**Introduction**

Data manipulation and aggregation is one of the classic tasks anyone working with data will come across. We of course can perform data transformation and aggregation [with base R](http://jozefhajnala.gitlab.io/r/categories/rcase4base/), but when speed and memory efficiency come into play, data.table is my package of choice.

In this post we will look at of the fresh and very useful functionality that came to data.table only last year – grouping sets, enabling us, for example, to create pivot table-like reports with sub-totals and grand total quickly and easily.

**Contents**

1. [Basic by-group summaries with data.table](https://jozefhajnala.gitlab.io/r/r912-datatable-grouping-sets/#basic-by-group-summaries-with-data.table)
2. [Quick pivot tables with subtotals and a grand total](https://jozefhajnala.gitlab.io/r/r912-datatable-grouping-sets/#quick-pivot-tables-with-subtotals-and-a-grand-total)
3. [Custom grouping sets](https://jozefhajnala.gitlab.io/r/r912-datatable-grouping-sets/#custom-grouping-sets)
4. [Cube and rollup as special cases of grouping sets](https://jozefhajnala.gitlab.io/r/r912-datatable-grouping-sets/#cube-and-rollup-as-special-cases-of-grouping-sets)
5. [References](https://jozefhajnala.gitlab.io/r/r912-datatable-grouping-sets/#references)

**Basic by-group summaries with data.table**

To showcase the functionality, we will use a very slightly modified dataset provided by Hadley Wickham’s [nycflights13](https://cran.r-project.org/package=nycflights13) package, mainly the flights data frame. Lets prepare a small dataset suitable for the showcase:

library(data.table)

dataurl <- "https://jozefhajnala.gitlab.io/r/post/data/"

flights <- readRDS(url(paste0(dataurl, "r006/flights.rds")))

flights <- as.data.table(flights)[month < 3]

Now, for those unfamiliar with data table, to create a summary of distances flown per month and originating airport with data.table, we could simply use:

flights[, sum(distance), by = c("month", "origin")]

## month origin V1

## 1: 1 EWR 9524521

## 2: 1 LGA 6359510

## 3: 1 JFK 11304774

## 4: 2 EWR 8725657

## 5: 2 LGA 5917983

## 6: 2 JFK 10331869

To also name the new column nicely, say distance instead of the default V1:

flights[, .(distance = sum(distance)), by = c("month", "origin")]

## month origin distance

## 1: 1 EWR 9524521

## 2: 1 LGA 6359510

## 3: 1 JFK 11304774

## 4: 2 EWR 8725657

## 5: 2 LGA 5917983

## 6: 2 JFK 10331869

For more on basic data.table operations, look at the [Introduction to data.table](https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html) vignette.

As you have probably noticed, the above gave us the sums of distances by months and origins. When creating reports, especially readers coming from Excel may expect 2 extra perks

* Looking at sub-totals and grand total
* Seeing the data in wide format

Since the wide format is just a reshape and data table has the [dcast()](https://www.rdocumentation.org/packages/data.table/versions/1.12.0/topics/dcast.data.table?tap_a=5644-dce66f&tap_s=10907-287229) function for that for quite a while now, we will only briefly show it in practice. The focus of this post will be on the new functionality that was only released in [data.table v1.11](https://github.com/Rdatatable/data.table/blob/master/NEWS.md#changes-in-v1110--01-may-2018) in May last year – creating the grand- and sub-totals.

**Quick pivot tables with subtotals and a grand total**

To create a “classic” pivot table as known from Excel, we need to aggregate the data and also compute the subtotals for all combinations of the selected dimensions and a grand total. In comes cube(), the function that will do just that:

# Get subtotals for origin, month and month&origin with `cube()`:

cubed <- data.table::cube(

flights,

.(distance = sum(distance)),

by = c("month", "origin")

)

cubed

## month origin distance

## 1: 1 EWR 9524521

## 2: 1 LGA 6359510

## 3: 1 JFK 11304774

## 4: 2 EWR 8725657

## 5: 2 LGA 5917983

## 6: 2 JFK 10331869

## 7: 1 27188805

## 8: 2 24975509

## 9: NA EWR 18250178

## 10: NA LGA 12277493

## 11: NA JFK 21636643

## 12: NA 52164314

As we can see, compared to the simple group by summary we did earlier, we have extra rows in the output

1. Rows 7,8 with months 1,2 and origin , – these are the subtotals per month across all origins
2. Rows 9,10,11 with months NA, NA, NA and origins EWR, LGA, JFK – these are the subtotals per origin across all months
3. Row 12 with NA month and origin – this is the Grand total across all origins and months

All that is left to get a familiar pivot table shape is to reshape the data to wide format with the aforementioned dcast() function:

# - Origins in columns, months in rows

data.table::dcast(cubed, month ~ origin, value.var = "distance")

## month EWR JFK LGA NA

## 1: 1 9524521 11304774 6359510 27188805

## 2: 2 8725657 10331869 5917983 24975509

## 3: NA 18250178 21636643 12277493 52164314

# - Origins in rows, months in columns

data.table::dcast(cubed, origin ~ month, value.var = "distance")

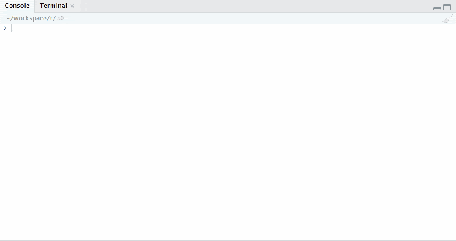
## origin 1 2 NA

## 1: EWR 9524521 8725657 18250178

## 2: JFK 11304774 10331869 21636643

## 3: LGA 6359510 5917983 12277493

## 4: 27188805 24975509 52164314



Pivot table with data.table

**Using more dimensions**

We can use the same approach to create summaries with more than two dimensions, for example, apart from months and origins, we can also look at carriers, simply by adding "carrier" into the by argument:

# With 3 dimensions:

cubed2 <- cube(

flights,

.(distance = sum(distance)),

by = c("month", "origin", "carrier")

)

cubed2

## month origin carrier distance

## 1: 1 EWR UA 5084378

## 2: 1 LGA UA 729667

## 3: 1 JFK AA 2013434

## 4: 1 JFK B6 3672655

## 5: 1 LGA DL 1678965

## ---

## 153: NA F9 174960

## 154: NA HA 293997

## 155: NA YV 21526

## 156: NA OO 733

## 157: NA 52164314

And dcast() to wide format which suits our needs best:

# For example, with month and carrier in rows, origins in columns:

dcast(cubed2, month + carrier ~ origin, value.var = "distance")

## month carrier EWR JFK LGA NA

## 1: 1 9E 46125 666109 37071 749305

## 2: 1 AA 415707 2013434 1344045 3773186

## 3: 1 AS 148924 NA NA 148924

## 4: 1 B6 484431 3672655 542748 4699834

## 5: 1 DL 245277 2578999 1678965 4503241

## 6: 1 EV 2067900 24624 86309 2178833

## 7: 1 F9 NA NA 95580 95580

## 8: 1 FL NA NA 226658 226658

## 9: 1 HA NA 154473 NA 154473

## 10: 1 MQ 152428 223510 908715 1284653

## 11: 1 OO NA NA 733 733

## 12: 1 UA 5084378 963144 729667 6777189

## 13: 1 US 339595 219387 299838 858820

## 14: 1 VX NA 788439 NA 788439

## 15: 1 WN 539756 NA 398647 938403

## 16: 1 YV NA NA 10534 10534

## 17: 1 9524521 11304774 6359510 27188805

## 18: 2 9E 42581 605085 34990 682656

## 19: 2 AA 373884 1817048 1207701 3398633

## 20: 2 AS 134512 NA NA 134512

## 21: 2 B6 456151 3390047 490224 4336422

## 22: 2 DL 219998 2384048 1621728 4225774

## 23: 2 EV 1872395 24168 112863 2009426

## 24: 2 F9 NA NA 79380 79380

## 25: 2 FL NA NA 204536 204536

## 26: 2 HA NA 139524 NA 139524

## 27: 2 MQ 140924 201880 812152 1154956

## 28: 2 UA 4686122 871824 681737 6239683

## 29: 2 US 301832 222720 293736 818288

## 30: 2 VX NA 675525 NA 675525

## 31: 2 WN 497258 NA 367944 865202

## 32: 2 YV NA NA 10992 10992

## 33: 2 8725657 10331869 5917983 24975509

## 34: NA 9E 88706 1271194 72061 1431961

## 35: NA AA 789591 3830482 2551746 7171819

## 36: NA AS 283436 NA NA 283436

## 37: NA B6 940582 7062702 1032972 9036256

## 38: NA DL 465275 4963047 3300693 8729015

## 39: NA EV 3940295 48792 199172 4188259

## 40: NA F9 NA NA 174960 174960

## 41: NA FL NA NA 431194 431194

## 42: NA HA NA 293997 NA 293997

## 43: NA MQ 293352 425390 1720867 2439609

## 44: NA OO NA NA 733 733

## 45: NA UA 9770500 1834968 1411404 13016872

## 46: NA US 641427 442107 593574 1677108

## 47: NA VX NA 1463964 NA 1463964

## 48: NA WN 1037014 NA 766591 1803605

## 49: NA YV NA NA 21526 21526

## 50: NA 18250178 21636643 12277493 52164314

## month carrier EWR JFK LGA NA

**Custom grouping sets**

So far we have focused on the “default” pivot table shapes with all sub-totals and a grand total, however the cube() function could be considered just a useful special case shortcut for a more generic concept – grouping sets. You can read more on grouping sets with [MS SQL Server](https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/bb522495(v%3Dsql.105)) or with [PostgreSQL](https://www.postgresql.org/docs/devel/queries-table-expressions.html#QUERIES-GROUPING-SETS).

The groupingsets() function allows us to create sub-totals on arbitrary groups of dimensions. Custom subtotals are defined by the sets argument, a list of character vectors, each of them defining one subtotal. Now let us have a look at a few practical examples:

**Replicate a simple group by, without any subtotals or grand total**

For reference, to replicate a simple group by with grouping sets, we could use:

groupingsets(

flights,

j = .(distance = sum(distance)),

by = c("month", "origin", "carrier"),

sets = list(c("month", "origin", "carrier")),

)

Which would give the same results as

flights[, .(distance = sum(distance)), by = c("month", "origin", "carrier")]

**Custom subtotals**

To give only the subtotals for each of the dimensions:

groupingsets(

flights,

j = .(distance = sum(distance)),

by = c("month", "origin", "carrier"),

sets = list(

c("month"),

c("origin"),

c("carrier")

)

)

## month origin carrier distance

## 1: 1 27188805

## 2: 2 24975509

## 3: NA EWR 18250178

## 4: NA LGA 12277493

## 5: NA JFK 21636643

## 6: NA UA 13016872

## 7: NA AA 7171819

## 8: NA B6 9036256

## 9: NA DL 8729015

## 10: NA EV 4188259

## 11: NA MQ 2439609

## 12: NA US 1677108

## 13: NA WN 1803605

## 14: NA VX 1463964

## 15: NA FL 431194

## 16: NA AS 283436

## 17: NA 9E 1431961

## 18: NA F9 174960

## 19: NA HA 293997

## 20: NA YV 21526

## 21: NA OO 733

## month origin carrier distance

To give only the subtotals per combinations of 2 dimensions:

groupingsets(

flights,

j = .(distance = sum(distance)),

by = c("month", "origin", "carrier"),

sets = list(

c("month", "origin"),

c("month", "carrier"),

c("origin", "carrier")

)

)

## month origin carrier distance

## 1: 1 EWR 9524521

## 2: 1 LGA 6359510

## 3: 1 JFK 11304774

## 4: 2 EWR 8725657

## 5: 2 LGA 5917983

## 6: 2 JFK 10331869

## 7: 1 UA 6777189

## 8: 1 AA 3773186

## 9: 1 B6 4699834

## 10: 1 DL 4503241

## 11: 1 EV 2178833

## 12: 1 MQ 1284653

## 13: 1 US 858820

## 14: 1 WN 938403

## 15: 1 VX 788439

## 16: 1 FL 226658

## 17: 1 AS 148924

## 18: 1 9E 749305

## 19: 1 F9 95580

## 20: 1 HA 154473

## 21: 1 YV 10534

## 22: 1 OO 733

## 23: 2 US 818288

## 24: 2 UA 6239683

## 25: 2 B6 4336422

## 26: 2 AA 3398633

## 27: 2 EV 2009426

## 28: 2 FL 204536

## 29: 2 MQ 1154956

## 30: 2 DL 4225774

## 31: 2 WN 865202

## 32: 2 9E 682656

## 33: 2 VX 675525

## 34: 2 AS 134512

## 35: 2 F9 79380

## 36: 2 HA 139524

## 37: 2 YV 10992

## 38: NA EWR UA 9770500

## 39: NA LGA UA 1411404

## 40: NA JFK AA 3830482

## 41: NA JFK B6 7062702

## 42: NA LGA DL 3300693

## 43: NA EWR B6 940582

## 44: NA LGA EV 199172

## 45: NA LGA AA 2551746

## 46: NA JFK UA 1834968

## 47: NA LGA B6 1032972

## 48: NA LGA MQ 1720867

## 49: NA EWR AA 789591

## 50: NA JFK DL 4963047

## 51: NA EWR MQ 293352

## 52: NA EWR DL 465275

## 53: NA EWR US 641427

## 54: NA EWR EV 3940295

## 55: NA JFK US 442107

## 56: NA LGA WN 766591

## 57: NA JFK VX 1463964

## 58: NA LGA FL 431194

## 59: NA EWR AS 283436

## 60: NA LGA US 593574

## 61: NA JFK MQ 425390

## 62: NA JFK 9E 1271194

## 63: NA LGA F9 174960

## 64: NA EWR WN 1037014

## 65: NA JFK HA 293997

## 66: NA JFK EV 48792

## 67: NA EWR 9E 88706

## 68: NA LGA 9E 72061

## 69: NA LGA YV 21526

## 70: NA LGA OO 733

## month origin carrier distance

**Grand total**

To give only the grand total:

groupingsets(

flights,

j = .(distance = sum(distance)),

by = c("month", "origin", "carrier"),

sets = list(

character(0)

)

)

## month origin carrier distance

## 1: NA 52164314

**Cube and rollup as special cases of grouping sets**

**Implementation of cube**

We mentioned above that cube() can be considered just a shortcut to a useful special case of groupingsets(). And indeed, looking at the implementation of the data.table method data.table:::cube.data.table, most of what it does is to define the sets to represent the given vector and all of its possible subsets, and passes that to groupingsets():

function (x, j, by, .SDcols, id = FALSE, ...) {

if (!is.data.table(x))

stop("Argument 'x' must be a data.table object")

if (!is.character(by))

stop("Argument 'by' must be a character vector of column names used in grouping.")

if (!is.logical(id))

stop("Argument 'id' must be a logical scalar.")

n = length(by)

keepBool = sapply(2L^(seq\_len(n) - 1L), function(k) rep(c(FALSE,

TRUE), times = k, each = ((2L^n)/(2L \* k))))

sets = lapply((2L^n):1L, function(j) by[keepBool[j, ]])

jj = substitute(j)

groupingsets.data.table(x, by = by, sets = sets, .SDcols = .SDcols,

id = id, jj = jj)

}

This means for example that

cube(flights, sum(distance), by = c("month", "origin", "carrier"))

## month origin carrier V1

## 1: 1 EWR UA 5084378

## 2: 1 LGA UA 729667

## 3: 1 JFK AA 2013434

## 4: 1 JFK B6 3672655

## 5: 1 LGA DL 1678965

## ---

## 153: NA F9 174960

## 154: NA HA 293997

## 155: NA YV 21526

## 156: NA OO 733

## 157: NA 52164314

Is equivalent to

groupingsets(

flights,

j = .(distance = sum(distance)),

by = c("month", "origin", "carrier"),

sets = list(

c("month", "origin", "carrier"),

c("month", "origin"),

c("month", "carrier"),

c("month"),

c("origin", "carrier"),

c("origin"),

c("carrier"),

character(0)

)

)

## month origin carrier distance

## 1: 1 EWR UA 5084378

## 2: 1 LGA UA 729667

## 3: 1 JFK AA 2013434

## 4: 1 JFK B6 3672655

## 5: 1 LGA DL 1678965

## ---

## 153: NA F9 174960

## 154: NA HA 293997

## 155: NA YV 21526

## 156: NA OO 733

## 157: NA 52164314

**Implementation of rollup**

The same can be said about rollup(), another shortcut than can be useful. Instead of all possible subsets, it will create a list representing the vector passed to by and its subsets “from right to left”, including the empty vector to get a grand total. Looking at the implementation of the data.table method data.table::rollup.data.table:

function (x, j, by, .SDcols, id = FALSE, ...) {

if (!is.data.table(x))

stop("Argument 'x' must be a data.table object")

if (!is.character(by))

stop("Argument 'by' must be a character vector of column names used in grouping.")

if (!is.logical(id))

stop("Argument 'id' must be a logical scalar.")

sets = lapply(length(by):0L, function(i) by[0L:i])

jj = substitute(j)

groupingsets.data.table(x, by = by, sets = sets, .SDcols = .SDcols,

id = id, jj = jj)

}

For example, the following:

rollup(flights, sum(distance), by = c("month", "origin", "carrier"))

Is equivalent to

groupingsets(

flights,

j = .(distance = sum(distance)),

by = c("month", "origin", "carrier"),

sets = list(

c("month", "origin", "carrier"),

c("month", "origin"),

c("month"),

character(0)

)

)