05 - Fast Manipulations for "Bigger" Data

R Workshop – Data wRestling

Adam Loy and Susan Vander Plas

Iowa State University

May 15, 2013

Corn production

We have data on corn production in the Cornbelt from 1958–2011 (Source: NASS Quick Stats)

```
corn <- read.csv("corn.csv")
names(corn)
```

```
## [1] "Year"          "State"         "StateFIPS"     "County"
## [5] "CountyCode"    "Crop"          "SubCrop"       "Measurement"
## [9] "Value"
```

```
levels(corn$Measurement)
```

```
## [1] "ACRES HARVESTED" "ACRES PLANTED"   "PRODUCTION"
## [4] "YIELD"
```

```
dim(corn)
```

```
## [1] 296377      9
```

Corn prodcution

Suppose we want to find the mean yield over the timespan for each county

```
plyr.agg <- ddply(corn, .(State, StateFIPS, County, CountyCode,
    Measurement), summarise, means = mean(Value))
```

Ok, that was fast, but what if we were interested in summaries by year?

- Some counties reported irrigated and nonirrigated land separately
- We need to aggregate within Year

- We just went from aggregating within 1,010 groups to 51,830 groups
- Adding Year to the ddply statement took almost 13 minutes on my laptop

Speeding things up

How can we speed up this aggregation?

- Manually subset
  1. Subset counties with numerous measurements
  2. Use ddply on the reduced set
- Find a faster one-line alternative

```
library(data.table)
help(data.table)
```

data.table

Pros

- Data tables are faster than data frames
  - What took 13 minutes with ddply took half a second with a data.table (not everything will be this big of an improvement)
- Data tables are also data frames (not everything breaks)
- Like the `**ply` statements, we can write compact and readable code
- Google uses data.tables (cool factor?)

Cons

- Data tables are definitely harder to use... at first

Getting started I

Creating a data.table

1. Creating a data table

```
## start with a data frame
df <- data.frame(x = rep(c("a", "b", "c"), each = 3), y = c(1,
    3, 6), z = 1:9)

## convert to a data table
dt <- data.table(df)
```

2. Functions used with data frames still work, but might look a little different

```
head(dt)
```

```
##    x y z
## 1: a 1 1
## 2: a 3 2
## 3: a 6 3
## 4: b 1 4
## 5: b 3 5
## 6: b 6 6
```

Getting started II

Creating a data.table

3. Indexing is a bit different

```
## Wrong way
dt[, 1]
```

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

```
dt[, "x"]
```

```
## [1] "x"
```

```
## Right way
dt[, x]
```

```
## [1] a a a b b b c c c
## Levels: a b c
```

```
dt[1:2, ]
```

```
##    x y z
## 1: a 1 1
## 2: a 3 2
```

Getting started III

Creating a data.table

But we can force data table to act more like the data frames

```
dt[, 1, with = FALSE]
```

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

Keys I

- Data frames have a single row name
- Data tables can have many names for a single row called a key
  - Each data table can only have 1 key
  - The data table is sorted by the key
- To see what keys are set we type

```
tables()
```

```
##      NAME NROW MB COLS  KEY
## [1,] dt      9 1  x,y,z
## Total: 1MB
```

Keys II

Keys allow for easy subsetting

```
setkey(dt, x)
tables()
```

```
##      NAME NROW MB COLS  KEY
## [1,] dt      9 1  x,y,z x
## Total: 1MB
```

```
dt["b", ]
```

```
##    x y z
## 1: b 1 4
## 2: b 3 5
## 3: b 6 6
```

Fast grouping I

Let's make a larger data table

```
grpsize <- ceiling(10e6 / 26^2)
DF <- data.frame(x = rep(LETTERS,each = 26 * grpsize),
                 y = rep(letters, each = grpsize),
                 v = runif(grpsize * 26^2),
                 stringsAsFactors = FALSE)
dim(DF)
```

```
## [1] 10000068        3
```

```
DT <- data.table(DF)
setkey(DT, x, y)
tables()
```

```
##      NAME       NROW  MB COLS  KEY
## [1,] dt            9   1 x,y,z x
## [2,] DT   10,000,068 229 x,y,v x,y
## Total: 230MB
```

Fast grouping II

```r
DT[, sum(v)]
```

```
## [1] 5e+06
```

- The second argument in `DT[i, j]` is used for aggregation
- You can put one or more expressions here
- To aggregate by group use `by`

```r
head(DT[, sum(v), by = x])
```

```
##    x     V1
## 1: A 192568
## 2: B 192283
## 3: C 192592
## 4: D 192228
## 5: E 192256
## 6: F 192462
```

Fast grouping III

To sum by both groups we simply add `y` to `by`

```
DT[, sum(v), by = "x,y"]  ## no space!
```

```
##       x y   V1
##   1: A a 7382
##   2: A b 7384
##   3: A c 7366
##   4: A d 7413
##   5: A e 7453
##  ---
## 672: Z v 7425
## 673: Z w 7420
## 674: Z x 7401
## 675: Z y 7382
## 676: Z z 7381
```

Your Turn

Return to the corn production data, and convert

```
# Don't run this!
plyr.agg <- ddply(corn, .(Year, State, StateFIPS, County, CountyCode,
    Measurement), summarise, means = mean(Value))
```

into a data table expression.

More complex queries

```
DT[i, j, by]
```

i
: restrict attention to certain rows by stringing statements together using `&` (and); `|` (or)

j
: use `list()` to string together multiple aggregation statements

by
: use `list()` to include multiple groups or by putting everything in quotes with no spaces

More complex queries I

We have data from the Cook's County Sheriff's Office that tracks location

```r
load("crime.rda")
str(crime)
```

```
## 'data.frame': 19207 obs. of  11 variables:
##  $ charges_citation       : chr  "720 ILCS 5 12-3.4(a)(2) [16145" "6
##  $ race                   : chr  "WH" "LW" "BK" "BK" ...
##  $ age_at_booking         : int  26 37 18 32 49 26 41 56 40 20 ...
##  $ gender                 : chr  "M" "M" "M" "F" ...
##  $ booking_date           : Date, format: "2013-01-20" ...
##  $ jail_id                : chr  "2013-0120171" "2013-0120170" "2013
##  $ bail_status            : chr  NA NA NA NA ...
##  $ housing_location       : chr  "05-" "05-" "05-L-2-2-1" "17-WR-N-A
##  $ charges                : chr  NA NA NA NA ...
##  $ bail_amount            : int  5000 10000 5000 50000 5000 5000 250
##  $ discharge_date_earliest: Date, format: NA ...
```

More complex queries II

```r
crime <- as.data.table(crime)

## A quick table
crime[, .N, by = bail_status]
```

```
##         bail_status     N
## 1:               NA 10318
## 2:          NO BOND  8086
## 3: Bond in Process   801
## 4:      10000000.00     1
## 5:      25000000.00     1
```

```r
crime[!is.na(bail_status), .N, by = bail_status]
```

```
##         bail_status    N
## 1:          NO BOND 8086
## 2: Bond in Process  801
## 3:      10000000.00    1
## 4:      25000000.00    1
```

Your turn

Create a summary of the Cook County data by race, gender, and age at booking. Include summaries for

- count
- average, standard deviation, minimum, and maximum bail

Tip: exclude cases with `NA`s in the calculations