Introduction

In this assignment, I will deal with 116GB csv files in shell and R. I used shell to do data filter, extract needed columns and pipe bash processing to R. We do high-level analysis in R using the data processed in shell. There are two general methods I used in R. One is the 'brute force' which is that we read the processed data in R and do analysis. The other is Parallel method by which we can make best of use the computer since R can only use one core to do a task.

Data Match

Before we clean data, we need to check if the data is matched. For example, the first rows in 'trip_data_1" csv file and the first rows in 'trip_fare_1' csv file should contain the same ID information. I selected 'medallion', 'hack_license' and 'pickup_datatime' columns from all 'data' csv files and save in a txt file. I also selected the same three columns from all 'fare' csv files. Since the data csv files and fare csv files are listed in the same order, i.e. the first data csv files is trip_data_10.csv and the first fare csv files is trip_fare_10.csv and so on, the two files should be the same if the data is matched. In the output of our function, data csv files are matched with trip csv files.

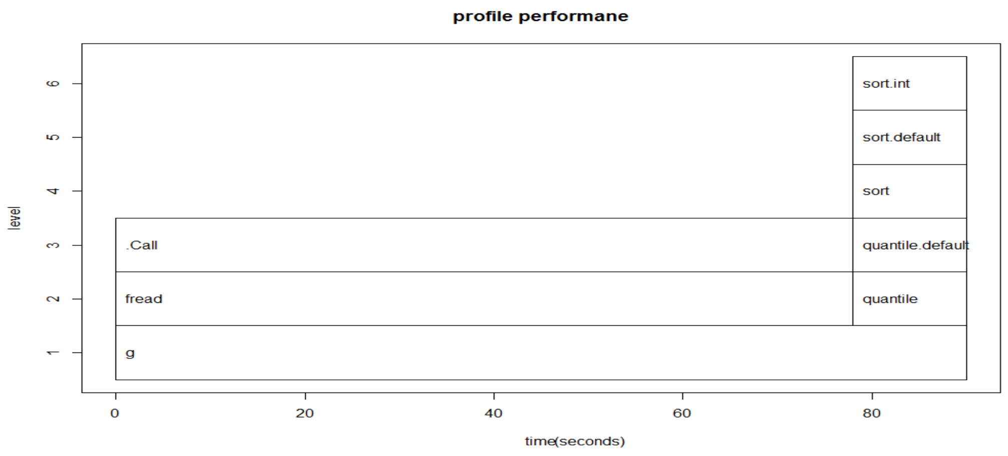
Data clean

The goal of cleaning data is to delete obvious error data and extract the columns we need in data analysis. Why do we clean data? First, large number of obvious error data will give us wrong results. Second, we need to make data contain enough useful information as well as make it as small as possible so that we can run it in R efficiently. In this assignment, I first selected 'tolls_amount', 'total_amount' and 'surcharge' from 'fare' csv files and save them by columns in a txt file. Then I recalculated 'triptime' by subtracting the 'pickup_datatime' from the 'dropoff_datatime' in 'data' csv files because for some observations, the 'trip_time_in_sec' is reported in seconds, and save them in another txt file. After this, I pasted the two txt files columns by columns. Since the data csv files and fare csv files are listed in the same order, i.e. the first data csv files is trip_data_10.csv and the first fare csv files is trip_fare_10.csv, two files are match row by row (mean it contain the same ID 's information). Then we deleted the rows whose 'tolls_amount' or 'total_amount' is negative. Then we subtracted 'tolls_amount' from 'total_amount' and we selected the rows whose 'totallesstoll' which represents tips is positive. After this, I saved the cleaned 'triptime' column in a txt file, 'surcharge' in a txt file and 'totallesstoll' in other file. The data is then clean and contain no redundant information and I will use this data for further analysis.

Compute the Deciles

1. **The 'brute way'**

   The 'brute' way means that we read straightly our data into R. The reason I do this is that the 'totallesstoll' txt file is not too big and 'fread' function in 'data.table' library is really faster to read the data in R. I use 'object_size' function in 'pryr' package to measure the size of 'totallesstol', it turned out to be 1.39 GB and the class of column is 'numeric'. The data can be handled by R to calculate decile by using quantile functions in my computer which has 4 cores and 4GB RAM. It took 91.03 of elapsed time in my computer. I think it is acceptable, because we used all data and it did not take two much time.



From the profile performance plot, we can see that 'fread' function took lots of time compared with 'quantile' function. We can see there is a internal C call in 'fread' function which makes it faster read data compared with other function like read.table or read.csv. We should really use C C++ in critical function, for example we should write loop in C code.
Below is the plot of quantile we calculated. I omitted the quantile of 100%, because it is too big (685908) to believe it is a normal value.

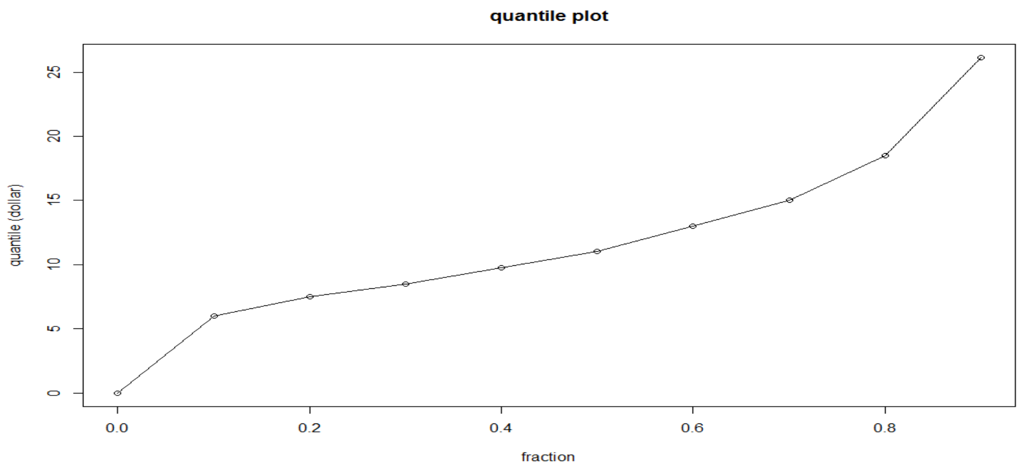


Fig.1 quantile plot

From the plot we can see that the first quarter and the median of tips (totallesstoll) are around 10 dollars, 90% driver receive tips under 26 dollar.
However, I can only do the quantile using this big data, if I used this data in lm function, my computer will crash, because it will copy data in the funciton. So, our goal is still to read small data in R. The 'brute way' method can only do simple calculation with bigdata.

2. **Parallel Reading**
In this method, I subtract totallesstoll from each fare csv file and read the data into a table. The reason I choose this method is that table take up small storage and we can do parallel on all files which improve performance. However in this case, I did not do data clean on each fare csv file. Dose this matter? In fact, the number data I deleted from file is 3thousand. Most of error are

outliers and usually small number of outlier does not affect main part of quantile. From the result quantile of this method, we can see that the significant difference is 0% quantile (uncleaning data give -1430) because of the negative value in csv file. However, we can see in Fig.2 that the main body of quantile using this data is quite close to the quantile using cleaned data

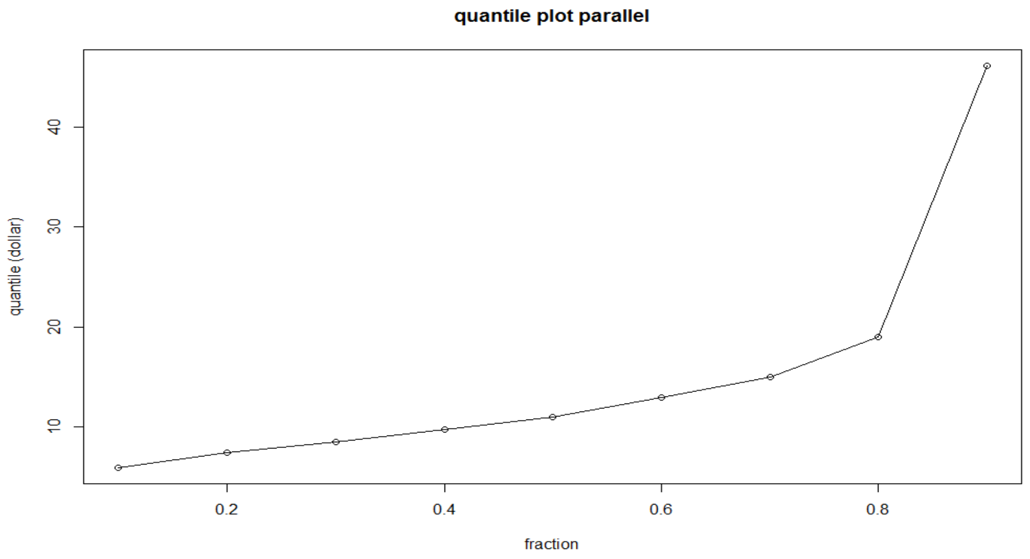


fig.2 quantile plot using parallel method

Now we compare the two methods time. I would say parallel method is much faster because It took 309 elapsed time with 1618 user time in second. Since the user time is pretty high, we may conclude that it should take less time if the server is not crowded. It seem that it took less time by 'brute way' method which is 91 elapsed time in second, but it took me 13 minutes to extract tolls and total, subtract them and save in a txt file on the serve when it did not crowded. What's more, it read data in table which make data very small.

Build linear model

1. Simple linear

We read 'totallesstoll' and 'triptime' into R by using fread function. 'triptime' txt file is 698 MB and 'totallesstoll' file is 651MB, but I found that it took 40 seconds to read 'triptime' while it took 80 seconds to read 'totallesstoll'. I check the type of colums in 'triptime' and it turned out to be integer and 'totallesstoll' was double. It seems that R read integer faster than reading double. The reason behind this is that double is store in 8 byte while integer is 4 byte. So sometimes we may convert to integer to do some calculation if the transformation does not change the results of analysis.

We use the formula of coefficients in simple linear regression to estimate β0 and β1.

$\widehat{\beta 1} = \frac{cov(x,y)}{sd(x)\char94 2}$ $\widehat{\beta 0} = \bar{Y} - \widehat{\beta 1}\bar{X}$. We get $\widehat{\beta 1} = 0.001951105$ and $\widehat{\beta 0} = 13.11718$. This means that trip time and tips has positive relation which means that the when the triptime increase 10 minutes (600 seconds) the tips will increase 0.17063 dollars (0.001951105*600), and the intercept means that the driver at least get 13.11718 tips. We can use $\hat{y} = 13.11718 + 0.00195105 * x$ to get prediction of y with a new x value. The MSE equal 4737.736, and the MSE is not big considering we have one hundred million data.

I calculated the coefficient in my computer and the elapsed time is 126.02 seconds including reading data, but it took most of time reading data, the computing time is short. Again, I can only do some simple computation on my computer with 2.6 GB dataframe read in R.

2. Multiple linear Using ‘bigmemory’ package

In this case, we have two predict variable, and I used ‘read.big.matrix’ to read the txt file which contains ‘triptime’,’surcharge’ and ‘totallesstoll’ on the server. I use the matrix form of formula $\beta = (t(x)x)^{-1}t(x)Y$ to calculate vector of β1 and β2. And then use $\widehat{\beta 0} = \bar{Y} - \widehat{\beta 1}\overline{X1} - \widehat{\beta 2}\overline{X2}$. X1 represents ‘triptime’ and X2 represents ‘surcharge’. It took me almost 31 elapsed time to get coefficients which is $\widehat{\beta 0} = 7.076736$, $\widehat{\beta 1} = 0.004087$, $\widehat{\beta 2} = 14.03385$. From the coefficient, we see that $\widehat{\beta 1}$which represent tips have positive relation with triptime and tips have high relation with surcharge. The MSE is 4779.66 which is small.

It took 189.225 elapsed time to read a 1.73GB data into a 616B big matrix, which is perfect. So this method can store data efficicents by put data in chunks.