Data Transformation with dplyr

Karen Chu Sam

2020-03-27T21:13:14-05:00

So it’s been about one month since I last published in the blog. I wanted to post a little bit earlier, but I also wanted to learn a little bit more in R before posting a new post. The coronavirus has kept me a little bit busy emotionally, but now I’ve learned to live with all the coronavirus news going around.

In the last month, I’ve been juggling between Statistical Inference from the Data Science specialization in *Coursera* and learning how to use the dplyr package by following *R for Data Science* (Wickham & Grolemund, 2016). The book is pretty good to learn about the tools for data science. Furthermore, the book provides a good amount of exercises to practice. I can strongly recommend doing the exercises because that’s how you internalize and digest the things learned.

In this post, I will explain about what I’ve learned with the dplyr package and how is the *Statistical Inference* course from *Coursera*.

# Dplyr

We will need the both dplyr and the nycflights13 package. The nycflights13 package contains information abouth flights departing from New York in 2013. We will use data tables from this package to perform data transformation with dplyr. The dlyr package contains a set of useful functions to perform the most common data transformation.

library(dplyr)

##   
## 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

library(nycflights13)

Note that the flights dataframe is a tibble. This means that….

dplyr package

dlyr basics: (i)filter(), (ii) arrange(), (iii) select(), (iv) mutate(), (v) summarize() and pipeline operator

### filter()

filter() allows you to filter rows based on their values. So let’s say you want to filter all flights that either departed or arrived on women’s international day, March 8th. filter(flights, month==3, day==8) You can also use logical operators in the functions. Let’s say you want to filter flights that departed with more than 1 hour delay. filter(flights, ( arr\_delay>=60)) x %in% select every row where x is one of the values in y.

### arrange()

arrange allows you to arrange the rows of the dataframe as you would like. So, let’s say you want to arrange the flights with a descending dep\_time, then you would have to use arrange(flights, desc(dep\_time)).

### select()

Let’s you select columns. So, you’re subsetting the dataframe and selecting only the variables you’re interested in.Say you want to select the tail number and the carrier of the flights dataframe. select(flights, tail\_num, carrier) Other options useful to know in select, when you want to select a couple of columns is select(flights, (year:day), -(carrier:air\_time), everything())

## mutate()

Mutate allows you to add new columns to the dataframe. You will always see the new variables or columnes at the end of the dataframe. Following the book, I’ll just add two columns. mutate(flights, gain=arr\_delay - dep\_delay, speed=distance/air\_time\*60)

## summarize()

let’s say I want to know the average arrival delay of the flights. summarize(flights, delay=mean(arr\_delay, na.rm=TRUE)) This results in the mean of the whole column arr\_delay. How about if I want to know the average arrival delay per day? I’ll have to use the function group\_by.

by\_day <- group\_by(flights, year, month, day)  
summarize(by\_day, delay=mean(dep\_delay, na.rm=TRUE))

## # A tibble: 365 x 4  
## # Groups: year, month [12]  
## year month day delay  
## <int> <int> <int> <dbl>  
## 1 2013 1 1 11.5   
## 2 2013 1 2 13.9   
## 3 2013 1 3 11.0   
## 4 2013 1 4 8.95  
## 5 2013 1 5 5.73  
## 6 2013 1 6 7.15  
## 7 2013 1 7 5.42  
## 8 2013 1 8 2.55  
## 9 2013 1 9 2.28  
## 10 2013 1 10 2.84  
## # ... with 355 more rows

What I am doing with group\_by() is grouping the flights dataframe by the 3 columns. But note that it doesn’t change how the data looks. So, if we call by\_day, we’ll see the same dataframe.

by\_day

## # A tibble: 336,776 x 19  
## # Groups: year, month, day [365]  
## year month day dep\_time sched\_dep\_time dep\_delay arr\_time sched\_arr\_time  
## <int> <int> <int> <int> <int> <dbl> <int> <int>  
## 1 2013 1 1 517 515 2 830 819  
## 2 2013 1 1 533 529 4 850 830  
## 3 2013 1 1 542 540 2 923 850  
## 4 2013 1 1 544 545 -1 1004 1022  
## 5 2013 1 1 554 600 -6 812 837  
## 6 2013 1 1 554 558 -4 740 728  
## 7 2013 1 1 555 600 -5 913 854  
## 8 2013 1 1 557 600 -3 709 723  
## 9 2013 1 1 557 600 -3 838 846  
## 10 2013 1 1 558 600 -2 753 745  
## # ... with 336,766 more rows, and 11 more variables: arr\_delay <dbl>,  
## # carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,  
## # air\_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time\_hour <dttm>

When I was learning group\_by(), I didn’t know about this fact and I couldn’t understand what was so special about this group\_by(). Then, when we call the summarize() funciton on the grouped data, by\_day, we’ll see that the mean is being calculated per day. We can also calcualte the average arrival delay per month. For this, we would have to group the data by year and month only.

by\_month <- group\_by(flights, year, month)  
summarize(by\_month, delay=mean(arr\_delay, na.rm=TRUE))

## # A tibble: 12 x 3  
## # Groups: year [1]  
## year month delay  
## <int> <int> <dbl>  
## 1 2013 1 6.13   
## 2 2013 2 5.61   
## 3 2013 3 5.81   
## 4 2013 4 11.2   
## 5 2013 5 3.52   
## 6 2013 6 16.5   
## 7 2013 7 16.7   
## 8 2013 8 6.04   
## 9 2013 9 -4.02   
## 10 2013 10 -0.167  
## 11 2013 11 0.461  
## 12 2013 12 14.9

Now that we have a basic kwowledge of summarize works with group\_by, it is useful to learn the Pipe operator or also %>%. First of all, a shortcut for the pipe is ctrl + shift + m. We observe that for using the summarize function in combination with group\_by(), we have the first group the dataframe and save it in a new variable. Then, we proceed to use the new variable in the summarize function(). But that just takes a lot of time. So instead, we can combine both code lines with the pipe operator:

flights %>%   
 group\_by(year, month, day) %>%   
 summarize(mean=mean(arr\_delay, na.rm=TRUE))

## # A tibble: 365 x 4  
## # Groups: year, month [12]  
## year month day mean  
## <int> <int> <int> <dbl>  
## 1 2013 1 1 12.7   
## 2 2013 1 2 12.7   
## 3 2013 1 3 5.73   
## 4 2013 1 4 -1.93   
## 5 2013 1 5 -1.53   
## 6 2013 1 6 4.24   
## 7 2013 1 7 -4.95   
## 8 2013 1 8 -3.23   
## 9 2013 1 9 -0.264  
## 10 2013 1 10 -5.90   
## # ... with 355 more rows

The pipe operator allows us to write everything in one section.

INCLUDE THE CHEATSHEET FOR DPLYR