SQL for R Users

Jae Yeon Kim

Contents

- Special thanks to Jacob Coblnetz (@Jacob_Coblnetz) for sharing his slides on the SQL workshop used at MIT.

Motivation

1. Designed vs. Found Data (Salganik 2017)

- Designed (e.g., Survey data, Experimental data; Small/medium size) vs.
- Found Data (e.g., Administrative data, Corporate data; Often Large)

2. The varieties of the datasets (Pradeep and Moy 2015):

- Small (what most of you have worked with)

- Medium (1-2 GB)
- Large (2 - 10 GB)
- Very large (> 10 GB)

3. Recipes for big data:

- Slice and dice: `read.csv("file_address", nrows = 20` or `data.table::fread()`
- Parallel processing: (`partition`, `summarize`, and `collect` from the `multidplyr` package)
- Multiple sessions: `bigmemory::read.big.matrix()` or `ff::read.csv.ffdf()`
- Distributed filesystem: RHadoop. Almost an only option for a dataset larger than 5TB.
- SQL: Extracting data from a database. SQL is a staple tool in the non-academic world to perform this task. Key idea: Local dataframe -> Database. The focus of today's workshop

What is SQL?

- Structured Query Language. Called SEQUEL and developed by IBM Corporation in the 1970s
- Remains the standard language for a relational database management system.
- It's a DECLARATIVE language (compute what you want to compute not how to compute it)
- Its main job is to define and query databases (i.e., two-dimensional tables).
- Great for keeping data type integrity, updating data frequently, joining different data sources, and doing quick data analyses

Learning objectives

  - Embracing a new mindset: from ownership (opening CSVs in your laptop) to access (accessing data stored in the database)
  - Learning how to access and query a database in R in a tidy way
- SQL and R

SQL	R
SELECT	select() for columns, mutate() for expressions, summarise() for aggregates
FROM	which data frame
WHERE	filter()
GROUP BY	group_by()
HAVING	filter() after group_by()
ORDER BY	arrange()
LIMIT	head()

Setup

- `pacman::p_load()` reduces steps for installing and loading several packages simultaneously.

```
rm(list = ls())

if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```
pacman::p_load(
 tidyverse, # tidyverse packages
 conflicted, # an alternative conflict resolution strategy
 pyrr, # for inspecting memory allocations
 dbplyr, # to use database with dplyr
 DBI, # for using SQL queries
 RSQLite, # for SQLite
 odbc, # backend engine; open data connectivity driver
 RPostgres, # PostgreSQL
 sqldf, # for running SQL in R
 tidyquery, # sqldf alternative
 nycflights13 # for test data
)
```

```
## Installing package into '/home/jae/R/x86_64-pc-linux-gnu-library/3.6'
## (as 'lib' is unspecified)
```

```
## Warning: package 'pyrr' is not available (for R version 3.6.3)
```

```
## Warning in p_install(package, character.only = TRUE, ...):
```

```
## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'pyrr'
```

```
## Warning in pacman::p_load(tidyverse, conflicted, pyrr, dbplyr, DBI, RSQLite, : Failed to install/loa
## pyrr
```

```
# Resoving conflicting functions
conflict_prefer("filter", "dplyr")
```

```
## [conflicted] Will prefer dplyr::filter over any other package
```

```
conflict_prefer("sql", "dplyr")
```

```
## [conflicted] Will prefer dplyr::sql over any other package
```

Data sets

We use the flight on-time performance data from the Bureau of Transpiration Statistics of the U.S. government. The data goes back to 1987 and its size is more than 20 gigabytes. For practice, we only use a small subset of the original data (flight data departing NYC in 2013) provided by RStudio.

Connect to the database

- This part draws heavily on the `dbplyr` package vignette.
- The `DBI` package provides a client-side interface that allows `dplyr` to work with databases. DBI is automatically installed when you installed `dbplyr`. However, you need to install a specific backend engine (a tool for communication between R and a database management system) for the database (e.g., `RMariaDB`, `RPostgres`, `RSQLite`, `obdc`, `bigrquery`). In this workshop, we use SQLite because it is the easiest to get started with. Personally, I love PostgreSQL because it's an open-source and also powerful to do many amazing things (e.g., text mining, geospatial analysis).
- If you want to connect to the database not manually, you can use the Connections interface in RStudio.
- Here's some information on the historical background of the package.

```
# Define a backend engine
```

```
drv <- RSQLite::SQLite()

# Create an empty in-memory database

con <- DBI::dbConnect(drv,
                      dbname = ":memory:")

#con <- DBI::dbConnect(RMariaDB::MariaDB(),
 # host = "database.rstudio.com",
 # user = "hadley",
 # password = rstudioapi::askForPassword("Database password")
#)

# Copy a local data frame to a DBI backend

copy_to(dest = con, # remote data source
        df = flights) # a local dataframe

copy_to(dest = con, # remote data source
        df = airports) # a local dataframe

# Note that we didn't load the data.

src_dbi(con)
```

```
## src:  sqlite 3.30.1 [:memory:]
## tbls: airports, flights, sqlite_stat1, sqlite_stat4
```

Show the list of tables.

```
# Return the name of the tables
dbListTables(con)
```

```
## [1] "airports"     "flights"      "sqlite_stat1" "sqlite_stat4"
```

Query using dbplyr

- The `tbl` object is lazily evaluated; It doesn't pull the data until you explicitly ask for it.

```
# Select all columns from flights table and show the first ten rows

dbGetQuery(con, "SELECT * FROM flights;") %>%
  head(10)
```

```
##    year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
## 1  2013     1   1      517            515         2      830            819
## 2  2013     1   1      533            529         4      850            830
## 3  2013     1   1      542            540         2      923            850
## 4  2013     1   1      544            545        -1     1004           1022
## 5  2013     1   1      554            600        -6      812            837
## 6  2013     1   1      554            558        -4      740            728
## 7  2013     1   1      555            600        -5      913            854
## 8  2013     1   1      557            600        -3      709            723
## 9  2013     1   1      557            600        -3      838            846
## 10 2013     1   1      558            600        -2      753            745
##    arr_delay carrier flight tailnum origin dest air_time distance hour minute
```

```
## 1         11      UA   1545  N14228    EWR  IAH      227     1400    5     15
## 2         20      UA   1714  N24211    LGA  IAH      227     1416    5     29
## 3         33      AA   1141  N619AA    JFK  MIA      160     1089    5     40
## 4        -18      B6    725  N804JB    JFK  BQN      183     1576    5     45
## 5        -25      DL    461  N668DN    LGA  ATL      116      762    6      0
## 6         12      UA   1696  N39463    EWR  ORD      150      719    5     58
## 7         19      B6    507  N516JB    EWR  FLL      158     1065    6      0
## 8        -14      EV   5708  N829AS    LGA  IAD       53      229    6      0
## 9         -8      B6     79  N593JB    JFK  MCO      140      944    6      0
## 10         8      AA    301  N3ALAA    LGA  ORD      138      733    6      0
##     time_hour
## 1  1357034400
## 2  1357034400
## 3  1357034400
## 4  1357034400
## 5  1357038000
## 6  1357034400
## 7  1357038000
## 8  1357038000
## 9  1357038000
## 10 1357038000
```

```
# Select dep_delay and arr_delay from flights table and show the first ten rows

dbGetQuery(con, "SELECT dep_delay, arr_delay FROM flights;") %>%
  head(10)
```

```
##    dep_delay arr_delay
## 1          2        11
## 2          4        20
## 3          2        33
## 4         -1       -18
## 5         -6       -25
## 6         -4        12
## 7         -5        19
## 8         -3       -14
## 9         -3        -8
## 10        -2         8
```

```
# Select dep_delay and arr_delay from flights table, show the first ten rows, then turn the result into

dbGetQuery(con, "SELECT dep_delay, arr_delay FROM flights;") %>%
  head(10) %>%
  as.tibble()
```

```
## Warning: `as.tibble()` is deprecated as of tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## # A tibble: 10 x 2
##    dep_delay arr_delay
##        <dbl>     <dbl>
##  1         2        11
##  2         4        20
```

```
##  3         2        33
##  4        -1       -18
##  5        -6       -25
##  6        -4        12
##  7        -5        19
##  8        -3       -14
##  9        -3        -8
## 10        -2         8
```

Tidy-way: dplyr -> SQL

One of the recent developments in the tidyverse. Working with a database using the `dplyr` syntax.

These examples are from the vignette of the `dbplyr` package.

```
# tbl select tables
flights <- con %>% tbl("flights")
airports <- con %>% tbl("airports")
```

select = SELECT

```
# Set to dplyr
conflict_prefer("filter", "dplyr")
```

```
## [conflicted] Removing existing preference
```

```
## [conflicted] Will prefer dplyr::filter over any other package
```

```
flights %>%
  select(contains("delay")) %>%
  show_query()
```

```
## <SQL>
## SELECT `dep_delay`, `arr_delay`
## FROM `flights`
```

mutate = SELECT AS

```
flights %>%
  select(distance, air_time) %>%
  mutate(speed = distance / (air_time / 60)) %>%
  show_query()
```

```
## <SQL>
## SELECT `distance`, `air_time`, `distance` / (`air_time` / 60.0) AS `speed`
## FROM `flights`
```

filter = WHERE

```
flights %>%
  filter(month == 1, day == 1) %>%
  show_query()
```

```
## <SQL>
## SELECT *
```

```
## FROM `flights`
## WHERE ((`month` = 1.0) AND (`day` = 1.0))
```

- Note that R and SQL operators are not exactly alike. R uses `!=` for `Not equal to`. SQL uses `<>` or `!=`. Also, some of SQL comparison operators are more intuitive than their R counterparts (`WHERE student_ID BETWEEN 1 and 100` `WHERE first_name LIKE 'Jae'`).

arrange = ORDER BY

```
flights %>%
  arrange(carrier, desc(arr_delay)) %>%
  show_query()
```

```
## <SQL>
## SELECT *
## FROM `flights`
## ORDER BY `carrier`, `arr_delay` DESC
```

summarise = SELECT AS and group by = GROUP BY

```
flights %>%
  group_by(month, day) %>%
  summarise(delay = mean(dep_delay)) %>%
  show_query()
```

```
## <SQL>
```

```
## Warning: Missing values are always removed in SQL.
## Use `mean(x, na.rm = TRUE)` to silence this warning
## This warning is displayed only once per session.
```

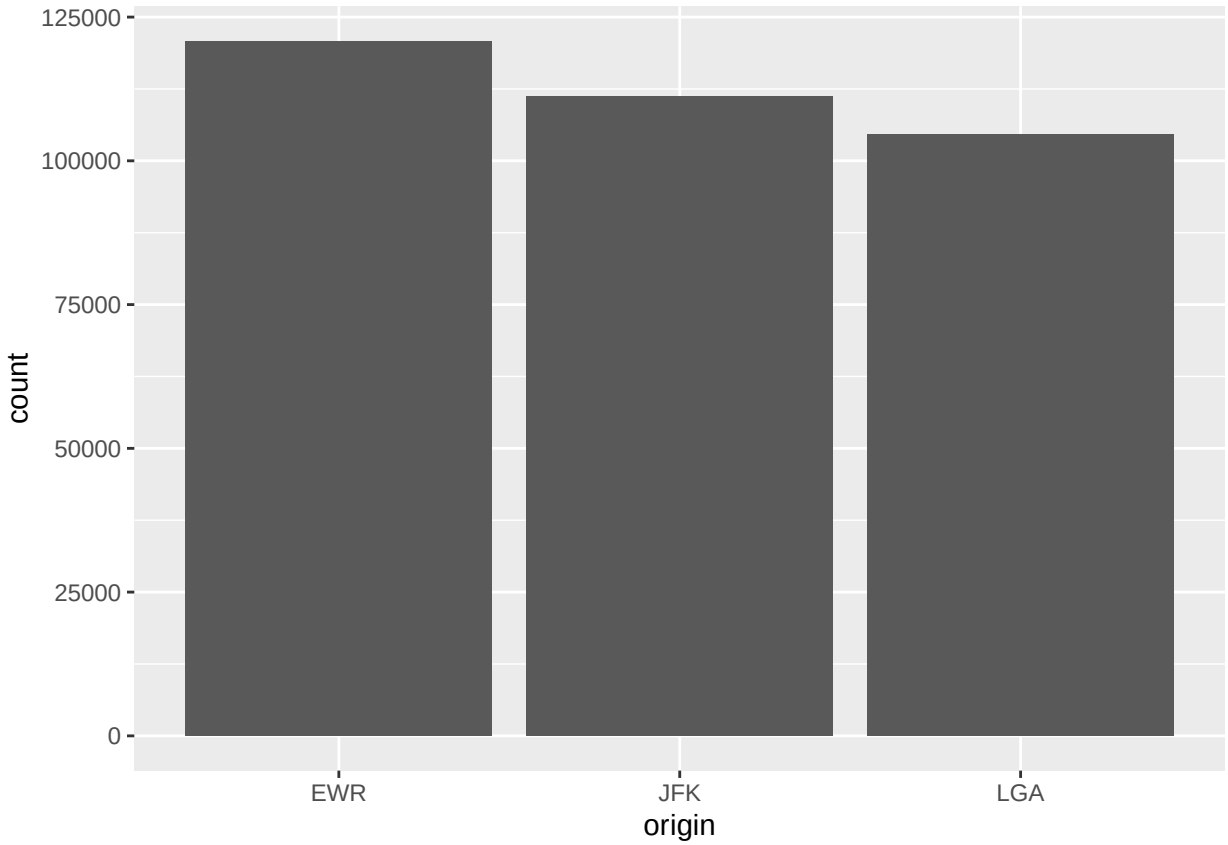
```
## SELECT `month`, `day`, AVG(`dep_delay`) AS `delay`
## FROM `flights`
## GROUP BY `month`, `day`
```

Data visualization

This part is from RStudio's DB best practices.

- A typical ggplot2

```
ggplot(flights) +
  geom_bar(aes(x = origin), stat = "count")
```

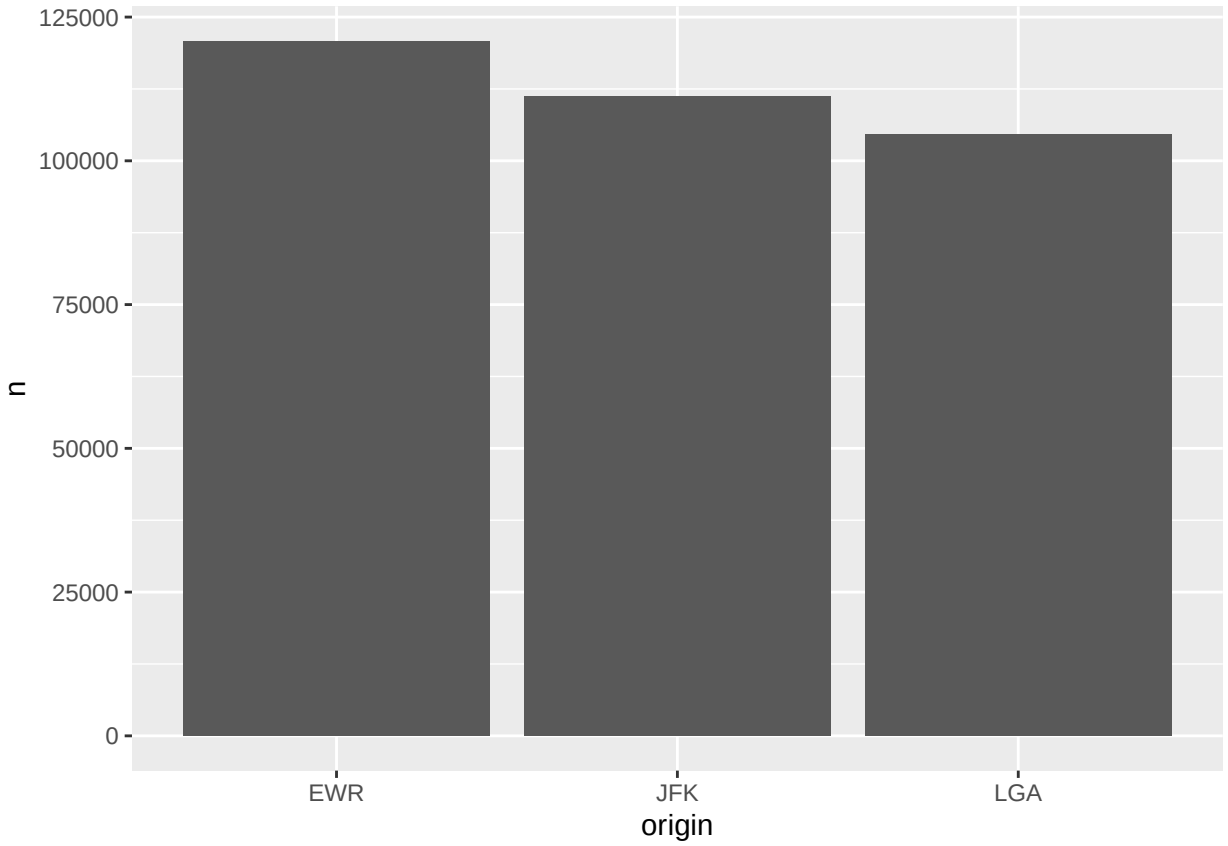


- Best practice: transforming data -> plotting results in R
- `collect()` is used to pull the data. Depending on the data size, it may take a long time to run.

```
df <- flights %>%
  group_by(origin) %>%
  tally() %>%
  collect()

# Shifted from geom_bar() to geom_col() because the heights of bar plots were calculated by tally()

ggplot(df) +
  geom_col(aes(x = origin, y = n))
```



SQL-way: SQL -> dplyr

`sqldf` has ben used to use SQL code in R. `tidyquery` is a good alternative because it has fewer dependencies.

`select` = `SELECT`

```
# Now switch to dbplyr
conflict_prefer("sql", "dbplyr")
```

```
## [conflicted] Removing existing preference
```

```
## [conflicted] Will prefer dbplyr::sql over any other package
```

```
# sqldf

con %>% tbl(sql("SELECT dep_delay, arr_delay FROM flights"))
```

```
## # Source:   SQL [?? x 2]
## # Database: sqlite 3.30.1 [:memory:]
##    dep_delay arr_delay
##        <dbl>     <dbl>
##  1         2        11
##  2         4        20
##  3         2        33
##  4        -1       -18
##  5        -6       -25
##  6        -4        12
```

```
##  7        -5        19
##  8        -3       -14
##  9        -3        -8
## 10        -2         8
## # ... with more rows
```

```
# tidyquery

query("SELECT dep_delay, arr_delay FROM flights")
```

```
## # Source:   lazy query [?? x 2]
## # Database: sqlite 3.30.1 [:memory:]
##    dep_delay arr_delay
##        <dbl>     <dbl>
##  1         2        11
##  2         4        20
##  3         2        33
##  4        -1       -18
##  5        -6       -25
##  6        -4        12
##  7        -5        19
##  8        -3       -14
##  9        -3        -8
## 10        -2         8
## # ... with more rows
```

```
# In case, if you want to translate the SQL code into tidyverse, then try the following code:
# parse_query("SELECT dep_delay, arr_delay FROM flights", tidyverse = TRUE)
```

mutate = SELECT AS

Also, note that you can combine `sqldf` and `dplyr`.

```
# sqldf
con %>% tbl(sql("SELECT distance, air_time, distance / air_time / 60.0 AS speed
FROM flights")) %>%
  arrange(desc(air_time))
```

```
## # Source:     SQL [?? x 3]
## # Database:   sqlite 3.30.1 [:memory:]
## # Ordered by: desc(air_time)
##    distance air_time speed
##       <dbl>    <dbl> <dbl>
##  1     4963      695 0.119
##  2     4983      691 0.120
##  3     4983      686 0.121
##  4     4983      686 0.121
##  5     4983      683 0.122
##  6     4983      679 0.122
##  7     4963      676 0.122
##  8     4983      676 0.123
##  9     4983      675 0.123
## 10     4963      671 0.123
## # ... with more rows
```

filter = WHERE

```
con %>% tbl(sql("
SELECT *
FROM flights
WHERE month = 1.0 AND day = 1.0
"))
```

```
## # Source:   SQL [?? x 19]
## # Database: sqlite 3.30.1 [:memory:]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dbl>
```

arrange = ORDER BY

```
con %>% tbl(sql("
SELECT *
FROM flights
ORDER BY carrier, arr_delay DESC
"))
```

```
## # Source:   SQL [?? x 19]
## # Database: sqlite 3.30.1 [:memory:]
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     2    16      757           1930       747     1013           2149
##  2  2013     7    24     1525            815       430     1808           1030
##  3  2013     7    10     2054           1459       355      102           1801
##  4  2013    11    27     1503            815       408     1628            952
##  5  2013    12    14     1425            825       360     1604            938
##  6  2013     2    27     1529            845       404     1639           1015
##  7  2013     7    22     2216           1620       356      116           1853
##  8  2013     6    25     1421            805       376     1602            950
##  9  2013     1    25       15           1815       360      208           1958
## 10  2013     3     1     1449            855       354     1701           1104
## # ... with more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dbl>
```

summarise = SELECT AS and group by = GROUP BY

```
con %>% tbl(sql("
SELECT month, day, AVG(dep_delay) AS delay
FROM flights
GROUP BY month, day
"))
```

```
## # Source:   SQL [?? x 3]
## # Database: sqlite 3.30.1 [:memory:]
##    month   day delay
##    <int> <int> <dbl>
##  1     1     1 11.5
##  2     1     2 13.9
##  3     1     3 11.0
##  4     1     4  8.95
##  5     1     5  5.73
##  6     1     6  7.15
##  7     1     7  5.42
##  8     1     8  2.55
##  9     1     9  2.28
## 10     1    10  2.84
## # ... with more rows
```

What we can't do

Check out the issue section of `queryparser` and that of `tidyquery` to see the latest developments.

Limitations

```
- Subqueries
- Unions
- Implict join notation
- Joins of three plus tables
- WITH clause
- OVER expressions
- Non-ASCII characters in queries
```

Non-Goals

```
- Translate other types of SQL statements (such as `INSERT` or `UPDATE`) and other more complex tasks (
```

References

- R Studio, Database using R
- Ian Cook, "Bridging the Gap between SQL and R" rstudio::conf 2020 slides
- Data Carpentry contributors, SQL database and R, Data Carpentry, September 10, 2019.
- Introduction to dbplyr
- Benjamin S. Baumer, Daniel T. Kaplan, and Nicholas J. Horton, Moden Data Science with R, 2nd ed., CRC Press, 2020-01-03.
- Josh Erickson, SQL in R, STAT 701, University of Michigan
- Deborah Nolan, STAT 133 class notes, University of California, Berkeley, Fall 2005
- Kane, Michael J, "Strategies for Exploring a 12 Gigabyte Data Set: Airline Flight Delays," Invited book chapter in Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving, 2015.

- Kane, Michael J., John Emerson, and Stephen Weston. "Scalable strategies for computing with massive data." Journal of Statistical Software 55.14 (2013): 1-19.
- Eduardo Arino de la Rubia, "Multicore Data Science with R and Python", Domino Data Lab, May 22, 2017.