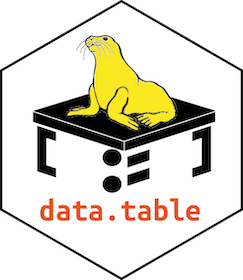
[johnmackintosh.net](https://johnmackintosh.net/blog/2019-06-30-datatable-by-a-dummy/)

**data.table by a dummy**

John MacKintosh

8-10 minutes

30 Jun 2019 on Rstats, Data.table



A dummies guide to the fabulous data.table package.

I like dplyr

I’ve used data.table in the past, though didn’t persevere with it enough. My happy safe space is dplyr & ggplot2. I love these packages and use them almost every time I use R. Every so often , I see mentions of cool data.table features and it has annoyed me that because I didn’t get the basics down, I could never progress to understand whether these features would help me. More recently, I came across some tremendously slow running code, and I was certain that data.table would have slashed the running time, but I didn’t know enough about it to fix it. I might be in a position to change that soon.

**I like data.table too** [**Permalink**](about:reader?url=https%3A%2F%2Fjohnmackintosh.net%2Fblog%2F2019-06-30-datatable-by-a-dummy%2F#i-like-datatable-too)

A week or so ago I was touting for blog post ideas and one that was suggested to me was ‘data.table for dummies’.

Thanks to Chris Billingham (@yobrenoops) for suggesting this.

This is nowhere near good enough, because I don’t know enough, so consider it ‘a bit about data.table, by a dummy’. It’s mainly a way for me to solidify what I’ve been refreshing my memory on over the last few days.

I originally thought about using the ‘flights’ dataset, because it is large, however, its used quite a lot already in data.table tutuorials.

So instead, I’ll convert some existing dplyr code that [I have used/demonstrated in the past](https://www.johnmackintosh.com/2017-12-21-flow/), to data.table.

Here is the dplyr code, which has been used and explained [elsewhere on the blog](https://www.johnmackintosh.com/2018-06-01-dplyr-for-the-win/). Basically, read in data from Excel, create a new variable to round down times to the nearest 15 minutes, and then create a counter variable which gives the total number of patients for each 15 minute segment, by 3 grouping criteria

data <- read\_xlsx("RedGreenGreyDots.xlsx", sheet = 1) #read raw data from Excel

plot\_data <- data %>%

mutate(Movement15 = lubridate::floor\_date(MovementDateTime,"15 minutes")) %>%

group\_by(IN\_OUT, Movement\_Type,Staging\_Post,Movement15) %>%

mutate(counter = case\_when (

IN\_OUT == 'IN' ~ 1,

IN\_OUT == 'OUT' ~ -1)) %>%

mutate(Movement\_15\_SEQNO =cumsum(counter)) %>%

ungroup()

# Change "Tranfer In" or "Transfer Out" to "Transfer"

plot\_data$Movement\_Type <- gsub("Transfer.\*","Transfer",x=plot\_data$Movement\_Type)

Here we go with data.table:

DT <- read\_xlsx("RedGreenGreyDots.xlsx", sheet = 1) #read raw data from Excel

#str(DT)

setDT(DT) # now a data.table, as well as data frame

#str(DT)

First, read the data in as before. When the data is read in, DT has the classes ‘tbl\_df’, ‘tbl’ and ‘data.frame’. We need to convert this to a data.table, using setDT. Once we do that, we can check that we now have a class of ‘data.table’ as well as ‘data.frame’.

Now we can start to create new columns.

For ease, I’ve kept each transformation on it’s own line - there are only 4.

DT[, Movement15 := (floor\_date(MovementDateTime,"15 minutes"))]

DT[,counter := (ifelse(IN\_OUT %chin% "IN",1L,-1L)), by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)]

DT[,Movement\_15\_SEQNO := cumsum(counter),by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)]

DT[Movement\_Type %like% ("Transfer\*"), Movement\_Type := "Transfer"][]

Data.table has the following syntax, where everything happens between the square brackets

This is supposed to be analogous to SQL (WHERE , SELECT, GROUP BY) for example. If you don’t have a SQL background, that might not be immediately meaningful, so we can also think of it as :

[filter or subset rows, select columns or create new variables , grouping conditions]

In truth, this is not remotely accurate, because there is so much more you can do in the ‘j’ part.

Hopefully, though, you can see that any filtering or predicates come before the first comma, you then ‘do stuff’ in the middle part, and list any grouping required in the last part.

If you don’t want to subset (in other words, whatever you are doing, will apply to every row) then you simply leave the ‘i’ part empty, so your code will look something like this:

Or, if you are grouping, it will look like this

DT[, do stuff, by = group\_colA]

And, if you have more than one grouping column, you might write this:

DT[, do stuff, by =.(group\_colA, group\_colB, group\_colC)]

Let’s go back to the first line:

DT[, Movement15 := (floor\_date(MovementDateTime,"15 minutes"))]

We’re not filtering here, so there is nothing before the first comma. We are creating a new column, so we use the ‘:=’ symbol, and then the expression, in this case using lubridate’s floor\_date function, to round times to the nearest 15 minutes.

Line 2:

DT[,counter := (ifelse(IN\_OUT %chin% "IN",1L,-1L)), by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)]

Again, no filtering or subsetting is happening, so there is nothing before the first comma.

We create a new ‘counter’ variable, this time with an ifelse statement, making use of data.table’s %chin% operator.

This is similar to %in%, which is familiar from dplyr. However %chin% is optimised for use with character vectors.

I could have written a test for equality , e.g.

ifelse(IN\_OUT == "IN",1L,-1L)

and I believe that ‘==’ is also optimised in data.table, but it seems sensible to make use of %chin%.

Note, that we also provide a set of grouping columns using ‘by’. I used the shorthand by =.(), which meant I could type the bare column names seperated by a comma inside the brackets.

I could also have written this:

DT[,counter := (ifelse(IN\_OUT %chin% "IN",1L,-1L)), by = c("IN\_OUT","Movement\_Type","Staging\_Post", "Movement15")]

but who wants to type loads of quotation marks? (Not me, I’d much rather fry my brain with tidy evaluation.)

Line 3 :

DT[,Movement\_15\_SEQNO := cumsum(counter),by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)]

Again, no filtering, just creating a new column, and applying the grouping criteria. Unlike dplyr, where grouping hangs around until you ungroup() again, you have to specify the grouping criteria for each line. Or at least, when I didn’t specify it for this line, things went horribly wrong ( my final results were incorrect).

Finally, line 4, and we do something in ‘i’ for the first time. For plotting purposes, I want to change ‘Transfer In’ or ‘Transfer Out’ to just ‘Transfer’ Here, we filter all the rows where the Movement Type is like ‘Transfer’ ( with the very handy %like% operator, and the wildcard) and then replace them all with ‘ Transfer’. There is no need to apply any grouping criteria here, so there is no ‘by’ section.

DT[Movement\_Type %like% ("Transfer\*"), Movement\_Type := "Transfer"][]

Those still awake will notice the [] at the end there. That simply allows us to return the final modified data.table.

Once more, without my blathering on (assume data already read in and DT assigned as a data.table):

DT[, Movement15 := (floor\_date(MovementDateTime,"15 minutes"))]

DT[,counter := (ifelse(IN\_OUT %chin% "IN",1L,-1L)), by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)]

DT[,Movement\_15\_SEQNO := cumsum(counter),by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)]

DT[Movement\_Type %like% ("Transfer\*"), Movement\_Type := "Transfer"][]

Pretty neat huh? You can also chain the steps, like this:

DT[, Movement15 := (floor\_date(MovementDateTime,"15 minutes"))

][,counter := (ifelse(IN\_OUT %chin% "IN",1L,-1L)), by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)

][,Movement\_15\_SEQNO := cumsum(counter),by = .(IN\_OUT,Movement\_Type,Staging\_Post, Movement15)

][Movement\_Type %like% ("Transfer\*"),Movement\_Type := "Transfer"][]

Note that the square brackets start and finish on separate lines, with the new line back to back with the previous closing bracket.

**Looking forwards** [**Permalink**](about:reader?url=https%3A%2F%2Fjohnmackintosh.net%2Fblog%2F2019-06-30-datatable-by-a-dummy%2F#looking-forwards)

There is so much more to data.table, and I’ve barely touched on any of it.

I still have a lot to learn, but crucially, my initial dabbling always left me with the idea that I needed to learn this. And the little refresher I’ve been through this week has certainly not dampened my enthusiasm, so perhaps this will be the first of a few data.table blog posts.