# Intro to data.table

*Steve Pittard*

*September 12, 2015*

## Motivations

Data frames are very powerful structures in R that let you describe observations and the attributes of those observations in a matrix like structure. You don't use R for very long without encountering data frames. data.frames are organized into rows and columns. We use "bracket notation" to address specific rows and columns

```
DF <- data.frame(x=c("B","A","B","A","B"),y=c(7,2,1,5,9))

str(DF)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ x: Factor w/ 2 levels "A","B": 2 1 2 1 2
##  $ y: num  7 2 1 5 9
```

```
DF
```

```
##   x y
## 1 B 7
## 2 A 2
## 3 B 1
## 4 A 5
## 5 B 9
```

```
DF[DF$y > 2, ]   # Get all rows where y > 2
```

```
##   x y
## 1 B 7
## 4 A 5
## 5 B 9
```

```
DF[DF$y > 2, 'x']   # Same as above but get just the x column
```

```
## [1] B A B
## Levels: A B
```

```
DF[DF$y > 2 & DF$x == "B",]  # Get all rows where y > 2 and x = "b"
```

```
##   x y
## 1 B 7
## 5 B 9
```

```
DF[DF$y > 2 & DF$x == "B",]$y   # Same as above but returns only the vector y
```

```
## [1] 7 9
```

```
mean(DF[DF$y > 2 & DF$x == "B",]$y)  # get the mean of y
```

```
## [1] 8
```

```
# DF[,sum(y)]   Won't work  with a data frame

sum(DF$y)
```

```
## [1] 24
```

Most people don't realize that the bracket notation is a function that is builtin to R. Just because it is a symbol "[" instead of say a word like "subset" they doesn't mean it cannot be a function.

```
`[`
```

```
## .Primitive("[")
```

```
`+`
```

```
## function (e1, e2)  .Primitive("+")
```

## data.table

The data.table package let's you create a data frame "substitute" that can behave almost exactly like a data frame when it is necssary. That is it will repsond to functions such as **nrow**. However, there are many interesting differences between a data table and a data frame structures that we will soon see.

The general form os the data.table is as follows. There are no methods or replacement functions to learn. That is, for example, data.table does not attempt to replace the standard functions in available in R like mean, sd, quantile, etc.

```
DT[where,select|update,group by][having][order by]
```

```
library(data.table)

DT <- data.table(DF)

class(DT)
```

```
## [1] "data.table" "data.frame"
```

```
DT[2:3,y]   # Get rows 2 and 3 and only column y (not quote needed around y)
```

```
## [1] 2 1
```

```
DT[!2:3]    # Get all rows but rows 2 and 3
```

```
##    x y
## 1: B 7
## 2: A 5
## 3: B 9
```

```
# DF[,sum(y)] # Let's get the sum of y column  Won't work  with a data frame

DT[,sum(y)]  # This works inside the bracket notation
```

```
## [1] 24
```

## Summary and aggregation

We frequently want to summarize by some numeric value in terms of factors. For example find the average value of y for each level of factor x. R has ways to do this.

```
tapply(DF$y, DF$x, mean)
```

```
##        A        B
## 3.500000 5.666667
```

```
# OR

aggregate(y~., data=DF, mean)
```

```
##   x        y
## 1 A 3.500000
## 2 B 5.666667
```

```
aggregate(y~.,data=DF, function(x) c(mean=round(mean(x),2), sd=round(sd(x),1)) )
```

```
##   x y.mean y.sd
## 1 A   3.50 2.10
## 2 B   5.67 4.20
```

But wouldn't it be nice to have this capability as part of the data table structure itself ? data.table allows us to do just that. Some advantages of **data.table** are:

- Extends the data frame bracket notation to do more
- Works well with huge data files
- Works with non data.table functions - that is it can act like a regular data frame when necessary

```
DT <- data.table(DF)   # We just created a data.table

class(DT)     # It has attributes of both a data frame and a data table
```

```
## [1] "data.table" "data.frame"
```

```
DT[,mean(y),by=x]  # replaces our aggregate function
```

```
##    x       V1
## 1: B 5.666667
## 2: A 3.500000
```

When considering how to best work a data table keep the following in mind. If we have a data table called DT we think of the associated bracket notation as being DT[i,j,by]. With DT we subset rows using ‘’i‘‘then calculate’‘j’‘as grouped by’‘k”. One is not obligated to take full advantage of the data table construct - we could use it simply to read in really large files which is something at which it excels. But let’s explore the expanded capabilities of data table’s bracket notation a little more.

```
DF[with(DF,order(-y)),]    # Sort by descending y using a data frame
```

```
##   x y
## 5 B 9
## 1 B 7
## 4 A 5
## 2 A 2
## 3 B 1
```

```
DT[order(-y)]              # Sort using a data table  simple
```

```
##    x y
## 1: B 9
## 2: B 7
## 3: A 5
## 4: A 2
## 5: B 1
```

```
DT[,list(total = sum(y), mean=mean(y))]    # Get the sum and of y for all rows
```

```
##    total mean
## 1:    24  4.8
```

```
DT[,list(total = sum(y), mean=mean(y)),by=x]  # Do it again but group by factor x
```

```
##    x total     mean
## 1: B    17 5.666667
## 2: A     7 3.500000
```

Note that if we have want to name the results of the function in the “j” position we need to create a list to contain it. You might also see the period notation in some examples of data.table which accomplishes the same thing.

```
DT[,sum(y),by=x]
```

```
##    x V1
## 1: B 17
## 2: A  7
```

```
# DT[,total=sum(y),by=x]   # Doesn't work

DT[, list(total=sum(y)),by=x]
```

```
##    x total
## 1: B    17
## 2: A     7
```

```
DT[, list(total=sum(y)),by=x]   # same as above but takes some getting used to
```

```
##    x total
## 1: B    17
## 2: A     7
```

```
DT[, list(total=sum(y), avg=mean(y)), by=x]
```

```
##    x total      avg
## 1: B    17 5.666667
## 2: A     7 3.500000
```

```
# The old way to do this - so to speak is to use the aggregate command

aggregate(y ~ x, data=DT, function(v) c(total=sum(v),avg=mean(v)))
```

```
##   x   y.total     y.avg
## 1 A  7.000000  3.500000
## 2 B 17.000000  5.666667
```

Let's do some some subsetting and some more aggregation

```
DT[, .N]    # the .N variable is a special variable in a data table
```

```
## [1] 5
```

```
DT[, .N, by=x]  # selects the number of rows  for each level of factor B
```

```
##    x N
## 1: B 3
## 2: A 2
```

```
DT[y > 2, .N, by = x]  # selects the number of rows where y > 2. We then group by x
```

```
##    x N
## 1: B 2
## 2: A 1
```

We can update existing columns or add new ones

```
DT[,y := y+1]          # Add 1 to every value of y
```

```
##    x  y
## 1: B  8
## 2: A  3
## 3: B  2
## 4: A  6
## 5: B 10
```

```
DT[,y := y-1]          # Remove 1 from
```

```
##    x y
## 1: B 7
## 2: A 2
## 3: B 1
## 4: A 5
## 5: B 9
```

```
DT[, avg := mean(y)]   # Add a column name avg
```

```
##    x y avg
## 1: B 7 4.8
## 2: A 2 4.8
## 3: B 1 4.8
## 4: A 5 4.8
## 5: B 9 4.8
```

```
DT[,avg := NULL]       # Remove the avg column
```

```
##    x y
## 1: B 7
## 2: A 2
## 3: B 1
## 4: A 5
## 5: B 9
```

```
DT[,avg :=mean(y), by=x] # Add the group averages to the respective rows
```

```
##    x y      avg
## 1: B 7 5.666667
## 2: A 2 3.500000
## 3: B 1 5.666667
## 4: A 5 3.500000
## 5: B 9 5.666667
```

```
DT[,avg := NULL]
```

```
##    x y
## 1: B 7
## 2: A 2
## 3: B 1
## 4: A 5
## 5: B 9
```

## Larger files

Let's examine a larger data set. If you cloned the repository that contains this document you also have a file called "hflights.csv". If you haven't cloned or downloaded the respository then the address is https://github.com/steviep42/BrownBag Check in the data.table folder.

This dataset contains all flights departing from Houston airports IAH (George Bush Intercontinental) and HOU (Houston Hobby). The data as part of the hflights package but to do some comparisons we'll read the data in from the .csv file.

Let's time how long it takes to read in this file using the standard **read.csv** function and then do the same thing using the **fread** function that comes as part of the data.table package.

```
system.time(dff <- read.csv("hflights.csv",sep=",",header=TRUE))
```

```
##    user  system elapsed
##   4.854   0.093   4.981
```

```
system.time(dft <- fread("hflights.csv",sep=",",header=TRUE))
```

```
##    user  system elapsed
##   0.223   0.014   0.240
```

We see that it takes much longer to read in the data frame version than the data table version. The larger the input the file the larger the difference between these two operations.We'll see an exmple of that later. Let's do some comparative aggregatio to see if using data.table is any faster than using say the aggregate function

```
str(dft)
```

```
## Classes 'data.table' and 'data.frame':   227496 obs. of  21 variables:
##  $ Year             : int  2011 2011 2011 2011 2011 2011 2011 2011 2011 2011 ...
##  $ Month            : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ DayofMonth       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ DayOfWeek        : int  6 7 1 2 3 4 5 6 7 1 ...
##  $ DepTime          : int  1400 1401 1352 1403 1405 1359 1359 1355 1443 1443 ...
##  $ ArrTime          : int  1500 1501 1502 1513 1507 1503 1509 1454 1554 1553 ...
##  $ UniqueCarrier    : chr  "AA" "AA" "AA" "AA" ...
##  $ FlightNum        : int  428 428 428 428 428 428 428 428 428 428 ...
##  $ TailNum          : chr  "N576AA" "N557AA" "N541AA" "N403AA" ...
##  $ ActualElapsedTime: int  60 60 70 70 62 64 70 59 71 70 ...
##  $ AirTime          : int  40 45 48 39 44 45 43 40 41 45 ...
##  $ ArrDelay         : int  -10 -9 -8 3 -3 -7 -1 -16 44 43 ...
##  $ DepDelay         : int  0 1 -8 3 5 -1 -1 -5 43 43 ...
##  $ Origin           : chr  "IAH" "IAH" "IAH" "IAH" ...
##  $ Dest             : chr  "DFW" "DFW" "DFW" "DFW" ...
##  $ Distance         : int  224 224 224 224 224 224 224 224 224 224 ...
##  $ TaxiIn           : int  7 6 5 9 9 6 12 7 8 6 ...
##  $ TaxiOut          : int  13 9 17 22 9 13 15 12 22 19 ...
##  $ Cancelled        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ CancellationCode : chr  "" "" "" "" ...
##  $ Diverted         : int  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Let's compute the average AirTime as grouped by DayOfWeek and then by Origin.

```
system.time(agg1 <- aggregate(AirTime~DayOfWeek+Origin,data=dff,mean))
```

```
##    user  system elapsed
##   2.044   0.030   2.091
```

```
agg1
```

```
##    DayOfWeek Origin   AirTime
## 1          1    HOU  87.03567
## 2          2    HOU  86.84831
## 3          3    HOU  86.71957
## 4          4    HOU  86.20807
## 5          5    HOU  86.47776
## 6          6    HOU  90.27321
## 7          7    HOU  87.62233
## 8          1    IAH 113.86052
## 9          2    IAH 113.98372
## 10         3    IAH 113.96404
## 11         4    IAH 113.80271
## 12         5    IAH 114.08602
## 13         6    IAH 116.98549
## 14         7    IAH 114.31497
```

```
system.time(agg2 <- dft[, list(avg=mean(AirTime,na.rm=T)), by=list(DayOfWeek,Origin)] )
```

```
##    user  system elapsed
##   0.007   0.001   0.007
```

```
agg2
```

```
##     DayOfWeek Origin       avg
##  1:         6    IAH 116.98549
##  2:         7    IAH 114.31497
##  3:         1    IAH 113.86052
##  4:         2    IAH 113.98372
##  5:         3    IAH 113.96404
##  6:         4    IAH 113.80271
##  7:         5    IAH 114.08602
##  8:         6    HOU  90.27321
##  9:         7    HOU  87.62233
## 10:         1    HOU  87.03567
## 11:         2    HOU  86.84831
## 12:         3    HOU  86.71957
## 13:         4    HOU  86.20807
## 14:         5    HOU  86.47776
```

The data.table approach will be faster. Also it's sometimes very obvious what the factors or categories are in a data frame by using the **str** function although this isn't alawys the case. We can then use an approach like the following to see how many unqiue values each column takes on. Those taking on only a few discrete values might be good candidates for being a factor.

```
sapply(dft,function(x) {length(unique(x))})
```

```
##              Year             Month        DayofMonth         DayOfWeek
##                 1                12                31                 7
##           DepTime           ArrTime     UniqueCarrier         FlightNum
##              1208              1284                15              3740
##           TailNum ActualElapsedTime           AirTime          ArrDelay
##              3320               436               399               464
##          DepDelay            Origin              Dest          Distance
##               430                 2               116               159
##            TaxiIn           TaxiOut         Cancelled  CancellationCode
##                97               144                 2                 5
##          Diverted
##                 2
```

```
# If we wanted to use the data.table approach

dft[,lapply(.SD,function(x) {length(unique(x))})]
```

```
##    Year Month DayofMonth DayOfWeek DepTime ArrTime UniqueCarrier FlightNum
## 1:    1    12         31         7    1208    1284            15      3740
##    TailNum ActualElapsedTime AirTime ArrDelay DepDelay Origin Dest
## 1:    3320               436     399      464      430      2  116
##    Distance TaxiIn TaxiOut Cancelled CancellationCode Diverted
## 1:      159     97     144         2                5        2
```

Note that in the second example we are using a special variable made available by data.table called **.SD** which basically stands for "subsetted data". In this case we aren't using the **by** operator so the "subsetted data" in this case is the entire data table. We then use the lapply command to apply our anonymous function over each column.

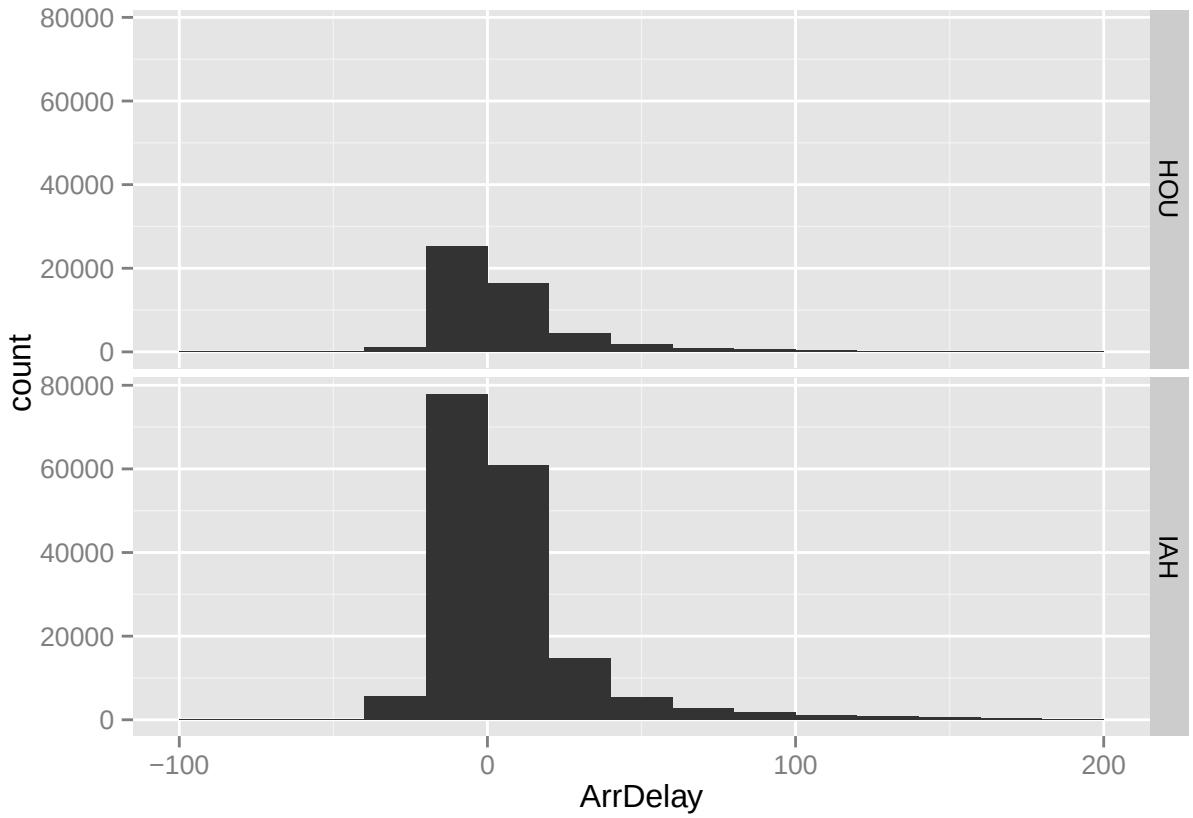
Let's look at the distribution of Arrival Delays from both airports. Let's make some histograms. We'll use ggplot and the chaining function from the dplyr package.

```
suppressMessages(library(dplyr))
library(ggplot2)

dft[,list(quantile=quantile(ArrDelay,na.rm=T)),by=Origin]
```

```
##     Origin quantile
##  1:    IAH      -70
##  2:    IAH       -8
##  3:    IAH        0
##  4:    IAH       11
##  5:    IAH      978
##  6:    HOU      -44
##  7:    HOU       -7
##  8:    HOU       -1
##  9:    HOU       11
## 10:    HOU      822
```

```
dft %>% ggplot(aes(x=ArrDelay)) + geom_bar(binwidth=20) +xlim(-100,200) + facet_grid(Origin ~.)
```




```
dft[,.N,by=Origin]
```

```
##    Origin      N
## 1:    IAH 175197
## 2:    HOU  52299
```

```
dft[,.N,by=Origin][order(-N)]
```

```
##    Origin      N
## 1:    IAH 175197
## 2:    HOU  52299
```

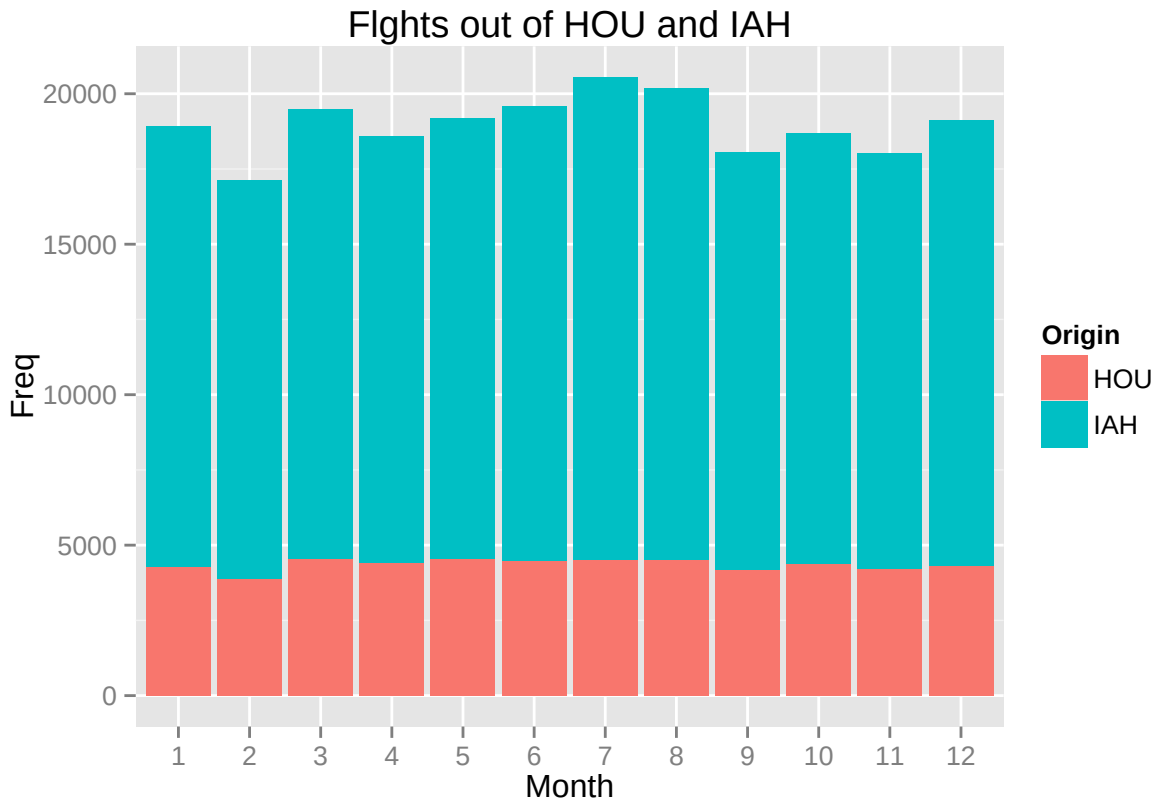
```
# how many flights are coming out of each airport each month ?

dft[,table(Origin,Month)]
```

```
##       Month
## Origin     1     2     3     4     5     6     7     8     9    10    11
##    HOU  4270  3884  4544  4420  4533  4499  4519  4505  4186  4405  4212
##    IAH 14640 13244 14926 14173 14639 15101 16029 15671 13879 14291 13809
##       Month
## Origin    12
##    HOU  4322
##    IAH 14795
```

```
# Let's create a stacked bar chart with this information

as.data.frame(dft[,table(Origin,Month)]) %>%
  ggplot(aes(x=Month,y=Freq,fill=Origin)) + geom_bar(stat="identity") +
  ggtitle("Flghts out of HOU and IAH")
```




```
#
```

## Processing an Even Larger File

I have a data set that contains a 150GB sample of the data. It includes a full 3 months of hourly page traffic statistics from Wikipedia (1/1/2011 - 3/31/2011), The breakdown is as follows:

- Contains hourly wikipedia article traffic stats covering a 3 mont period
- Each of the 2,161 log files is named with the date and time of collection
- Each line has 4 fields: projectcode, pagename, pageviews, and downloads in bytes

The data is available from Amazon Public Data sets at

https://aws.amazon.com/datasets/wikipedia=page-traffic-statistic-v3/

I took some of these files and combined them into a single file of size 1.4GB and 31,164,567 records with 4 columns. This is a small fraction of the total data. I ran this test on an Amazon Web Services instance that has 4GB of memory and 2 cores. Many laptops have at least this and usually more.

```
$ ls -lh combined_wiki.txt
-rw-r--r-- 1 ubuntu root 1.4G Sep 3 19:27 combined_wiki.txt

$ wc -l combined_wiki.txt
31164567 combined_wiki.txt
```

## Interesting things

The following two are equivalent and take about the same time to execute

```
system.time(dft[,mean(ArrDelay,na.rm=T),by=Origin])
```

```
##    user  system elapsed
##   0.005   0.000   0.005
```

```
#

system.time(dft[,lapply(.SD,mean,na.rm=T),by=Origin,.SDcols=c(12)])
```

```
##    user  system elapsed
##   0.006   0.000   0.006
```