…getting started multiple bare variable names in data.table functions –

Flexible functions in data.table

I’m getting slightly more experienced with data.table, and I really like it.

My learning method was to get pretty deep for a month, reading everything I could
and replicating my dplyr code in data.table.

I then stopped using it for a month, and carried on with dplyr.

Then I tried switching back to data.table again. Some of it stuck, some of it didn’t, but I persevered.
I’m still struggling with joining tables, (for some reason the default right-joins
really throw my mental model), but I really enjoy working with it, and I know there is a lot more for me to learn.

When in use interactively, there are some nice little shortcuts that allow you to explore a dataset reasonably quickly, and I have been able to create some little helper functions without too much effort.

However, I am passing in column names wrapped in quotes, which shouldn’t really be a big deal, but working with dplyr for so long has spoiled me.

So this post is a way to note some potential ways round it.

N.B. not a data.table expert, some of this is probably horrendous, use the comments below / reach out otherwise and educate me.
It will be appreciated.

Let’s get set up with the flights dataset:

```
library(nycflights13)
library(data.table)
data(flights) # bring flights into the environment
setDT(flights)
```

Normal use and a brief .SD explainer

```
flights[,head(.SD,5), .SDcols = 'dep_delay']
```

```
##    dep_delay
## 1:         2
## 2:         4
## 3:         2
## 4:        -1
## 5:        -6
```

This does nothing earth shattering, just grabbing the first few rows from the ‘dep_delay’ column.
.SD means to take a subset of the data , and I specify the columns with .SDcols (note, not .SDCols as my brain
seems to want to type)

You can of course pass in multiple column names like this:

```
flights[,head(.SD,5), .SDcols = c('dep_delay','carrier','sched_dep_time')]
```

```
##    dep_delay carrier sched_dep_time
## 1:         2      UA            515
## 2:         4      UA            529
## 3:         2      AA            540
## 4:        -1      B6            545
## 5:        -6      DL            600
```

Or you can do this:

```
columns_of_interest <-  c('dep_delay','carrier','sched_dep_time')
flights[,head(.SD,5), .SDcols = columns_of_interest]
```

```
##    dep_delay carrier sched_dep_time
## 1:         2      UA            515
## 2:         4      UA            529
## 3:         2      AA            540
## 4:        -1      B6            545
## 5:        -6      DL            600
```

Single column functions – quoted column names

Of course we don’t want to have to do this repeatedly so we can create a function.

Here is a simple one, which will return unique values for a column of our choosing.
There are a few ways we can do this by passing in a quoted column name:

```
unique_dots <- function(DT,target_col) {

  vec <- unique(DT[,..target_col])

  vec

}
```

See the two dots before ‘target_col’ in the function body. That’s the magic right there. Don’t believe me?

```
unique_dots(flights, 'dep_delay')
```

```
##      dep_delay
##   1:         2
##   2:         4
##   3:        -1
##   4:        -6
##   5:        -4
##  ---
## 524:       358
## 525:       602
## 526:       593
## 527:      1014
## 528:       422
```

```
unique_dots(flights,'sched_dep_time')
```

```
##       sched_dep_time
##    1:            515
##    2:            529
##    3:            540
##    4:            545
##    5:            600
##   ---
## 1017:           1058
## 1018:            516
## 1019:           2153
## 1020:           2246
## 1021:           2208
```

```r
unique_dots(flights,'carrier')
```

```
##     carrier
##  1:      UA
##  2:      AA
##  3:      B6
##  4:      DL
##  5:      EV
##  6:      MQ
##  7:      US
##  8:      WN
##  9:      VX
## 10:      FL
## 11:      AS
## 12:      9E
## 13:      F9
## 14:      HA
## 15:      YV
## 16:      OO
```

Cool, we have a function that works.

But wait, we can also do this:

```r
# using with = FALSE

unique_with <- function(DT,target_col) {


  vec <- unique(DT[,target_col, with = FALSE])
  vec
}
```

```r
unique_with(flights, 'dep_delay')
```

```
##      dep_delay
##   1:         2
##   2:         4
##   3:        -1
##   4:        -6
##   5:        -4
##  ---
## 524:       358
## 525:       602
## 526:       593
## 527:      1014
## 528:       422
```

```r
unique_with(flights,'sched_dep_time')
```

```
##       sched_dep_time
##    1:            515
##    2:            529
##    3:            540
##    4:            545
##    5:            600
##   ---
## 1017:           1058
## 1018:            516
## 1019:           2153
## 1020:           2246
## 1021:           2208
```

```
unique_with(flights,'carrier')
```

```
##     carrier
##  1:      UA
##  2:      AA
##  3:      B6
##  4:      DL
##  5:      EV
##  6:      MQ
##  7:      US
##  8:      WN
##  9:      VX
## 10:      FL
## 11:      AS
## 12:      9E
## 13:      F9
## 14:      HA
## 15:      YV
## 16:      OO
```

And a cursory check that the results are the same for both functions :

```
all.equal(unique_dots(flights, 'dep_delay'),
          unique_with(flights,'dep_delay'))
```

```
## [1] TRUE
```

Well, that all seems marvellous.

But wait, there's even more. We can pass in a quoted column name and use 'get'.
Note, I wrapped the call to get in brackets to return a data.table, rather than a vector.

```
unique_get <- function(DT, target_col){
  vec <- unique(DT[,.(get(target_col))]) # ugly but returns a DT
  vec
}
```

A marginally less horrible way would be this, which returns a vector:

```
unique_get2 <- function(DT, target_col){
     vec <- unique(DT[,get(target_col)])
    vec
 }
```

Anyway, despite the hideousness, it still works

```
unique_get(flights, 'dep_delay')
```

```
##        V1
##   1:    2
##   2:    4
##   3:   -1
##   4:   -6
##   5:   -4
##  ---
## 524:  358
## 525:  602
## 526:  593
## 527: 1014
## 528:  422
```

```
unique_get(flights,'sched_dep_time')
```

```
##         V1
##    1:  515
##    2:  529
##    3:  540
##    4:  545
##    5:  600
##   ---
## 1017: 1058
## 1018:  516
## 1019: 2153
## 1020: 2246
## 1021: 2208
```

```
unique_get(flights,'carrier')
```

```
##     V1
##  1: UA
##  2: AA
##  3: B6
##  4: DL
##  5: EV
##  6: MQ
##  7: US
##  8: WN
##  9: VX
## 10: FL
## 11: AS
## 12: 9E
## 13: F9
## 14: HA
## 15: YV
## 16: OO
```

Enough of this. Give me multiple unquoted column names

No, I will not do that. Instead, have a function that takes a single unquoted column name

```
bare_col <- function(dt,n,target_col) {

  target_col <- deparse(substitute(target_col))

  dt[,head(.SD,n), .SDcols = target_col]
}
```

If you are thinking, “Dude, this is standard base R stuff” then yes, you are correct. Which is kind of the point.. Does it work? Oh yes..

```
bare_col(flights,5, dep_delay)
```

```
##    dep_delay
## 1:         2
## 2:         4
## 3:         2
## 4:        -1
## 5:        -6
```

```
bare_col(flights, 20, origin)
```

```
##     origin
##  1:    EWR
##  2:    LGA
##  3:    JFK
##  4:    JFK
##  5:    LGA
##  6:    EWR
##  7:    EWR
##  8:    LGA
##  9:    JFK
## 10:    LGA
## 11:    JFK
## 12:    JFK
## 13:    JFK
## 14:    EWR
## 15:    LGA
## 16:    JFK
## 17:    EWR
## 18:    LGA
## 19:    LGA
## 20:    EWR
```

I literally hate you. Give me multiple unquoted columns now..

Well, seeing as you asked nicely.. As a reminder, we can do this kind of thing *with quotes*

```
flights[,head(.SD,10), .SDcols = c('origin','distance','tailnum')]
```

```
##     origin distance tailnum
##  1:    EWR     1400  N14228
##  2:    LGA     1416  N24211
##  3:    JFK     1089  N619AA
##  4:    JFK     1576  N804JB
##  5:    LGA      762  N668DN
##  6:    EWR      719  N39463
##  7:    EWR     1065  N516JB
##  8:    LGA      229  N829AS
##  9:    JFK      944  N593JB
## 10:    LGA      733  N3ALAA
```

And we can do this..

```
getcols <- function(dt,n, ...) {

  sdcols <- eval(substitute(alist(...)))
  sdcols <- sapply(as.list(sdcols), deparse)
  dt[,head(.SD,n),.SDcols = sdcols]
}
```

And look – no quotes necessary :

```
getcols(flights, 10, origin, distance , tailnum)
```

```
##     origin distance tailnum
##  1:    EWR     1400  N14228
##  2:    LGA     1416  N24211
##  3:    JFK     1089  N619AA
##  4:    JFK     1576  N804JB
##  5:    LGA      762  N668DN
##  6:    EWR      719  N39463
##  7:    EWR     1065  N516JB
##  8:    LGA      229  N829AS
##  9:    JFK      944  N593JB
## 10:    LGA      733  N3ALAA
```

```
getcols(flights, 20, dep_time, sched_dep_time, carrier)
```

```
##     dep_time sched_dep_time carrier
##  1:      517            515      UA
##  2:      533            529      UA
##  3:      542            540      AA
##  4:      544            545      B6
##  5:      554            600      DL
##  6:      554            558      UA
##  7:      555            600      B6
##  8:      557            600      EV
##  9:      557            600      B6
## 10:      558            600      AA
## 11:      558            600      B6
## 12:      558            600      B6
## 13:      558            600      UA
## 14:      558            600      UA
## 15:      559            600      AA
## 16:      559            559      B6
## 17:      559            600      UA
## 18:      600            600      B6
## 19:      600            600      MQ
## 20:      601            600      B6
```

2020-01-20-boom.gif

This also works :

```
getcols2 <- function(dt,n, ...) {

  sdcols <- eval(substitute(alist(...)))
  sdcols <- sapply(sdcols, deparse)
  dt[,head(.SD,n),.SDcols = sdcols]
}
```

```
getcols2(flights, 10, origin, distance , tailnum)
```

```
##     origin distance tailnum
##  1:    EWR     1400  N14228
##  2:    LGA     1416  N24211
##  3:    JFK     1089  N619AA
##  4:    JFK     1576  N804JB
##  5:    LGA      762  N668DN
##  6:    EWR      719  N39463
##  7:    EWR     1065  N516JB
##  8:    LGA      229  N829AS
##  9:    JFK      944  N593JB
## 10:    LGA      733  N3ALAA
```

Again, usual disclaimers apply. I’m not a data.table expert. Indeed I’m not even a full time R user, much to my general displeasure. Which is why I’m faffing about with this at midnight on a Sunday. Anyway, I digress… there are no doubt a load of better ways of doing this, but this will hopefully serve as a starter.. if you have better ways of creating a flexible function that will accept multiple unknown columns, don’t be shy in sharing them

Thanks