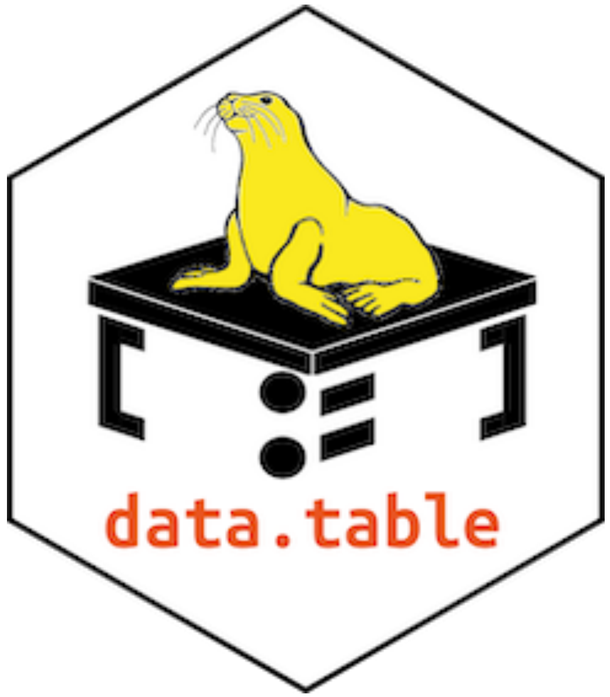
Computational Efficiency of R's `data.table` Package

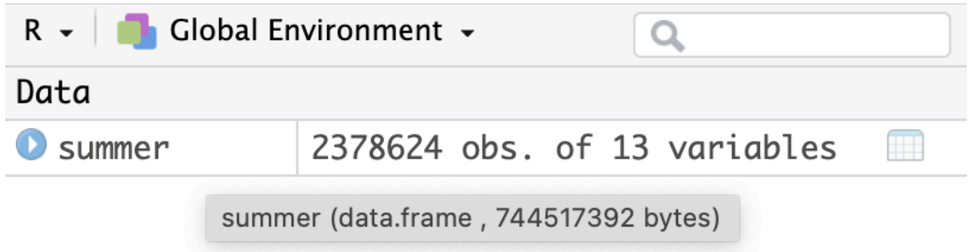
CMU StatBytes: March 12, 2025



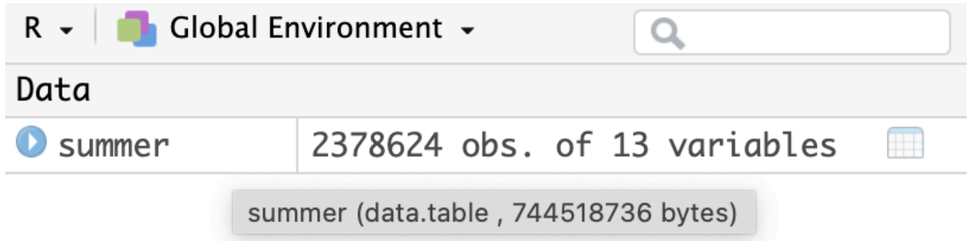
Erin Franke & Sara Colando

What is `data.table`?

- High-performance version of base R's `data.frame`
- Benefits include:
  - Efficiency!!
    - Low level parallelism (multithreading)
  - Concise syntax
  - No dependencies
  - Updates continuously tested against old versions of R (from April 2014)




```
summer <- data.table(summer)
```



What do you mean, computationally efficient?

In reading/writing files

In data manipulation tasks
(e.g. filtering, joining, aggregating)

In memory usage

Presentation Outline

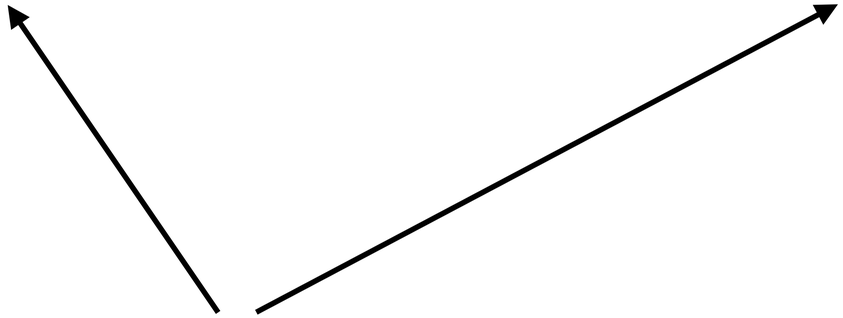
1. Reading/writing files
2. General syntax
3. Data manipulation: keys & indices
4. Pivoting and joining
5. Visualizing/modeling by a grouping variable
6. `dtplyr` & `tidytable`
7. Live demo!

Source Code: https://github.com/efranke22/data.table

`fread()`: Efficiency

```
my_data <- fread("mydata.csv")
```

```
fwrite(my_data, "mydata.csv")
```



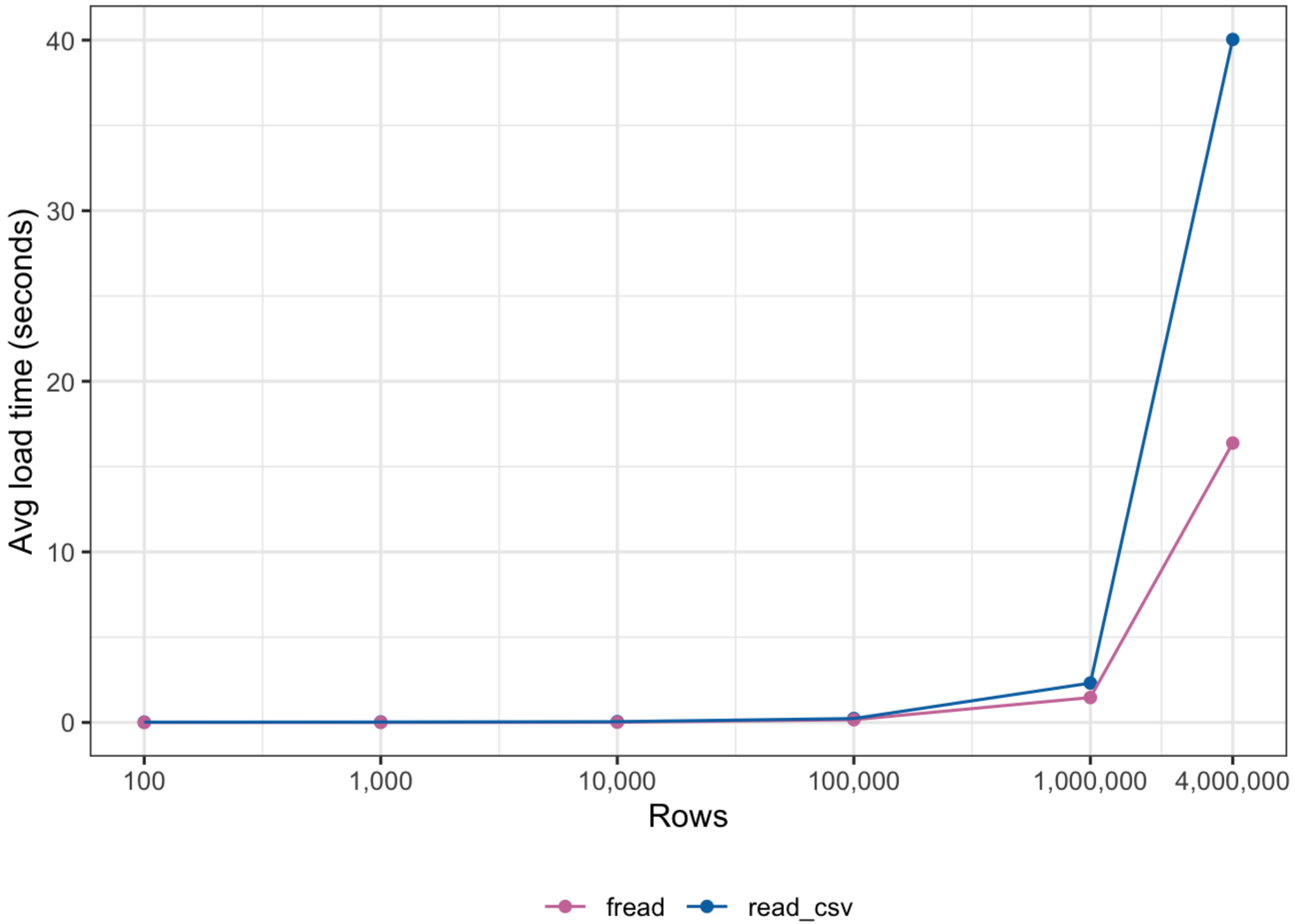
Compatible file types

.csv & .tsv

Other delimited files (semicolon, colon, pipe)

Compressed files with .gz, .bz2 extension

Reading/Writing with `fread()` and `fwrite()`



`fread()`: Why it works and special features

- Efficiency over `read_csv()` comes from:
  - Column type inference process: lazy on-demand memory map
  - Parallel processing
  - Memory efficient data structure — returns a tidytable object

```
fread("mydata.csv", nrows=Inf, skip="_auto_", select=NULL, drop=NULL)
```

`data.table` Syntax

rows (filter/reorder)

grouping (group by)

```
dt[i, j, by] + some more
```

columns (select/mutate)

"Take **dt,** subset/reorder rows using **i**, then calculate **j**, grouped by **by**."

Row-wise data.table object creation

- Convenient and readable for small datasets
  - E.g., useful for toy examples and in teaching
- Simple way to create a new row to bind to an already existing data.table object

```
rowwiseDT(
  name =, age =, pets = ,
  “Erin”, 24, c(“cat”, “hermit crab”),
  “Sara”, 23, NA
  :
  :
)
```

```
data.table(
  name = c(“Erin”, “Sara”, …),
  age = c(24, 23, …),
  pets = list(c(“cat”, “hermit crab”),
  NA, …)
```

Data manipulation efficiency

[8.2 million rows, 30 columns]

Case Number	Date	Year	Block	IUCR	Primary Type
HV106221	05/01/2001 12:00:00 AM	2001	064XX S MAPLEWOOD AVE	1754	OFFENSE INVOLVING CHILDI
JG436438	09/23/2001 09:11:00 PM	2001	053XX W POTOMAC AVE	0486	BATTERY
JG437940	09/25/2001 03:50:00 AM	2001	017XX W 90TH PL	1320	CRIMINAL DAMAGE
JG446200	09/29/2001 11:00:00 PM	2001	082XX S COTTAGE GROVE AVE	0486	BATTERY
JG447387	03/02/2001 12:30:00 AM	2001	036XX W DOUGLAS BLVD	0910	MOTOR VEHICLE THEFT
JG467702	01/01/2001 12:00:00 AM	2001	045XX N CENTRAL PARK AVE	0281	CRIMINAL SEXUAL ASSAULT
JG465686	12/14/2001 12:01:00 AM	2001	043XX S CALIFORNIA AVE	0266	CRIMINAL SEXUAL ASSAULT
JH548973	01/01/2001 12:00:00 AM	2001	051XX W MEDILL AVE	1751	OFFENSE INVOLVING CHILDI
JJ103774	01/01/2001 02:00:00 PM	2001	086XX S PAULINA ST	0820	THEFT
JH548896	01/01/2001 12:00:00 AM	2001	051XX W MEDILL AVE	1751	OFFENSE INVOLVING CHILDI
JG488521	11/02/2001 01:00:00 PM	2001	037XX W 55TH ST	0810	THEFT
JG491345	10/31/2001 06:00:00 PM	2001	043XX S ALBANY AVE	1320	CRIMINAL DAMAGE
JG197641	02/06/2001 12:01:00 AM	2001	007XX E PERSHING RD	1752	OFFENSE INVOLVING CHILDI
JG494727	08/17/2001 03:00:00 PM	2001	025XX N SAYRE AVE	0820	THEFT
JH248855	04/27/2001 12:00:00 AM	2001	046XX S ELLIS AVE	0560	ASSAULT
JG522946	11/29/2001 05:28:00 PM	2001	057XX W SUPERIOR ST	2826	OTHER OFFENSE
JG525402	03/02/2001 12:39:00 AM	2001	013XX W TAYLOR ST	1320	CRIMINAL DAMAGE
JG527147	11/30/2001 08:00:00 AM	2001	064XX N SHERIDAN RD	1195	DECEPTIVE PRACTICE
JH255107	05/06/2001 04:00:00 PM	2001	079XX S EAST END AVE	0460	BATTERY

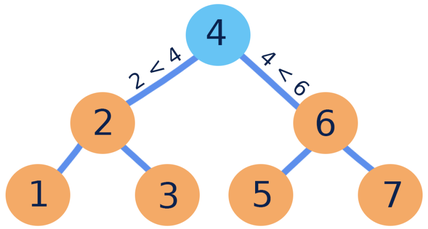
```
data.frame(microbenchmark(
  dplyr_way = crime %>% dplyr::filter(Beat == 511),
  dt_way = crime[Beat == 511], times = 10L))
```

```
# A tibble: 2 × 2
  expr      avg_time
  <fct>        <dbl>
1 dplyr_way    0.147
2 dt_way       0.0640
```

~2.2x faster!

Why is this so efficient?

- **Main reason**: data.table's order function used to identify groups
- **Even faster:** keys!
- Before we call `crime[.(511)]`, we call `setkeyv(crime, "Beat")`
  - This sets a **key** which:
    1. Physically reorders rows of data.table by column(s) provided
    2. Marks specified column(s) as *key* columns
  - This allows for **binary search**!
    - Complexity O(log n) as opposed to O(n) without a key, where `n=nrow(crime)`



In Order Traversal: 1 2 3 4 5 6 7

Keys versus Secondary Indices

- Keys are great!
    - But physically reordering the table could be time consuming depending on number of rows and columns
- Secondary indices are like keys, but *don't* physically reorder the entire data.table in RAM
    - Creates an order vector for column(s) provided, stores this in an attribute *index*
    - Not very time consuming — data.table uses **true radix sorting**
    - Easily reusable!

Implementing Secondary Indices

Set the secondary index: `setindexv(crime, “Year”, “Primary Type”)`

See indexed columns: `indices(crime)`

```
[1] "Year__Primary Type"
```

Implementing indices on the fly (will not be stored as an attribute):

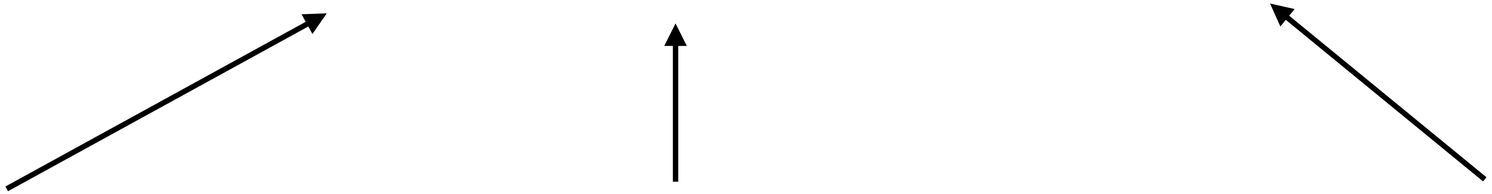
- Existing indices will not be recomputed.

```
crime[.(2023, “AUTOMOBILE”), on = c(“Year”, “Description”)]
```

Pivoting large data.table objects

`dcast`: long-to-wide reshaping with same compuionally efficient processes

```
dcast(data, formula, fun.aggregate = NULL)
```



data.table object

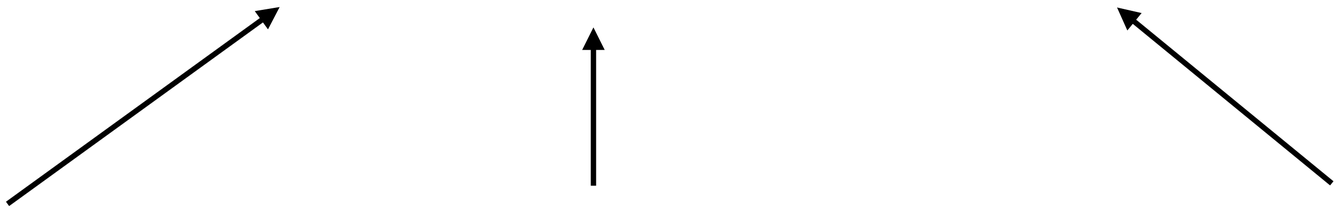
[keys/ids] ~ [new columns]

function(s) for aggregating data pre-pivot

Pivoting large data.table objects

`melt`: wide-to-long reshaping with same computationally efficient processes

```
melt(data, id.vars, measure.vars)
```



data.table object

variables to use as id variables

variables to use as measures corresponding to id variables

Joining operations in `data.table`

primary data.table/list/data.frame

what to do with non-matches between x and i

```
x[i, on,  nomatch = NULL]
```

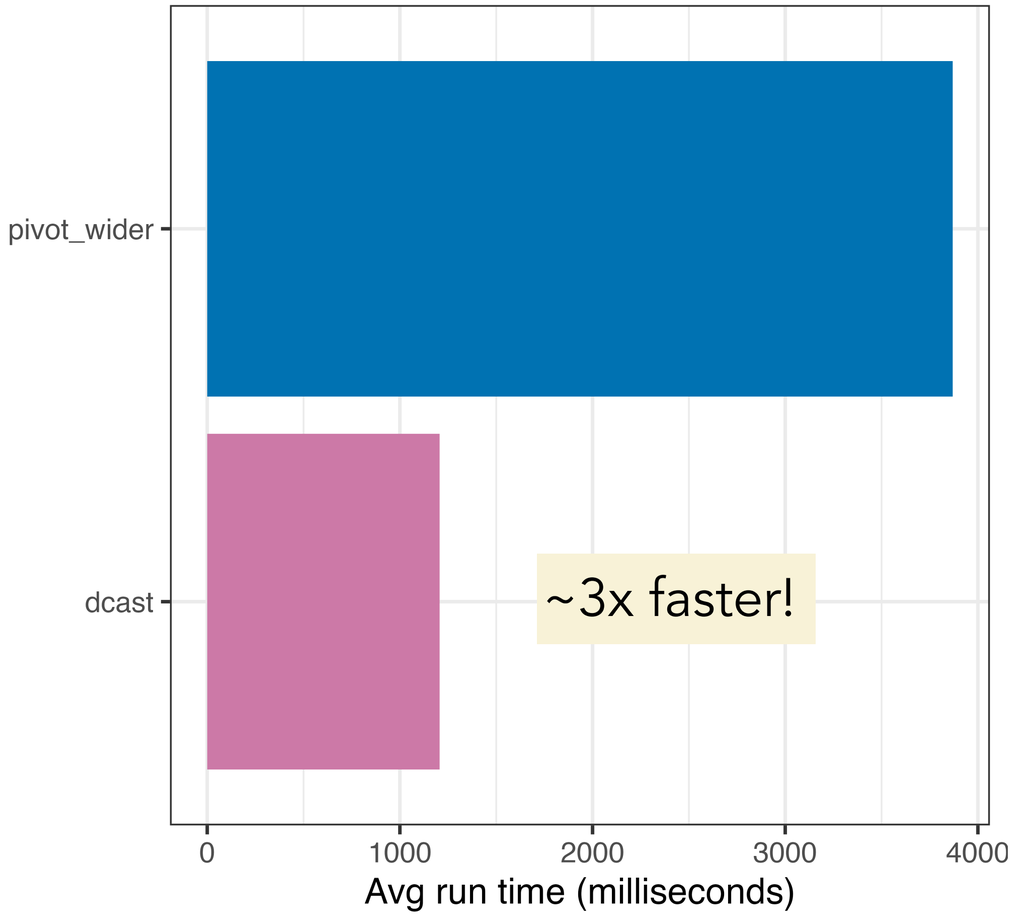
secondary data.table

character vector/list defining match logic

- **Equi Joins:** match rows with common (equal) elements
- **Non-Equi Joins:** match rows based on comparison operators other than equality (e.g., > or <)
- **Overlapping Joins:** match rows based on overlapping ranges between elements
- **Rolling Joins:** match rows based on the nearest value in a sorted column

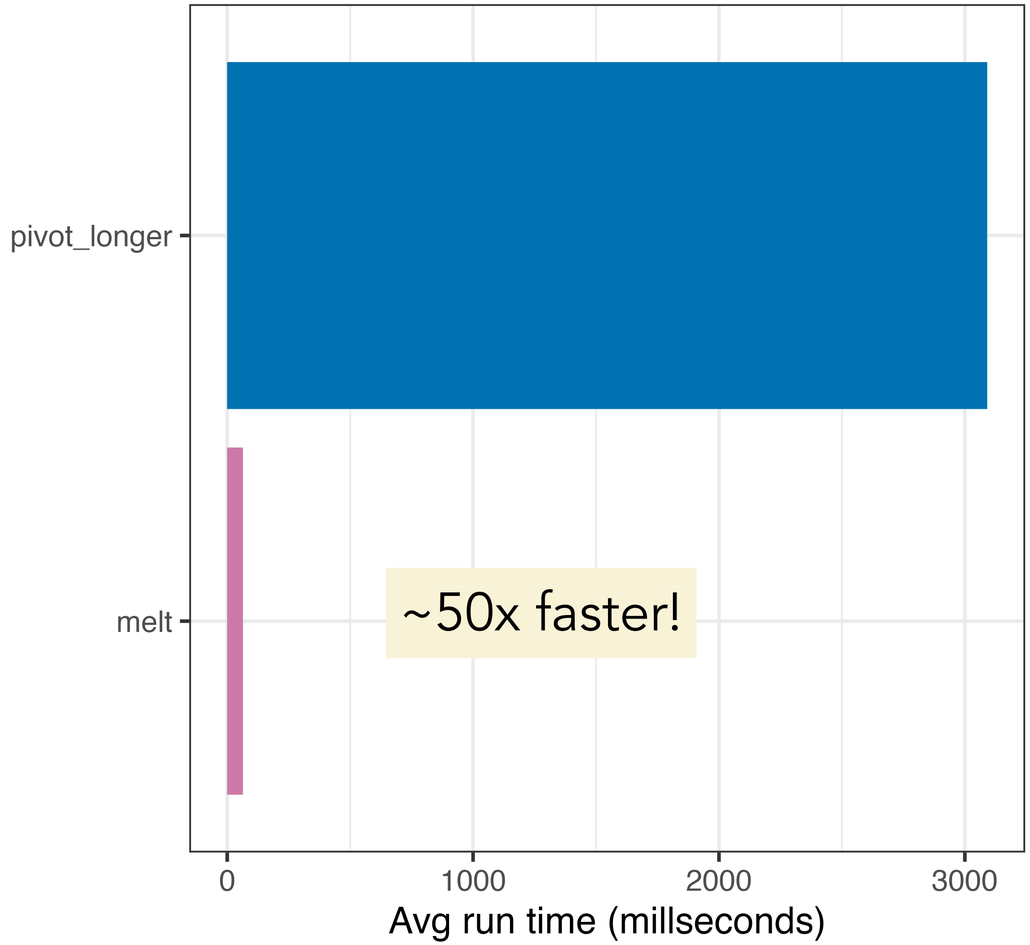
Pivot speed tests (100 repetitions)

`dcast` vs **`pivot_wider`** (tidyr)



[8,092,500 x 3] to [25 x 323,701]

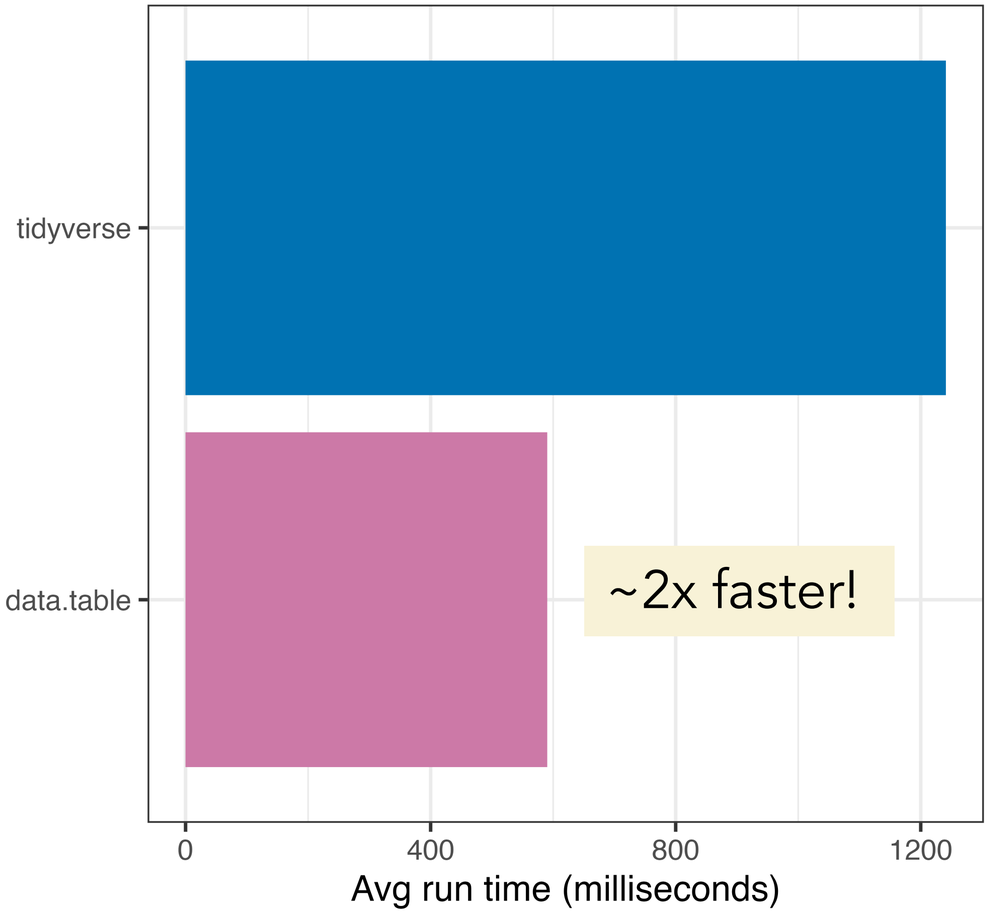
`melt` vs **`pivot_longer`** (tidyr)



[25 x 323,701] to [8,092,500 x 3]

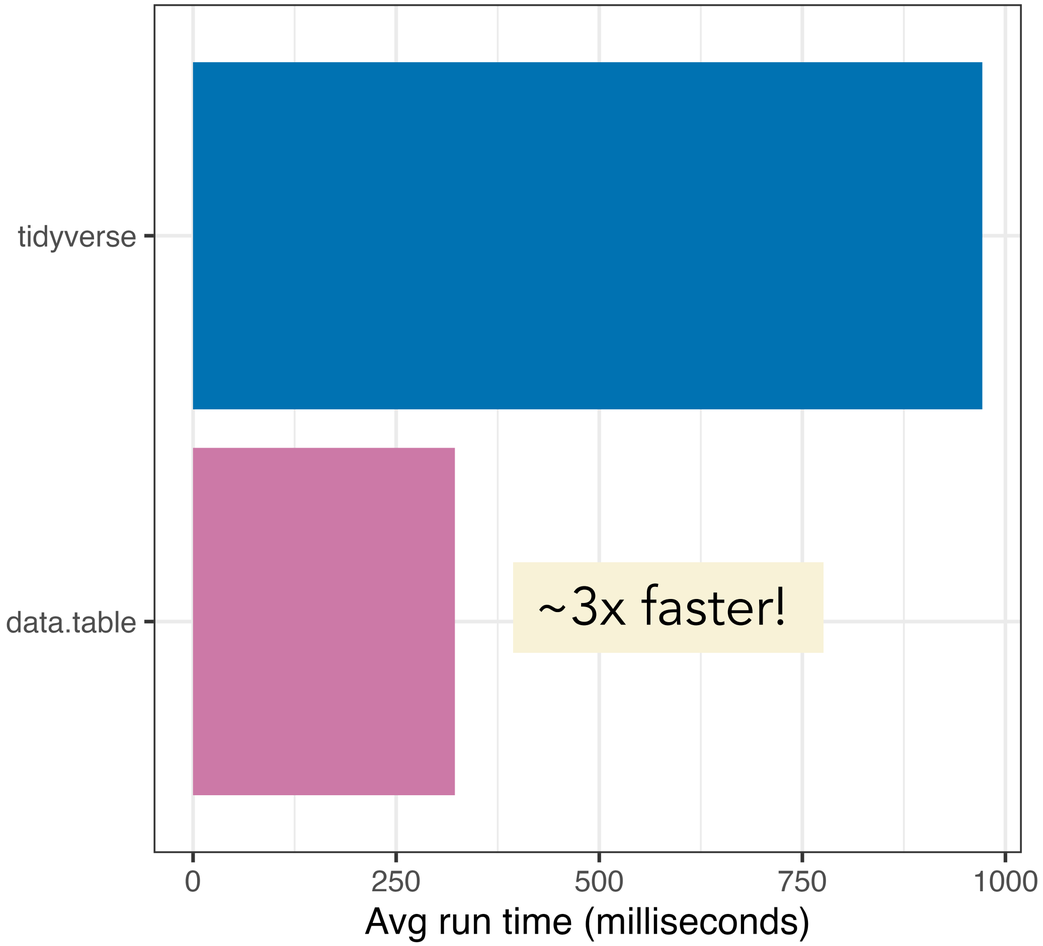
Joins speed tests (100 repetitions)

`data.table` vs **`left_join`** (dplyr)



Left = [8,092,500 x 3] & Right = [323,700 x2]

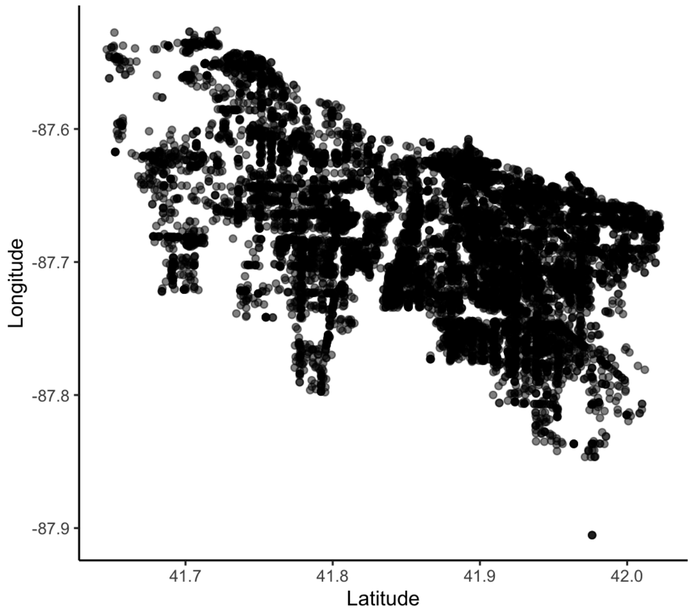
`data.table` vs **`inner_join`** (dplyr)



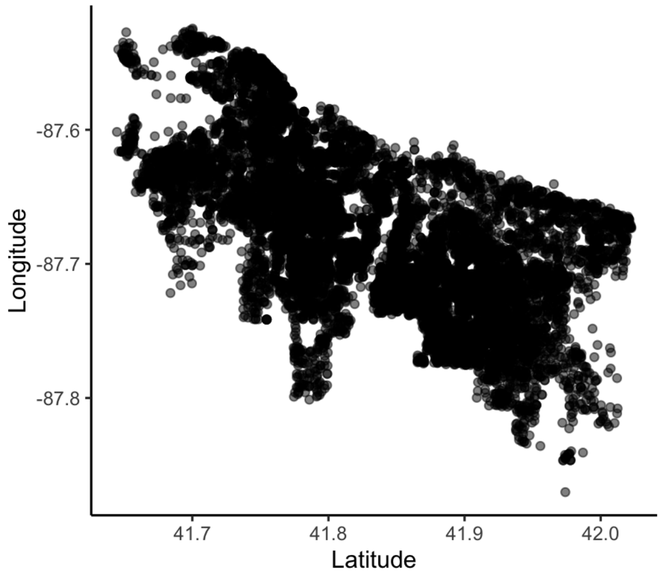
Left = [8,092,500 x 3] & Right = [97,110 x2]

Easy visualization and modeling by a grouping variable

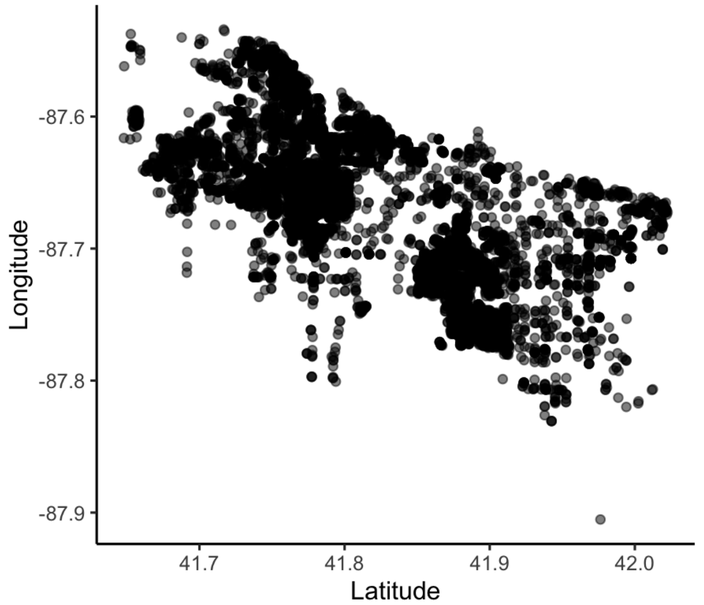
```
crime_sub[, print(
     ggplot(.SD, aes(x = Latitude, y = Longitude)) +
       geom_point(alpha = 0.5) +
       theme_classic()), by = .(`Primary Type`)]
```



Liquor law



Arson



Gambling

dtplyr and tidytable

```
library(dtplyr)

crime2 <- lazy_dt(crime)

crime_sub <- crime2 %>% filter(`Primary Type` %in% c("ARSON", "GAMBLING", "LIQUOR LAW VIOLATION")) %>%
  select(`Primary Type`, Latitude, Longitude) %>% na.omit() %>%
  as_tibble()
```

10 repetitions of microbenchmark:

```
# A tibble: 2 × 2
  expr        avg_time
  <fct>          <dbl>
1 dtplyr_way    0.0270
2 dplyr_way     0.542
```

Relative speed of `dtplyr` and `tidytable`

`dtplyr` and `tidytable` are still usually a little slower than `data.table`

Two Key Reasons for This:

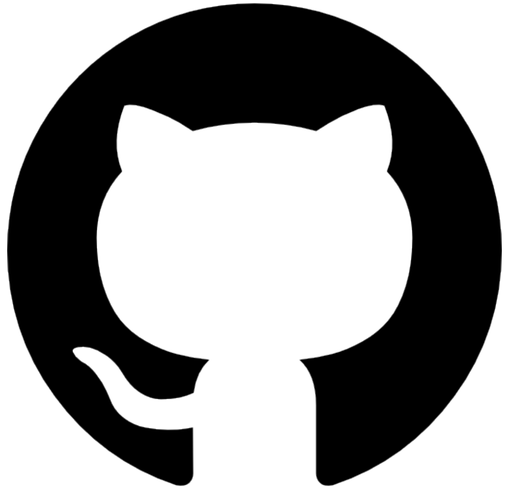
1. `dplyr/tidyr` verbs must translate to `data.table` syntax for fast back-end computation
2. `mutate()` expressions make extra copies to match `dplyr` semantics (doesn't happen in `data.table` package)

Example Speed Tests

Ongoing community `data.table` projects

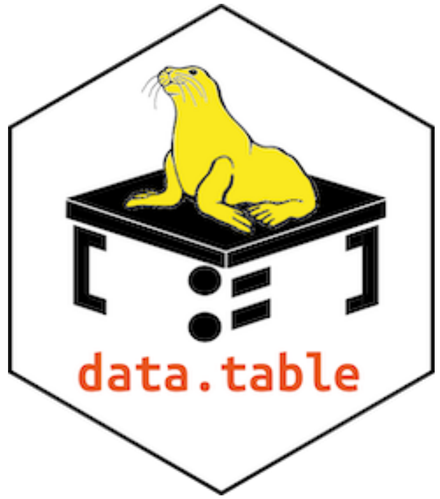
Language Translation Projects

Seal of Approval



Open Source Development

Live demo!



References

- Data.table github page: https://github.com/Rdatatable/data.table
- fread(): https://www.rdocumentation.org/packages/data.table/versions/1.16.4/topics/fread
    - https://github.com/Rdatatable/data.table/wiki/Convenience-features-of-fread
- Keys and Secondary Indices:
    - https://cran.r-project.org/web/packages/data.table/vignettes/datatable-keys-fast-subset.html
    - https://cran.r-project.org/web/packages/data.table/vignettes/datatable-secondary-indices-and-auto-indexing.html
    - https://gist.github.com/arunsrinivasan/dacb9d1cac301de8d9ff
- Visualize by grouping var: https://tysonbarrett.com/assets/JSM_2024/Barrett_JSM2024.pdf
- tidytable & dtplyr:
    - https://markfairbanks.github.io/tidytable/
    - https://dtplyr.tidyverse.org/
    - https://www.reddit.com/r/rprogramming/comments/hakb1l/tidytable_vs_dtplyr/
- Packages: data.table, dtplyr, tidytable, tidyverse, microbenchmark

References (continued)

- https://stackoverflow.com/questions/21435339/data-table-vs-dplyr-can-one-do-something-well-the-other-cant-or-does-poorly
- Syntax slide inspiration:
  - https://tysonbarrett.com/assets/JSM_2024/Barrett_JSM2024.pdf
  - https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html
- Row-wise data.table creation: https://rdatatable.gitlab.io/data.table/reference/rowwiseDT.html
- Pivots: https://cloud.r-project.org/web/packages/data.table/vignettes/datatable-reshape.html
- Joins:
  - https://cran.rstudio.com/web/packages/data.table/vignettes/datatable-joins.html
  - https://medium.com/analytics-vidhya/r-data-table-joins-48f00b46ce29
  - https://www.r-bloggers.com/2016/06/understanding-data-table-rolling-joins/
- Speed Test Inspiration: https://markfairbanks.github.io/tidytable/articles/speed_comparisons.html

Presentation Summary

`data.table` provides a high-performance version of base R's `data.frame` that offers:

- Concise syntax
- Computational efficiency — often faster and less memory-intensive than `tidyverse` equivalents
- Additional features for convenience (rolling joins, visualization with grouping, `rowwiseDT`)
- Powerful integration with Tidyverse-like syntax (`dtplyr` and `tidytable`)

Source Code: https://github.com/efranke22/data.table