

2022-12-17 統計数理研究所共同利用研究集会「データ解析環境Rの整備と利用」

Rによる大規模データの処理

瓜生真也（徳島大学デザイン型AI教育研究センター）

  u ribo

 https://github.com/u ribo/talk_221217_rjusers

一押しポイント

読み込み・集計加工処理が
速い

ファイル・メモリサイズが
小さい

データの操作・管理が
効率的

大規模データセットの効率的な処理と転送のために設計

多言語ツールボックスとして、さまざまなプログラミング言語で実装

C++, R, Python, Rust, Julia, DuckDB など

インメモリのカラム型データを取り扱う（シリアル化不要）

parquetに対応。パーティショニングによる管理

{**arrow**} パッケージ csvファイルからのdplyrやdata.tableでの処理と比較して、**かなり高速**
dplyrバックエンドを提供。パイプ演算子による処理を適用可能

lubridate、rlang、stringr、tidyselectなどの関数にも対応
未対応な関数を{duckdb}パッケージがカバー

Apache Arrow

{**arrow**} パッケージ

自己紹介

うりゅう しんや

瓜生 真也

2021年10月～徳島

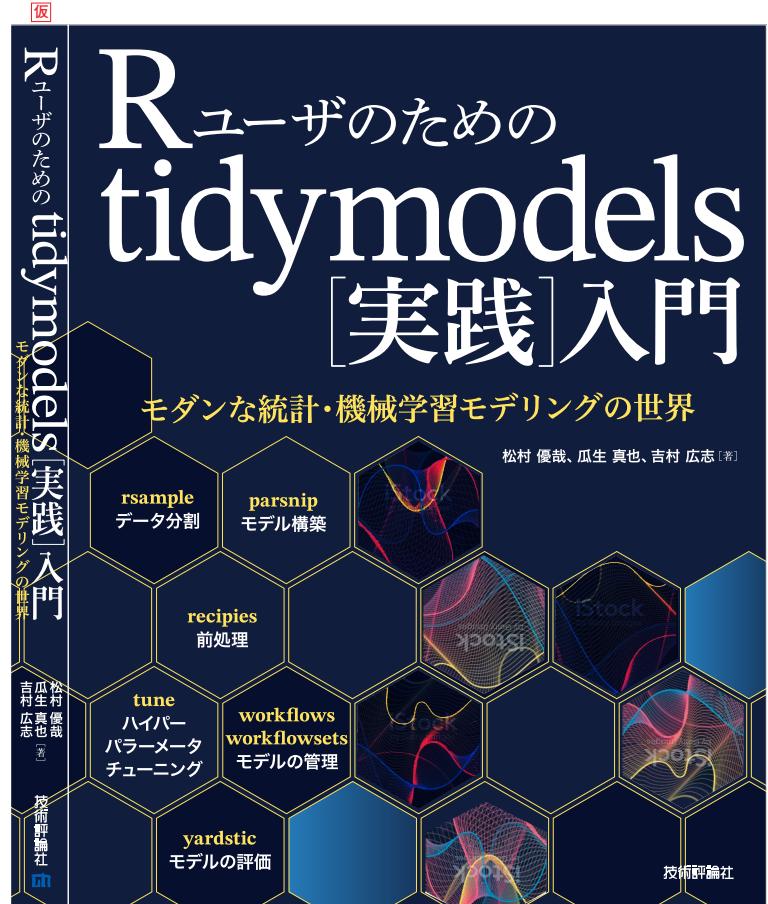
地理空間データ解析

統計、データサイエンス

#データ可視化 #R言語



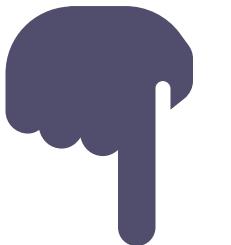
来年1月出版！



大きいデータの処理、どうしていますか？

ここでは「大きいデータ」を「現実的に扱えそうな物」とします

1. データ基盤（DB、Hadoop、Amazon Athena、BigQuery）等を利用しない
单一のテキスト・バイナリファイル
2. awkなどで前処理をしない程度の大きさ
一つのファイルが最大で数GB、全体で100GB以内程度



ローカル・クラウドで処理… **メモリをたくさん積んだコンピューターで処理させる**

なるべく安価・低性能の環境で済ませたい

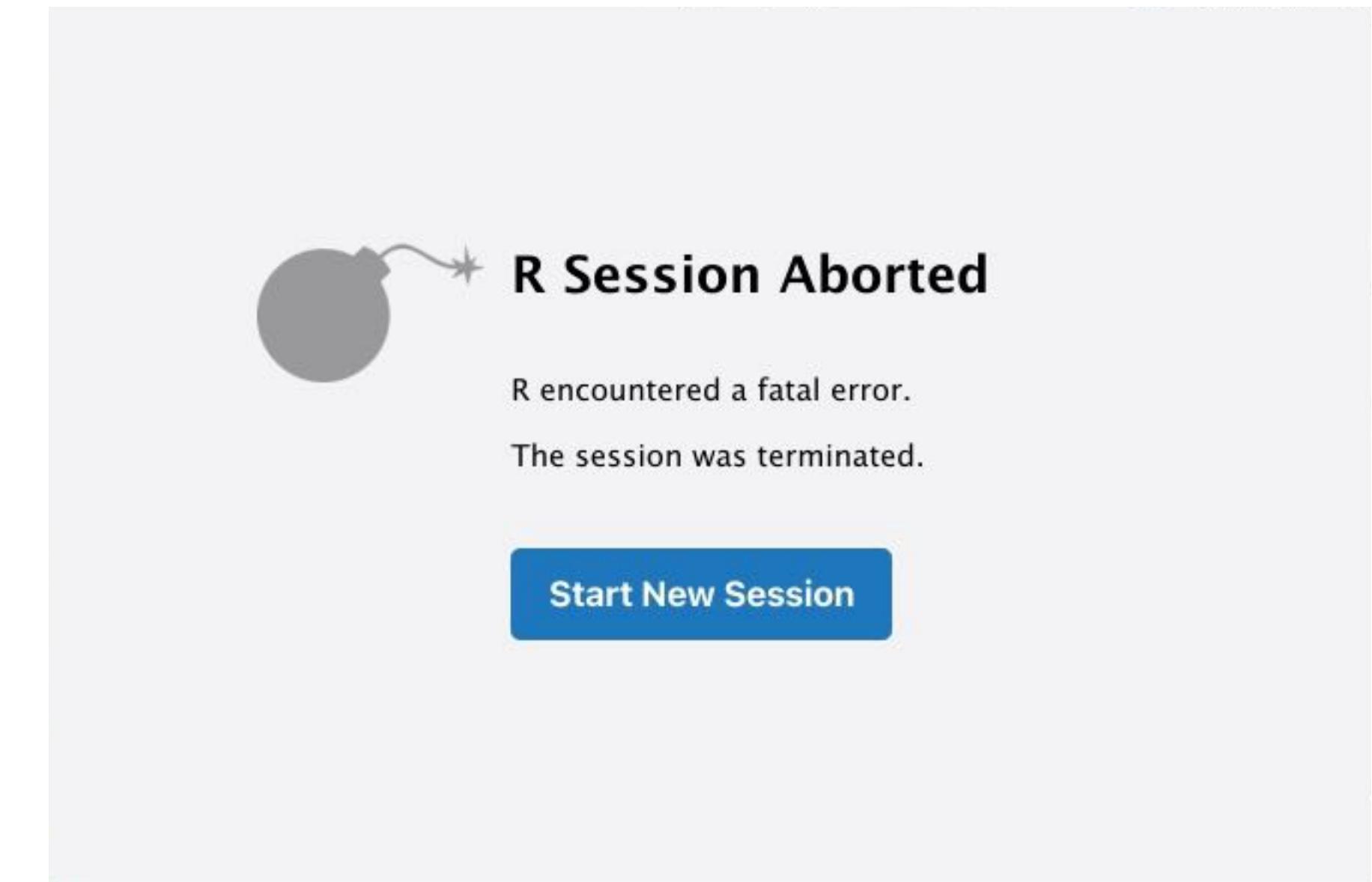
並列化

大きいデータの処理、どうしていきますか？

R、RStudioあるある

Error: vector memory exhausted (limit reached?)

処理の過程でメモリを使い切る



RStudioが爆発する

Apache Arrow

Apache Arrow

1. 複数言語対応 のデータ分析ツールのプラットフォーム

共通の実装を言語を超えて共有できる

データ交換時の性能改善による高速な処理を実現

2. インメモリでの処理を実現

メモリ内のデータを効率的に構造化して処理（シリアル化不要、高速）

列指向のデータ（カラム型データ）を扱うことに特化

*.parquet

Apache Arrow形式… *.arrow

カラム型データ：データ分析に特化

列指向

列単位でデータを保持・取り出し

→指定された列のみ読み込み

大量の行に対する集約処理が効率的

食肉類	レッサーパンダ	63.5	6
霊長類	チンパンジー	85.0	60
霊長類	マントヒヒ	80.0	20
食肉類	ライオン	250.0	225
鳥類	フンボルトペンギン	69.0	6

行指向 CSVやMySQLなどのデータベース

行単位でデータを保持・取り出し

特定列の値への操作

→全部の行・列へのアクセスを伴う

食肉類	レッサーパンダ	63.5	6
霊長類	チンパンジー	85.0	60
霊長類	マントヒヒ	80.0	20
食肉類	ライオン	250.0	225
鳥類	フンボルトペンギン	69.0	6



{arrow}パッケージ

{arrow}パッケージの特徴

1. Arrow C++ライブラリへのインターフェースを提供

{dplyr}をはじめとして{lubridate}、{stringr}、{tidyselect}などの関数にも対応

→**{dplyr}によるパイプ処理やRの関数で大規模データの操作が実現**

2. (メモリに乗り切らない規模の) 複数のファイルを読み込み、処理

Arrow Arrowが扱うarrow, parquet形式のデータ

クラウド環境を含めた、**複数ファイルへの高速な読み書き**

3. {duckdb}パッケージ、Pythonとの連携

データのコピーをする過程でも **インメモリで処理**

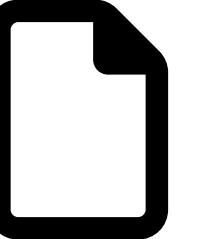
{arrow}で対応しない処理を実行するため、別のパッケージと言語でデータを共有

{arrow}を使った処理の流れ

1. データの読み込み

```
d <- read_parquet("data/zoo.parquet.parquet",  
                      as_data_frame = FALSE)
```

parquet csv
feather tsv txt
json



2. {dplyr}などの関数による処理

```
result <- d |>  
  select(name, taxon, body_length_cm) |>  
  filter(taxon %in% c("靈長類", "齧齒類", "鳥類")) |>  
  group_by(taxon) |>  
  summarise(body_length_mean = mean(body_length_cm))
```



3. データフレームとして返り値を得る

```
result |>  
  collect()
```

データをメモリ上に読み込む
遅延評価

4. 集約されたデータを分析・可視化

{duckdb}パッケージ

動作するコンピューターのメモリ上で動作する軽量データベースシステム

オランダの国立研究所のメンバーが中心になって開発

C++で書かれたAPIを多言語にラップして提供

{duckdb}はそのうちのR言語向けラッパーパッケージ

{DBI}によってデータフレームを操作

{arrow}側で{duckdb}との連携のために `to_duckdb()` を提供

→{arrow}側で対応しない一部の関数を実行できる

{arrow}が対応しない関数？

前述のとおり、{arrow}ではArrow C++へのインターフェイスを提供

`list_compute_functions()`

{dplyr}の関数も利用可能

`?acero` (Arrow query engine)

→すべてのRの関数が{arrow}で実行可能なわけではない

data.frameに変換して処理 or エラー

```
d <-  
  read_parquet(here::here("data/typeB/36_tokushima/year=2022/month=10/part-0.parquet"),  
               as_data_frame = FALSE)
```

次のスライドで例を示すためにarrowオブジェクトを用意

1. data.frameに変換して処理

大きいデータのままだと処理に時間がかかるかも

```
d |>  
  select(datetime) |>  
  mutate(is_jholiday = zipangu::is_jholiday(datetime))  
#> Warning: Expression  
#> zipangu::is_jholiday(datetime) not supported in Arrow;  
#> pulling data into R  
#>   datetime is_jholiday  
#> 1 2022-10-01 FALSE  
#> 2 2022-10-01 FALSE  
#> 3 2022-10-01 FALSE  
#> 4 2022-10-01 FALSE  
#> 5 2022-10-01 FALSE  
#> 6 2022-10-01 FALSE
```

2. エラーで処理を停止

```
d |>  
  group_by(location_no) |>  
  slice_min(order_by = traffic, n = 1) |>  
  collect()  
#> Error: Slicing grouped data not supported in Arrow
```

→{duckdb}への変換で対応

```
d |>  
  group_by(location_no) |>  
  to_duckdb() |>  
  slice_min(order_by = traffic, n = 1) |>  
  collect()  
#> # A tibble: 25,237 × 10  
#> # Groups:   location_no [274]  
#>   datetime           source_...¹  locat...²  locat...³  meshc...⁴  link_...⁵  link_no  traffic  
#>   <dttm>             <chr>      <int>    <chr>     <chr>      <int>    <int>    <int>  
#> 1 2022-10-29 17:50:00 3028          54 西ノ丸... 513404        2       662      2  
#> 2 2022-10-29 19:50:00 3028          54 西ノ丸... 513404        2       662      2
```

{arrow}パッケージでの Parquetファイルの扱い

Apache Parquet

大規模データを扱うために広く使われている標準的な形式を提供

列指向データを圧縮されたバイナリ形式で保存、CSVなどの非圧縮テキスト形式と比較してファイルサイズが小さくなる

効率的な列データ表現の利点を利用できるように設計

元はHadoopエコシステムの中で機能するように開発

parquetの読み込み: 単一ファイル

データフレームとして読み込み（デフォルト）

```
read_parquet("data/typeB/36_tokushima/year=2022/month=10/part-0.parquet")
```

ArrowTableとして読み込み→インメモリで処理可能

```
read_parquet("data/typeB/36_tokushima/year=2022/month=10/part-0.parquet",
             as_data_frame = FALSE,
             # tidyselectでの列選択
             col_select = c(datetime, ends_with("no")))
```

AWS S3、Google Cloud Storage (GSC)からの読み込みもOK

```
read_parquet("s3://[access_key:secret_key@]bucket/path[?region]")
read_parquet("gs://anonymous@bucket/path")
```

parquurtの読み込み：複数ファイル

共通の列配列をもつ複数のparquetファイルを読み込む（この段階ではスキーマのみ）

```
open_dataset(here::here("data/typeB/13_tokyo/year=2021/"))
#> FileSystemDataset with 12 Parquet files
#> datetime: timestamp[us, tz=Asia/Tokyo]      月毎に分かれた12個のファイル
#> source_code: string
#> location_no: int32
#> location_name: string
#> meshcode10km: string
#> link_type: int32
#> link_no: int32
#> traffic: int32
#> to_link_end_10m: string
#> link_ver: int32
#> month: int32
#>
#> See $metadata for additional Schema metadata
```

`open_dataset(..., format = "csv")` csvでもOK

パーティショニング

一つの大きなデータセットを複数の細かなファイルに分割する戦略

必要なデータに素早くアクセス、処理の負担が軽減

分割を的確に行うことが重要 例えば… 年、月（12）、都道府県（47）

```
d |>  
  write_dataset(path = "data",  
                 partitioning = c("year", "month"))
```

データセットを年、月に分けて保存

Hive形式… **key=value** data/year=2021/month=1/part-0.parquet

month=2/part-0.parquet

month=3/part-0.parquet

...

data/year=2022/month=1/part-0.parquet

month=2/part-0.parquet

month=3/part-0.parquet

...

スキーマ (型の指定)

列指向のデータ形式… 列ごとに決まった型（まとまった配置）がある

豊富なデータタイプで列配列の値を定義する

64ビット整数 (int64) と32ビット整数(int32)は区別される

```
open_dataset("data",
             schema = schema(
                 datetime = timestamp(unit = "ms",
                                      timezone = "Asia/Tokyo"),
                 source_code = utf8(),
                 location_no = int32(),
                 ...))
#> FileSystemDataset with 12 Parquet files
#> datetime: timestamp[ms, tz=Asia/Tokyo]
#> source_code: string
#> location_no: int32
#> location_name: string
#> ...
```

{arrow}パッケージによる 大規模データの処理

一般道路の断面交通量情報データ

<https://www.jartic.or.jp/>

公益財団法人日本道路交通情報センターがオープンデータとして公開

クリエイティブコモンズ 表示4.0 国際 (CC BY 4.0)

各都道府県警察が車両感知器などの計測機器で収集した断面交通量に関する情報を警察庁においてとりまとめたもの

一般道路の「断面交通量情報」(2022年10月分)

更新日 2022年12月01日

※全国すべての2次メッシュを
網羅するわけではない

各都道府県警察が車両感知器などの計測機器で収集した断面交通量に関する情報です。
リンク番号の詳細及び最新情報については、公益財団法人日本交通管理技術協会のホームページ
(<https://www.tmt.or.jp/research/index9.html>)に
掲載しています。
情報は毎月月初の更新を基本としておりますが、作業状況により更新が遅れる場合がありますのでご了承ください。
※一般道路の「断面交通量情報」の説明書 [\(必ずお読みください。\)](#)



一般道路の断面交通量情報データ

<https://github.com/uribo/jarticr>

zipファイル展開後のcsvをdata.tableとして読み込む関数を用意

```
remotes::install_github("uribo/jarticr")
dplyr::glimpse(
  jarticr::read_jartic_traffic("data-raw/typeB_tokushima_2022_10/徳島県警_202210.csv")
)
#> Rows: 2,384,773
#> Columns: 10
#> $ datetime          <dttm> 2022-10-01, 2022-10-01, 2022-10-01, 2022-10-01, 2022-...
#> $ source_code        <chr> "3028", "3028", "3028", "3028", "3028", "3028", "3028"...
#> $ location_no       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ...
#> $ location_name     <chr> "徳島本町→県庁前", "県庁前→徳島本町", "新助任橋北詰→徳...
#> $ meshcode10km      <chr> "513404", "513404", "513404", "513404", "513404", "513...
#> $ link_type         <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
#> $ link_no           <int> 710, 705, 681, 676, 679, 678, 11, 375, 6, 5, 6, 5, 842...
#> $ traffic           <int> 21, 24, 25, 27, 25, 28, 30, 16, 42, 42, 27, 25, 16, 26...
#> $ to_link_end_10m   <chr> "6", "6", "11", "27", "21", "29", "26", "8", "7", "92"...
#> $ link_ver          <int> 202100, 202100, 202100, 202100, 202100, 202100, 202100, 202100...
```

検証: データ

サイズの異なるデータを4種用意

地域	期間	データサイズ		ファイルサイズ	
		行数	列数	csv	parquet
Small	東京都 1ヶ月間	1920万9389	10	1.59GB	102MB
Medium	東京都 1年間	2億2801万2986	10	19.3GB	1.17GB
Large	全国 1年間	39億2376万7686	10	260.3GB*	13.5GB
Huge	全国 4年間	156億7727万6121	10	643.2GB*	54.1GB

*Large、Hugeのcsvファイルサイズはzip圧縮時の大ささ
(展開後はさらに大きい)

検証: 環境

Apple M1 Mac

メモリ: 64GB (現行のMBPでは最大)

R 4.2.1 いずれのパッケージもCRAN最新版

```
library(arrow) # 10.0.1  
  
library(readr) # 2.1.3  
  
library(data.table) # 1.14.6  
  
library(dplyr) # 1.0.10  
  
library(dtplyr) # 1.2.2  
  
library(duckdb) # 0.6.1
```



検証: 3つの処理を比較

1. 読み込み csvファイルの読み込み速度

`arrow::read_csv_arrow()`

`readr::read_csv()`

`data.table::fread()`

2. グループ化と集計

{arrow}

{duckdb}… arrowオブジェクトから変換

{dplyr}

{data.table}

{dtplir} … data.tableオブジェクトをdplyrの関数で処理
バックエンドで{data.table}が機能

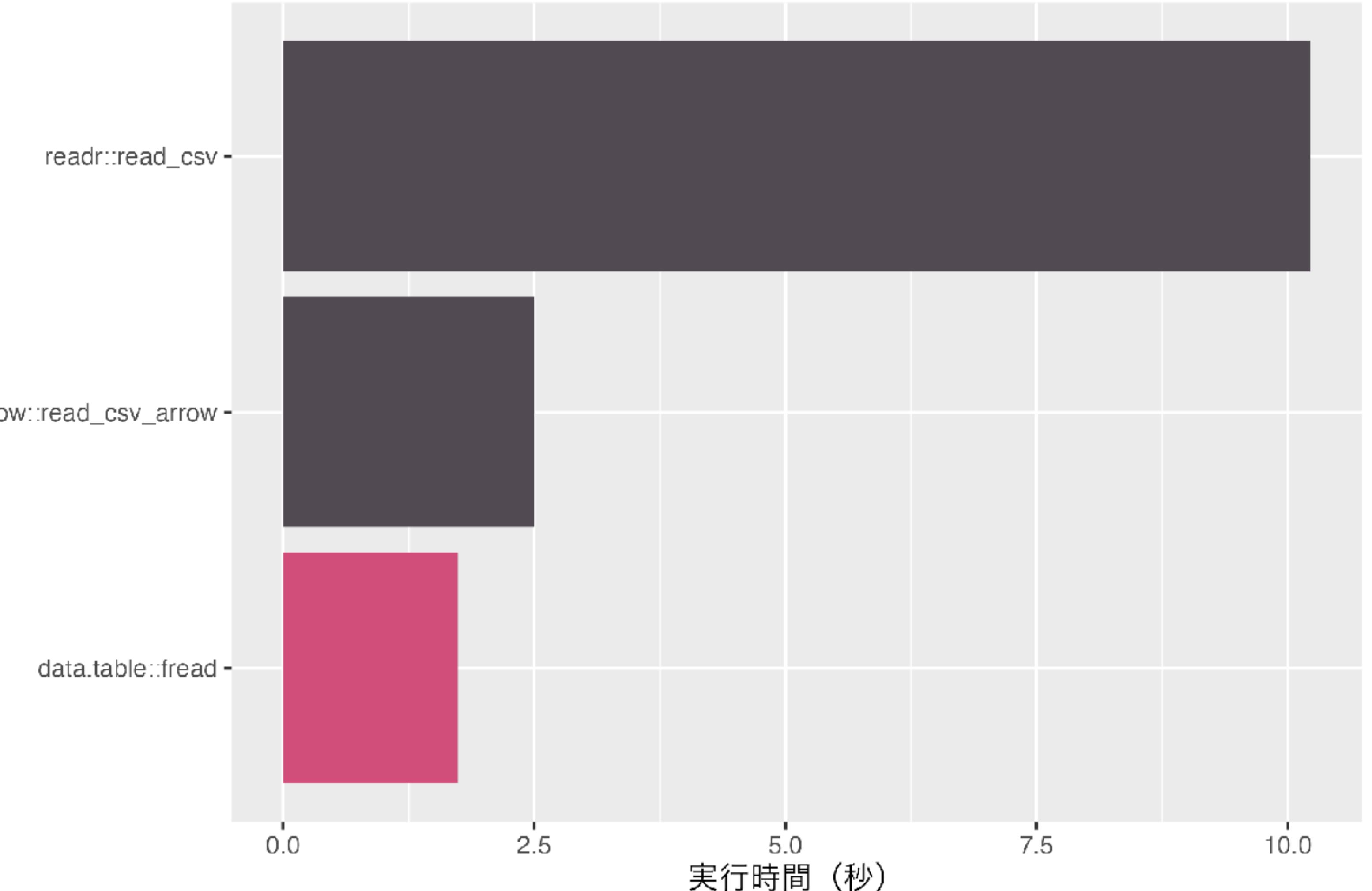
3. 結合

結果1. ファイルの読み込み

Small

⭐ data.table

タスク: csvファイルの読み込み
データサイズ: Small (1920万9389件)

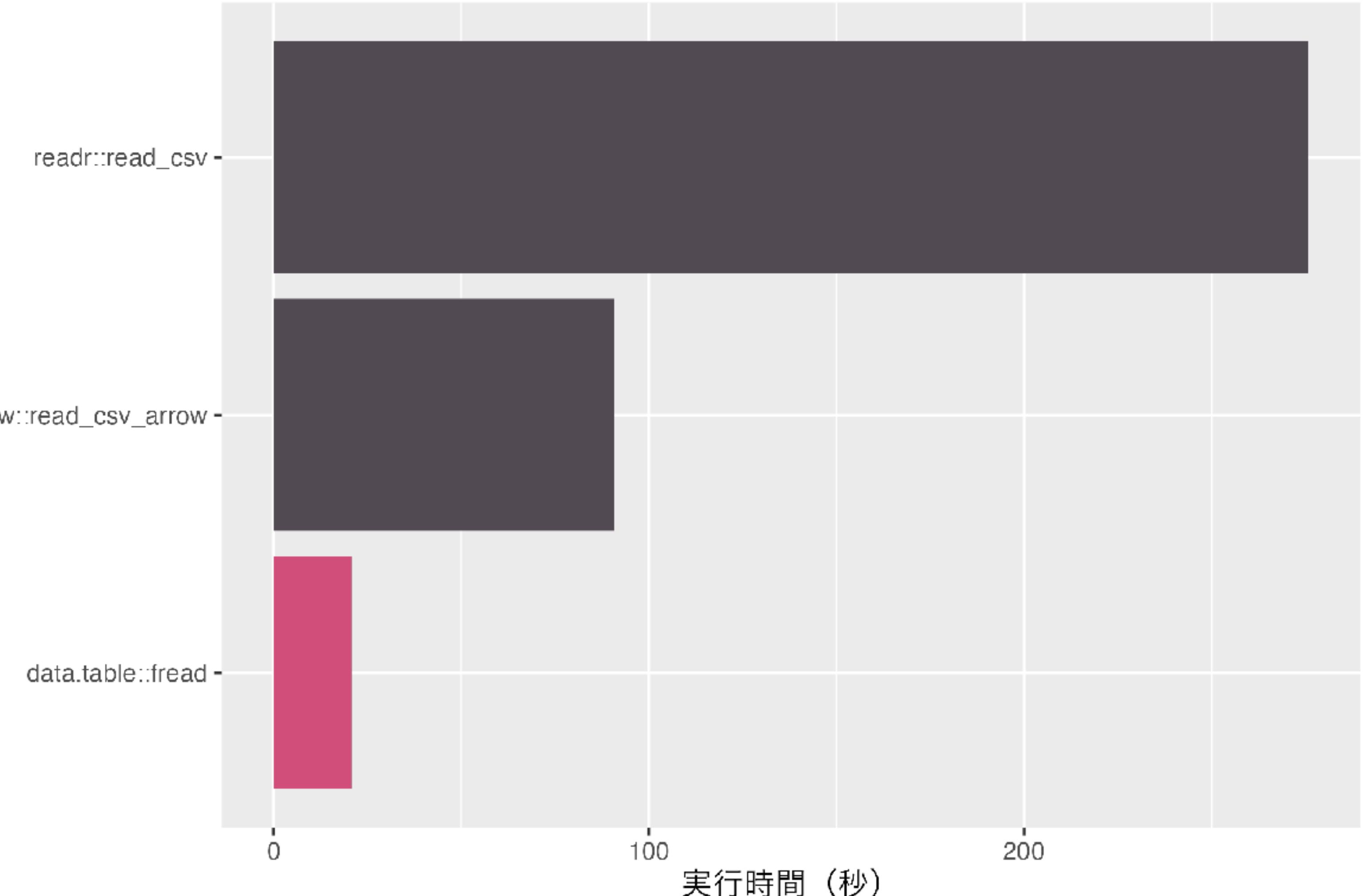


結果1. ファイルの読み込み

Medium

⭐ data.table

タスク: csvファイルの読み込み
データサイズ: Medium (2億2801万2986件)



結果1. ファイルの読み込み

Large, Hugeのcsv用意できず

同規模のparqurtファイルを `arrow::open_dataset()` で読み込む… OK

処理の内容によっては結果を得られる状態

Large

```
# 全国1年分(2021年)
traffic_large <-
  open_dataset(here::here("data/typeB/"),
               schema = jartic_typeB_schema) |>
  filter(year == 2021)
dim(traffic_large)
#> [1] 3923767686           12
lobstr::obj_size(traffic_large)
#> 487.10 kB
```

Huge

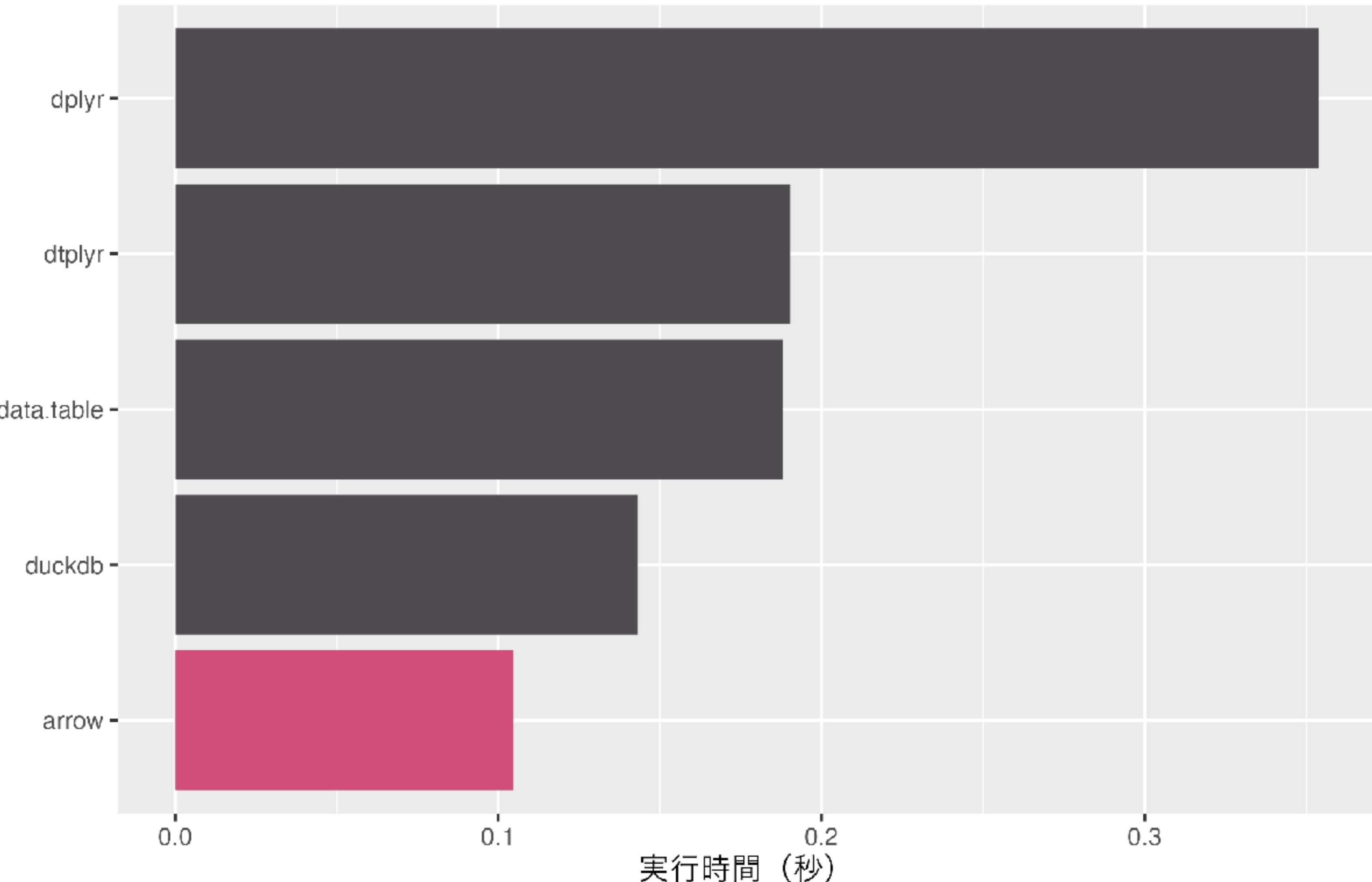
```
# 全国4年分
traffic_huge <-
  open_dataset(here::here("data/typeB/"),
               schema = jartic_typeB_schema)
dim(traffic_huge)
#> [1] 15677276121          12
lobstr::obj_size(traffic_huge)
#> 261.59 kB
```

結果2. グループ化と集計

Small



タスク: 計測地点番号と10kmメッシュごとに断面交通量を合計
データサイズ: Small (1920万9389件)

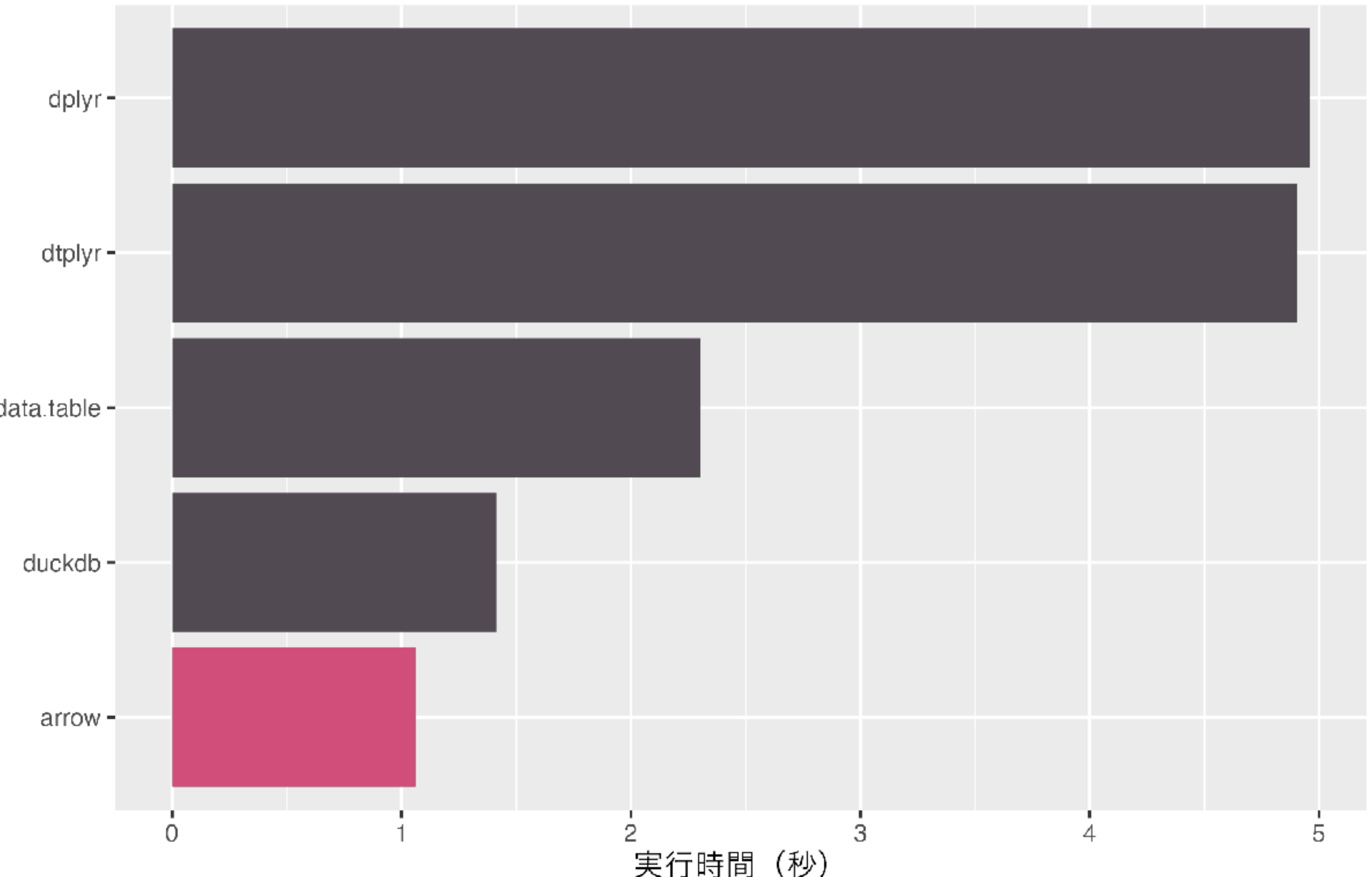


結果2. グループ化と集計

Medium



タスク: 計測地点番号と10kmメッシュごとに断面交通量を合計
データサイズ: Medium (2億2801万2986件)

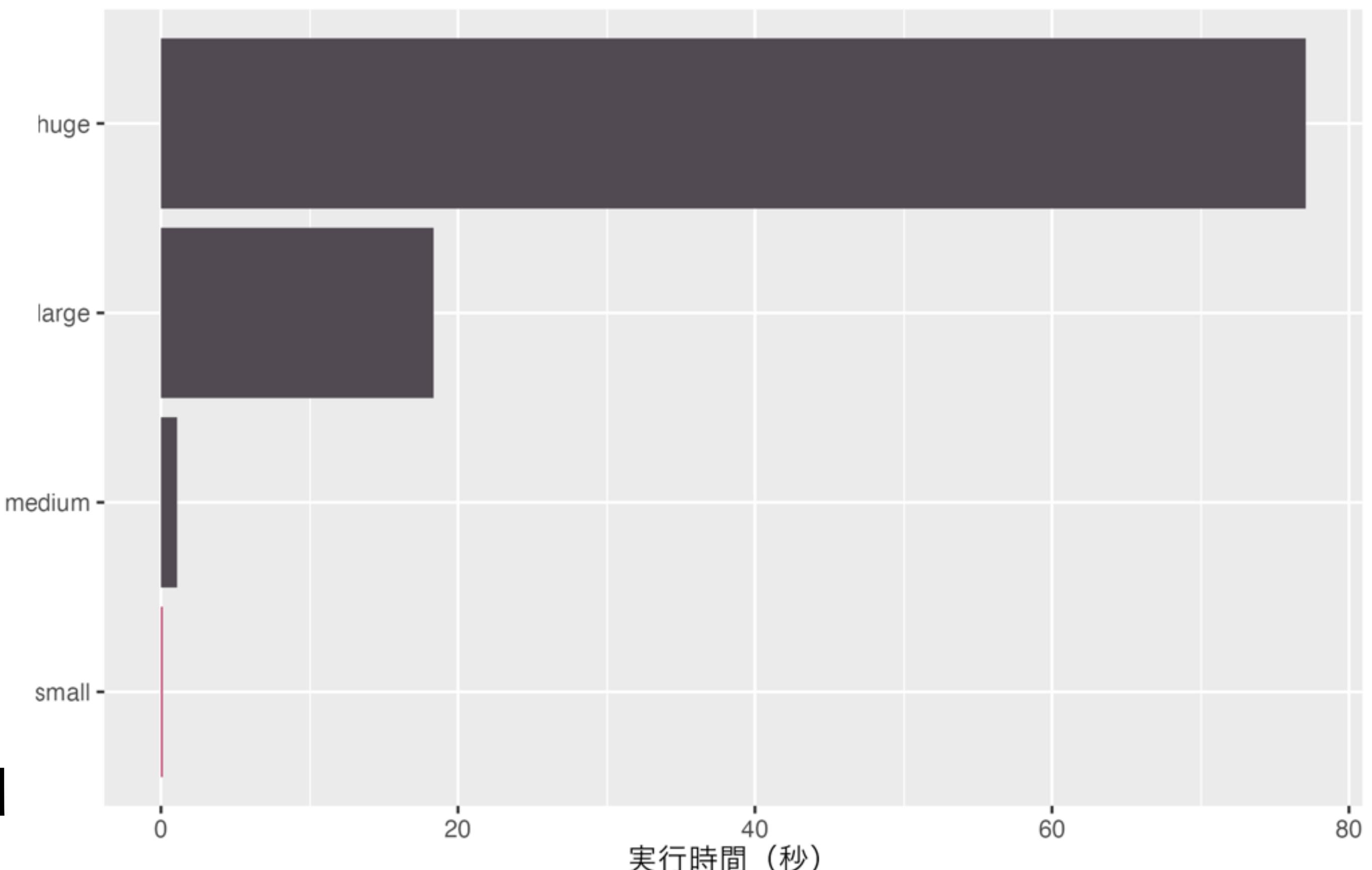


結果2. グループ化と集計

4種のデータサイズの結果を比較

タスク: 計測地点番号と10kmメッシュごとに断面交通量を合計

データサイズ: 4通り



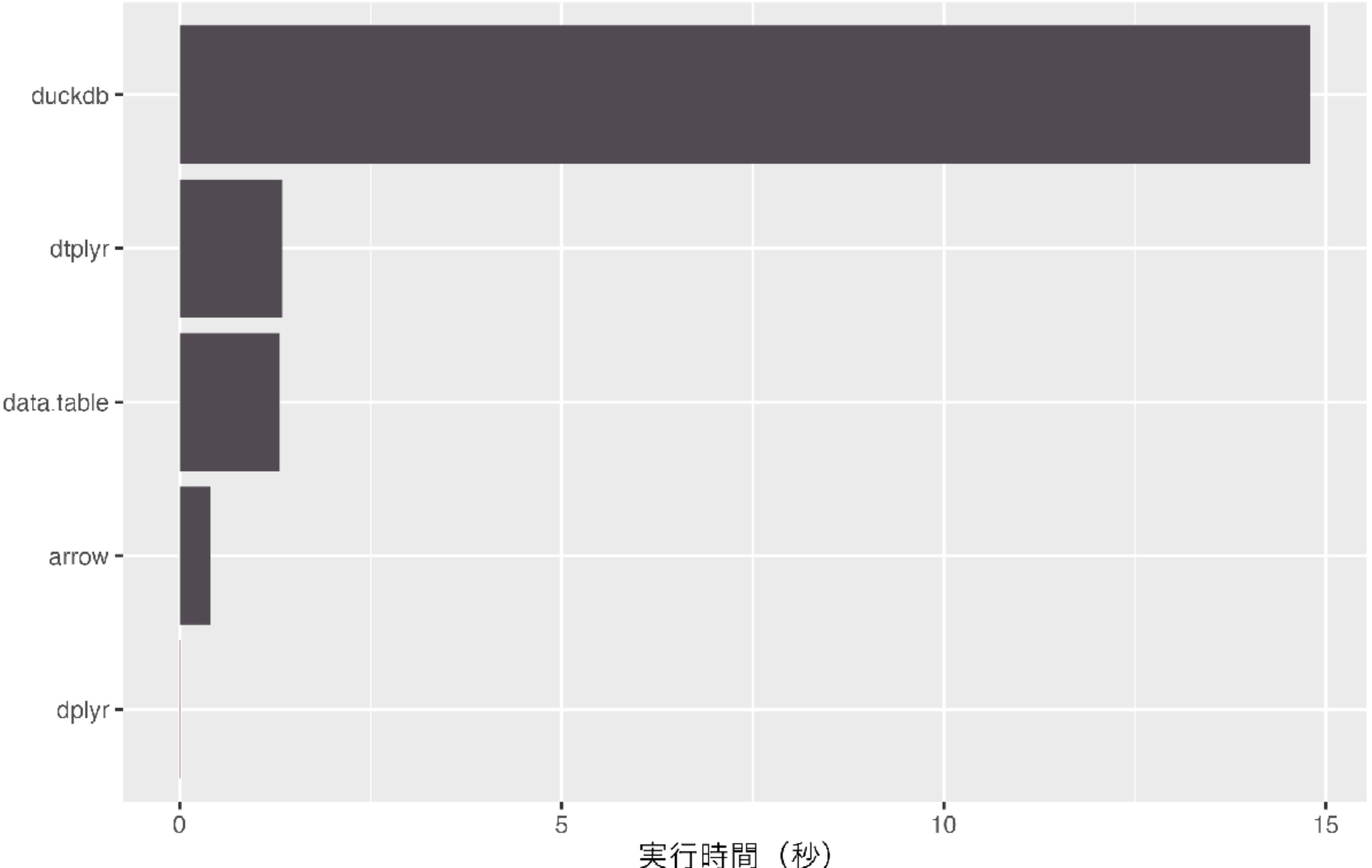
データサイズに応じて実行時間も増加

結果3. 結合

Small



タスク: メッシュコードを持つデータと結合
データサイズ: Small (1920万9389件)

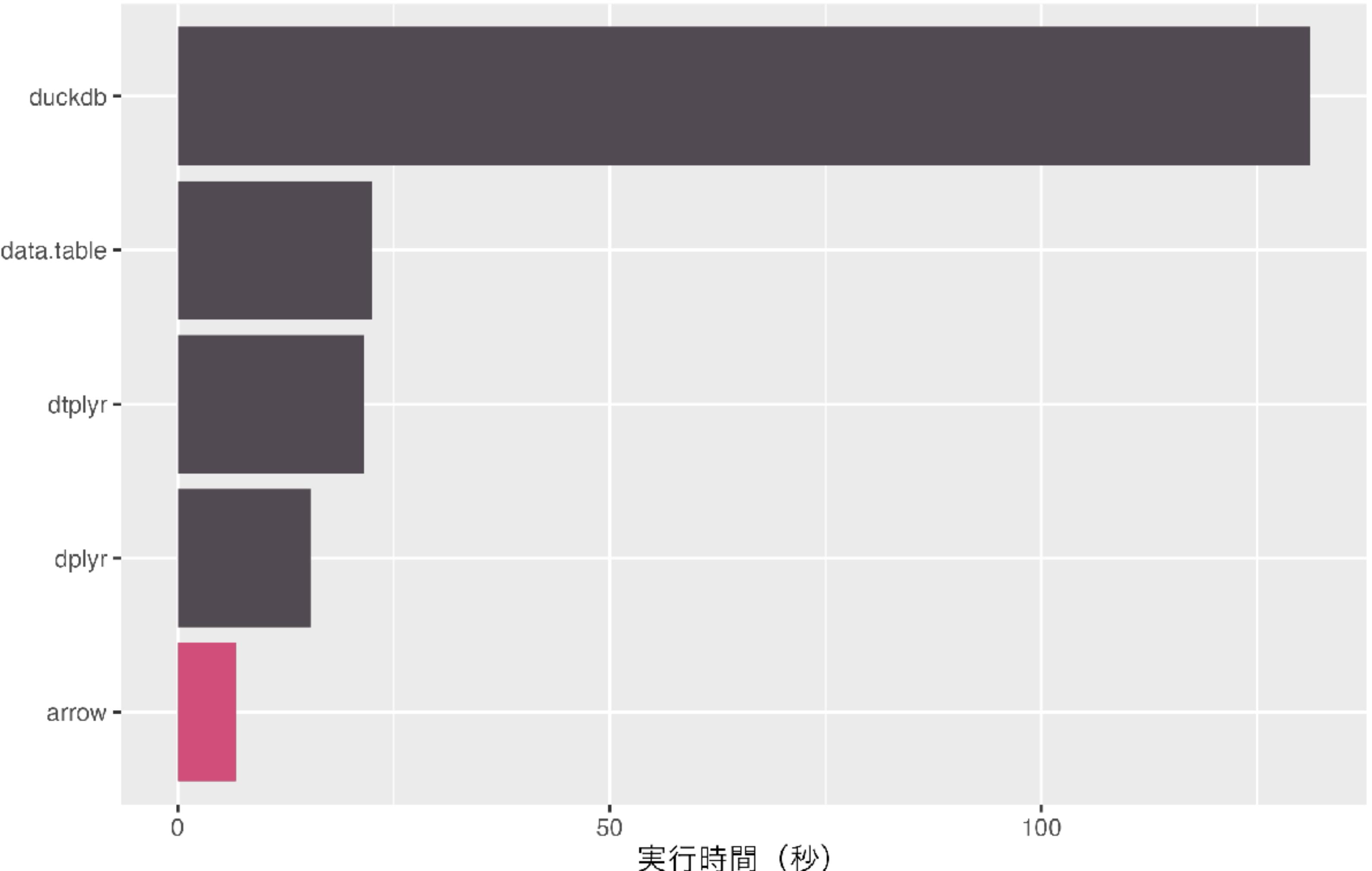


結果3. 結合

Medium



タスク: メッシュコードを持つデータと結合
データサイズ: Medium (2億2801万2986件)



結果3. 結合

Large, Hugeとともに失敗

filterやdistinctをせずに結合したため？

distinctを挟めばOK

Large

```
tictoc::tic();
traffic_large |>
  distinct(location_no, meshcode10km) |>
  left_join(arrow_mesh10km,
             by = c("meshcode10km" = "meshcode10km")) |>
  collect();
tictoc::toc()
#> 16.746 sec elapsed
```

Huge

```
tictoc::tic();
traffic_huge |>
  distinct(location_no, meshcode10km) |>
  left_join(arrow_mesh10km,
             by = c("meshcode10km" = "meshcode10km")) |>
  collect();
tictoc::toc()
#> 67.593 sec elapsed
```

所見

{arrow}で大規模なデータを処理できる（少なくともインメモリの状態に）

大規模なファイルは parquet で保存しておくのが良さそう

パーティショニングの基準の決め方が重要

どんなとき（サイズ・処理内容）でも{arrow}がベストな選択肢ではない

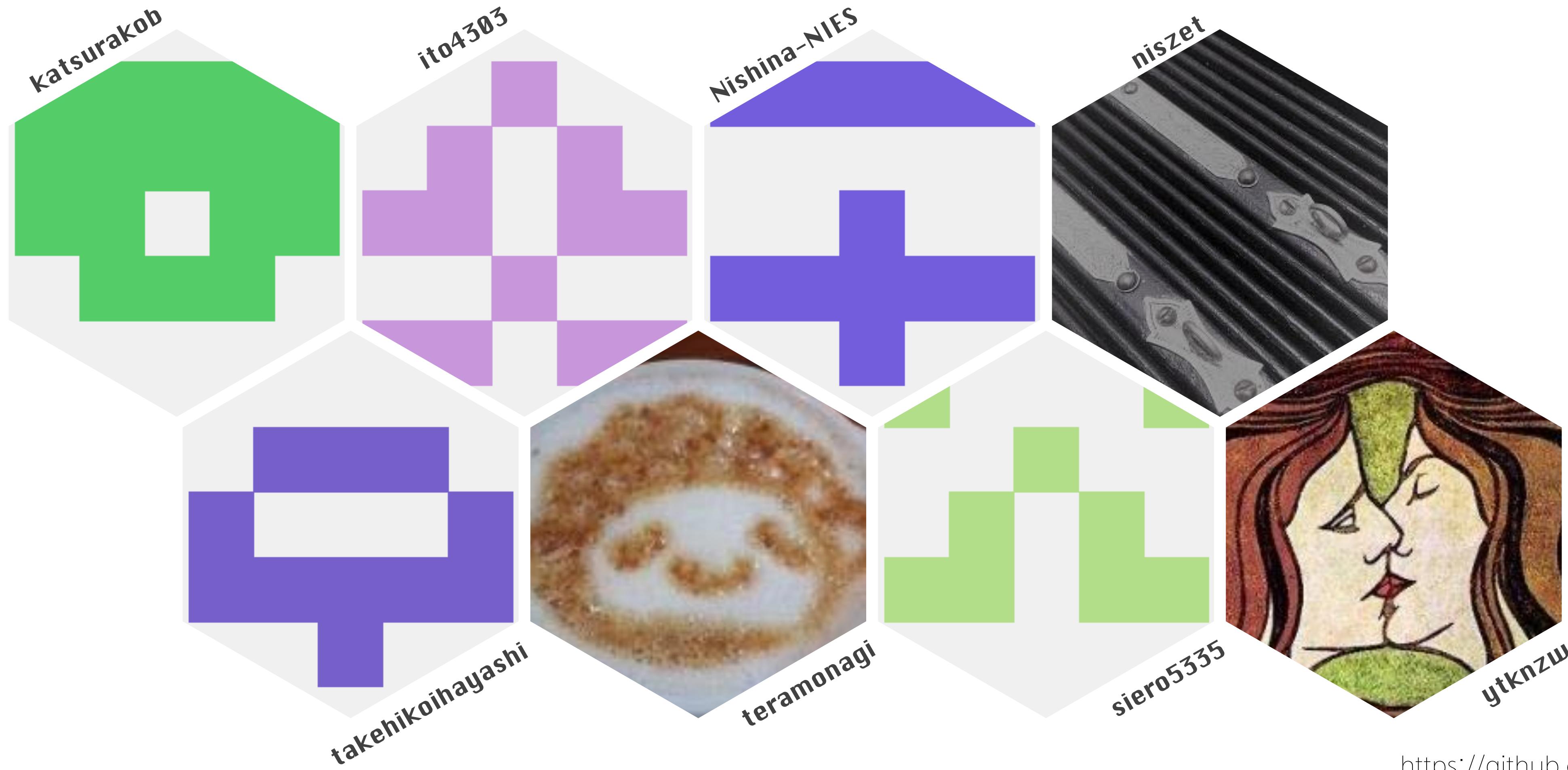
サポートしない関数→{duckdb}へ渡す

{data.table}、{dplyr}も頑張っている

謝辞

この発表はGitHub Sponsorsで支援してくださる方々の提供でお送りしました。

この場を借りて感謝申し上げます。



参考文献・URL

そろそろRユーザーもApache ArrowでParquetを使ってみませんか？

<https://notchained.hatenablog.com/entry/2019/12/17/213356>

R for Data Science (2nd edition)

<https://r4ds.hadley.nz/arrow.html>

Apache Arrowフォーマットはなぜ速いのか

<https://slide.rabbit-shocker.org/authors/kou/db-tech-showcase-online-2020/>

Apache Arrow 鬼はええ！このままCSV全部Parquetに変換していこうぜ！

https://eitsupi.github.io/tokyorslide/tokyor_97/

Larger-Than-Memory Data Workflows with Apache Arrow

<https://arrow-user2022.netlify.app>

Apache Arrow R Cookbook

<https://arrow.apache.org/cookbook/r/index.html>

【R】Apache Arrowとduckdbを試してみる - Qiita

<https://qiita.com/eitsupi/items/ce3e1b1fb0e45e0d45e3>