Jared Yu 914640019

STA 141C

HW 6

Good practices

1. Frequent use of vectorized code makes it such that the code is fast and optimized. When using large datasets such as this, having vectorized code will make it possible to quickly program and find results efficiently.
    a. Many different functions are used for a vectorized process: `count(), split(), subset(), spread(), seq(), bind_rows(), prop.table(),` etc.
2. Parallelization is an important trick when dealing with data that is 'embarrassingly parallel,' i.e. a task which needs to be done repetitively regardless of the order of the data. Using mclapply() is a great way to find results faster and more efficiently than using a regular lapply, and takes advantage of a situation where use of parallelization is beneficial.
    a. `freq_by_id = mclapply(df_split_by_id, freq_count)`
3. The code is in general quite clean and easy to follow. The different variable names make sense, and they are updated in a manner which is not difficult to understand.
    a. The function `freq_count` uses local variables such as: `temp, result,` and `empty_df.`
4. There was a frequent use of apply functions within the code, and no use of any for-loop. Using an apply() function over a for-loop is good practice for writing faster code, especially when it comes to large data sets such as this one.
    a. `kld_by_id = apply(p_table, 1, kld)`

Sub-optimal practices

1. Using a variable name which is already in use is not good, in case when need to use the original function in the future. This is like creating a variable named 'c,' when 'c' is already the function that is used to create a vector, i.e., `c(1,2,3)` will create a vector with the numbers 1, 2, and 3.
    a. `file = fread('dup_removed_dataset.csv', colClasses = c(rep('character', 3)))`
    b. The function `file()` is already within base R.
2. Leaving column names as default names, is in general not a good idea when thinking in the long term of using the code. Often, us or someone else will have to look back to try and understand what our code is doing and having column names which makes sense are good for the purpose of further analysis.
    a. Below is an example of V2, V3 being used together:

```
i.   file = file[file$V2 != '' && file$V2 != ' '] file$V2 =
     substr(file$V2, 1, 1)
ii.  file = file[file$V2 != '0' & file$V2 != '-' &
     file$V2 != '']
iii. freq_table = count(file, "V3")
iv.  remove_list = freq_table$V3[freq_table$freq < 100]
v.   file = file[!(file$V3 %in% remove_list)]
```

vi. `file = subset(file, select = c('V2', 'V3'))`

3. When writing functions, it is okay to use global variables, however we should set them equal to local variables, and use the local variables within the function. Hiding global variables in functions without saying anything can be a bad practice and will possibly lead to mistakes in the future.
   a. `kld = function(element) { sum (element * log(element / q_table), na.rm = TRUE) }`
   b. solution: `kld = function(element, local_q_table = q_table) { sum (element * log(element / local_q_table), na.rm = TRUE) }`
4. There are few comments within the R code, this makes it difficult to understand the full meaning of what certain sections of code are doing. Often it is good to provide brief comments for short sections of code, along with comments within functions to explain their purpose, even if it is simple and short.
   a. There are no comments to explain certain chunks of code, and none of the functions provide any comments to explain their purpose (counterexample rather than an example). The only comments seem to be `#qn3`, `#Bootstrap`, and `#qn4`.