Using `data.table`.

Erik Swanson

November 4, 2013

Outline

Section 1

Better than data.frame

Going Faster

names	graduated	gpa
Michael	2000	2.9
Jayden	1999	2.5
Emily	2000	3.6
William	1999	3.8
Olivia	2000	3.4
Emma	2000	3.2
Madison	1999	3.9
Abigail	1999	3.7
Liam	2000	2.1
Jacob	1999	2.2
Isabella	1999	3.3
Mason	2000	2.6
Elizabeth	2000	4.0
Sophia	1999	3.1
Alexander	1999	3.3
Ethan	1999	3.0
Aiden	2000	3.7
Mia	2000	3.8
Noah	2000	3.4
Ava	1999	3.5

Going Faster

names	graduated	gpa
Jacob	1999	2.2
Jayden	1999	2.5
Ethan	1999	3.0
Sophia	1999	3.1
Alexander	1999	3.3
Isabella	1999	3.3
Ava	1999	3.5
Abigail	1999	3.7
William	1999	3.8
Madison	1999	3.9
Liam	2000	2.1
Mason	2000	2.6
Michael	2000	2.9
Emma	2000	3.2
Noah	2000	3.4
Olivia	2000	3.4
Emily	2000	3.6
Aiden	2000	3.7
Mia	2000	3.8
Elizabeth	2000	4.0

Section 2

Basic Syntax

Create a data.table

- Use the fuction `data.table`

Create a data.table

- Use the fuction `data.table`
- Syntax just like `data.frame`

Create a data.table

- Use the fuction `data.table`
- Syntax just like `data.frame`

```
> library(data.table)
> my.dt <- data.table(maj=rep(c("math","art"), 100),
+                     yrs.exp=rep(c(1:10), 20),
+                     salary=10000
+                      * rep(c(2,1), 100)
+                      * rep(seq(1.1, 2, .1), 20)
+                      * rgamma(200,shape=4,scale=.75)
+                    )
```

Setting Keys & Reviewing Your Data Tables

- Speedup from sorting comes by setting “keys”

Setting Keys & Reviewing Your Data Tables

- Speedup from sorting comes by setting "keys"

```
> setkey(my.dt, maj, yrs.exp)
> key(my.dt)

[1] "maj"     "yrs.exp"
```

Setting Keys & Reviewing Your Data Tables

- Speedup from sorting comes by setting "keys"

```
> setkey(my.dt, maj, yrs.exp)
> key(my.dt)

[1] "maj"     "yrs.exp"
```

- If you ever want to see all your tables and keys.

Setting Keys & Reviewing Your Data Tables

- Speedup from sorting comes by setting "keys"

```
> setkey(my.dt, maj, yrs.exp)
> key(my.dt)
[1] "maj"     "yrs.exp"
```

- If you ever want to see all your tables and keys.

```
> tables()
     NAME              NROW MB COLS                  KEY
[1,] my.dt              200 1  maj,yrs.exp,salary maj,yrs.exp
[2,] res.j              200 1  maj,yrs.exp,diff
[3,] result.i.scan       20 1  maj,yrs.exp,salary maj,yrs.exp
[4,] result.i.search     20 1  maj,yrs.exp,salary maj,yrs.exp
Total: 4MB
```

Arguments

- A `data.table` uses ‘`[`‘ like a `data.frame`
- There are a number of different arguments to ‘`[`‘
- Going to talk about `i, j, by`

Arguments

- A `data.table` uses ‘`[`‘ like a `data.frame`
- There are a number of different arguments to ‘`[`‘
- Going to talk about `i`, `j`, `by`

```
> my.dt["art", mean(salary), by=yrs.exp]

   yrs.exp       V1
1:       2 44194.87
2:       4 47928.68
3:       6 46805.52
4:       8 64216.43
5:      10 79742.68
```

Argument: `i`

- Choose which rows you're working with.
- Can take a logical vector just like the first argument in a dataframe.

Argument: i

- Choose which rows you're working with.
- Can take a logical vector just like the first argument in a dataframe.

```
> result.i.scan <- my.dt[maj=="math" & yrs.exp==7, ]
> head(result.i.scan)

    maj yrs.exp    salary
1: math       7  94742.87
2: math       7 156974.10
3: math       7 144571.56
4: math       7  38946.83
5: math       7  77173.01
6: math       7  82694.64
```

- This is still a linear scan. Avoid this!

Argument: `i`

- Use another `data.table` for `i`.

Argument: i

- Use another data.table for i.
- Or use the J shortcut function to lookup by keys.

```
> result.i.search <- my.dt[J("math", 7), ]
> head(result.i.search)

    maj yrs.exp    salary
1: math       7  94742.87
2: math       7 156974.10
3: math       7 144571.56
4: math       7  38946.83
5: math       7  77173.01
6: math       7  82694.64

> identical(result.i.scan, result.i.search)

[1] TRUE
```

Argument: `j`

- Choose which columns to display.

Argument: `j`

- Choose which columns to display.
- Takes a `list` of column names, or functions of column names

Argument: j

- Choose which columns to display.
- Takes a `list` of column names, or functions of column names

```
> m <- mean(my.dt$salary)
> res.j <- my.dt[,list(maj, yrs.exp, diff=salary - m)]
> head(res.j)

   maj yrs.exp      diff
1: art       2 -28703.30
2: art       2 -45414.34
3: art       2 -45480.56
4: art       2  29356.18
5: art       2 -35876.52
6: art       2 -31431.88
```

Argument: by

- Lets you group by any combination of keys.

Argument: by

- Lets you group by any combination of keys.
- Any function in the `j` argument will be performed on each set of grouped values.

Argument: by

- Lets you group by any combination of keys.
- Any function in the `j` argument will be performed on each set of grouped values.
- Much faster than packages such as `plyr`.

Argument: by

```
> my.dt[ , list(mean=mean(salary), sd=sd(salary)),
+           by="maj,yrs.exp"]
```

	maj	yrs.exp	mean	sd
1:	art	2	44194.87	22475.34
2:	art	4	47928.68	20104.82
3:	art	6	46805.52	24487.78
4:	art	8	64216.43	40519.32
5:	art	10	79742.68	42918.28
6:	math	1	64513.57	25183.13
7:	math	3	62735.37	31906.28
8:	math	5	79092.90	41003.04
9:	math	7	101123.27	54486.12
10:	math	9	105876.22	68535.64

Fast Updates

- The operator `:=` in the `j` argument causes side effects.

Fast Updates

- The operator `:=` in the `j` argument causes side effects.
- Fast and expressive.

Fast Updates

- The operator `:=` in the `j` argument causes side effects.
- Fast and expressive.
- Works with the `by` argument.

Fast Updates

- The operator `:=` in the `j` argument causes side effects.
- Fast and expressive.
- Works with the `by` argument.

```
> my.dt[, maj.sal.sd := sd(salary), by=maj]

> head(my.dt)

   maj yrs.exp   salary maj.sal.sd
1: art       2 40919.65   33819.94
2: art       2 24208.61   33819.94
3: art       2 24142.39   33819.94
4: art       2 98979.13   33819.94
5: art       2 33746.43   33819.94
6: art       2 38191.07   33819.94
```

Section 3

In Action

Speed Demo

```
> df <- read.csv("~/current_playlist_2006-2012.csv",
                 stringsAsFactors = FALSE)
> nrow(df)
[1] 805604
> names(df)
[1] "id"   "date"   "title"  "artist"   "starrating"
> system.time(ddply(df,
                    .(artist, title),
                    function(df) { nrow(df) })
              )
   user  system elapsed
 16.712   0.334  17.570
```

Speed Demo

```
> dt <- data.table(df)
> setkey(dt, artist, title)
> system.time(dt[, list(nrow = .N), by="artist,title"])
   user  system elapsed
  0.095   0.005   0.101
> 17.57 / .101
[1] 173.9604
```