Beyond data.frame: using data.table

Emmy Cheng

December 3, 2017

Introduction

Motivation and Definition

In class, we already learned the basics behind data manipulation using *data.frame*. It is an easy package to use and contains many useful functions for data analysis, but *data.table*goes beyond that. I found this new package while researching more on data manipulation, and found it to be a very useful and not difficult package that can further our data analysis skills. Thus, I hope to share it with you in this post. Simply put, *data.frame* is another package that we install into R and contains a series of functions that we can use to manipulate data. It enhances thee functions that we already have in *data.table*. I'll be going over various functions in *data.table* and how they differ or expand on things that already exist in *data.frame*.

Let's start by installing the necessary packages to use this package. Install the data.table package and the curl package and load them.

```
#install.packages("data.table")
#install.packages("curl")

library("data.table")
```

```
## Warning: package 'data.table' was built under R version 3.4.3
```

```
library("curl")
```

```
## Warning: package 'curl' was built under R version 3.4.3
```

Adding Data

Let's start by getting data that I'll be using to demonstrate different functions in this post. We'll be using data from flights that deported from NYC airports in Jan-Oct 2014.

data.table uses the function `fread` to load the data. Like in *data.frame* we can use https or a file name to get the data. In addition, this function requires the package curl to use, so don't forget to install it!

```
flights <- fread("flights14.csv")
#website for data#
#https://raw.githubusercontent.com/wiki/arunsrinivasan/flights/NYCflights14/flights14.csv
head(flights)
```

```
##    year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014     1   1      914        14     1238        13         0      AA
## 2: 2014     1   1     1157        -3     1523        13         0      AA
## 3: 2014     1   1     1902         2     2224         9         0      AA
## 4: 2014     1   1      722        -8     1014       -26         0      AA
## 5: 2014     1   1     1347         2     1706         1         0      AA
## 6: 2014     1   1     1824         4     2145         0         0      AA
##    tailnum flight origin dest air_time distance hour min
## 1:  N338AA      1    JFK  LAX      359     2475    9  14
## 2:  N335AA      3    JFK  LAX      363     2475   11  57
## 3:  N327AA     21    JFK  LAX      351     2475   19   2
## 4:  N3EHAA     29    LGA  PBI      157     1035    7  22
## 5:  N319AA    117    JFK  LAX      350     2475   13  47
## 6:  N3DEAA    119    EWR  LAX      339     2454   18  24
```

Basics

We'll first start by going over how to create our own *data.table*, and how to manipulate rows and columns.

i. Creating our own data.table

Using the `data.table()` function, we can make our own *data.table* without importing data from elsewhere.

```
data_table = data.table(ID=c("a","b","c"), a=1:3, b=4:6,c=7:9)
data_table
```

```
##    ID a b c
## 1:  a 1 4 7
## 2:  b 2 5 8
## 3:  c 3 6 9
```

data.table never sets or uses row names and columns of character type are never converted to factors.

ii. general data.table syntax

```
data_table[i, j, by]
```

This is read as subsetting rows by *i*, calculate *j*, then group *by*

iii. subsetting rows

We want to get all the flights starting at "JFK" in June.

```
jfk_june <- flights[origin=="JFK" & month == 6L]
head(jfk_june)
```

```
##    year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014     6   1      851        -9     1205        -5         0      AA
## 2: 2014     6   1     1220       -10     1522       -13         0      AA
## 3: 2014     6   1      718        18     1014        -1         0      AA
## 4: 2014     6   1     1024        -6     1314       -16         0      AA
## 5: 2014     6   1     1841        -4     2125       -45         0      AA
## 6: 2014     6   1     1454        -6     1757       -23         0      AA
##    tailnum flight origin dest air_time distance hour min
## 1:  N787AA      1    JFK  LAX      324     2475    8  51
## 2:  N795AA      3    JFK  LAX      329     2475   12  20
## 3:  N784AA      9    JFK  LAX      326     2475    7  18
## 4:  N791AA     19    JFK  LAX      320     2475   10  24
## 5:  N790AA     21    JFK  LAX      326     2475   18  41
## 6:  N785AA    117    JFK  LAX      329     2475   14  54
```

If columns are variables, they can be referred to by name. For example, we can just say `day` or `year` to refer to the column. However, `flights$day` or `flights$year` also work.

Now, let's get the first two rows of `flights`

```
first_two <- flights[1:2]
first_two
```

```
##    year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014     1   1      914        14     1238        13         0      AA
## 2: 2014     1   1     1157        -3     1523        13         0      AA
##    tailnum flight origin dest air_time distance hour min
## 1:  N338AA      1    JFK  LAX      359     2475    9  14
## 2:  N335AA      3    JFK  LAX      363     2475   11  57
```

We can sort the fliights by column `dest` in ascending order and `origin` in descending order By using "-", we sort the column in decreasing order.

```
sort <- flights[order(dest, -origin)]
head(sort)
```

```
##    year month day dep_time dep_delay arr_time arr_delay cancelled carrier
## 1: 2014     1   1     2011        10     2308         4         0      B6
## 2: 2014     1   2     2215       134      145       161         0      B6
## 3: 2014     1   7     2006         6     2314         6         0      B6
## 4: 2014     1   8     2009        15     2252       -15         0      B6
## 5: 2014     1   9     2039        45     2339        32         0      B6
## 6: 2014     1  10     2029        35     2326        19         0      B6
##    tailnum flight origin dest air_time distance hour min
## 1:  N766JB     65    JFK  ABQ      280     1826   20  11
## 2:  N507JB     65    JFK  ABQ      252     1826   22  15
## 3:  N652JB     65    JFK  ABQ      269     1826   20   6
## 4:  N613JB     65    JFK  ABQ      259     1826   20   9
## 5:  N598JB     65    JFK  ABQ      267     1826   20  39
## 6:  N519JB     65    JFK  ABQ      269     1826   20  29
```

iv. Selecting columns

This will select a column and return it as a vector

```
column <- flights[,hour]
head(column)
```

```
## [1]  9 11 19  7 13 18
```

We can also select a column but return it as a *data.table* instead.

```
col1 <- flights[, list(hour)]
head(col1)
```

```
##    hour
## 1:    9
## 2:   11
## 3:   19
## 4:    7
## 5:   13
## 6:   18
```

There are two ways to select multiple columns.

```
this <- flights[, .(arr_delay, dep_delay)]
or <- flights[, list(arr_delay, dep_delay)]
head(this)
```

```
##    arr_delay dep_delay
## 1:        13        14
## 2:        13        -3
## 3:         9         2
## 4:       -26        -8
## 5:         1         2
## 6:         0         4
```

```
head(or)
```

```
##    arr_delay dep_delay
## 1:        13        14
## 2:        13        -3
## 3:         9         2
## 4:       -26        -8
## 5:         1         2
## 6:         0         4
```

They should both look the same!!

Aggregation

We can tell that the data included in the flights csv file is quite large and requires a lot of time to process, but *data.table* does this job very quickly. Previously, we manipulated `i` and `j` from the general form. Now, we can combine them with `by` to perform operations by *group*. This is very useful when we want to use multiple variables to aggregate data.

Let's get the number of trips corresponding to each origin airport .N is a special variable that holds the number of rows in the current group. `by` accepts character vector for column names. We don't need the "`.() when there is only one expression or column referred to.

```
origins <- flights[, .(.N), by = .(origin)]
origins
```

```
##    origin     N
## 1:    JFK 81483
## 2:    LGA 84433
## 3:    EWR 87400
```

keyby provides us a way to directly sort the variables we grouped by, rather than retaining the original order of groups.

```
we_sort <- flights[carrier == 'AA',
                   .(mean(arr_delay), mean(dep_delay)),
                   keyby = .(origin,dest,month)]
head(we_sort)
```

```
##    origin dest month        V1        V2
## 1:    EWR  DFW     1  6.427673 10.012579
## 2:    EWR  DFW     2 10.536765 11.345588
## 3:    EWR  DFW     3 12.865031  8.079755
## 4:    EWR  DFW     4 17.792683 12.920732
## 5:    EWR  DFW     5 18.487805 18.682927
## 6:    EWR  DFW     6 37.005952 38.744048
```

Of course, there are many other ways to aggregate data, but these are just some of the basic ways to do so.

Conclusion

Key takeaways

Hopefully in this post, you were able to learn more about the *data.table* package and some of its functions. The main point is that it expands on what we have learned in class (*data.frame*) and offers more functions to analyze larger sets of data faster. It offers quicker data aggregation and more flexibility when we want to sort our data. More information can be found in the references I've provided below. For the post's purposes and easier understanding, I've only chosen to show a few topics and explained each block of code. I hope you've learned a lot!

References

References

1. https://www.datacamp.com/community/tutorials/data-table-cheat-sheet
2. https://www.r-bloggers.com/intro-to-the-data-table-package/
3. https://github.com/Rdatatable/data.table/wiki
4. https://campus.datacamp.com/courses/data-table-data-manipulation-r-tutorial/chapter-one-datatable-novice?ex=1
5. https://www.r-bloggers.com/two-of-my-favorite-data-table-features/
6. https://www.rdocumentation.org/packages/data.table/versions/1.10.4-2
7. https://www.dezyre.com/data-science-in-r-programming-tutorial/r-data-table-tutorial