Exploring Unix Shell, R, and SQLite

Shuhua Liang

March 5, 2013

The task of this project is to combine the use of the UNIX shell, R, and SQLite as a database. When there are large amounts of data, processing everything in one language, such as R, can be extremely slow, and the process takes up much memory of the CPU. Here, we have 22 .csv files containing airline delay data from the year 1987 to 2008. All files add up to approximately 12 GB; if we were to load all of them into R and process, it would take up tremendous amount of time.

The shell is a relatively fast tool to read in data, so we can read the 22 .csv files into the shell and pass the information to R. In this set of airline delay data, there are four airports, include SFO, LAX, SMF, and OAK, that we are particularly interest in. To find the total number of flight going out of those airports, we would split the data in the shell by the four airports and find their corresponding counts. As a result, we get the following:

- $LAX \rightarrow 4057452$
- $SFO \rightarrow 2711958$
- $SMF \rightarrow 806133$
- $OAK \rightarrow 1151897$

There are several approaches to extract data from the files while avoid long processing time and high capacity of memory. First, we consider combining R and the shell. We can first process the files in the shell, where we extract the useful data from each file and combine them into a new .csv file, called file.csv. Then we can read file.csv into R as we would normally do in R Console. After this, we can work out different kinds of computations in R. For example, here we computed the

means and standard deviations for the airports that we are interested in. Using this method, the results are:

Airline	Mean	Standard Deviation
LAX	6.276653	27.36568
OAK	5.095424	22.03457
SFO	8.228922	30.35535
SMF	5.451920	25.14183

Table 1: Mean and Standard Deviation from Combining Shell and R

The challenge with this part of the project is that there are two files with abnormal inputs, so we cannot read the files in with the same command. Alternatively, we would need to create separated files to save the data from those two abnormal files and the other normal files. After all, we can combine those files with a loop in the shell.

Next, we can consider using a connection between R and the shell. This is different from the previous case because we are calling for the data directly from R to be processed in the shell. We will not have to read in a separated file in R. Specifically, the pipe function was used in R to connection to the shell. This allows us to extract particular data of interest by specifying the conditions through the shell command that is called in the R function. In our case, the columns of Origin and Arrival Delay are called.

These data comes in a set of strings, and it includes every airport. In R, we would need to do some data cleaning to convert the strings in to a two column data frame, and then we need to subset the data to distinguish the four airports' data. For this approach, the mean and standard deviations computed and results are:

Airline	Mean	Standard Deviation
LAX	6.007046	27.30415
OAK	5.073360	22.05919
SFO	7.951510	30.28721
SMF	5.359327	25.09679

Table 2: Mean and Standard Deviation from Connection

Another method that we would consider is operating a database in SQLite. Here,

we need to create a table of data, the database, in SQLite and import all data into the table. This database gives a separate file that we can refer to using R functions. In R, we can simply call the SQL functions to extract data with specific conditions that we would imply. This process output a data frame in R, and we can work it directly. The mean and standard deviation calculation results are:

Airline	Mean	Standard Deviation
LAX	5.892965	27.05606
OAK	5.007034	21.92210
SFO	7.774897	29.97188
SMF	5.299447	24.96255

Table 3: Mean and Standard Deviation from Database

The mean and standard deviation outputs from the three approaches are slightly different, but they show the same ordering of the airports.

Among the different approaches, we can check time used to process each of them using the function system.time in R. This shows that R take a relatively longer time to process data in the connection approach; moreover, splitting the strings took up most of the processing time. On the other hand, although calling the SQLite function in R is fast, it took a while to import all files into the table. In terms of memory, the first approach, where we combined the shell and R was a little problematic because we had to go through each file, extract data and create a new file, which in some way we are making a duplicate of the original data.

In other analysis, the first approach would be useful when the same file is shared amount programs. The connection approach is useful for reading large data into R or other programs. And, the database approach is most useful because it is easy to construct, and a flat database contains all the information about a record.

In terms of block size from connections and database, the speed of the connection approach is high effected because the amount of data that needs to be extract from the file slows the process in R. However, block size does not effect the process in database as much because we call function in R with SQLite commands, so all the blocking work is done in SQLite, and the program is meant to be fast in this kind of operations.

I explore the interaction between different programs and how they simplify the process of data cleaning and analysis. The challenges were to get the programs

to link, understand different command lines in different programs, and recognize when it is appropriate to read in a command line from other programs in R. Overall, these tools are very useful in the future, when I need to process large amount of data in a less time consuming way.