ILLINOIS

Data Wrangling with data.table

Lecture 13

Dirk Eddelbuettel

STAT 430: Data Science Programming Methods (Fall 2019)

Department of Statistics, University of Illinois

Topics

- The `[i, j, by]` idiom
- `fread` and `fwrite`
- `dcast` and `melt`
- examples / case study in appendix
- Resources:
  - data.table cheatsheet
  - data.table wiki

Previous Lecture

- Data Wrangling with Base R
- `data.frame` object, subsetting and indexing
- `aggregate()`, the `*apply()` functions and more

Today

- One more powerful extension of `data.frame`
- We will see another next lecture

Matrices

- Two-dimensional data structure
- All elements must be of the same type
  - Indexing by position, name or logical expression:

```
M[1, 4:5]
M[1:10, "rates"]        # if col 'rates' exists
M[ M[,"rates"] > 0, 2:5]
```

- Also:

```
M <- matrix(1:9,3); I <- matrix(c(2,3,1,2),2)
M[ I ]
```

Data.Frame

- Core R Data Structure
- Different column types allowed, must have same length
- Indexing by position, name or logical expression:

```
DF[1, 4:5]
DF[1:10, "rates"]          # if col 'rates'
DF[ DF[,"rates"] > 0, 2:5]
```

Data.Table

A Powerful and Popular Package

ILLINOIS

- Written by
  - Matt Dowle
  - Arun Srinivasan
  - and many other contributors
- Highly-optimised replacement for `data.frame`:
  - fast aggregation
  - fast (ordered) join
  - fast add/modify/delete by group with copies
  - fast input/output
- Very stable and mature
- High performance, efficient, concise

ILLINOIS

A Few Starting Points

- General Overview at `https://r-datatable.com`
- CRAN starting point with vignettes, FAQ,
- Introduction to data.table vignette
- Getting started Wiki
- Cheat Sheet

List of Vignettes

- Introduction to Data.Table
- FAQs about the data.table package
- Efficient reshaping using data.tables
- Keys and fast binary search based subset
- Reference Semantics
- Using .SD for Data Analysis
- Benchmarking data.table

All (and more) at

`https://cloud.r-project.org/web/packages/data.table`

`fread()` and `fwrite()`

- a *very* fast (parallel) and general reader / writer for csv files
- general alternative to `read.csv()` and `write.csv()`

`rbindlist()`

- more efficient than `do.call(rbind, listObj)`
- uses less memory, runs faster

```
dt[i, j, by]
```

where

- *row-selection i*
- *column expression j*
- optional grouping **by**

```
dt[i, j, by]
```

or as in SQL

- `where` *i* expression
- `select` *j* expression
- `group by` expression

```
library(data.table)    ## load data.table

## 2014 (Jan - Oct) flights data in data.table sources
url <- paste0("https://github.com/Rdatatable/data.table/",
              "blob/master/vignettes/flights14.csv?raw=true")

flights <- fread(url)  ## returns a data.table object

flights                ## quick head + tail summary view

head(flights)          ## just like on data.frame object
dim(flights)
```

```
## General idea:     DT[i, j, by]
## SQL equivalent:   where i    select    group by

## work on i -- Subset rows, here two criteria
ans <- flights[origin == "JFK" & month == 6L]

## work on i -- or just two rows
flights[1:2]

## work on i -- sort by two criteria
ans <- flights[order(origin, -dest)]
```

Note how we get direct access to the column names inside `[ ... ]`

```
## select in j -- select several and rename
## the .() is shorthand for list() and very useful
ans <- flights[, .(delay_arr = arr_delay,
                   delay_dep = dep_delay)]

## select in j -- compute (more on next slide)
ans <- flights[, sum((arr_delay + dep_delay) < 0)]
```

```
## using both i and j -- subset and compute
##
## important: in the j part, any R function is available
## grouping as well via { }
ans <- flights[origin == "JFK" & month == 6L,
               .(m_arr = mean(arr_delay),
                 m_dep = mean(dep_delay))]
```

```
## i,j,by -- delays at JFK across year
## very powerful: group by month, select JFK only, avg delay
ans <- flights[origin == "JFK",
               .(m_arr = mean(arr_delay),
                 m_dep = mean(dep_delay)), by=month]

# how many trips from JFK in June?
flights[origin == "JFK" & month == 6L, length(dest)]
flights[origin == "JFK" & month == 6L, .N]   # shorthand
```

```
## what if we want columns by name as in data.frame?
ans <- flights[, c("arr_delay", "dep_delay")]

## NB: that is a new-ish feature
## earlier releases may need to add 'with=FALSE' for names
ans <- flights[, c("arr_delay", "dep_delay"), with=FALSE]
```

```
## how many trips departing at each airport?
ans <- flights[, .(.N), by=.(origin)]

## use 'keyby' for different sort order
ans <- flights[carrier == "AA",
               .(avgarr = mean(arr_delay),
                 avgdep = mean(dep_delay)),
               keyby = .(origin, dest, month)]
```

```
## 'chaining' to efficiently combine queries
ans <- flights[carrier == "AA", .N,
               by=.(origin, dest)][order(origin, -dest)]

## expressions in by -- try this too and reason about it
ans <- flights[, .N, .(dep_delay>0, arr_delay>0)]
```

Using *i*: "where"

- Subset rows similar to `data.frame`
- ... but without repeated naming of `DT$`
- Sort using `order()` using fast internal order function
- Can do much more in `i` by keying `data.table`
- This allow fast subsets and (inner, outer, ...) joins

Using *j*: "select"

```
## select columns the data.table way
DT[, .(colA, colB)]
DT[, c("colA", "colB"), with=FALSE]  # older versions
DT[, c("colA", "colB")]              # current version

## compute aggregations of data -- here whole table
DT[, .(sum(colA), mean(colB))]
DT[, .(sA=sum(colA), mB=mean(colB))]  # can name too

## combine aggregation with  i  selection
DT[colA > value, sum(colB)]
```

Using by: "group by"

- Group by columns by
  - specifying a list of columns or
  - character vector of column names or expressions
- Handle multiple columns and also expressions.
- Use `keyby` grouping to sort the grouped result.
- Use `.SD` and `.SDcols` in *j* to operate on multiple columns using already familiar base functions.

The := operator to update by reference

- In `data.frame()`, assignment often leads to full copy
- (Though more recent versions of R improved that)
- In `data.table()` use `:=` in one of two ways:

```
DT[, c("colA","colB",...) := list(valA,valB,...)]
DT[, `:=`(colA = valA,  # valA is assigned to colA
          colB = valB,  # valB is assigned to colB
          ...
          )]
```

- Assigns 'within' so no need for new DT object

ILLINOIS

Examples of `:=`

```
flights[, `:=`(speed = distance/(air_time/60), # in km/hr
               delay = arr_delay+dep_delay)]   # in min

flights[, delay := NULL]  # deletes instantly

flights[, max_speed := max(speed), by=.(origin, dest)]

in_cols  <- c("dep_delay", "arr_delay")
out_cols <- c("max_dep_delay", "max_arr_delay")
flights[, c(out_cols) := lapply(.SD, max),
        by = month, .SDcols = in_cols]
```

DCAST AND MELT

A 'tall and narrow' data set

```
R> names(ChickWeight) <- tolower(names(ChickWeight))
R> data.table(ChickWeight)[]
#      weight time chick diet
#   1:     42    0     1    1
#   2:     51    2     1    1
#   3:     59    4     1    1
#  ---
# 576:    234   18    50    4
# 577:    264   20    50    4
# 578:    264   21    50    4
```

Measurements of weight over time for different chickens and diet.

We need both *long* and *wide* datasets.

dcast and melt: Initial Example

ILLINOIS

```
R> ## first we melt with weight as the value var
R> ## calls melt.data.table; id here by position
R> DT <- melt(as.data.table(ChickWeight), id=2:4)
R> DT[]
#       time chick diet variable value
#    1:    0     1    1   weight    42
#    2:    2     1    1   weight    51
#    3:    4     1    1   weight    59
#   ---
# 576:   18    50    4   weight   234
# 577:   20    50    4   weight   264
# 578:   21    50    4   weight   264
```

```
R> W <- dcast(DT, diet + chick ~ time, drop=FALSE)
R> W[]
#       diet chick  0  2  4  6   8  10  12  14  16  18  20  21
#    1:    1    18 39 35 NA NA  NA  NA  NA  NA  NA  NA  NA  NA
#    2:    1    16 41 45 49 51  57  51  54  NA  NA  NA  NA  NA
#    3:    1    15 41 49 56 64  68  68  67  68  NA  NA  NA  NA
#   ---
# 198:    4    50 41 54 67 84 105 122 155 175 205 234 264 264
# 199:    4    42 42 49 63 84 103 126 160 174 204 234 269 281
# 200:    4    48 39 50 62 80 104 125 154 170 222 261 303 322
```

Here using formula to select **diet** and **check** as variables – leaving **weight** as the measurement split over columns.

Long to wide: Start with a long dataset

```
R> library(data.table)
R> dt <- data.table(mtcars)
R> dt[,1:10]  # 32 rows, 11 cols
#      mpg cyl disp  hp drat    wt  qsec vs am gear
#  1: 21.0   6  160 110 3.90 2.620 16.46  0  1    4
#  2: 21.0   6  160 110 3.90 2.875 17.02  0  1    4
#  3: 22.8   4  108  93 3.85 2.320 18.61  1  1    4
# ---
# 30: 19.7   6  145 175 3.62 2.770 15.50  0  1    5
# 31: 15.0   8  301 335 3.54 3.570 14.60  0  1    5
# 32: 21.4   4  121 109 4.11 2.780 18.60  1  1    4
```

```
R> dcast(dt, gear ~ cyl, value.var = c("disp", "hp"),
+        fun = list(mean, sum))
#    gear disp_mean_4 disp_mean_6 disp_mean_8 hp_mean_4 hp_mean_6
# 1:    3      120.10        241.5      357.62         97     107.5
# 2:    4      102.62        163.8         NaN         76     116.5
# 3:    5      107.70        145.0      326.00        102     175.0
#    hp_mean_8 disp_sum_4 disp_sum_6 disp_sum_8 hp_sum_4 hp_sum_6
# 1:    194.17      120.1      483.0     4291.4        97      215
# 2:       NaN      821.0      655.2        0.0       608      466
# 3:    299.50      215.4      145.0      652.0       204      175
#    hp_sum_8
# 1:     2330
# 2:        0
# 3:      599
```

```
R> dcast(dt, gear ~ cyl, value.var = c("disp", "hp"),
+         fun = list(mean, sum))
```

What did this do?

- use `gear` as index column
- `mean` and `sum` will be calculated
  - for variables `disp` and `hp`
  - for every `gear` and `cyl` combination
- more operations possible, see `help(dcast.data.table)`

Wide to long

```
R> melt(dt, c("cyl", "gear"), measure = "disp")
#      cyl gear variable value
#  1:   6    4     disp   160
#  2:   6    4     disp   160
#  3:   4    4     disp   108
# ---
# 30:   6    5     disp   145
# 31:   8    5     disp   301
# 32:   4    4     disp   121
```

Wide to long

```
R> melt(dt, c("cyl", "gear"), measure = "disp")
```

What did this do?

- using `cyl` and `gear` as index colums
- using `disp` as the variable selected
- many more options, see `help(melt.data.table)`

Not enough time in initial lecture for these advanced topics

- indexing and keys
- various `join` operations including very powerful rolling joins
- numerous `f*` functions including rolling functions
- 'gforce' replacement functions
- and much more

Some Resources

- Package / Project page
- GitHub repo and Wiki
- CRAN page
- Many talks and videos on the Wiki page

Appendix

Source

- Bill Gold talk at NY Data (Sep 2018)
- Code example at his GitHub repo

```
R> library (data.table)
R>
R> dt.mtcars <- data.table(mtcars, keep.rownames = TRUE)
R> # rownames becomes new column 'rn'
R> # subset columns so that results fit slides
R> dt.mtcars <- dt.mtcars[, 1:7]
R> dt.mtcars[1:5, ]      # will explain indices later
#                   rn  mpg cyl disp  hp drat    wt
# 1:          Mazda RX4 21.0   6  160 110 3.90 2.620
# 2:      Mazda RX4 Wag 21.0   6  160 110 3.90 2.875
# 3:         Datsun 710 22.8   4  108  93 3.85 2.320
# 4:     Hornet 4 Drive 21.4   6  258 110 3.08 3.215
# 5: Hornet Sportabout 18.7   8  360 175 3.15 3.440
```

The `i` are filters

```
R> # filter by column value
R> dt.mtcars[cyl == 8,  ]
#                      rn  mpg cyl disp  hp drat   wt
#  1: Hornet Sportabout 18.7   8  360 175 3.15 3.44
#  2:         Duster 360 14.3   8  360 245 3.21 3.57
#  3:         Merc 450SE 16.4   8  276 180 3.07 4.07
# ---
# 12:  Pontiac Firebird 19.2   8  400 175 3.08 3.85
# 13:    Ford Pantera L 15.8   8  351 264 4.22 3.17
# 14:     Maserati Bora 15.0   8  301 335 3.54 3.57
```

The `i` are filters

```
R> # filter by multiple conditions
R> # use '&', '|', '!' for Booleans
R> # not use of '%like%
R> dt.mtcars[cyl == 8   &
+              wt < 4   &
+              rn %like% 'Merc' ]
#             rn  mpg cyl  disp  hp drat   wt
# 1:  Merc 450SL 17.3   8 275.8 180 3.07 3.73
# 2: Merc 450SLC 15.2   8 275.8 180 3.07 3.78
```

The `i` are filters

```
R> # to displa first 5 rows
R> dt.mtcars[ 1:5 ]
#                    rn  mpg cyl disp  hp drat    wt
# 1:          Mazda RX4 21.0   6  160 110 3.90 2.620
# 2:      Mazda RX4 Wag 21.0   6  160 110 3.90 2.875
# 3:         Datsun 710 22.8   4  108  93 3.85 2.320
# 4:     Hornet 4 Drive 21.4   6  258 110 3.08 3.215
# 5: Hornet Sportabout 18.7   8  360 175 3.15 3.440
```

The j do select-list

```
R> # select-clause one variable, vector output
R> # (second [] explained later)
R> dt.mtcars[, rn ] [1:20]
#  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
#  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
#  [7] "Duster 360"          "Merc 240D"           "Merc 230"
# [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
# [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwoo
# [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
# [19] "Honda Civic"         "Toyota Corolla"
```

The j do select-list

```
R> # same as previous, much faster
R> # (second [] explained later)
R> dt.mtcars[['rn']] [1:20]
#  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
#  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
#  [7] "Duster 360"          "Merc 240D"           "Merc 230"
# [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
# [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwoo
# [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
# [19] "Honda Civic"         "Toyota Corolla"
```

The `j` do select-list

```
R> # select-clause one variable, data.table output
R> # (second [] explained later)
R> dt.mtcars[1:5, list(rn) ] [1:20]
#                rn
#  1:     Mazda RX4
#  2: Mazda RX4 Wag
#  3:    Datsun 710
# ---
# 18:           <NA>
# 19:           <NA>
# 20:           <NA>
```

The `j` do select-list

```
R> # multiple columns
R> dt.mtcars[1:5, list(rn, cyl, hp)]
#                    rn cyl  hp
# 1:          Mazda RX4   6 110
# 2:      Mazda RX4 Wag   6 110
# 3:         Datsun 710   4  93
# 4:     Hornet 4 Drive   6 110
# 5: Hornet Sportabout   8 175
```

The `j` do select-list

```
R> # same as previous as .() equals list()
R> dt.mtcars[1:5, .(rn, cyl, hp) ]
#                  rn cyl  hp
# 1:          Mazda RX4   6 110
# 2:      Mazda RX4 Wag   6 110
# 3:         Datsun 710   4  93
# 4:     Hornet 4 Drive   6 110
# 5: Hornet Sportabout   8 175
```

The `j` do select-list

```
R> # until recently (or on older data.table)
R> # add `,with=FALSE` as final argument
R> dt.mtcars[1:5, c('rn', 'cyl', 'hp') ]
#                  rn cyl  hp
# 1:          Mazda RX4   6 110
# 2:      Mazda RX4 Wag   6 110
# 3:         Datsun 710   4  93
# 4:     Hornet 4 Drive   6 110
# 5: Hornet Sportabout   8 175
```

The `j` do select-list

```
R> # same as previous two
R> dt.mtcars[1:5,    c(1, 3, 5) ]
#                    rn cyl  hp
# 1:          Mazda RX4   6 110
# 2:      Mazda RX4 Wag   6 110
# 3:         Datsun 710   4  93
# 4:     Hornet 4 Drive   6 110
# 5: Hornet Sportabout   8 175
```

The `j` do select-list

```
R> # select columns rn thru cyl
R> # (and .SD is "sub-data" of current selection)
R> dt.mtcars[1:5 , .SD, .SDcols = rn:cyl ]
#                  rn  mpg cyl
# 1:          Mazda RX4 21.0   6
# 2:      Mazda RX4 Wag 21.0   6
# 3:         Datsun 710 22.8   4
# 4:     Hornet 4 Drive 21.4   6
# 5: Hornet Sportabout 18.7   8
```

The `j` do select-list

```
R> # use a variable column name
R> # together with .. evaluation in parent
R> variable.col.name <- 'rn'
R> dt.mtcars[1:5, ..variable.col.name ]
#                    rn
# 1:          Mazda RX4
# 2:      Mazda RX4 Wag
# 3:         Datsun 710
# 4:     Hornet 4 Drive
# 5: Hornet Sportabout
```

This is more advanced use outside of core course content.

Grouping by **by=**

```
R> # group by  for cond. mean
R> dt.mtcars[, .( mean (mpg) ), by = cyl ]
#    cyl      V1
# 1:   6 19.743
# 2:   4 26.664
# 3:   8 15.100
```

Grouping by **by=**

```
R> # group by, output variable named mpg
R> dt.mtcars[, .( mpg = mean (mpg) ), by = cyl ]
#    cyl    mpg
# 1:   6 19.743
# 2:   4 26.664
# 3:   8 15.100
```

Grouping by **by=**

```
R> # group  by 'cyl'
R> # select at cols 'mpg' to 'wt'
R> # and sweep mean() function over them
R> dt.mtcars[, lapply ( .SD, mean )
+             , .SDcols = mpg:wt
+             , by = cyl ]
#    cyl    mpg cyl   disp      hp   drat     wt
# 1:   6 19.743   6 183.31 122.286 3.5857 3.1171
# 2:   4 26.664   4 105.14  82.636 4.0709 2.2857
# 3:   8 15.100   8 353.10 209.214 3.2293 3.9992
```

chaining - having

```
R> # having, aggregate then filter
R> # the aggregation via chaining
R> dt.mtcars[, .(mpg = mean(mpg)), by=cyl][mpg > 16]
#    cyl    mpg
# 1:   6 19.743
# 2:   4 26.664
```

chaining - order by

```
R> # order by
R> dt.mtcars[, .(mpg = mean(mpg)), by=cyl][order(-mpg)]
#    cyl    mpg
# 1:   4 26.664
# 2:   6 19.743
# 3:   8 15.100
```

data.table and vectors

```
R> 1:2
# [1] 1 2
R> 1:6
# [1] 1 2 3 4 5 6
R> 1:2 %in% 1:6
# [1] TRUE TRUE
R> 1:6 %in% 1:2
# [1]  TRUE  TRUE FALSE FALSE FALSE FALSE
```

data.table and vectors

```
R> dt.mtcars [ , cyl ]
#  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8
R> dt.mtcars [ , cyl ] %in% c(4,6)
#  [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  T
# [12] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  T
# [23] FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FA
```

data.table and vectors

```
R> dt.mtcars[cyl %in% c(4,6)] [1:5] [order(cyl)]
#                rn  mpg cyl disp  hp drat    wt
# 1:     Datsun 710 22.8   4  108  93 3.85 2.320
# 2:      Mazda RX4 21.0   6  160 110 3.90 2.620
# 3:  Mazda RX4 Wag 21.0   6  160 110 3.90 2.875
# 4: Hornet 4 Drive 21.4   6  258 110 3.08 3.215
# 5:       Valiant 18.1   6  225 105 2.76 3.460
```

joins

```
R> # create a new aggregated data.table
R> dt.mtcars.cyl.aggr <-
+     dt.mtcars[, .(mpg.mean.cyl = mean(mpg)
+                 , mpg.sd.cyl   = sd(mpg)
+                 , hp.mean.cyl  = mean(hp)
+                 , hp.sd.cyl    = sd(hp))
+               ,by = cyl]
R> dt.mtcars.cyl.aggr
#    cyl mpg.mean.cyl mpg.sd.cyl hp.mean.cyl hp.sd.cyl
# 1:   6       19.743     1.4536     122.286    24.260
# 2:   4       26.664     4.5098      82.636    20.935
# 3:   8       15.100     2.5600     209.214    50.977
```

joins

```
R> # sort / key dt.mtcars by cyl
R> setkeyv(dt.mtcars,c('cyl'))
R> # sort / key dt.mtcars.cyl.aggr by cyl
R> setkeyv(dt.mtcars.cyl.aggr,c('cyl'))
R> # joing -- which defaults to using key-ed cols
R> DT <- dt.mtcars [ dt.mtcars.cyl.aggr ]
R> DT[ 1:2 ]
#            rn  mpg cyl  disp hp drat   wt
# 1: Datsun 710 22.8   4 108.0 93 3.85 2.32
# 2:  Merc 240D 24.4   4 146.7 62 3.69 3.19
#    mpg.mean.cyl mpg.sd.cyl hp.mean.cyl hp.sd.cyl
# 1:       26.664     4.5098      82.636    20.935
# 2:       26.664     4.5098      82.636    20.935
```

update

```
R> dt.mtcars <- data.table(mtcars, keep.rownames = TRUE)
R> # Add new column N, value is always 1
R> dt.mtcars [, N := 1]
R> dt.mtcars [1:2]
#                rn mpg cyl disp  hp drat    wt  qsec
# 1:      Mazda RX4  21   6  160 110  3.9 2.620 16.46
# 2: Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02
#    vs am gear carb N
# 1:  0  1    4    4 1
# 2:  0  1    4    4 1
```

update

```
R> # create a manufacture vector with the first word in rn
R> v.manuf <- gsub("([A-Za-z]+).*", "\\1", dt.mtcars [ , rn ] )
R> # add new variable manufacture
R> dt.mtcars [ , manufacturer := v.manuf ]
R> dt.mtcars [ 1:2 ]
#                rn mpg cyl disp  hp drat    wt  qsec
# 1:     Mazda RX4  21   6  160 110  3.9 2.620 16.46
# 2: Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02
#    vs am gear carb N manufacturer
# 1:  0  1    4    4 1        Mazda
# 2:  0  1    4    4 1        Mazda
```

update

```
R> # create is.merc indicator
R> dt.mtcars [ manufacturer == 'Merc', is.merc := 1 ]
R> # .N counts rows in current sub-data, here groups
R> dt.mtcars [ , .N, by = is.merc ]
#    is.merc  N
# 1:      NA 25
# 2:       1  7
```

data.table & plot

```
R> library(ggplot2)
R> plot.All.XY.by.Z <- function (dt, x, y, z) {
+    #  numerics only
+    dt[, (y):= lapply(.SD, function(x) {
+              as.numeric(as.character(x)) }),
+       .SDcols = y]
+    dts <- melt(dt, id = c(x,z), measure = y)
+    p <- ggplot(dts, aes_string(x = colnames(dt)[x],
+                                y = "value",
+                                colours = colnames(dt)[z])) +
+        geom_line() +
+        facet_wrap(~ variable)
+    print (p)
+ }
```

data.table & plot