Data Manipulation with Data.Table.

Expanding the open-source ecosystem around data.table in R

Doris Amoakohene Afriyie

What is data.table

- Enhanced data.frame, inherits from and extends data.frame
- Columnar data structure
- Every column must be of the same length but can be of different type
- Concise and consistent syntax , in terms of rows, columns, and groups

Data.table Syntax

DT[i, j, by]

I = on which rows

j= what to do

By = what should be grouped by

Creating a data.table and data.frames

Syntax: The syntax is slightly different between data.table and data.frame.

Efficiency: data.table is known for its efficient and fast performance, especially with large datasets. It uses optimized algorithms and data structures, which can result in faster filtering compared to data.frame for certain operations.

Additional features: data.table provides additional features like the ability to modify data in-place, perform joins, and set keys for faster indexing. These features can be advantageous for complex data manipulation tasks.

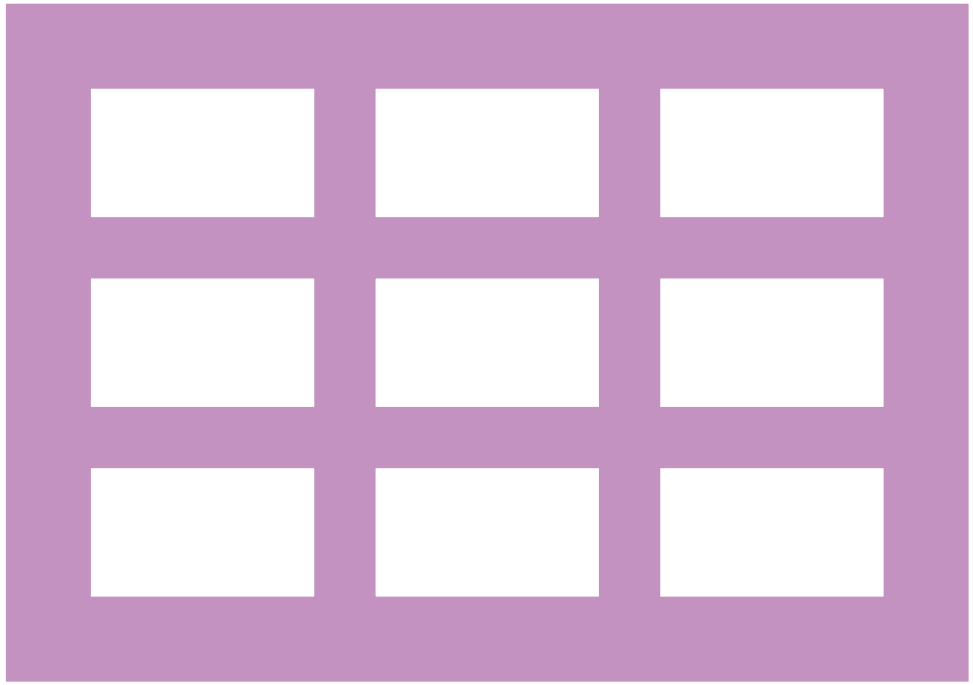
Creating a data.table and data.frames

There are three ways of creating a data.table

- data.table()
- as.data.table()
- Fread()

Creating a data.frames

- data.frame()
- as.data.frames



Creating a data.table using data.table()

- Data.frame

```
x_df <- data.frame(id = 1:2, name = c("a," "b"))
x_df
```

Id	name
1	A
2	B

- Data.table

```
library(data.table)
x_dt <- data.table(id = 1:2, name = c("a, "b"))
x_dt
```

Id	name
1	A
2	B

Creating a data.table using as.data.table

- Making a list a data.table

```
y <- list(id = 1:2, name = c(“a, ”b”))
y
```

- Data.frame

```
y.df = as.data.frame(y)
y.df
```

- Data.table

```
library(data.table)
y.dt <- as.data.table(y)
y.dt
```

Creating a data.table Using Fread

```
# Create a CSV file with some sample data
data <- "Name, Age, Salary
John,25,50000
Alice,30,60000
Bob,35,70000“

# Save the data to a CSV file
```

- fwrite(data, "sample_data.csv")

- # Read the CSV file into a data.table using fread
- dt <- fread("sample_data.csv")
- print(dt)

Function used on data.frame can also be used on data.table

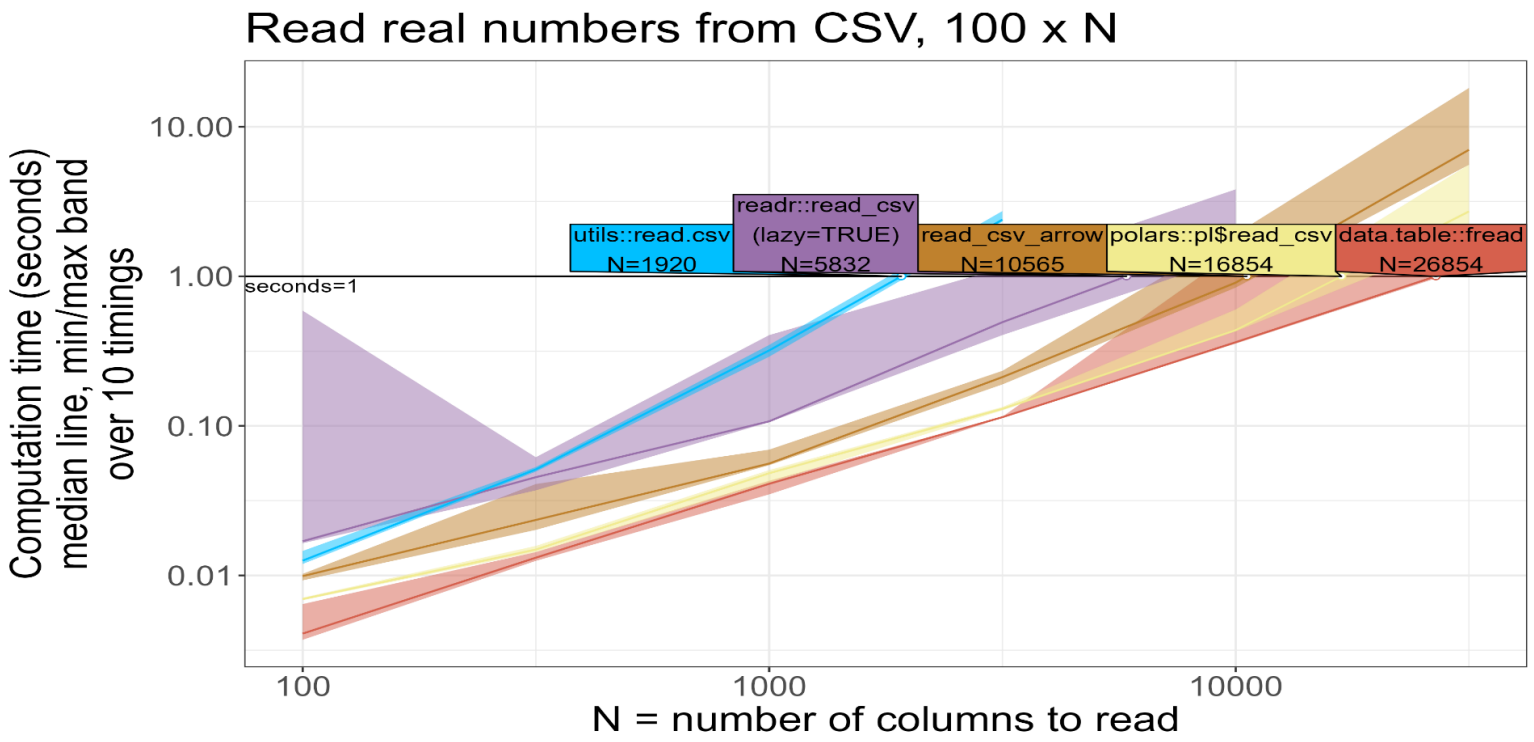
Functions used to query data.frames also work on data.tables

- nrow
- ncol
- dim

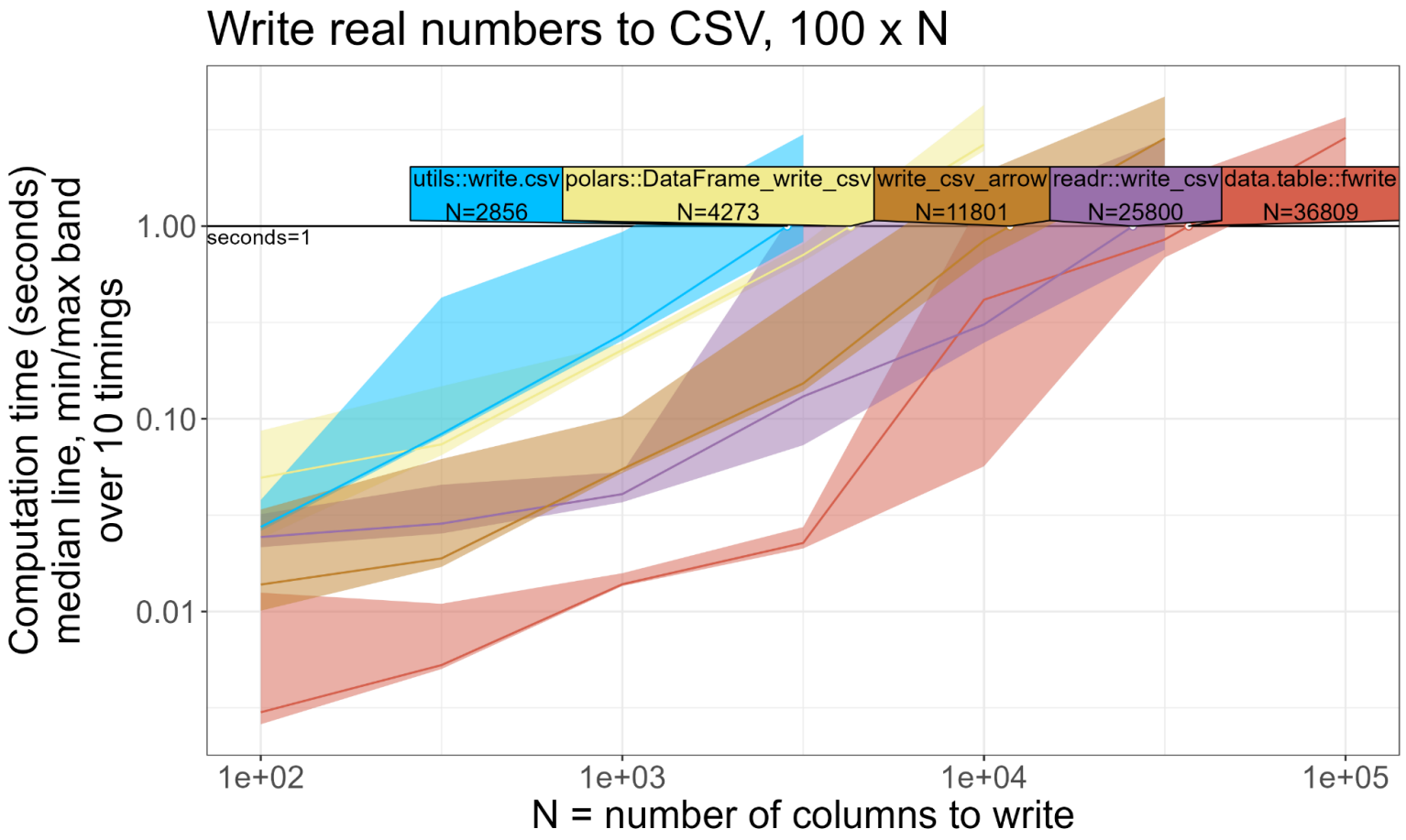
A data table never automatically converts character columns to factors

```
x_df <- data.frame(id = 1:2, name = c("a," "b"))
x_df
class(x_df$name) = "factor"
x_dt <- data.table(id = 1:2, name = c("a," "b"))
x_dt
class(x_dt$name) = "character"
```

Reading CSV Files



Writing CSV Files



Filtering

- Subset by row: You can use the [operator along with the i argument to filter rows based on a condition
- Subset by row and column: You can use the [operator with both i and j arguments to filter rows and select specific columns

	Data.table
how to subset the 3rd and 4th rows from a dataset	dataset[3:4] dataset[3:4,]
Subset everything except first five rows	dataset[-(1:5),] dataset[-(1:5),]

Helpers for Filtering

[] (subset operator): The [] operator is used to subset or filter rows in a data.table based on specified conditions.

Example

```
library(data.table)
# Create a sample data.table
dt <- data.table(x = c(1, 2, 3, 4, 5))
# Filter rows where 'x' is greater than 2
filtered_dt <- dt[x > 2]
# Print the filtered data.table
print(filtered_dt)
```

x
3
4
5

Helpers for Filtering

%in% (element matching operator): This operator is used to filter rows where a column value matches any element in a specified vector.

Example

```
library(data.table)
# Create a sample data.table
dt <- data.table(x = c("A", "B", "C", "D"))
# Filter rows where 'x' matches elements in a vector
filtered_dt <- dt[x %in% c("B", "C")]
# Print the filtered data.table
print(filtered_dt)
```

x
B
C

DT = data.table(Name=c("Mary","George","Martha"), Salary=c(2,3,4)) DT[Name %like% "^Mar"] DT[Name %ilike% "mar"] DT[Name %flike% "Mar"]

Helpers for Filtering

%Like%(pattern matching operator): This function is used to filter rows where a column value matches a specified pattern using wildcard characters.(*,?,.)

Example

library(data.table)

dt <- data.table(x = c("apple", "banana", "orange", "grape"))

 # Filter rows where 'x' starts with 'a' and ends with 'e'

*filtered_dt <- dt[x %like% "^a.*e$"]*

 # Print the filtered data.table

print(filtered_dt)

x
Apple

Helpers for Filtering

%between% (range filtering): This function is used to filter rows where a column value falls within a specified range.

Example

```
library(data.table)
# Create a sample data.table
dt <- data.table(x = c(10, 20, 30, 40))
# Filter rows where 'x' is between 15 and 35 (inclusive)
Filtered_dt<-dt[x %between% c(15,35)]

# Print the filtered data.table
print(filtered_dt)
```

x
20
30

Helpers for Filtering

%chin% operator(fast membership testing): It checks if elements in a vector are present in another vector. It is particularly useful when you want to check if multiple values in one vector are present in another vector efficiently**.**

Example

```
library(data.table)

# Create two vectors

vector1 <- c("apple", "banana", "orange")

vector2 <- c("orange", "grape", "kiwi")

# Check membership using %chin%

membership <- vector1 %chin% vector2

# Print the result

Membership

[1] FALSE FALSE TRUE
```

Symbols

".N" is used to represent the total number of rows in a data.table.

Example

library(data.table)

 # Create a sample data.table

dt <- data.table(x = c("A", "B", "C", "A", "B"))

 # Calculate the frequency of each value in the 'x' column

dt[, .N, by = x]

x	N
A	2
B	2
C	1

Symbols

:= (assignment by reference): This operator is used to create or modify columns in a data.table without making a copy of the entire object.

Example

```
library(data.table)

# Create a sample data.table

dt <- data.table(x = c(1, 2, 3))

# Add a new column 'y' based on 'x'

dt[, y := x * 2]

# Print the modified data.table

print(dt)
```

x	y
1	2
2	4
3	6

Symbols

.SD (Subset of Data): This special symbol is used to refer to the subset of data for each group in a grouped operation.

Example

library(data.table)

 # Create a sample data.table

dt <- data.table(x = c(1, 2, 3), y = c("A", "B", "C"))

 # Group by 'y' and calculate the mean of 'x' for each group

dt[, .(mean_x = mean(x)), by = y]

y	Mean_x
A	1
B	2
C	3

Symbols

setkey() (set keys for fast indexing): This function is used to set keys on one or more columns in a data.table, enabling faster indexing and sorting operations

Example

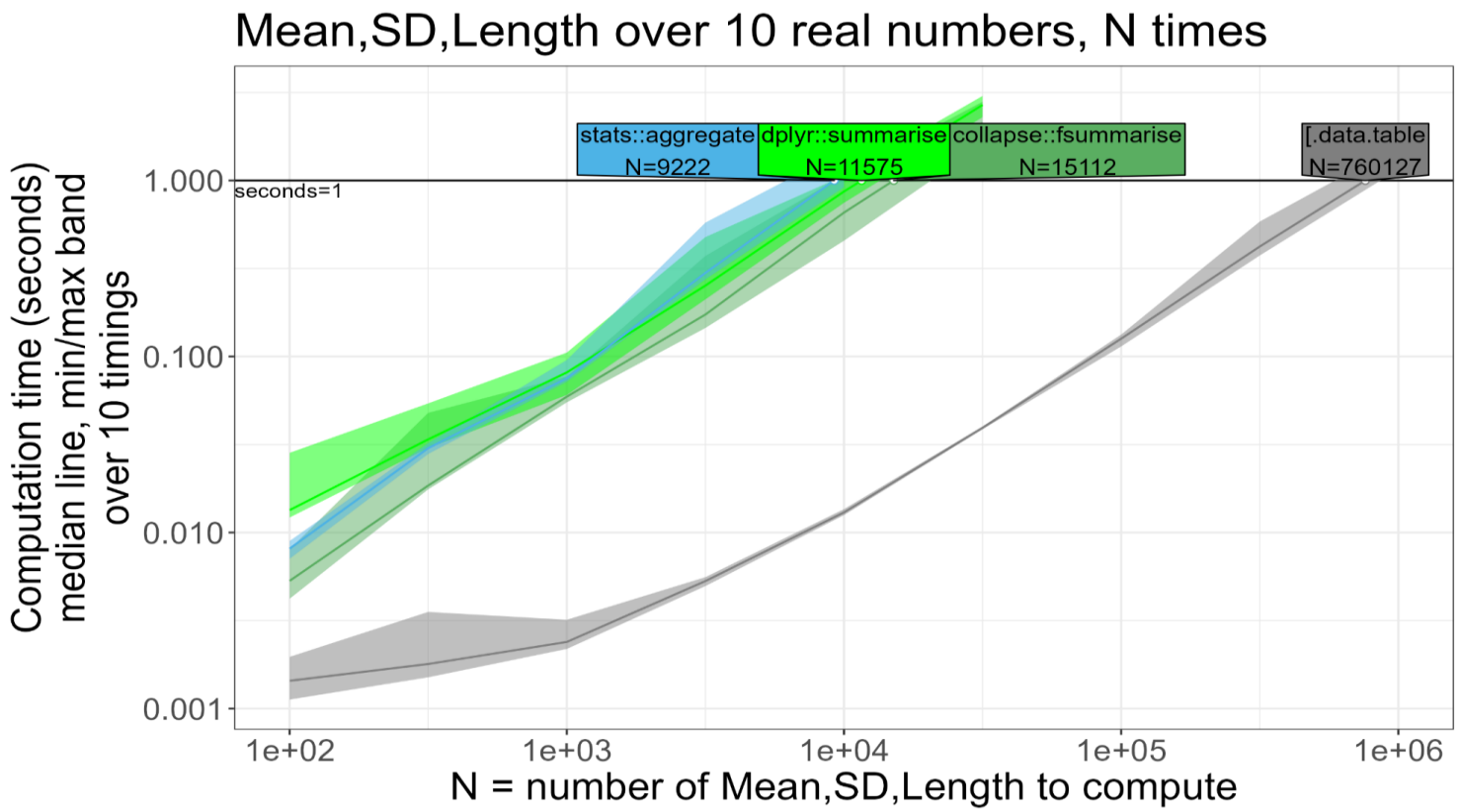
```
library(data.table)
# Create a sample data.table
dt <- data.table(x = c(3, 1, 2), y = c("Z", "X", "Y"))
# Set 'x' as the key column
setkey(dt, x)
# Sort the data.table by the key column
dt <- dt[]
# Print the sorted data.table
print(dt)
```

x	y
1	X
2	Y
3	Z

Aggregating

- The data.table package provides efficient aggregation functions to summarize data.
- You can calculate various summary statistics such as mean, sum, count, and more using functions like sum(), mean(), count(), etc.

Creating an Expanded Data summary



Sorting

- With data.table, you can sort data by one or more columns using the order() function or the setorder() function.
- Sorting can be performed in ascending or descending order.
- Sorting can also be done while creating an index using the setkey() function.

THANK YOU ???