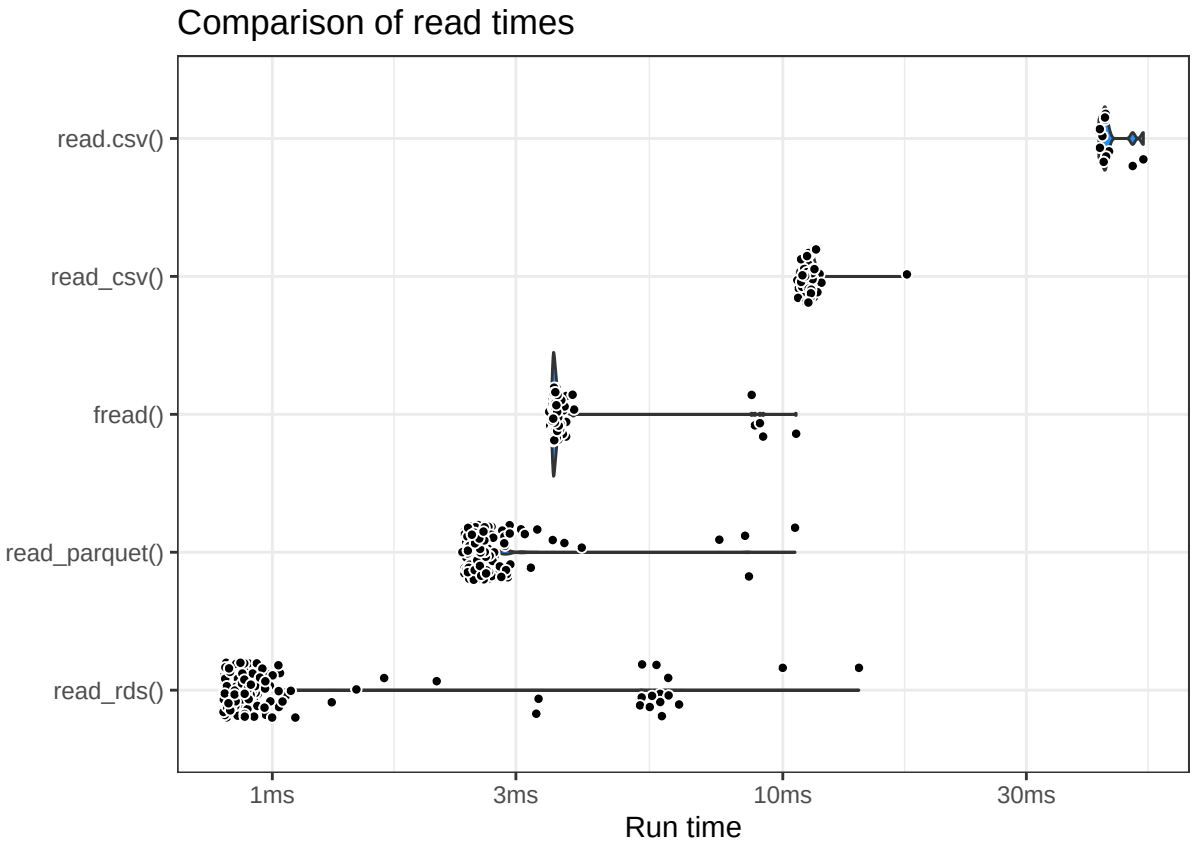
##Table displaying the varying import speed of different R function

```
# 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 = "[(]")

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

expression	min	median	itr/sec	mem_alloc
read.csv()	41.6ms	42.76ms	23.11675	4.5MB
read_csv()	10.68ms	11.28ms	88.87054	867.19KB
fread()	3.49ms	3.57ms	278.47087	1.79MB
read_parquet()	2.36ms	2.53ms	387.01646	3.27MB
read_rds()	801.67us	853.54us	1138.19339	786.5KB

2(d)Create a violin plot to display the varying speeds of the three different functions for importing the data.

```
# 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()
```



2(e)Based on the violin plot, which function was typically the fastest? read_rds function has the shortest median time import hence the fastest.

3

(a)Import data on Taylor Swift songs directly from the URL using the read.csv(), read_csv(), and the fread() functions, comparing the read times using the mark function from the bench package, storing the results of the mark function in an object called readTimes. Specify a minimum of 5 iterations in the mark function.

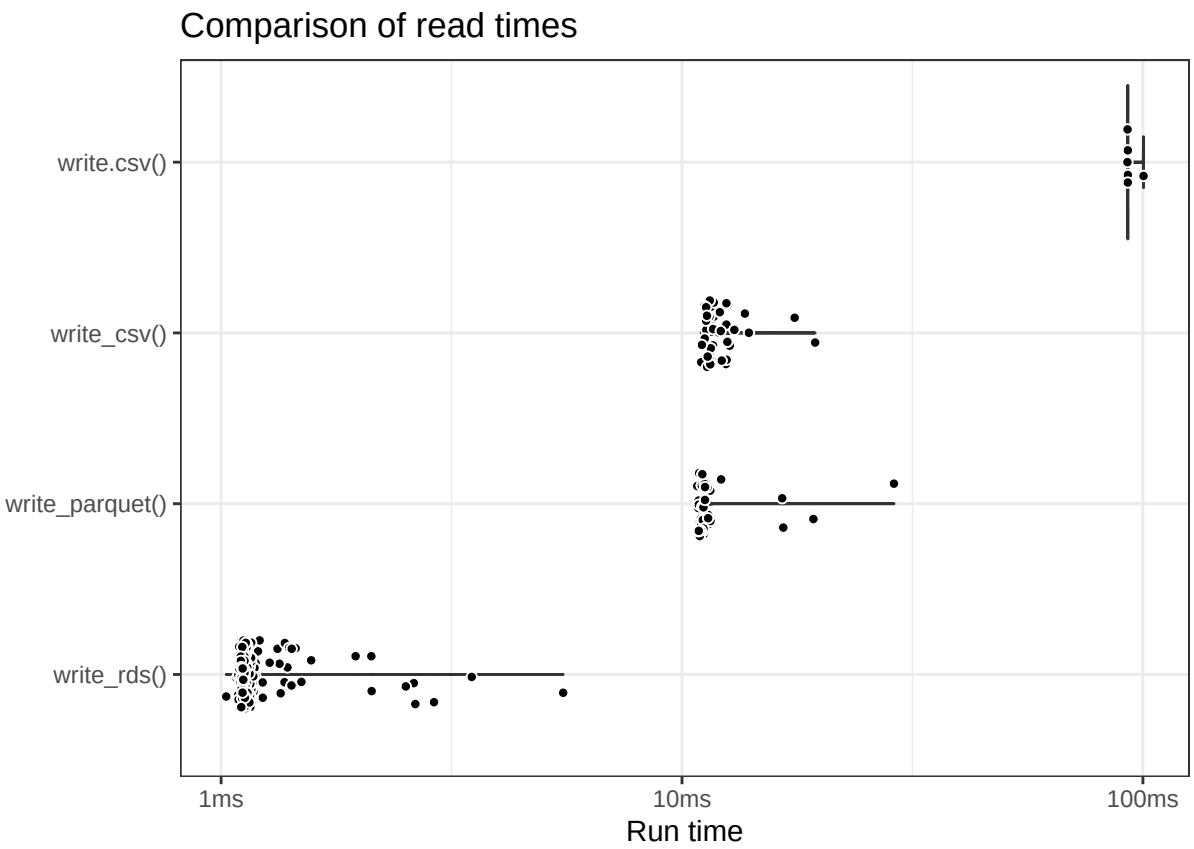
```
##Table displaying the varying export speed of different R function

# Comparing run times
```

```
writeBmResult <- mark(write.csv(bigData,"bigData.csv"), write_csv(bigData,"bigData.csv"),  write_rds(bi
     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 = "[(]")

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

expression	min	median	itr/sec	mem_alloc
write.csv()	92.55ms	92.73ms	10.64473	520.42KB
write_csv()	11.01ms	11.58ms	83.08033	102.62KB
write_parquet()	10.79ms	11.09ms	85.16097	15.12KB
write_rds()	1.03ms	1.11ms	860.71594	8.63KB

3(b)Create a violin plot to display the varying speeds of the three different functions for importing the data.

```
# 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 read times", y = "", x = "Run time") + theme_bw()
```



3(c)Based on the violin plot, which function was typically the fastest? read_rds function has the shortest median time export hence the fastest.