你只需要library(data.table)

data.table是一个稳定、高效、强大的数据处理工具

谭显英 @shrektan 2020.12.20

“具体问题具体分析”是……

每一种工具或最佳实践都有自己适合的场景

“只library(data.table)”适用的场景

- 脚本的大部分都是在进行数据处理/变换
- 注重代码的可维护性和稳定性
- 希望“试验代码”和“生产代码”差异比较小
- 对于代码的执行效率和写作效率有较高的要求
- 愿意花上一点时间去熟悉一种“新”的数据处理“语法”和“思维方式”

尽量避免library()

一般来说，“生产”脚本的最佳实践

为什么要尽量避免library()

library()虽然能减少书写代码时的一些麻烦，但有一定弊端

一. 可能导致函数冲突的问题，包的载入顺序可能会影响代码结果

二. 由于不清楚函数来自哪个包，未来进行代码阅读或者进行修改时会有一定的困难

三. pkg::fun()的形式可以很清晰地知道每一个函数是来自于哪里的，日后进行替换或者修改非常方便

为什么可以 library(data.table)?

“稳定、高效、强大” —— 就和base包一样

稳定

= 可以放心地在生产环境中使用data.table

极为稳定

- 彻底的零依赖，底层使用C语言和R API交互，避免依赖的包发生变更而导致代码错误
- 可以放心升级，体验到最新的特性和效率优化，而不用担心代码报错
- data.table继承了R语言的传统，把历史兼容性放在首位，目前最老的R版本支持R 3.1.0（2014年，6年）
- 任何函数或者API的变化，格外慎重

data.table改变API的流程

1.尽可能地避免更改

2.通过reverse check来避免意料之外的更改或监测更改的影响

3.万一有不得不做的更改：对于影响范围较大的更改，可能会先保留当前行为，但是给以警告，从而让用户有充足的时间去调整。对于影响范围确定很小的更改就可能是直接更改，但是提供很明确的警示或错误，并且会同时提供一个便捷的选项，如options(data.table.revert-to-old-behavior)，让用户能很方便地恢复之前的行为。

37. Using `with=FALSE` together with `:=` was deprecated in v1.9.4 released 2 years ago (Oct 2014). As warned then in release notes (see below) this is now a warning with advice to wrap the LHS of `:=` with parenthesis; e.g. `myCols=c("colA","colB"); DT[,(myCols):=1]`. In the next release, this warning message will be an error message.

高效

= 计算速度快 + 对内存友好

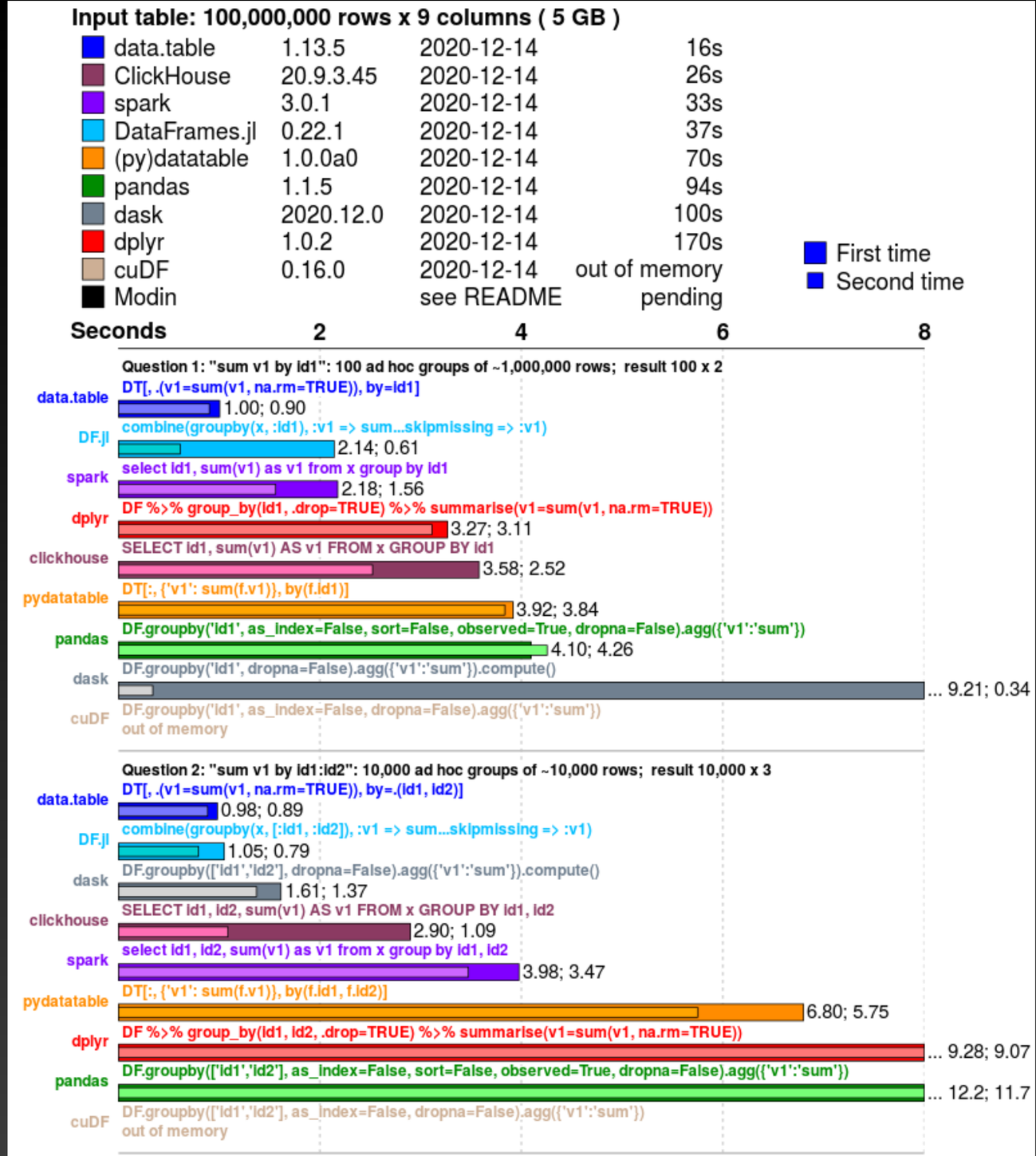
计算速度重要吗?

如果能不费劲就能写出运算速度快的代码 —— Why Not?

- 同样的“开发”速度下，计算速度越快越好
- 1毫秒和10毫秒的差异不大，但1分钟和10分钟、1小时和10小时的差异非常大——计算效率往往是倍数关系
- 研究工作往往意味着不停地重复——不停地提出想法（假设），不停地验证（代码）——更快的速度 = 更快的迭代 = 更高效的工作

计算速度

https://h2oai.github.io/db-benchmark/



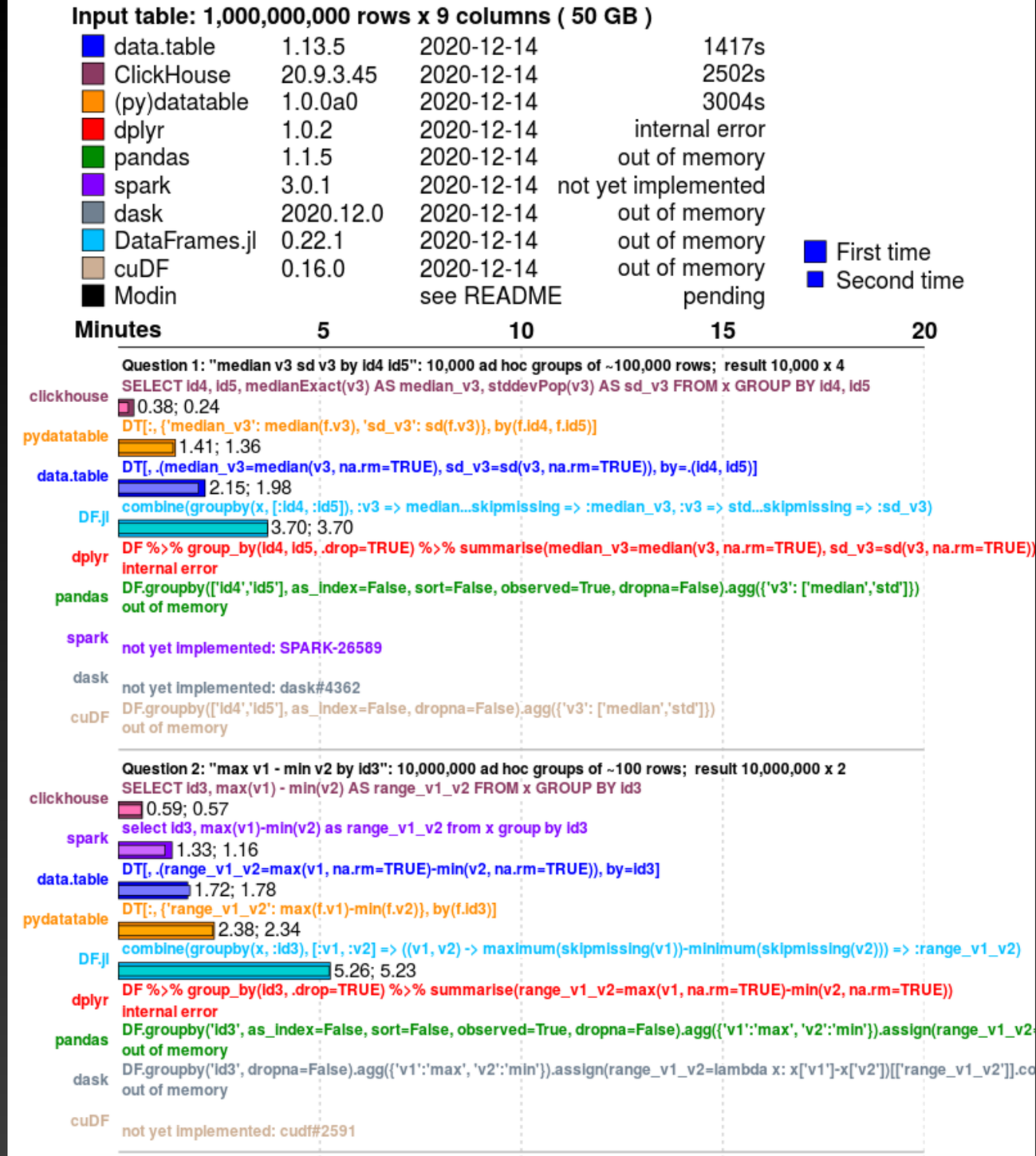
内存友好是什么？

也许很多时候你都不需要，一旦有需要就会觉得"真香"

- 一般来说，数据需要被载入内存后才能被计算机处理
- 因此，内存的大小决定了可以被处理的数据的大小
- 16GB的可用内存是否意味着可以处理16GB的数据呢？一般来说，只能处理1/2甚至1/4的数据（也就是8GB或者4GB）
- 内存友好意味着，一次性能够处理更多的数据 —— 而且往往也更快 —— 因为避免了"复制"

内存效率

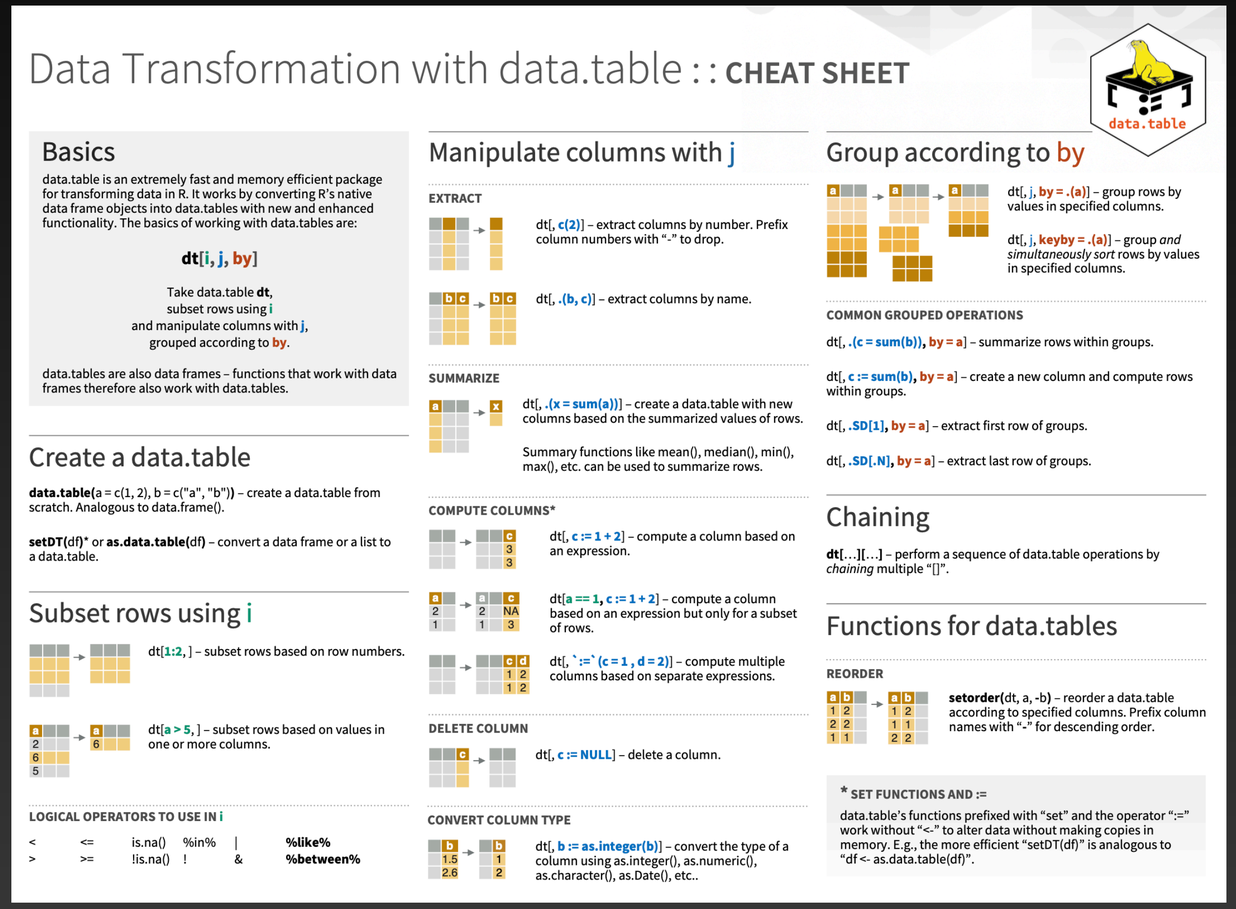
https://h2oai.github.io/db-benchmark/



强大

= 完善的功能 + 简洁的语法 + 丰富的表达力

具备完善和丰富的数据处理功能



Data Transformation with data.table : : **CHEAT SHEET**

data.table

Basics

data.table is an extremely fast and memory efficient package for transforming data in R. It works by converting R's native data frame objects into data.tables with new and enhanced functionality. The basics of working with data.tables are:

dt[i, j, by]

Take data.table **dt**,
subset rows using **i**
and manipulate columns with **j**,
grouped according to **by**.

data.tables are also data frames – functions that work with data frames therefore also work with data.tables.

Create a data.table

data.table(a = c(1, 2), b = c("a", "b")**)** – create a data.table from scratch. Analogous to data.frame().

setDT(df**)*** or **as.data.table(**df**)** – convert a data frame or a list to a data.table.

Subset rows using i

dt[**1:2**,] – subset rows based on row numbers.

dt[**a > 5**,] – subset rows based on values in one or more columns.

LOGICAL OPERATORS TO USE IN i

<	<=	is.na()	%in%	|	**%like%**
>	>=	!is.na()	!	&	**%between%**

Manipulate columns with j

EXTRACT

dt[, **c(2)**] – extract columns by number. Prefix column numbers with "-" to drop.

dt[, **.(b, c)**] – extract columns by name.

SUMMARIZE

dt[, **.(x = sum(a))**] – create a data.table with new columns based on the summarized values of rows.

Summary functions like mean(), median(), min(), max(), etc. can be used to summarize rows.

COMPUTE COLUMNS*

dt[, **c := 1 + 2**] – compute a column based on an expression.

dt[**a == 1, c := 1 + 2**] – compute a column based on an expression but only for a subset of rows.

dt[, **`:=`(c = 1 , d = 2)**] – compute multiple columns based on separate expressions.

DELETE COLUMN

dt[, **c := NULL**] – delete a column.

CONVERT COLUMN TYPE

dt[, **b := as.integer(b)**] – convert the type of a column using as.integer(), as.numeric(), as.character(), as.Date(), etc..

Group according to by

dt[, j, **by = .(a)**] – group rows by values in specified columns.

dt[, j, **keyby = .(a)**] – group *and simultaneously sort* rows by values in specified columns.

COMMON GROUPED OPERATIONS

dt[, **.(c = sum(b)), by = a**] – summarize rows within groups.

dt[, **c := sum(b), by = a**] – create a new column and compute rows within groups.

dt[, **.SD[1], by = a**] – extract first row of groups.

dt[, **.SD[.N], by = a**] – extract last row of groups.

Chaining

dt[…][…] – perform a sequence of data.table operations by *chaining* multiple "[]".

Functions for data.tables

REORDER

setorder(dt, a, -b**)** – reorder a data.table according to specified columns. Prefix column names with "-" for descending order.

*** SET FUNCTIONS AND :=**

data.table's functions prefixed with "set" and the operator ":=" work without "<-" to alter data without making copies in memory. E.g., the more efficient "setDT(df)" is analogous to "df <- as.data.table(df)".

read & write files

IMPORT

fread("file.csv"**)** – read data from a flat file such as .csv or .tsv into R.

fread("file.csv", select = c("a", "b")**)** – read specified columns from a flat file into R.

EXPORT

fwrite(dt, "file.csv"**)** – write data to a flat file from R.

Reshape a data.table

RESHAPE TO WIDE FORMAT

id	y	a	b
A	x	1	3
A	z	2	4
B	x	1	3
B	z	2	4

→

id	a_x	a_z	b_x	b_z
A	1	2	3	4
B	1	2	3	4

dcast(dt,
id ~ y,
value.var = c("a", "b")**)**

Reshape a data.table from long to wide format.

dt	A data.table.
id ~ y	Formula with a LHS: ID columns containing IDs for multiple entries. And a RHS: columns with values to spread in column headers.
value.var	Columns containing values to fill into cells.

RESHAPE TO LONG FORMAT

id	a_x	a_z	b_x	b_z
A	1	2	3	4
B	1	2	3	4

→

id	y	a	b
A	1	1	3
B	1	1	3
A	2	2	4
B	2	2	4

melt(dt,
id.vars = c("id"),
measure.vars = patterns("^a", "^
variable.name = "y",
value.name = c("a", "b")**)**

Reshape a data.table from wide to long format.

dt	A data.table.
id.vars	ID columns with IDs for multiple entries.
measure.vars	Columns containing values to fill into cells (often in pattern form).
variable.name, value.name	Names of new columns for variables and values derived from old headers.

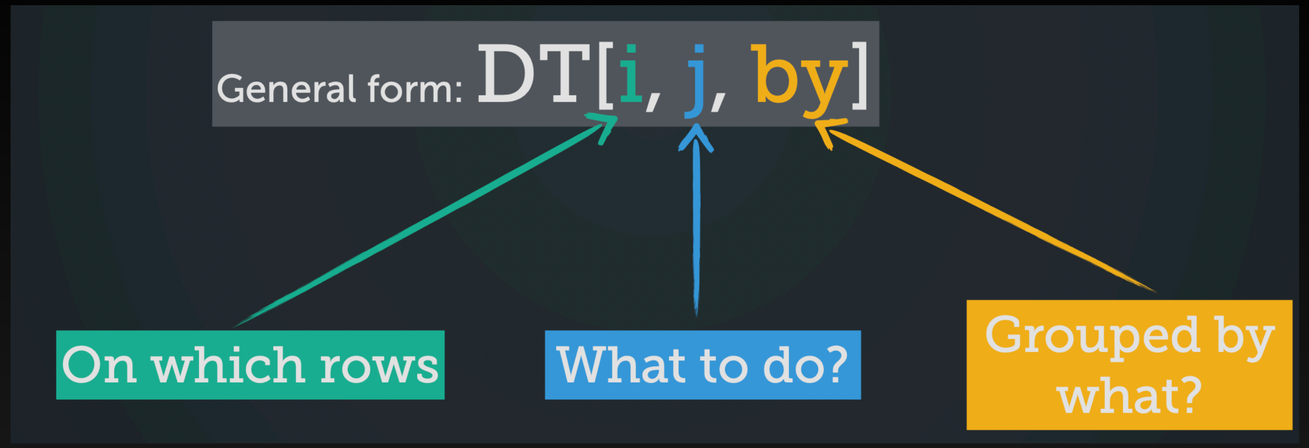
- fifelse, fcase
- nafill, fcolaese
- %between%, %like%
- .(), J(), CJ()
- rbindlist()
- ……

Picture Source: https://raw.githubusercontent.com/rstudio/cheatsheets/master/datatable.pdf

简洁的语法

最懒的程序员是最有效率的

- 语法简洁到其基本形式只有一个：DT[i(行), j(列), by(组)]
- 简洁却不简单：其语法设计我个人认为是"天才"性的（感谢作者@mattdowle），它非常巧妙地与data.frame的语法、R语言的特性结合了起来
- 一旦熟悉后就会觉得这种语法用来处理表格类数据才是最合适的




```
library(data.table)
iris_dt <- as.data.table(iris)
iris_dt[Species %in% c("setosa", "virginica"), .(Avg_Speal_Len = mean(S
epal.Length)), by = Species]
#>       Species Avg_Speal_Len
#> 1:     setosa         5.006
#> 2: virginica         6.588
```

1.选取Species为setosa或virginica的数据

2. 组别为Species

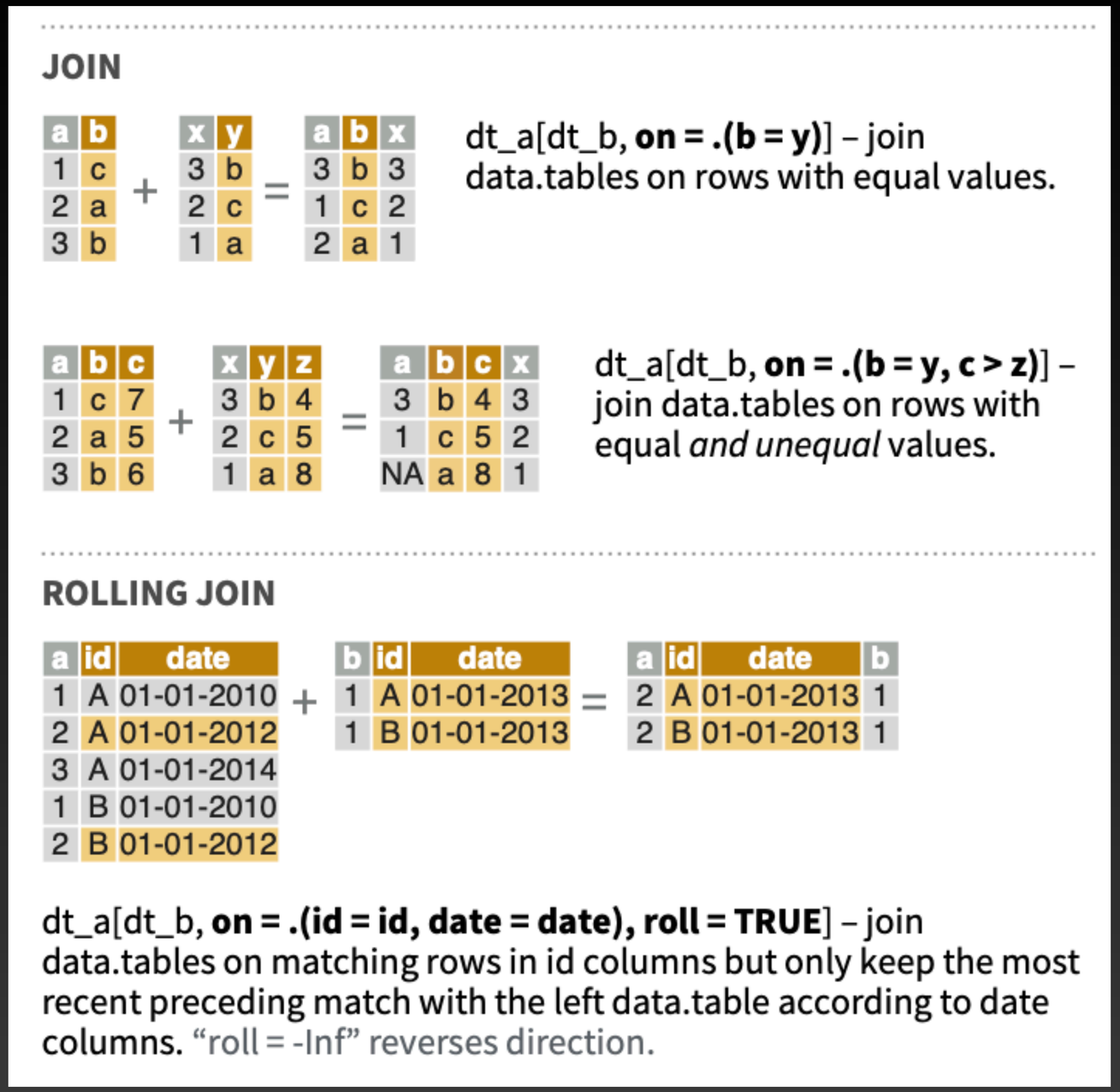
3.计算筛选后数据中，对各组别分别统计Sepal.Length的平均值

Picture Source: https://github.com/arunsrinivasan/user2017-data.table-tutorial/blob/master/user2017-arun-tutorial.pdf

丰富的表达力

简洁的语法反而具有更丰富的表达力

- 写的更少但表达得更多：如支持非常丰富的表联结(join)功能——rolling join, non-equi join, aggreagated join等
- 写的更少但写得更快：敲的字更少、更容易敲正确
- 写的更少但算得更快：一行语句可以表达多个含义，为在底层C代码中进行单独优化提供了可能性



Picture Source: https://raw.githubusercontent.com/rstudio/cheatsheets/master/datatable.pdf

感受一下

三个真实世界中的小案例

案例1

找到"最近可获得"的证券价格

- 表A：存储着市场上所有证券的日度价格信息，其字段有：证券代码，日期，价格 。证券可能由于停牌或其他原因导致并不是每个时点上都有数据。
- 表B：某个基金的持仓表，字段有：证券代码，日期。
- 需求：在表B上增加一列"价格"，等于每行证券代码和日期对应的价格。如果当前日期没有价格，则去取"最近可获得价格的时点"的价格。但是，如果"最近的可获得价格的时点"早于该日期10天，则认为该价格已经过时了，应该填写NA。

```
library(data.table)
A = data.table(
  证券代码 = c("A", "A", "A", "B", "B", "B"),
  日期 = c(1, 2, 3, 1, 2, 3),
  价格 = c(10.1, 10.2, 10.3, 20.1, 20.2, 20.3)
)
B = data.table(
  证券代码 = c("A", "B", "A", "B"),
  日期 = c(13, 13, 14, 14)
)
A
#>    证券代码 日期 价格
#> 1:        A    1 10.1
#> 2:        A    2 10.2
#> 3:        A    3 10.3
#> 4:        B    1 20.1
#> 5:        B    2 20.2
#> 6:        B    3 20.3
B
#>    证券代码 日期
#> 1:        A   13
#> 2:        B   13
#> 3:        A   14
#> 4:        B   14
B[, 价格 := {
  A[B, 价格, roll = 10, on = c("证券代码", "日期")]
}]
B
#>    证券代码 日期 价格
#> 1:        A   13 10.3
#> 2:        B   13 20.3
#> 3:        A   14   NA
#> 4:        B   14   NA
```

案例2

统计任意区间内证券的平均交易金额

- 表A：存储着市场上所有证券的日度交易信息，其字段有：证券代码，日期，交易金额 。
- 表B：某个基金的历史证券持有信息，字段有：证券代码，起始日期，终止日期。
- 需求：在表B上增加一列"市场交易金额"，等于该行证券代码在该时段交易金额的总和。

```r
library(data.table)
n = 10
A = data.table(
  证券代码 = c(rep("A", n), rep("B", n)),
  日期 = rep(seq_len(n), 2),
  交易金额 = c(10 + seq_len(n), 20 + seq_len(n))
)
B = data.table(
  证券代码 = c("A", "B", "A", "B"),
  起始日期 = c(3, 1, 2, 7),
  终止日期 = c(5, 10, 4, 10)
)
A
#>      证券代码 日期 交易金额
#>  1:         A    1        11
#>  2:         A    2        12
#>  3:         A    3        13
#>  4:         A    4        14
#>  5:         A    5        15
#>  6:         A    6        16
#>  7:         A    7        17
#>  8:         A    8        18
#>  9:         A    9        19
#> 10:         A   10        20
#> 11:         B    1        21
#> 12:         B    2        22
#> 13:         B    3        23
#> 14:         B    4        24
#> 15:         B    5        25
#> 16:         B    6        26
#> 17:         B    7        27
#> 18:         B    8        28
#> 19:         B    9        29
#> 20:         B   10        30
B
#>    证券代码 起始日期 终止日期
#> 1:         A        3        5
#> 2:         B        1       10
#> 3:         A        2        4
#> 4:         B        7       10
B[, 市场交易金额 := {
  A[B, .(VALUE = sum(交易金额)),
    by = .EACHI,
    on = c("证券代码", "日期>=起始日期", "日期<=终止日期")][["VALUE"]]
}]
B
#>    证券代码 起始日期 终止日期 市场交易金额
#> 1:         A        3        5          42
#> 2:         B        1       10         255
#> 3:         A        2        4          39
#> 4:         B        7       10         114
```

案例3

更新某列信息--多个数据来源

- 表A1 - A3：为三个不同的境外债券数据源，字段为：证券代码，属性X。
- 表B：某个基金的持仓信息，字段有：证券类型，证券代码，属性X。
- 需求：更新表B中"属性X"的值。"属性X"的定义为，先去A1里面查找，如果找不到再去A2里面查找，如果还找不到再去A3里面，如果还找不到就保留原始的属性X。

```
library(data.table)
A1 = data.table(
  证券代码 = c("A"),
  属性X = 1L,
  key = "证券代码"
)
A2 = data.table(
  证券代码 = c("A", "B"),
  属性X = c(2L, 3L),
  key = "证券代码"
)
A3 = data.table(
  证券代码 = c("C"),
  属性X = c(4L),
  key = "证券代码"
)
B = data.table(
  证券代码 = c("A", "B", "C", "D"),
  属性X = 11:14,
  key = "证券代码"
)
A1
#>    证券代码 属性X
#> 1:        A     1
A2
#>    证券代码 属性X
#> 1:        A     2
#> 2:        B     3
A3
#>    证券代码 属性X
#> 1:        C     4
B
#>    证券代码 属性X
#> 1:        A    11
#> 2:        B    12
#> 3:        C    13
#> 4:        D    14
```

```
B[, 属性X := {
  fcoalesce(
    A1[B, 属性X],
    A2[B, 属性X],
    A3[B, 属性X],
    属性X
  )
}]
B
#>    证券代码 属性X
#> 1:        A     1
#> 2:        B     3
#> 3:        C     4
#> 4:        D    14
```

如何学习？

学习资源

http://r-datatable.com

- 最好能耐心地把Vignettes通读一遍
- 尤其是这四篇：“Introduction to data.table“ + “Keys and fast binary search based subset” + “Reference semantics” + “Frequently Asked Questions”
- `?data.table::data.table`
- 三个概念：Key & Query, Modify by Reference, DT[where, how, by = what, on = which]
- [Datatable Cheat Sheet](https://raw.githubusercontent.com/rstudio/cheatsheets/master/datatable.pdf)
- Stackoverflow, https://github.com/Rdatatable/data.table/issues

谢谢！

Github: @shrektan / Email: shrektan@126.com