Lesson 2 Lab: Data Wrangling with R & data.table

In this lab, we will get comfortable with an R package called data.table and explore how its syntax is fairly synonymous to SQL syntax. The goal of this lab is to show a way to perform SQL like operations using R syntax. There a few aspects of data.table package that make it unique:

1. **Extremely robust and fast file reading:** The fread function in data.table package allows us to read gigabityes of data in the order of minutes. This is significantly faster than any other file reading function available in R.
2. **SQL Like Syntax:** The syntax of data.table package can be synonymous to SQL in many ways making it easier for us to translate SQL queries in R native. This becomes particularly useful when we have large operations - In these cases, we could use R as the querying interface (using RODBC package) to **GET DATA** from databases and continue to wrangle or **TRANSFORM DATA** at high speeds from within R using data.table. This allows us to keep most of our GET DATA and TRANSFORM DATA steps within R.
3. **Fast Calculations:** A dataset in data.table package is called “datatable”. A datatable performs significantly faster than a regular data.frame for almost all mathematical operations. This is a result of various indexing and inherent storing techniques used in the background in data.table

The D[i, j, by] forms the core syntax of dtaa.table. This syntax is quite analogous to SQL in the following ways: i - This is WHERE condition j - This is where you define the SELECT statement by - This is where you define the GROUP BY statement

An example: Let’s say we have a dataset called dt which shows daily temperature (recorded at noon) for Seattle for year 2017. To get the average temperature for the month June and July , you will use the following data.table syntax in SQL:

SELECT month(Recording\_Date) as Month,  
 avg(Temp) as Avg\_Monthly\_Temp  
 FROM dt  
 WHERE month(Recording\_Date) in (5,6)  
 GROUP BY month(Recording\_Date)

In data.table, the syntax to achieve the same is:

dt[month(Recording\_Date) = 5 | month(Recording\_Date) = 6 ,   
 Avg\_Monthly\_Temp = mean(Temp),   
 by = month(Recording\_Date)]

Follow the steps to complete the R Lab.

**0. SETTING UP your R studio environment**

First, create a new project in R studio

Go to File>New Project

Select New Directory

New Project

Directory: L02HW

Click on Create project

Second, Create and R script

Go to File> New File> R script

Third, install and load the packages you will need.

install.packages("data.table")

install.packages("sqldf")

install.packages("curl")

library(data.table) # Loading `data.table` package

library(sqldf) # Loading `sqldf` package

library(curl) # Loading `curl` package

**1. GET Data**

Enter the location where you have saved the “Lab Data Files”" folder for Lesson 2 if you are using your own installation of R studio or use the course storage path if you are using R studio Virtual lab. This folder contains 3 csv files named 2017\_05\_FlightTimes.csv, 2017\_06\_FlightTimes.csv and 2017\_07\_FlightTimes.csv.

folder\_path = "/Users/viveksnh/Documents/Personal/ProjectUW/Lesson 2/Lab Data Files/"

or

for virtual lab users

folder\_path = "https://library.startlearninglabs.uw.edu/DATAAVS210/L02-RLab/"

**1.1 Reading the CSV files using fread function**

The typical syntax for using the fread function is : fread(file\_path, header = True/False)

library(data.table) # Loading `data.table` package  
dt\_may = fread(paste(folder\_path, "2017\_05\_FlightTimes.csv", sep = ""), header = TRUE)  
dt\_jun = fread(paste(folder\_path, "2017\_06\_FlightTimes.csv", sep = ""), header = TRUE)  
dt\_jul = fread(paste(folder\_path, "2017\_07\_FlightTimes.csv", sep = ""), header = TRUE)

**2. TRANSFORM Data**

**2.1 Combine May, June and July datasets into one large dataset**

dt = rbind(dt\_may, dt\_jun) # combining may and june datasets  
dt = rbind(dt, dt\_jul) # adding july dataset to it  
  
#See Preview  
head(dt)

## FL\_DATE CARRIER FL\_NUM ORIGIN DEST DEP\_TIME DEP\_DELAY TAXI\_OUT  
## 1: 2017-05-01 HA 1 LAX HNL 0739 -6 14  
## 2: 2017-05-01 HA 2 HNL LAX 1441 1 22  
## 3: 2017-05-01 HA 3 LAX HNL 0958 -2 18  
## 4: 2017-05-01 HA 4 HNL LAX 2130 0 15  
## 5: 2017-05-01 HA 7 LAS HNL 0903 -12 18  
## 6: 2017-05-01 HA 8 HNL LAS 2225 -10 20  
## WHEELS\_OFF CANCELLED V11  
## 1: 0753 0 NA  
## 2: 1503 0 NA  
## 3: 1016 0 NA  
## 4: 2145 0 NA  
## 5: 0921 0 NA  
## 6: 2245 0 NA

**2.2 Add a New Column**

In data.table notation, := is used to denote the creation of a new column. Typical syntax for creating a new column is: dataset\_name[, New\_Col\_Name := some\_function(existing\_col\_name), ]

To demonstrate the use of := let’s create a new column that indicates whether a flight departed delayed. To do this, we will need to read the values of the column DEP\_DELAY and for any value > 0, we will need to add a flag that represents that the flight was delayed.

First, let’s create a new column called DELAYED\_DEP\_FLAG and have it assume value of 0 when the DEP\_DELAY<=0 . THis will flag all of our on-time and before-time departures with value 0

# this code is basically filtering the dataset where DEP\_DELAY<=0 and then adding a new column called DELAYED\_DEP\_FLAG with value = 0.  
dt[DEP\_DELAY <= 0, DELAYED\_DEP\_FLAG := 0, ]

Now, let’s handle the delayed departures

# this code is basically filtering the dataset where DEP\_DELAY > 0 and then adding a new column called DELAYED\_DEP\_FLAG with value = 0. In this case, new column will not overwrite the existing DELAYED\_DEP\_FLAG because our filters are different - hence it will simply populate the DELAYED\_DEP\_FLAG values for rows that meet the criteria of DEP\_DELAY > 0  
dt[DEP\_DELAY > 0, DELAYED\_DEP\_FLAG := 1, ]

Notice that we are splitting the operation into two parts, one for each value of the flag.

We can combine the two operations into one - although the syntax gets a little overwehlming.

This is an example of how we can create a new column using any function with inputs as existing columns (in this case, ifelse)

dt[, DELAYED\_DEP\_FLAG := ifelse(DEP\_DELAY<=0, 0, 1), ]  
head(dt)

## FL\_DATE CARRIER FL\_NUM ORIGIN DEST DEP\_TIME DEP\_DELAY TAXI\_OUT  
## 1: 2017-05-01 HA 1 LAX HNL 0739 -6 14  
## 2: 2017-05-01 HA 2 HNL LAX 1441 1 22  
## 3: 2017-05-01 HA 3 LAX HNL 0958 -2 18  
## 4: 2017-05-01 HA 4 HNL LAX 2130 0 15  
## 5: 2017-05-01 HA 7 LAS HNL 0903 -12 18  
## 6: 2017-05-01 HA 8 HNL LAS 2225 -10 20  
## WHEELS\_OFF CANCELLED V11 DELAYED\_DEP\_FLAG  
## 1: 0753 0 NA 0  
## 2: 1503 0 NA 1  
## 3: 1016 0 NA 0  
## 4: 2145 0 NA 0  
## 5: 0921 0 NA 0  
## 6: 2245 0 NA 0

**2.3 Fix date column format**

Currently the date column FL\_DATE is of class character. Lets correct the variable type to date.

dt[, FL\_DATE := as.Date(FL\_DATE, "%Y-%m-%d"),]  
# the := notation is used to   
# Checking the class of the FL\_DATE column to confirm that it has been correctly formatted as date  
class(dt$FL\_DATE)

## [1] "Date"

**2.4 Create Aggregations**

Create an aggregated dataset, with FL\_DATE, ORIGIN, and an aggregated column showing the Number of Flights departed on that day from that ORIGIN airport.

daily\_flights = dt[, .(Num\_of\_Flights = sum(!is.na(DEP\_TIME))), by = list(FL\_DATE, ORIGIN)]

**Explanations:**

.(Num\_of\_Flights = sum(!is.na(DEP\_TIME))) : The “.( )” notiation is basically creating a list of new columns that we are creating..in this case we are creating just 1 new column called Num\_of\_Flights. General syntax: “.(new column declarations separated by commas )”

Num\_of\_Flights = sum(!is.na(DEP\_TIME)) : This creates a new column called Num\_of\_Flights but since we didnt use := notation, it doesn’t add this new column to the dt dataset. When we don’t use the := notation, the new columns are created in temporary memory but isn’t stored. When you are aggregaing data, you will need to assign the aggregated data to a new dataset - that is where the aggregated columns will be. In this case, the aggregated dataset is called daily\_flights whicn contains the Num\_of\_FLights column.

by = list(FL\_DATE, ORIGIN) : performs the aggregation grouped by FL\_DATE and ORIGIN . THese two columns will apprear in the aggregated dataset by default - because we are grouping by them.

Let’s see a preview of the aggregated dataset

head(daily\_flights)

## FL\_DATE ORIGIN Num\_of\_Flights  
## 1: 2017-05-01 LAX 606  
## 2: 2017-05-01 HNL 130  
## 3: 2017-05-01 LAS 435  
## 4: 2017-05-01 SFO 479  
## 5: 2017-05-01 SAN 235  
## 6: 2017-05-01 SMF 125

Now let’s create some more aggregated fields. To the above aggregation, we are going to add an Average Departure Delay column. The logic to calculate this column is to get a mean of the DEP\_DELAY field but only for those flights that departed delayed. We will use the DELAYED\_DEP\_FLAG that we had previously created to identify flights that had departed delayed.

daily\_delay = dt[, .(AVG\_DEP\_DELAY = mean(ifelse(DELAYED\_DEP\_FLAG==1, DEP\_DELAY, NA), na.rm = TRUE)), by = list(FL\_DATE, ORIGIN)]  
  
head(daily\_delay)

## FL\_DATE ORIGIN AVG\_DEP\_DELAY  
## 1: 2017-05-01 LAX 38.28146  
## 2: 2017-05-01 HNL 10.51852  
## 3: 2017-05-01 LAS 34.67081  
## 4: 2017-05-01 SFO 42.64052  
## 5: 2017-05-01 SAN 20.78462  
## 6: 2017-05-01 SMF 16.40541

Another “hacky”" way to achieve similar results is to filter the dataset dt for rows where DELAYED\_DEP\_FLAG==1 and then creating a calculated column AVG\_DEP\_DELAY based on just that data

daily\_delay = dt[DELAYED\_DEP\_FLAG==1, .(AVG\_DEP\_DELAY = mean(DEP\_DELAY) ), by = list(FL\_DATE, ORIGIN)]  
head(daily\_delay)

## FL\_DATE ORIGIN AVG\_DEP\_DELAY  
## 1: 2017-05-01 HNL 10.51852  
## 2: 2017-05-01 SAN 20.78462  
## 3: 2017-05-01 SMF 16.40541  
## 4: 2017-05-01 SEA 34.00000  
## 5: 2017-05-01 OGG 17.61111  
## 6: 2017-05-01 JFK 70.84000

The slight problem with the second approach is that it will not have all the airports on the list for all dates - it will only have those airports that have at least 1 delayed flight on that day

**2.5 Perform JOIN Operations**

Just like in SQL, data.table can be used to perform really fast join operations. Next, we are going to try two types of joins: INNER JOIN and LEFT OUTER JOIN

In R, we can use the merge function to perform JOIN operations. Let’s use it to perform INNER JOIN of daily\_flights and daily\_delay datasets.

# INNER JOIN  
daily\_flights\_and\_delay = merge(daily\_flights, daily\_delay, by = c("FL\_DATE", "ORIGIN"))  
  
nrow(daily\_flights\_and\_delay) # Number of rows in the resulting inner joined dataset

## [1] 20296

head(daily\_flights\_and\_delay)

## FL\_DATE ORIGIN Num\_of\_Flights AVG\_DEP\_DELAY  
## 1: 2017-05-01 ABE 6 29.66667  
## 2: 2017-05-01 ABQ 54 28.70000  
## 3: 2017-05-01 ABY 3 89.50000  
## 4: 2017-05-01 ACV 3 17.00000  
## 5: 2017-05-01 ACY 10 34.33333  
## 6: 2017-05-01 AEX 9 128.50000

# LEFT OUTER JOIN  
daily\_flights\_and\_delay = merge(daily\_flights, daily\_delay, all.x = TRUE, by = c("FL\_DATE", "ORIGIN"))  
  
nrow(daily\_flights\_and\_delay) # Num of rows in the resulting left outer joined dataset

## [1] 26977

head(daily\_flights\_and\_delay)

## FL\_DATE ORIGIN Num\_of\_Flights AVG\_DEP\_DELAY  
## 1: 2017-05-01 ABE 6 29.66667  
## 2: 2017-05-01 ABQ 54 28.70000  
## 3: 2017-05-01 ABR 2 NA  
## 4: 2017-05-01 ABY 3 89.50000  
## 5: 2017-05-01 ACV 3 17.00000  
## 6: 2017-05-01 ACY 10 34.33333

The all.x = TRUE is what makes this a LEFT OUTER JOIN. The by = character vector of col names used to join the two datasets, similar to the ON clause in SQL. If the names of the columns are different between the two datasets then one can use by.x = and by.y = keywords to provide column names in the two datasets.