Lab 07 STATS 769 Due 4:00pm Tuesday 1 October

The main point of this lab is to experiment with some **large data solutions**. We will again generate a literate report that fits a linear regression model, but we will keep it even simpler and just fit the model to all of the available data (no cross-validation).

The Data

The data set for this lab is a single large CSV file containing electric vehicle trips (a larger version of what we have been using in previous labs). This file is available on the VMs at the following location:

`/course/data.austintexas.gov/Dockless_Vehicle_Trips.csv`

The Task

1. We estimated in the last lab that a smallish laptop (4GB RAM) might struggle to work with the full CSV file. The virtual machines for this course provide us with an opportunity to use the simple solution of using a bigger machine.

   Use the following code to load the entire CSV file into R (and tidy it and transform it), perform a linear regression fit, and report the coefficients of the model. Notice that the first line of data contains only NA values, so we do not read that first row of data in.

   Report the memory size of the full data frame in R and measure the maximum memory used by these calculations.

   ```
   colnames <-
       gsub(" ", ".",
            scan("/course/data.austintexas.gov/Dockless_Vehicle_Trips.csv",
                 sep=",", what="character", strip.white=TRUE, nlines=1))
   trips <-
       read.csv("/course/data.austintexas.gov/Dockless_Vehicle_Trips.csv",
                header=FALSE, skip=2, col.names=colnames)
   trips <- subset(trips, Trip.Duration > 0 & Trip.Distance > 0)
   trips$logDuration <- log(trips$Trip.Duration)
   trips$logDistance <- log(trips$Trip.Distance)
   fit <- lm(logDuration ~ logDistance, trips)
   coef(fit)
   ```

   This code will be slow (especially the `read.csv()` call) and it will use a couple of Gb of memory, so do NOT evaluate this code in your R Markdown report; just run the code in an R session and record the result in your report.

2. One way to limit the R's memory use is to load only the data that we need.

   Use `awk` to generate a new CSV file that only contains the distance and duration variables.

   Measure the peak memory usage of your `awk` command and compare it to the memory required to load the entire data set in to R.

3. Read the new CSV file (just containing distance and duration) into R with `read.csv()` and with `data.table::fread()` and report the sizes of the resulting `data.frame` and `data.table`.

4. To keep things simple, and to retain the focus on memory usage, our Tidy and Transform steps will just involve removing non-positive values and logging both variables.

   Perform this subsetting and transformation with a normal data frame and measure the memory usage.

   Then try to use less memory by repeating the task with the **data.table** package.

   NOTE: see the section at the end of this handout for a discussion of how to get a fair comparison of memory usage between the two approaches.

5. Fit a linear regression using both data frame and data table. Is the memory usage any different?

6. Use a streaming approach and the **biglm** package to fit the linear regression. Show that this approach requires significantly less memory.

7. Check that the linear regression coefficients are the same from all four linear regression fits (naive use of original CSV, data frame from new CSV, data table from new CSV, and streaming fit).

8. Discuss what challenges might arise, if any, if we wanted to calculate the Mean Square Error for each of the four model fits.

The Report

Your submission should consist of a *tar ball* (as generated by the `tar` shell tool) that contains an R Markdown document *and* a Makefile *and* a processed version of your R Markdown document, submitted via Canvas.

You should write your document and your Makefile so that the tar ball can be extracted into a directory anywhere on one of the virtual machines provided for this course (`sc-cer00014-04.its.auckland.ac.nz` or `sc-cer00014-05.its.auckland.ac.nz`) and the markdown document can be processed just by typing `make`.

Your report should include:

- A description of the data format.
- A description of each of "The Task"s.
- A conclusion summarising the analysis.

Your report should NOT be longer than **10 pages**.

Marks will be lost for:

- Submission is not a tar ball.
- More than 10 pages in the report.
- R Markdown file does not run.
- A "Task" is missing.
- R Markdown file is missing.
- Processed file (pdf or html) is missing.
- Makefile is missing.
- Significantly poor R (or other) code.

Fair comparisons of memory usage

So far we have considered measuring memory usage like this ...

```
gc(reset=TRUE)
## Perform task
gc()
```

This gives us an idea of the peak memory usage required to perform the task. However, R also automatically performs garbage collections by itself (whenever it hits the "gc trigger" limit). This means that the estimate of peak memory can be different depending on what the "gc trigger" value is (whether R performs garbage collection itself, or not).

The following code just generates two 8Mb vectors.

```
> x <- runif(1e6)
> y <- runif(1e6)
```

If we now measure the peak memory for calling `lm()` with these vectors, we see a difference of approximately 110Mb, but that reflects the fact that R has garbage collected itself and cleaned up some memory along the way.

```
> gc(reset=TRUE)
```

```
         used (Mb) gc trigger (Mb) max used (Mb)
Ncells  288865 15.5     592000 31.7   288865 15.5
Vcells 2592517 19.8    4135294 31.6  2592517 19.8
```

```
> fit <- lm(y ~ x)
> gc()
```

```
           used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  1297013  69.3    2164898 115.7  1299359  69.4
Vcells 14069762 107.4   20672602 157.8 17074082 130.3
```

The following code gratuitously generates a large vector to force the "gc trigger" much higher.

```
> invisible(object.size(integer(1e8)))
```

Now, if we repeat the peak memory measurement for `lm()`, we get a different answer (approximately 200Mb). This reflects the fact that R has not garbage collected itself (because the trigger limit is never reached).

```
> gc(reset=TRUE)
```

```
           used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  1297032  69.3    2164898 115.7  1297032  69.3
Vcells 14069852 107.4   61721922 471.0 14069852 107.4
```

```
> fit <- lm(y ~ x)
> gc()

           used  (Mb) gc trigger  (Mb) max used  (Mb)
Ncells  1297041  69.3    2164898 115.7  1305270  69.8
Vcells 14069882 107.4   49377537 376.8 40624015 310.0
```

This does not mean that our measurement of peak memory is wrong, it just means that the measure will depend on the "gc trigger" at the time. And that means that if we are trying to compare peak memory for two different approaches, we need the "gc trigger" to be the same for both approaches or the comparison will not be fair.

There are two things we can do to make a comparison more fair.

- Create a large object (temporarily) that sets the garbage collection trigger very high.

  ```
  object.size(integer(5e8))
  ```

- Use the total allocations reported by `profmem::profmem()`.

  ```
  p <- profmem({
     ## Perform task
  })
  total(p)
  ```

Both of these will overestimate the real peak memory usage (by avoiding any intermediate garbage collections), but will allow a more fair comparison between different approaches.
