Dplyr 0.8.0 launched recently, which you probably already know, but just in case you missed it..

Two new functions have been catching my eye : group\_map and group\_split.

The aim of this post – take a first look at these and try and get a new blog post up on github before February is out.

Time now: 23:38(UK)

**What it says on the tin**

Load packages and create a function for demo purposes , basically grab the first 5 rows of whatever we throw at it:

library(nycflights13)

## Warning: package 'nycflights13' was built under R version 3.5.2

library(dplyr)

## Warning: package 'dplyr' was built under R version 3.5.2

##

## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':

##

## filter, lag

## The following objects are masked from 'package:base':

##

## intersect, setdiff, setequal, union

data(flights)

header <- function(x,n = 5){

head(x,n)

}

Let’s check the function works first of all:

## # A tibble: 5 x 19

## # Groups: origin [3]

## year month day dep\_time sched\_dep\_time dep\_delay arr\_time

##

## 1 2013 1 1 517 515 2 830

## 2 2013 1 1 533 529 4 850

## 3 2013 1 1 542 540 2 923

## 4 2013 1 1 544 545 -1 1004

## 5 2013 1 1 554 600 -6 812

## # ... with 12 more variables: sched\_arr\_time , arr\_delay ,

## # carrier , flight , tailnum , origin , dest ,

## # air\_time , distance , hour , minute ,

## # time\_hour

Good, that just returns 5 rows from the entire datset. Now let’s replace group\_by with group\_split, and group by origin

split\_test <- flights %>%

group\_by(origin)

Then we use group\_split to create individual tibbles for each group

group\_split(split\_test)

## [[1]]

## # A tibble: 120,835 x 19

## year month day dep\_time sched\_dep\_time dep\_delay arr\_time

##

## 1 2013 1 1 517 515 2 830

## 2 2013 1 1 554 558 -4 740

## 3 2013 1 1 555 600 -5 913

## 4 2013 1 1 558 600 -2 923

## 5 2013 1 1 559 600 -1 854

## 6 2013 1 1 601 600 1 844

## 7 2013 1 1 606 610 -4 858

## 8 2013 1 1 607 607 0 858

## 9 2013 1 1 608 600 8 807

## 10 2013 1 1 615 615 0 833

## # ... with 120,825 more rows, and 12 more variables: sched\_arr\_time ,

## # arr\_delay , carrier , flight , tailnum ,

## # origin , dest , air\_time , distance , hour ,

## # minute , time\_hour

##

## [[2]]

## # A tibble: 111,279 x 19

## year month day dep\_time sched\_dep\_time dep\_delay arr\_time

##

## 1 2013 1 1 542 540 2 923

## 2 2013 1 1 544 545 -1 1004

## 3 2013 1 1 557 600 -3 838

## 4 2013 1 1 558 600 -2 849

## 5 2013 1 1 558 600 -2 853

## 6 2013 1 1 558 600 -2 924

## 7 2013 1 1 559 559 0 702

## 8 2013 1 1 606 610 -4 837

## 9 2013 1 1 611 600 11 945

## 10 2013 1 1 613 610 3 925

## # ... with 111,269 more rows, and 12 more variables: sched\_arr\_time ,

## # arr\_delay , carrier , flight , tailnum ,

## # origin , dest , air\_time , distance , hour ,

## # minute , time\_hour

##

## [[3]]

## # A tibble: 104,662 x 19

## year month day dep\_time sched\_dep\_time dep\_delay arr\_time

##

## 1 2013 1 1 533 529 4 850

## 2 2013 1 1 554 600 -6 812

## 3 2013 1 1 557 600 -3 709

## 4 2013 1 1 558 600 -2 753

## 5 2013 1 1 559 600 -1 941

## 6 2013 1 1 600 600 0 851

## 7 2013 1 1 600 600 0 837

## 8 2013 1 1 602 610 -8 812

## 9 2013 1 1 602 605 -3 821

## 10 2013 1 1 623 610 13 920

## # ... with 104,652 more rows, and 12 more variables: sched\_arr\_time ,

## # arr\_delay , carrier , flight , tailnum ,

## # origin , dest , air\_time , distance , hour ,

## # minute , time\_hour

We can verify how the grouping was carried out by using group\_keys:

group\_keys(split\_test)

## # A tibble: 3 x 1

## origin

##

## 1 EWR

## 2 JFK

## 3 LGA

Now, let’s get rid of that, and try group\_map, again, by origin. We may as well use my highly pointless function. If all goes well, we will return a tibble of 15 rows

rm(split\_test)

test5 <- flights %>%

group\_by(origin) %>%

group\_map(~ header(.x))

The dplyr help tells us that ‘.x’ refers to the subset of rows for each group that is passed in. That returned 5 rows per origin, as expected.Let’s try again and obtain some more rows per group:

test10 <- flights %>%

group\_by(origin) %>%

group\_map(~ header(.x,10L))

Now we have 30 observations in our new tibble, so this all looks good. Can we split and map together?

#test\_group\_and\_map <- flights %>%

# group\_split(origin) %>%

#group\_map(~ first5(.x,20))

No we cannot. Group\_map works on grouped tibbles, and by splitting, we’re passing it a list. Instant fail. RTFM folks.

At this point, I can see group\_map being pretty useful, but I will need to do a bit more work on the use case for group\_split – at present, I’m not seeing what it offers over and above group\_by, for staying in a purely dplyr workflow.  
I understand if you want to throw some purrr into the mix, it becomes more useful.  
As always, feel free to chip in if you have any thoughts or answers..I will try to play around with this more as well.