Data Table Lecture

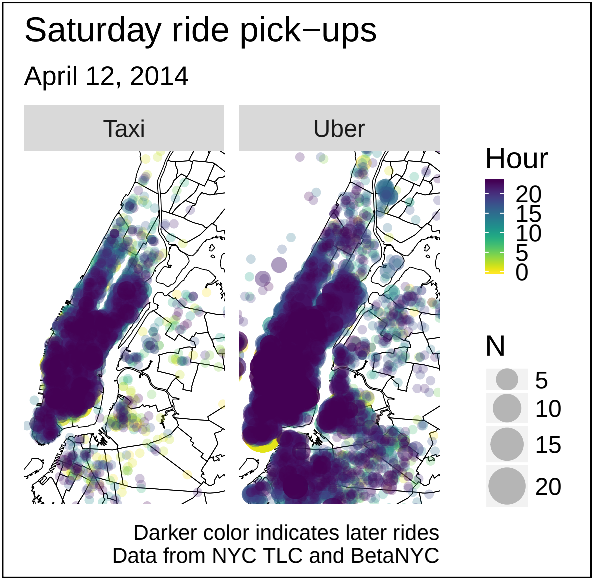
Ian de Medeiros

November 27, 2017

How far does Uber driving get you?

- Uber advertised as great money-making solution, is that true
  - What days and times are most profitable to drive
- Use data.table to work with large data sets and create maps to investigate profitability of Uber

Uber vs Taxi: Saturday



Introduction: data.table Overview

- Allows for fast sub-setting, querying, and manipulating of data
  - Similar to dplyr
  - Simple and consistent syntax
- Fast and efficient at reading and processing data
- More info can be found here: https://github.com/Rdatatable/data.table/wiki/Getting-started

Introduction: Simple Example

```
DT = data.table(x=letters[1:10], y=1:10, z=rep( c("odd", "e
DF = as.data.frame(DT)

#1.Basic aggregation
DT[, sum(y), by = z]                                # data
DF %>% group_by(z) %>% summarise(sum(y))            # dply

#2.Basic update operation
DT[, y := cumsum(y), by = z]
DF %>% group_by(z) %>% mutate(y = cumsum(y))
```

Reading in data

- `fread(input, ...)` quickly reads in large data sets
  - `input` may be quoted file path or URL
- Other helpful arguments include
  - `sep` — may specify character delimiter in file; generally not needed
  - `nrows` — limit number of rows read in
  - `na.strings` — specify character vector used to determine missing values
  - `colClasses` — tell whether column is character or numeric
  - `select`/`drop` — subset the data using vector of column names or column numbers

Taxi data

- Data obtained from NYC Taxi and Limousine Commission under Freedom of Information Act
  - FiveThirtyEight provides data on their *github site*
- Include code cookbook for NYC Yellow cab data can be found *here*
- NAs are coded as blank characters
- Using a random sample of 200,000 rides in April, 2014

```
yellow.all <- fread(paste0(path, "yellow_ss_2014-04.csv"),
head(yellow.all, n = 3)
yellow.all[, V1 := NULL]
```

Subsetting the data with i

- Basic structure of a data.table — `DT[i, ...]`
- `i` takes row sub-setting operations
  - Identical to basic row sub-setting with additional options
- Do not need to quote column names or use $ notation
- For example:

```
# only want first 1000 observations
yellow.all[1:1000,]

# only want people who paid with a credit card
yellow.all[payment_type=="CRD",]

# only want people who tipped with a credit card
yellow.all[payment_type=="CRD" & tip_amount > 0,]
```

- What `dplyr` verb could we use to subset the data?

Using Keys

- Filtering uses scan and sort method — can take long on unsorted data
- Setting key value sorts data.table by key
  - Sub-setting on key columns much faster
  - `setkey()` function
  - Numeric variables may require additional wrapper functions

```
setkey(yellow.all,payment_type)
yellow.all[J("CRD")]
```

Selecting Columns with j

- `DT[i, j, ...]`
- `j` works by taking a **list** of column names
  - Do not need to put names in quotes
  - Must use `J()`, `list()`, or `.()`
  - `DT[, .(col1, col2, ...)]`
- May mix `i` and `j` to further subset

```
# only keep payment type and fare amount
yellow.all[, .(payment_type, fare_amount)]
```

In class Exercise 1:

- Create a data.frame of only payment type and fare amount for rides that cost more than $10 in **both** data.table and dplyr

With argument

- What if we want to pass a vector of column numbers instead of column names?
  - Can't do this in data.table
  - CAN do this in data.frame
- Setting `with` = FALSE regains data.frame functionality
  - `DT[i,j, with = , ...]`
  - Does not get rid of key functionality
  - May pass column names as quoted strings
  - `with` = TRUE is default

```
yellow.all[, 2:5, with = FALSE]

yellow.all[.("CRD"), 2:5, with = FALSE]
```

Renaming and mutating variables with j

- Do not need to memorize any verbs in order to mutate

```
yellow.all[,
    .(pickup_time = ymd_hms(pickup_datetime, tz = "EST"),
      dist = trip_distance, fare = fare_amount)]
```

Taxi Data at a Glance

- NYC is around 40 latitude and -80 longitude
- Notice anything strange?

Table 1: Summary of Unemployment Rate

Statistic	Mean	St. Dev.	Min	Max
passenger_count	1.696	1.360	0	6
trip_distance	2.899	3.421	0.000	76.500
pickup_longitude	−72.494	10.364	−82.334	0.000
pickup_latitude	39.935	5.709	0.000	51.721
dropoff_longitude	−72.434	10.562	−86.783	0.000
dropoff_latitude	39.903	5.819	0.000	50.381
fare_amount	12.470	10.209	2.500	310.300
tip_amount	1.473	2.241	0.000	113.330

Final Sample Selection

- Only include taxi rides that are paid for by credit card
- Change lengthy variable names
- Drop:
  - `store_and_forward` entries
  - Trips with no distance traveled
  - Trips with 0 longitude or 0 latitude
  - Trips with 0 passengers
- **In-Class Exercise #2**: Translate dplyr code for final sample into data.table syntax

Updating Data in Place

- Can either change change values of existing column or add new column in place
- Columns are added directly to the new data set

```
# Create a rounded tip variable
yellow.all[, tip_whole := round(tip_amount)]
```

- may also update multiple columns at once

```
yellow.all[, c("fare_whole","total_whole") := .(
  round(fare_amount),round(total_amount))]
```

In-Class Exercise #3:

- Fill in the code below and create four new columns:
  - A dummy indicating the hour of the day
  - A dummy indicating the week of the year
  - A variable for the total time of the trip
  - A dummy indicating the day of the week
- Hint: Recall the properties of date objects and the lubridate package

```
day_names <- c("Monday","Tuesday","Wednesday",
               "Thursday","Friday","Saturday","
               Sunday")
yellow.ss <- as.data.table(yellow.ss)
yellow.ss[, c( , , , ) := .( , ,
    as.numeric( , unit = "secs")/60,
    factor( ,levels = day_names,
           labels = day_names))]
```

Summarizing the Data

- Can also use j option to summarize data
  - Use summary functions **without assigning** them to new variables

```
# what is the average distance (in miles) and time
# (in minutes) of trips in april?
  yellow.ss[,.(mean(dist), mean(trip_time))]
```

- by = argument allows us to group variables and perform calculations
  - DT[i,j, by =, ...]

```
# what is the average distance (in miles) and time
# (in minutes) of trips in april by weekday?
 setkey(yellow.ss, weekday, hour)
 yellow.ss[,.(avg_distance = mean(dist),
              avg_time = mean(trip_time)),
          by = .(weekday)]
```

In-Class Exercise #4:

- Perhaps people tip better later in the evening?
- Translate the following code into dplyr:

```
yellow.ss[hour >= 22 | hour < 2,
          .(mean(fare), mean(tip),
            mean(tip/fare)),
          by = .(weekday, hour)]
```

The .N argument

- Special argument within the `j` argument
- Corresponds to total number of rows for group that is passed through
  - Similar to `n()` function in `dplyr`
  - Like `j`, can use `i` and `by` arguments

```
# number of rides on each weekday
  yellow.ss[,.(num_rides = .N), by=weekday]
# number of rides that did not tip
 yellow.ss[tip == 0,.(.N), by=weekday]
# recreate the average hourly fare, tips,
 yellow.ss[tip == 0,.(test = sum(fare)/.N), by=weekday]
```

Chaining Argument

- So far only single data.table commands, `DT[...]`
- Can utilize sequences of commands in single call
  - `DT[...][....][....]`
  - Each call executes on the previous
  - Very similar to %>%

```
# calculate the the average amount of
# gross earnings earnings per mile for
# trips over 2 miles
yellow.ss[,
          gross_earnings := fare + tip][,
          gepm := gross_earnings/dist][dist > 2,
         .(max(gepm))]
```

Are taxi rides profitable?

- In 2014 average maintenance were $0.06/mile
- Using average weekly gas prices can calculate mpg
  - Suppose each taxi is a Ford Crown Victoria with 17 mpg
- can use data.table to merge data tables with `on =` option
  - Default is to merge by key
  - `DT1[DT2, on = c("merge_col1","merge_col2", ...)]`
- columns used to merge do not have to have same name
  - `DT1[DT2, on = c("col1_DT1"="col1_DT2", "col2_DT1"="col2_DT2",...)]`

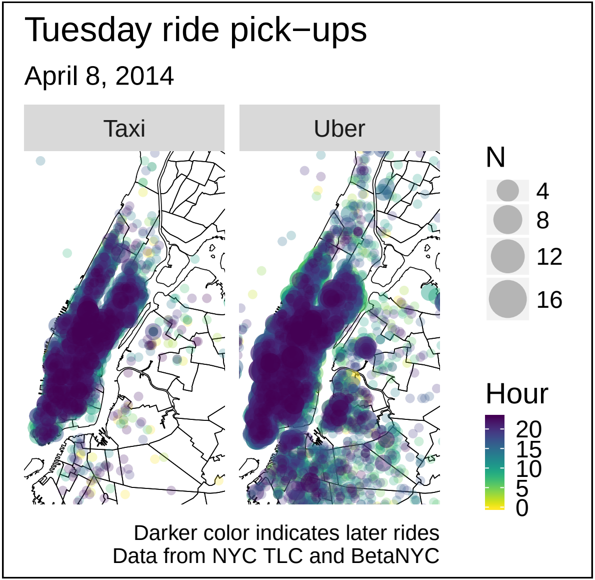
```
gas_prices <- read.csv(paste0(path, "nyc_gas_prices.csv"))
gas_prices <- as.data.table(gas_prices)

yellow.mpg <- yellow.ss[gas_prices, on = "week"]
```

- **In-Class Exercise 5:** use the merged data and `data.table` chains to:
  - Create a new variable net earnings (Hint: How should we

Mapping the data

- GPS coordinates provided by NYC TLC
  - Pinpoint most frequented and profitable location
- Use downloaded shape file to create base of map
  - `ggmap` downloads maps directly from Google Maps API

Uber vs Taxis: Tuesday



In-Class Exercise 6:

- Create a map that compares the number and time of Taxi cab rides on the average Tuesday to the number versus the number of Taxi cab rides on the average Saturday

In-Class Exercise 7:

- Create a map that plots the most profitable locations for Taxi cap pick up
- *Hint*: you will use one less aes option