FAQs about the **data.table** package in R

Matthew Dowle

September 1, 2010

The first section, Beginner FAQs, is intended to be read in order from start to finish. It may be read before reading the Introduction vignette.

Contents

1 Beginner FAQs

1.1 Why does `DT[,5]` return `5` ?

Because, by default, unlike a `data.frame`, the 2nd argument is an *expression* which is evaluated within the scope of `DT`. 5 evaluates to 5. It is generally bad practice to refer to columns by number rather than name. If someone else comes along and reads your code later, they may have to hunt around to find out which column is number 5. Furthermore, if you or someone else changes the column ordering of `DT` higher up in your R program, you might get bugs if you forget to change all the places in your code which refer to column number 5.

Say column 5 is called "region", just do `DT[,region]` instead. Notice there are no quotes around the column name. This is what we mean by `j` being evaluated within the scope of the `data.table`. That scope consists of an environment where the column names are variables.

You can write *any* R expression in the `j` e.g. `DT[,colA*colB/2]`. Further, `j` may be a `list()` of many R expressions, including calls to any R package e.g. `DT[,fitdistr(d1-d1,"normal")]`.

Having said this, there are some circumstances where referring to a column by number is ok, such as a sequence of columns. In these situations just do `DT[,5:10,with=FALSE]` or `DT[,c(1,4,10),with=FALSE]`. See `?"[.data.table"` for an explanation of the `with` argument.

Note that `with()` has been a base function for a long time. That's one reason we say `data.table` builds upon base functionality. There is little new here really, `data.table` is just making use of `with()` and building it into the syntax.

1.2 Why does `DT[,"region"]` return `"region"`?

See answer to 1.1 above. Try `DT[,region]` instead. Or `DT[,"region",with=FALSE]`.

1.3 Why does `DT[,region]` return a vector? I'd like a 1-column `data.table`. There is no drop argument like I'm used to in `data.frame`.

Try `DT[,list(region)]` instead.

1.4 Why does DT[,x,y,z] not work? I wanted the 3 columns x,y and z.

The j expression is the 2nd argument. The correct way to do this is DT[,list(x,y,z)].

1.5 I assigned a variable mycol="x" but then DT[,mycol] returns "x". How do I get it to look up the column name contained in the mycol variable?

This is what we mean when we say the j expression 'sees' objects in the calling scope. The variable mycol does not exist as a column name of DT so R then looked in the calling scope and found mycol there, and returned its value "x". This is correct behaviour. Had mycol been a column name, then that column's data would have been returned. What you probably meant was DT[,mycol,with=FALSE], which will return the x column's data as you wanted. Alternatively, since a data.table is just a list, you can do DT[["x"]] or DT[[mycol]].

1.6 Ok but I don't know the expressions in advance. How do I programatically pass them in?

To create expressions use the quote() function. We refer to these as *quote()-ed* expressions to save confusion with the double quotes used to create a character vector such as c("x"). The simplest quote()-ed expression is just a column name on it's own :

```
q = quote(x)
DT[,eval(q)] # returns the column x as a vector
q = quote(list(x))
DT[,eval(q)] # returns the column x as a 1-column data.table
```

Since these are *expressions*, we are not restricted to column names only :

```
q = quote(mean(x))
DT[,eval(q)] # identical to DT[,mean(x)]
q = quote(list(x,sd(y),mean(y*z)))
DT[,eval(q)] # identical to DT[,list(x,sd(y),mean(y*z))]
```

If it's just simply a vector of column names it may be simpler to pass a character vector to j and use with=FALSE.

1.7 This is really hard. What's the point ?

j doesn't have to be just column names. You can put any R expression of column names directly as the j, e.g., DT[,mean(x*y/z)]. The same applies to i. You have been used to i being row numbers or row names only. It's nice to simply write DT[x>1000, sum(y*z)]. What does that mean? It just runs the j expression on the set of rows where the i expression is true. i can be any expression of column names that evaluates to logical. You don't even need to return data, e.g., DT[x>1000, plot(y,z)]. When we get to compound table joins we will see how i and j can themselves be other data.table queries. We are going to stretch i and j much further than this, but to get there we need you on board first with FAQs 1.1-1.6.

1.8 OK, I'm starting to see what data.table is about, but why didn't you enhance data.frame in R? Why does it have to be a new package?

As FAQ 1.1 highlights, j in [.data.table is fundamentally different from j in [.data.frame. Even something as simple as DF[,1] would break existing code in many packages and user code. This is by design, and we want it to work this way for more complicated syntax to work. There are other differences, too.

However, we *have* proposed enhancements to R wherever possible, and some of these have been accepted and included. We intend to continue attempts to contribute to R itself whenever an opportunity arises.

1.9 Why are the defaults the way they are? Why does it work the way it does?

The simple answer is because the author designed it for his own use, and he wanted it that way. He finds it a more natural, faster way to write code, which also executes more quickly.

1.10 Isn't this already done by `with()` and `subset()` in base?

Some of the features discussed so far are, yes. The package builds upon base functionality. It does the same sorts of things but with less code required, and executes many times faster if used correctly.

1.11 Why does `x[y]` just return the columns from x? Shouldn't it return the y columns too?

Good question. The thinking was that, more often than not, you don't actually want the columns from y which aren't in the key. By default, we try to keep things efficient. In general, we don't want to create memory for the union of things, only to select out a few columns from it in the end. So if you want the computer to do more work, then you need to tell it do more work. You can either do `cbind(y,x[y])` or `merge(x,y)`. There are many different ways to do the same thing in `data.table`. It's your choice to understand the differences and write good code. The other consideration is that `x[y]` is after all a subset of x. The "[" operator does mean subset. By default, we thought it was more consistent with base R for `x[y]` to just return the columns from x. However, it is now apparent that `x[y]` returning all columns from both tables would be useful, so an argument `inci` will be added. We mentioned the merge method for `data.table` too, but does `merge(x,y)` mean `x[y]` or `y[x]`? Those are different things. Again, it is your choice which syntax you prefer and find clearer.
Finally, although it appears as though `x[y]` does not return the columns in y, you can actually use the columns from y in the `j` expression. This is what we mean by *join inherited scope*. Why not just return the union of all the columns from x and y and then run expressions on that? It boils down to efficiency of code and what is quicker to program. When you write `x[y,foo*boo]`, `data.table` automatically inspects the `j` expression to see which columns it uses. It will only subset, or group, those columns only. Memory is only created for the columns the `j` uses. Let's say foo is in x, and boo is in y (along with 20 other columns in y). Isn't x[y,foo*boo] quicker to program and quicker to run than a merge step followed by another subset step ?

2 General syntax

2.1 How can I avoid writing a really long `j` expression? You've said I should use the column names, but I've got loads of them.

There is a special `.SD` object, which stands for Sub Data. The `j` expression can use column names as variables, as you know, but it can also use `.SD`, which refers to the sub `data.table` as a whole. So to sum up all your columns it's just `DT[,lapply(.SD,sum),by=grp]`. It might seem tricky, but it's fast to write and fast to run. Notice you don't have to create an anonymous function(). See the timing vignette and comparison to other methods. The `.SD` object is efficiently implemented internally and more efficient than passing an argument to a function. Please don't do this though : `DT[,.SD[,"sales",with=FALSE],by=grp]`. That 'works', but is very inefficient and inelegant. This is what was intended: `DT[,sum(x),by=grp]` and could be 100's of times faster if DT contains many columns. No `data.table` may contain a column called .SD, that's why it has a "." at the start as you are unlikely to really want a column called ".SD".

2.2 Why is the default for mult "first" ? If there are duplicates in the key, shouldn't it return them all by default?

Possibly, yes. The default might be changed to "all".

In versions up to v1.3, "all" was slower. Internally, "all" was implemented by joining using "first", then again from scratch using "last", after which a diff between them was performed to work out the span of the matches in `x` for each row in `i`. Most often we join to single rows, though, where "first","last" and "all" return the same result. We prefer maximum performance for the majority of situations so the default chosen was "first". If you are working with a non-unique key then you need to specify "all".

In v1.4 the binary search in C was changed to branch at the deepest level to find first and last. Note that branch will likely occur within the same final pages. So there should no longer be a speed disadvantage in defaulting mult to 'all'. The 'all' default would be simpler and easier for the user as this FAQ comes up quite often.

A future version of `data.table` may allow a distinction between a key and a unique key. The `mult` argument could then be defaulted more wisely. `data.table` would then need to add checks on insert and update to make sure a unique key is maintained. That would be one of the advantages to the user of specifying a unique key; i.e., `data.table` would make sure a duplicate could not be inserted.

Note that when `i` (or `i`'s key if it has one) has fewer columns than x's key, mult is automatically set to "all". This is why grouping by `i` works; e.g., `DT[J(id),mean(v)]` where `key(DT)` has 2 or more columns.

2.3 I'm using `c()` in the j and getting strange results.

This is a common source of confusion. In `data.frame` you are used to, for example:

```
> df = data.frame(x=1:3,y=4:6,z=7:9)
> df

  x y z
1 1 4 7
2 2 5 8
3 3 6 9

> df[,c("y","z")]

  y z
1 4 7
2 5 8
3 6 9
```

which returns the two columns. In `data.table` you know you can use the column names directly and might try :

```
> dt = data.table(df)
> dt[,c(y,z)]

[1] 4 5 6 7 8 9
```

but this returns one vector. Remember that the `j` expression is evaluated within the environment of `dt`, and `c()` returns a vector. If 2 or more columns are required, use list() instead:

```
> dt[,list(y,z)]

     y z
[1,] 4 7
[2,] 5 8
[3,] 6 9
```

`c()` can be useful in a `data.table` too, but its behaviour is different from that in a `data.frame`.

2.4 I have built up a complex table with many columns. I want to use it as a template for a new table; i.e., create a new table with no rows, but with the column types copied from my table. Can I do that easily?

Yes. If your complex table is called `DT`, try `DT[0]`.

2.5 Is a NULL `data.table` the same as `DT[0]`?

No, despite the print method indicating otherwise. Strictly speaking, it's not possible to have `is.null(data.table(NULL))` return FALSE. [Perhaps look at this again.]

2.6 Why has the `DT()` alias been removed?

`DT` was introduced originally as a wrapper for a list of `j` expressions. Since `DT` was an alias for `data.table`, this was a convenient way to take care of silent recycling in cases where each item of the `j` list evaluated to different lengths. The alias was one reason grouping was slow, though. As of v1.3, list() should be passed instead to the `j` argument. list() is a primitive and is much faster, especially when there are many groups. Internally, this was a nontrivial change. Vector recycling is done internally, along with several other speed enhancements for grouping. Some users have come to rely on the `DT` alias, though. If there is a lot of code that depends on `DT()`, you can easily create the alias yourself: `DT = function(...)  data.table(...)`

2.7 But my code uses `j=DT(...)` and it works. The previous FAQ says that `DT()` has been removed.

`[.data.table` inspects the `j` expression that is passed to it. If it finds that the expression starts with a call to `DT()` it automatically replaces it with a call to `list()`. This is to help existing users. Please don't use `j=data.table(...)` as that may be slow. Use `j=list(...)`.

2.8 What are the scoping rules for j expressions?

Think of the subset as an environment where all the column names are variables. When a variable is used in the `j` expression, it is looked for in the following order :

1. The scope of the subset, i.e., the column names.
2. The scope of the calling frame; e.g., the line that appears before the `data.table` query.
3. Exercise for reader: does it then ripple up or go straight to .GlobalEnv?
4. The global environment .GlobalEnv

This is *lexical scoping* as explained in R FAQ 3.3.1. The environment that the function was created is not relevant, though, because there is no function. No anonymous *function* is passed to the `j`. Instead, an anonymous *body* is passed to the `j`; for example,

```
> DT = data.table(x=rep(c("a","b"),c(2,3)),y=1:5)
> DT
     x y
[1,] a 1
[2,] a 2
[3,] b 3
[4,] b 4
[5,] b 5
> DT[,{z=sum(y);z+3},by=x]
     x V1
[1,] a  6
[2,] b 15
```

2.9 Can I trace the j expression as it runs through the groups?

Try something like this:

```
> DT[,{
+   cat("Objects:",paste(objects(),collapse=","),"\n")
+   cat("Trace: x=",as.character(x)," y=",y,"\n")
+   sum(y)
+ },by=x]

Objects: x,y
Trace: x= a a  y= 1 2
Objects: x,y
Trace: x= b b b  y= 3 4 5
     x V1
[1,] a  3
[2,] b 12
```

2.10 Inside each group, why is the group variable a long vector containing the same value repeated?

This is correct. We saw that in the previous FAQ. x was `"a"` repeated twice in the first group, and `"b"` repeated three times in the second group. When you group, think of the data being split up. Sometimes we want to use the value of the group in the expression, though. In that case, we just use the first value.

```
> DT[,list(g=1,h=2,i=3,j=4,repeatgroupname=x,sum(y)),by=x]     # not intended

     x g h i j repeatgroupname V6
[1,] a 1 2 3 4               a  3
[2,] a 1 2 3 4               a  3
[3,] b 1 2 3 4               b 12
[4,] b 1 2 3 4               b 12
[5,] b 1 2 3 4               b 12

> DT[,list(g=1,h=2,i=3,j=4,repeatgroupname=x[1],sum(y)),by=x]  # intended

     x g h i j repeatgroupname V6
[1,] a 1 2 3 4               a  3
[2,] b 1 2 3 4               b 12
```

In the first attempt, the aggregate `sum(y)` was recycled to match the length of `x`. Recycling can be useful, but wasn't intended here.

2.11 Only the first 10 rows are printed, how do I print more?

Try `print(DT,nrows=Inf)` to print all rows, or set nrows to the number of rows you require.

3 Questions relating to compute time

3.1 I have 20 columns in `data.table` x. Why is an expression of one column so quick?

Several reasons:

- Only that column is grouped, the other 19 are ignored because `data.table` inspects the j expression and realises it doesn't use the other columns.

- One memory allocation is made for the largest group only, then that memory is re-used for the other groups, there is very little garbage to collect.
- R is an in memory column store i.e. the columns are contiguous in RAM. Page fetches from RAM into L2 cache are minimised.

3.2 I don't have a key on a large table, but grouping is still really quick. Why is that?

`data.table` uses radix sorting. This is significantly faster than other sort algorithms. Radix is specifically for integers only, see `?base::sort.list(x,method="radix")`.

This is also one reason why `setkey()` is quick.

When no key is set, or we group in a different order from that of the key, we call it an *ad hoc by*.

3.3 Why is grouping by columns in the key faster than an ad hoc by?

Because each group is contiguous in RAM, thereby minimising page fetches, and memory can be copied in bulk rather than looping.

4 Error messages

4.1 `Could not find function "DT"`

See FAQ 2.6 and FAQ 2.7.

4.2 I have created a package that depends on `data.table`. It works fine in development but when I release it as a package I get `data.frame` errors.

Make sure you use `require(data.table)` rather than `library(data.table)`. The `require()` function is very similar to `libary()` but also adds `data.table` to the hidden `.Depends` vector and this makes your package *data.table aware*. This is important from v1.5 when `data.table` started to inherit from `data.frame`. Calls to `[.data.table` from non `data.table` aware packages, such as base and ggplot2, are redirected to `[.data.frame` by `[.data.table`. When those packages use the usual `[.data.frame` syntax (e.g. `x[,5]`) when x is a `data.table`, x appears to behave as a `data.frame` would. You have likely been developing and testing your package in .GlobalEnv which is always considered `data.table` aware by `data.table`.

4.3 Why does `[.data.table` now have a `drop` argument from v1.5?

So that `data.table` can inherit from `data.frame` without using `....` If we used `...` then invalid argument names would not be caught and the convenience of test 147 would be lost.

The drop argument is never used by `[.data.table`; it is a placeholder for non `data.table` aware packages when they treat the `data.table` as if it were a `data.frame`. See previous FAQ.

5 General questions about the package

5.1 v1.3 appears to be missing from the CRAN archive ?

That is correct. v1.3 was available on R-Forge only. There were several large changes internally and these took some time to test in development.

5.2 Is `data.table` compatible with S-plus ?

Not currently.

- A few core parts of the package are written in C and use internal R functions and R structures.
- The package uses lexical scoping which is one of the differences between R and S-plus explained by R FAQ 3.3.1.

5.3 Is it available for Linux, Mac and Windows?

Yes, for both 32-bit and 64-bit on all platforms. Thanks to CRAN and R-Forge. There are no special or OS-specific libraries used.

5.4 I think it's great. What can I do?

Please send suggestions, bug reports and enhancement requests to datatable-help@lists.r-forge.r-project.org. This helps make the package better. The list is public and archived.

You can vote for packages at `http://crantastic.org/`. This helps encourage the developers. If you have time to write a comment too, that helps others in the community; e.g., some users have mentioned the types of data they use `data.table` to analyse.

You can join the project and change the code and/or documentation yourself.

5.5 I think it's not great. How do I warn others about my experience?

Please put your vote and comments on `http://crantastic.org/`. Make it constructive, so we have a chance to improve.

5.6 I have a question. I know the posting guide tells me to contact the maintainer, but that's just one person. Isn't there a group of people I can ask ?

Yes. You can post to datatable-help@lists.r-forge.r-project.org. It's like r-help, but just for this package. Feel free to answer questions there, too.

5.7 Where are the datatable-help archives?

`http://lists.r-forge.r-project.org/pipermail/datatable-help/`

5.8 I'd prefer not to contact datatable-help, can I mail just one or two people privately?

Sure. You're more likely to get a faster answer from datatable-help, though.

5.9 Why is this FAQ a pdf? Can we have the FAQ on a website?

This FAQ is a vignette written using Sweave. The benefits of Sweave include the following:

- We include R code in the answers. This is actually run when the file is created, not copy and pasted.
- This document is reproducible. Grab the .Rnw and you can run it yourself.
- This file is built every night on R-Forge so it's another way we check the package.
- This file is bound into each version of the package. The package is not accepted on CRAN unless this file passes checks. Each version of the package will have its own FAQ file which will be relevant for that version. Contrast this to a single website, which can be ambiguous if the answer depends on the version.

- You can open it offline, from your R prompt using vignette().
- You can extract the code from the document and play with it using `edit(vignette("datatable-timings"))`.
- It prints out easily.
- It's quicker and easier for us to write and maintain the FAQ in .Rnw form.