Christine Cai
STA250 HW1
Winter 2014

I downloaded the files and extracted them on gumbel as follows:

wget http://eeyore.ucdavis.edu/stat250/Data/Airlines/Delays1987_2013.tar.bz2
tar jxf Delays1987_2013.tar.bz2 -C data

Method 1 (Method 5 in prompt):

The file method1fun.R contains a function which takes in a list of files, a number specifying which file in this list, and makes a frequency table for arrival delay for that file.

It also contains another function which loops through all the csv files that we were given.

Running method1run.R with the appropriate input and output directories will result in a list (of txt files) of frequency tables, one for each csv file. (Simply open method1run.R and replace the directories as desired. My output directory is named tables.) These freq tables have name format tableYear.txt or tableYear_month.txt, depending on whether they are from before or after 2008.

I manually put the tables for 2008 and after into a separate directory, tables2. This allows me to run the following in shell, first change directory to tables2,

```
for f in table*
do
echo $f
cat $f | tail -n +2 | head -n -1 >> ctable.txt
done
```

Now change directory to tables,

```
for f in table*
do
echo $f
cat $f | head -n -2 >> ctable.txt
done
```

The former takes care of 2008 and after, the latter takes care of pre-2008. This will result in two concatenated tables with only the integer arrival delay times we care about; i.e. entries like

1 "ArrDelay"

are not included. Change back to the directory containing the directories tables and tables2, and oncatenate these two ctable.txt's into one ctable.txt with:

cat tables/ctable.txt tables2/ctable.txt > ctable.txt

Running method1pro.R will read the final ctable.txt, merge entries across files to get a final freq table, and calculate the desired statistics with the wtd.stats() functions in the Hmisc package.

Method 2 (Method 1 in prompt):

This method uses read.csv() to read in an entire file, extracts the arrival delays column, then makes and stores a freq table. This process is looped over all the files. The frequency tables are then merged, and the desired statistics are calculated using the Hmisc package.

To run this method, simply run method2_3.R. Then call the function with

method2.3.fun(method = 2, direc = 'data', verbose = TRUE)

where direc is the directory containing (only) the csv files to be processed. Set verbose = TRUE if you would like to print the progress.

Method 3 (Method 7 in prompt):

This method utilizes the FastCSVSample package to sample observations from each file. The observations are then parsed so that we can extract the arrival delays, and make a freq table. This process is looped over all the files. Similar to the other two methods, when we have all the frequency tables, they are merged and the statistics are calculated using the Hmisc package.

Unlike the previous 2 methods, this one comes with a theoretical caveat. How many observations should we sample from each file? At first, just to get things going, I sampled 10,000 observations from each file. However, this is a faulty sampling scheme. Since we want the mean of all arrival delays, we need to sample uniformly across all arrival delays. 1987.csv is about 121mb, whereas 2007.csv is about 670mb. So if we sample 10,000 observations from 1987 and 10,000 observations from 2007, we are giving the observations in 1987 a higher chance of being picked.

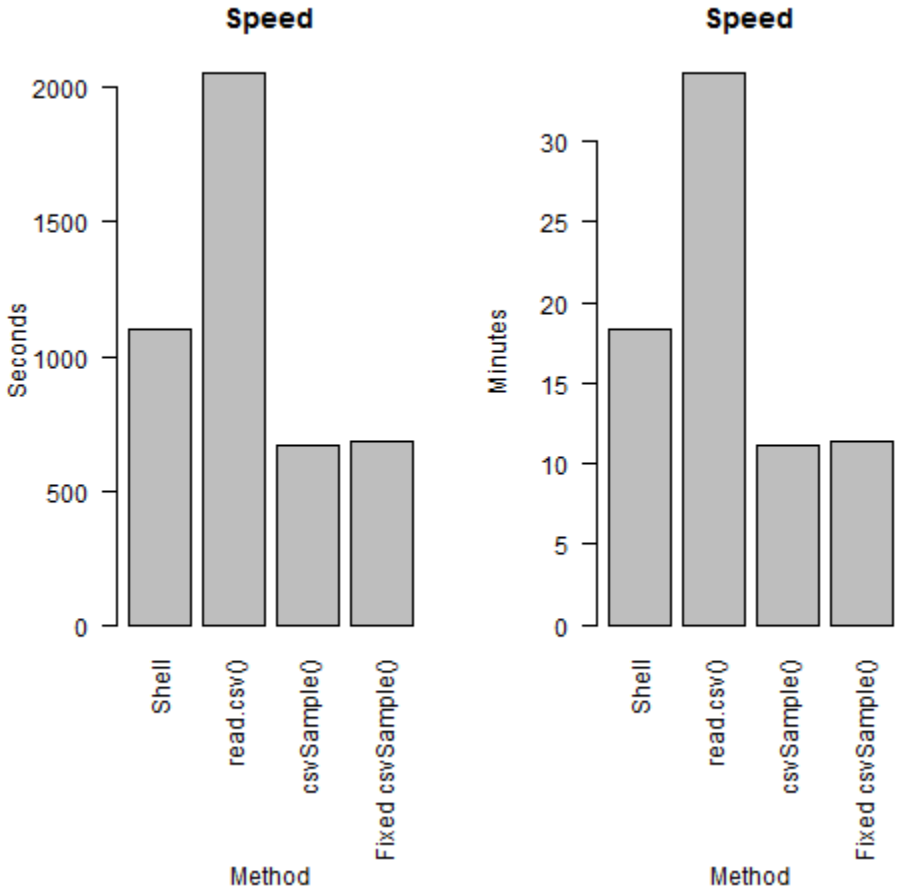
A naïve but computationally fast fix for this is to scale the sample size to the file size. This is taken care of in the file sampleSize.R. Further discussion is provided in the results section below.

To run this method, run method2_3.R. Then call the function with

method2.3.fun(method = 3, direc = 'data', verbose = TRUE)

Again, direc is the directory containing (only) the csv files to be processed. Set verbose = TRUE to print progress.

Results

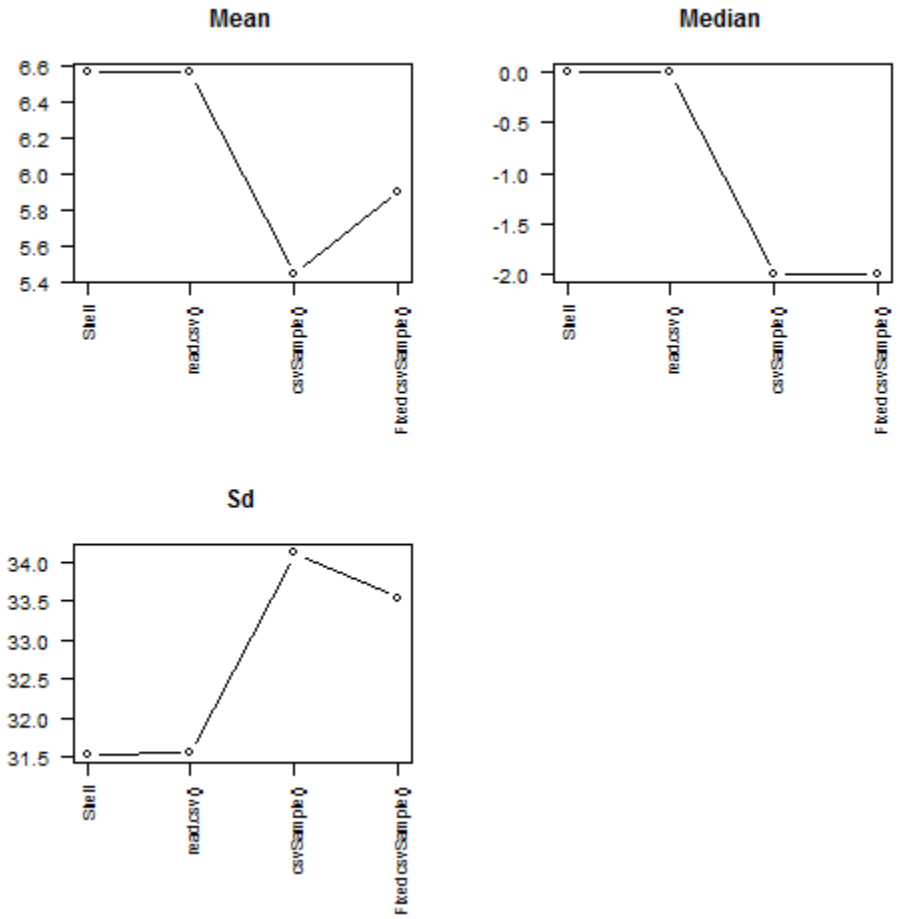


Note: in the plot above, csvSample() refers to sampling an equal number of observations from each file, while Fixed csvSample() refers to scaling the sample sizes according to file sizes.

Looking at the speed, the sampling method (using FastCSVSample) is the fastest. For all three methods, merging the frequency tables and calculating the statistics are fast; almost all of the computation time is eaten up before this step. The read.csv() method is the slowest, probably because it requires the most memory. Each file needs to be stored in a data frame, and these files are not small. For FastCSVSample, I believe only one line is read at a time. After we are done with that line, it is discarded; I think the lack of memory use makes this method speedy. Speed is one thing, but we also need to compare accuracy.

Please see the plot below for the means, medians, and standard deviations. Since the shell and read.csv() methods both utilize all entries to find the weighted means, medians, and standard deviations from frequency tables, I am confident in the accuracy of their solutions. It is also a good sign that the solutions from these two methods are almost identical. The sampling method however, is not very accurate. For the mean and sd, we do a little better if we base the sample sizes on the sizes of the files, but they're

still a ways off.



Since the speed for taking scaled sample sizes across files is almost the same as taking equal sample sizes, and the solutions are more accurate if we take scaled sample sizes, it is definitely worth it to spend the time on a scaled sampling size scheme.

Out of the three methods attempted, I would say the shell is the most efficient. It provided accurate answers with moderate speed. However, I have to say I was surprised that using read.csv() to read in entire files only took about half an hour. I expected this method to take hours, if not a day, to run. As an R user who is still not very familiar with shell commands, I honestly preferred the read.csv() method.