# Using the data.table package

Steve Pittard wsp@emory.edu

November 6, 2015

# Motivations

- A data frame is a set of columns. Every column is same length but of possibly different types.
- It has characteristics of both a matrix, (each row is the same data type),
- Each column can be a different data type
- Bracket notation offers a convenient way to search through the data drame

# Motivations

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

DT[2:3,]
  x y
2 A 2
3 B 1
```

# Get Rows 2 and 3

DT[2:3,]

| X | Y |
|---|---|
| B | 7 |
| A | 2 |
| B | 1 |
| A | 5 |
| B | 9 |

# Find all rows where x is equal to B and y greater than 2

```
DT[DT$x=="B" & DT$y > 2,]
```

| X | Y |
|---|---|
| B | 7 |
| A | 2 |
| B | 1 |
| A | 5 |
| B | 9 |

# Find all rows where x is equal to B and y greater than 2

If DT were a table in some relational database system such as MySQL, SQLite, Oracle, Postgress, Access, etc then you could use SQL to extract the information. So if you know SQL then this approach might be more attractive to you.

```
select * from DT where x = 'B' and y > 2
```

| X | Y |
|---|---|
| B | 7 |
| A | 2 |
| B | 1 |
| A | 5 |
| B | 9 |

# Find all rows where x is equal to B and y greater than 2

Note that there is an R package called **sqldf** that lets you treat a data frame as if it were a table in a relational database.

```
library(sqldf)
sqldf("select * from DT where x = 'B' and y > 2")
  x y
1 B 7
2 B 9
```

| X | Y |
|---|---|
| B | 7 |
| A | 2 |
| B | 1 |
| A | 5 |
| B | 9 |

# Motivations

But some things that are natural to want to do do not work with standard data fames

```
DT[,'y']
[1] 7 2 1 5 9

# I think the following should work but it does not

DT[,sum('y')]
Error in sum("y") : invalid 'type' (character) of argument

sum(DT$y)          # You have to do it this way
[1] 24
```

# Summarizing by Factors

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. There are functions we can call

```
tapply(DT$y, DT$x, mean)
       A        B
3.500000 5.666667


# OR


aggregate(y~., data=DT, mean)
  x        y
1 A 3.500000
2 B 5.666667
```

# Summarizing by Factors

But wouldn't it be nice to have summary capability as part of the data frame structure ?

**data.table** allows us to do just that. There are packages such as dplyr that do this als but we will look at **data.table** first.

Some advantages of **data.table** is that:

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

# Summarizing by Factors

We will turn DT into a data.table - it is easy and it will still act like a data frame if you use the standard data frame functions

```
NDT <- data.table(DT)

class(NDT)
[1] "data.table" "data.frame"

nrow(NDT)
[1] 5

NDT[,sum(y)]  # Aha - this will not work with typical data frames
[1] 24
```

# Summarizing by Factors

Aggregating with **data.table** follows a pattern very similar to SQL although you don't need to know SQL to memorize it

**DT[i, j, by]**

Reading the above out loud would sound like: "With DT, subset rows using i, then calculate j as grouped by k.

| | | | | |
|---|---|---|---|---|
| R | : | i | j | by |
| SQL | : | WHERE | SELECT | GROUP BY |

# Summarizing by Factors

```
# Using NDT, for all rows calculate the sum and mean of y

NDT[,.(total = sum(y), mean = mean(y))]
   total mean
1:    24  4.8

# For all rows calculate sum and mean of y and group by x

NDT[,.(total = sum(y), mean = mean(y)),by=x]
   x total     mean
1: B    17 5.666667
2: A     7 3.500000

# Old way - you have to use a function like aggregate

aggregate(y~x,data=NDT,function(v) c(total=sum(v),mean=mean(v)))
  x   y.total    y.mean
1 A  7.000000  3.500000
2 B 17.000000  5.666667
```

# Summarizing by Factors

```
# How many rows in NDT ?

NDT[,.N]
[1] 5

# How many observations in each group

NDT[,.N,by=x]
   x N
1: B 3
2: A 2

# For rows where y > 2 how many observations in each group

NDT[y > 2,.N,by=x]
   x N
1: B 2
2: A 1
```

# Summarizing by Factors

Let's do some work with the built in mtcars data frame

```
tcars <- data.table(mtcars)

str(tcars)
Classes data.table and data.frame:  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

# Summarizing by Factors

```
# For all rows calculate the mean mpg as grouped by trans type

tcars[,.(mean_mpg = mean(mpg)),by=.(trans=am)]
   trans mean_mpg
1:     1 24.39231
2:     0 17.14737

# For all rows calculate mean mpg as grouped by trans type and cylinder

tcars[,.(mean_mpg = mean(mpg)),by=.(trans=am,cyl=cyl)]
   trans cyl mean_mpg
1:     1   6 20.56667
2:     1   4 28.07500
3:     0   6 19.12500
4:     0   8 15.05000
5:     0   4 22.90000
6:     1   8 15.40000
```

# Summarizing by Factors

```
# To order the result in descending order of mean_mpg we just treat it like
# data frame - which it can act like !

tcars[,.(mean_mpg = mean(mpg)),by=.(trans=am,cyl=cyl)][order(-mean_mpg),]
   trans cyl mean_mpg
1:     1   4 28.07500
2:     0   4 22.90000
3:     1   6 20.56667
4:     0   6 19.12500
5:     1   8 15.40000
6:     0   8 15.05000

# Could have done

tmpdf <- tcars[,.(mean_mpg = mean(mpg)),by=.(trans=am,cyl=cyl)]
tmpdf[order(-tmpdf$mean_mpg),]
```

# Summarizing by Factors

```
# where car weight > 2.5 tons calculate the mean mpg as grouped by gears

tcars[wt > 2.5,.(mean_mpg = mean(mpg)), by=.(gear=gear)]
   gear mean_mpg
1:    4 21.08571
2:    3 15.72143
3:    5 16.83333

tcars[,.N]   # How many rows ?
[1] 32

tcars[,.N,by=am]  # Number of obs per trans type
   am  N
1:  1 13
2:  0 19

tcars[,.N,by=cyl] # Number of obs per cyl group
   cyl  N
1:   6  7
2:   4 11
3:   8 14
```

# Example: Reading in a “large” .csv file

- Consider a .csv file that has 334,142 rows relating to calls to the Chicago police in 2012
- `http://steviep42.bitbucket.org/YOUTUBE.DIR/chi_crimes.csv`
- Use the default **read.csv** function to create a data frame
- Let’s time this function call to see how long it will take on my MacBook Pro with 8GB of RAM:

```
system.time(df.crimes <- read.csv("chi_crimes.csv",
                                  header=TRUE,sep=","))
   user  system elapsed
 30.251   0.283  30.569

nrow(df.crimes)
[1] 334141
```

# Example: Reading in a "large" .csv file

Let's read in this file using **data.table** function. The **fread** function returns a data table

```
library(data.table)

system.time(dt.crimes <- fread("chi_crimes.csv",
                               header=TRUE,sep=","))
   user  system elapsed
  1.045   0.037   1.362

attributes(dt.crimes)$class     # dt.crimes is also a data.frame
[1] "data.table" "data.frame"

nrow(df.crimes)
[1] 334141

dt.crimes[,.N]
[1] 334141
```

# Aggregation on a large file

This data frame has information on every call to Chicago police in the year 2013.

So we'll want to see what factors there are in the dataframe so we can do some summaries across groups.

```
names(df.crimes)
 [1] "Case.Number"          "ID"                   "Date"
 [5] "IUCR"                 "Primary.Type"         "Description"
 [9] "Arrest"               "Domestic"             "Beat"
[13] "Ward"                 "FBI.Code"             "X.Coordinate"
[17] "Y.Coordinate"         "Year"                 "Latitude"
```

# Aggregation on a large file

Let's see how many unique values there are for each column. Looks like 30 FBI codes so maybe we could see the number of calls per FBI code. What about District ? There are 25 of those.

```
sapply(df.crimes,function(x) {length(unique(x))})
        Case.Number                  ID                Date
             334114              334139              121484
       Primary.Type         Description Location.Description
                 30                 296                 120
               Beat            District                Ward
                302                  25                  51
     Community.Area        Y.Coordinate                Year
                 79               89895                   1
          Longitude            Location
             180393              178534
```

# Aggregation on a Large File

How many calls per District were there ?

```
> nt[,.N,by=District]
    District     N
 1:       11 21798
 2:        1 12107
 3:       15 14385
 4:        2 13448
 5:        3 17649
 6:       20  5674
 7:        8 22386
 8:        5 15258
 9:        7 20150
10:        6 19232
11:       18 14178
12:       16 10753
13:       12  8774
14:        4 19789
15:       17  9673
16:       24  9498
17:        9 16656
18:       19 15608
19:       22 10745
20:       14 12537
21:       10 15016
22:       25 19658
23:       13  7084
24:       NA  2079
25:       31     6
```

# Aggregation on a Large File

How many calls per District were there ?

```
nt[,.N,by=District]
    District     N
 1:       11 21798
 2:        1 12107
 3:       15 14385
 4:        2 13448
 5:        3 17649
 6:       20  5674
 7:        8 22386
 8:        5 15258
 9:        7 20150
10:        6 19232
11:       18 14178
12:       16 10753
13:       12  8774
14:        4 19789
15:       17  9673
16:       24  9498
17:        9 16656
18:       19 15608
19:       22 10745
20:       14 12537
21:       10 15016
22:       25 19658
23:       13  7084
24:       NA  2079
25:       31     6
```

# Aggregation on a Large File

- How many calls per District were there ?
- Notice that when we use a function designed for a native data frame that it will work
- However, as we will soon see, it works much slower than using the data.table approach

```
table(nt$District)

    1     2     3     4     5     6     7     8     9    10    11    12    13
12107 13448 17649 19789 15258 19232 20150 22386 16656 15016 21798  8774  7084

   14    15    16    17    18    19    20    22    24    25    31
12537 14385 10753  9673 14178 15608  5674 10745  9498 19658     6
```

# Aggregation on a Large File

The difference in time is pretty significant although from the user point of view it might not be that different. The utility of data.table becomes apparent when reading in really large files (millions of rows and lots of attributes/columns)

```
system.time( nt[,.N,by=District] )
   user  system elapsed
  0.003   0.000   0.002

system.time( table(nt$District) )
   user  system elapsed
  0.089   0.004   0.093
```

# Aggregation on a Large File

Let's randomly sample 500 rows and then find the mean calls to the cops as grouped by FBI.Code (whatever that corresponds to) check `https://www2.fbi.gov/ucr/nibrs/manuals/v1all.pdf` to see them all

```
nt[sample(1:.N,500),.(mean=mean(.N)),by=FBI.Code]
```

```
    FBI.Code mean
 1:      08A   28
 2:      08B   97
 3:       06   98
 4:       24    8
 5:       03   21
 6:       11   13
 7:       22    3
 8:       26   38
 9:       05   33
10:      04A   10
11:       14   51
12:       18   47
13:       07   18
14:       15   15
```

# Wikipedia Page Traffic Statistics

This dataset contains a 150 GB sample of the data used to power trendingtopics.org.

It includes a full 3 months of hourly page traffic statistics from Wikipedia (1/1/2011-3/31/2011)

- Contains hourly wikipedia article traffic statistics dataset covering 3 month period from January 01 2011 to March 31 2011
- Each of the 2,161 log files is named with the date and time of collection: pagecounts-20090430-230000.gz
- Each line has 4 fields: projectcode, pagename, pageviews, bytes

Data available from Amazon Public Data Sets at `https://aws.amazon.com/datasets/wikipedia-page-traffic-statistic-v3/`

# Wikipedia Page Traffic Statistics

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 am running 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
```

# Wikipedia Page Traffic Statistics

Let's read this file in using the native **read.csv** function and then with the **fread** function supplied with the data.table package.

```
system.time(df <- read.csv("combined_wiki.txt",sep=" ",header=F))
   user  system elapsed
660.344   4.244 665.825

nrow(df)
[1] 31164567

head(df,5)
    V1                                V2 V3   V4
1 aa.b                         Main_Page  1 5565
2 aa.b             MediaWiki:Image_sample  1 5179
3 aa.b        MediaWiki:Upload_source_file  1 5195
4 aa.b           Wikibooks:Privacy_policy  1 4925
5 aa.d   MediaWiki:Group-abusefilter-member  1 4912
```

# Wikipedia Page Traffic Statistics

Let's read this file in using the native **read.csv** function and then with the **fread** function supplied with the data.table package.

```
system.time(dt <- fread("combined_wiki.txt"))
Read 31164567 rows and 4 (of 4) columns from 1.307 GB file in 00:02:22
   user  system elapsed
140.476   1.064 156.104

dt[1:5,]
     V1                                V2 V3   V4
1: aa.b                         Main_Page  1 5565
2: aa.b            MediaWiki:Image_sample  1 5179
3: aa.b      MediaWiki:Upload_source_file  1 5195
4: aa.b          Wikibooks:Privacy_policy  1 4925
5: aa.d MediaWiki:Group-abusefilter-member  1 4912

dt[,.N]
[1] 31164567
```

# Wikipedia Page Traffic Statistics

For all rows where page access is > 5 what is the mean number of page accesses as grouped by project code ?

```
> dt[V3 > 5,.(mean=mean(V3)),by=V1]
                V1      mean
  1:            ab  8.777778
  2:           ace  9.785714
  3:            af 27.277778
  4:           als 13.964286
  5:            am 10.900000
 ---
597: zh-min-nan.d  6.000000
598:          zu.b  6.000000
599:         ms.mw  6.000000
600:          mt.d  6.000000
601:          tg.d 10.000000
```

# Wikipedia Page Traffic Statistics

For all rows where page access is $> 5$ what is the mean number of page accesses as grouped by project code ?

```
system.time( dt[V3 > 5,.(mean=mean(V3)),by=V1] )
   user  system elapsed
  0.412   0.080   0.492
```

# setkey function

It is possible to sort a data table by using the **setkey(DT, key)** function that is part of the **data.table** package

This will reorganize the data table DT by key where key is a column name

This isn't necessary unless perhaps you want to pre sort the data table for some aggregation in which case pre sorted the data can help.

to make this clear let's look at the smaller data frame from the first slides

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

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

# setkey function

```
setkey(DT,x)  # Will sort the table by x
   x y
1: A 2
2: A 5
3: B 7
4: B 1
5: B 9

setkey(DT,y) # Will sort the table by y
   x y
1: B 1
2: A 2
3: A 5
4: B 7
5: B 9
```

# setkey function

```
setkey(DT,x,y)  # Sort by x and y

DT

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

# Review

Why did I show you all this ? Let's review the good stuff about data.table

- Excellent for reading in very large data sets fast
- Provides a way to do aggregation within the brackets
- Can behave just like a data frame if need be
- It is easy to turn an existing data frame into a data table

Okay why wouldn't you use data.table ?

- Do you REALLY need to read in a 100 million row dataset ?
- Just read in a fraction of the data
- The bracket notation can get busy
- Maybe you should use a data base / SQL approach

# sqldf

There is another way to do this. Turn the data frame into a SQLite database We will explore SQL and databses in another class

```
library(sqldf)

sqldf("attach cwiki as new")     # create a new SQLite databse

# Read in the .txt file into a table

system.time( read.csv.sql("combined_wiki.txt",
   sql="create table info as select * from file",
   dbname="cwiki",header=FALSE,sep=" ")
)
   user  system elapsed
 85.832  10.676 134.189
```

# sqldf

```
# Prove that we got all the records

sqldf("select count(*) from info", dbname="cwiki")
  count(*)
1 31164567

# How many distinct Project Codes are there ?

sqldf("select count(distinct(V1)) from info",dbname="cwiki")
  count(distinct(V1))
1                1266

# How many records correspond to project code aa.b ?

sqldf("select count(*) from info where V1 = 'aa.b' ",
      dbname="cwiki")
  count(*)
1       52
```

# sqldf

```
# Create a data frame holding all rows relating to aa.b

df <- sqldf("select * from info where V1 = 'aa.b' ",
            dbname="cwiki")

head(df)
    V1                         V2 V3    V4
1 aa.b                  Main_Page  1  5565
2 aa.b       MediaWiki:Image_sample  1  5179
3 aa.b MediaWiki:Upload_source_file  1  5195
4 aa.b    Wikibooks:Privacy_policy  1  4925
5 aa.b                  Main_Page  3 27714
6 aa.b            Special:Imagelist  1   540
```

# sqldf

```
# Compute the average number of megabytes downloaded for
# the top 5 unique project codes

sqldf("select distinct(V1) as ProjCode,avg(V4)/1000000 as MB
       from info group by V1 order by MB
       desc limit 5",dbname="cwiki")

  ProjCode        MB
1    en.mw 77518.224
2    ja.mw  9126.983
3    fr.mw  2020.454
4    ru.mw  1311.165
5    de.mw  1214.592
```

# sqldf

Well if you knew that you just wanted the Project codes aa.b from the beginning then you could have specified this when creating the database originally.

This would save some time in the long run since we wouldn't be reading in data for other project codes

```
sqldf("attach cwiki as new")

# We just select the aa.b project codes

read.csv.sql("combined_wiki.txt",
          sql = "create table info as select * from file
          where V1 = 'aa.b'",
          dbname = "cwiki", header = FALSE, sep = " ")

sqldf("select count(*) from info",dbname="cwiki")
  count(*)
1       52
```