# Task 2: Initial Data Transformation

James Mbewu

13/06/2021

**Note**: Unfortunately it appears RStudio has a problem running the future library in a notebook, so please don't be alarmed by all the errors when run. The correct file to run the code is initial_data_transformation.

## Overview

This R Markdown document serves as the report and code for Task 2: Initial Data Transformation for the City of Cape Town Data Science Unit Code Challenge. The code by itself can be found in the file initial_data_transformation.R.

## Brief

Join the file `city-hex-polygons-8.geojson` to the service request dataset, such that each service request is assigned to a single H3 hexagon. Use the `sr_hex.csv` file to validate your work.

For any requests where the `Latitude` and `Longitude` fields are empty, set the index value to `0`.

Include logging that lets the executor know how many of the records failed to join, and include a join error threshold above which the script will error out. Please also log the time taken to perform the operations described, and within reason, try to optimise latency and computational resources used.

## Report

The aim of this task is to join the H3 level 8 hex data in `city-hex-polygons-8.geojson` to the service request dataset ,`sr.csv`, that has been provided. To do this I used the h3jsr library to calculate the H3 level 8 hex index of the hex that contains the latitude and longitude of each service request. I then used this index to join the two datasets together so that the combined dataset contains H3 hex data for the hex it is in.

The calculation of H3 indexes was sped up by splitting the data into batches and then processing each of these batches in parallel using the package future. The data was split into batches so that the program could abort if not enough of the H3 indexes could be calculated and to provide progress updates to the user.

The join was then validated by comparing H3 hex level 8 index in the constructed joined dataset with a provided dataset (`sr_hex.csv`) that includes service request data as well as the corresponding H3 hex level 8 index. Having said that, this doesn't really validate the join, but more like the calculation of the H3 level 8 index so perhaps the intention was for me to calculate the H3 level 8 index of the service request from `city-hex-polygons-8.geojson` data myself. In any case, I think it is better and more efficient to just use the off-the-shelf H3 R-bindings provided by h3jsr.

**Note**: Unfortunately I only had access to my 2 core laptop to run this dataset on so I wasn't able to process the whole dataset or investigate the scaling of the parallelisation. But from my minimal tests it did seem to make a good improvement, for example running 10 000 service requests took 46s with 1 process and 27s with 2 processes to calculate the H3 indexes.

**Note**: When installing the package h3jsr I had an error that could be fixed by installing/updating some extra packages in the Unix command line. The command to fix this was:
`sudo apt-get -y update && apt-get install -y libudunits2-dev libgdal-dev libgeos-dev`

`libproj-dev`
and then to install the package in R terminal:
`remotes::install_github("obrl-soil/h3jsr")`

### Parmeters

There are a number of parameters that can be set below for this script including the number of processors to use (num_procs), the number of lines to read and process (num_lines), the number of batches to split the data into (num_batches_input), the threshold at which the program closes if not enough H3 indexes could be calculated (join_error_threshold).

```
# PARAMETERS ======================================================================
# Number of processors to use
num_procs <- 1
# Number of lines to read (service requests to process), Inf for whole file
num_lines <- 10000
# Number of batches to split data into (minimum set to 10)
num_batches_input <- 10
# Error threshold to terminate program if not enough h3 indexes can be calculated
join_error_threshold <- 0.2

plan(multiprocess, workers = num_procs)
```

### Loading Data

We first need to download the datasets from the remote location and save them to a temporary staging folder before being read in by R as a data frame. The service request dataset is particularly large so it might be a good idea to load and process it in batches. For now in the testing phase I am simply processing a subset of the data.

```
# LOAD DATA ======================================================================


# Note that I was having some troubles getting h3jsr to work out of the box
# and I needed to do the following (on linux):
# 1) apt-get install -y libudunits2-dev libgdal-dev libgeos-dev libproj-dev (in unix terminal)
# 2) remotes::install_github("obrl-soil/h3jsr" (in R)


start <- Sys.time()

# Download data if not present
dir.create("data",showWarnings = FALSE)

hex_8_data_filename <- "data/city-hex-polygons-8.geojson"
if(!file.exists(hex_8_data_filename)) {
  hex_8_data_url <- "https://cct-ds-code-challenge-input-data.s3.af-south-1.amazonaws.com/city-hex-polyg
  download.file(hex_8_data_url, hex_8_data_filename)
}

sr_data_filename <- "data/sr.csv"
sr_gz_data_filename <- "data/sr.csv.gz"
options(timeout=180)
if(!file.exists(sr_data_filename)) {
  sr_data_url <- "https://cct-ds-code-challenge-input-data.s3.af-south-1.amazonaws.com/sr.csv.gz"
```

```
  download.file(sr_data_url, sr_gz_data_filename)
  gunzip(sr_gz_data_filename, remove=FALSE)
}

end_download <- Sys.time()
download_time <- difftime(end_download, start)
print(paste("download_time =",download_time))



start_load <- Sys.time()

# Read in data
# For testing purposes we are only going to read in a subset of the data
hex_8_data <- read_json(hex_8_data_filename,simplifyVector = TRUE)
sr_data <- read_csv(sr_data_filename,n_max = num_lines,skip = 0, col_types = cols());
n_rows <- nrow(sr_data)

end_load <- Sys.time()
load_time <- difftime(end_load, start_load)
print(paste("load_data_time =",load_time))
```

### Joining the data

To join the data, it is simply a case of calculating the H3 hex level 8 index of the service request from its latitude and longitude using the package h3jsr. Then we use these indexes to join the service request data and the H3 hex level 8 data.

```
# JOIN DATA ================================================================

start_calc_h3 <- Sys.time()

# Specify num_procs to future
plan(multiprocess, workers = num_procs)


# Initialise some variables for batch loop
sr_data$h3_level8_index <- 0
lat_idx <- which(colnames(sr_data)=="Latitude")
lon_idx <- which(colnames(sr_data)=="Longitude")
options(digits=9)

num_fail <- 0
num_attempts <- 0
portion_complete <- 0

min_batches <- 10
num_batches <- max(num_batches_input,min_batches)
batch_size <- as.integer(n_rows/num_batches)

start_idxs <- seq(1,n_rows,batch_size)
end_idxs <- start_idxs + batch_size
end_idxs[length(end_idxs)] <- n_rows
```

```
# Loop over batches
for(i in 1:length(start_idxs))  {
  start_idx = start_idxs[i]
  end_idx = end_idxs[i]

  # Parallel loop over each batch
  sr_data$h3_level8_index[start_idx:end_idx] <-
    future_apply(sr_data[start_idx:end_idx,],1, function(sr_row) {

    lat <- as.numeric(sr_row[lat_idx])
    lon <- as.numeric(sr_row[lon_idx])

    # h3 index is 0 if
    h3_level8_index <- 0
    if(!is.na(lat) && !is.na(lon)) {
      # find the h3 index of this
      sfc_point <- st_sfc(st_point(c(lon, lat)), crs = 4326)
      h3_level8_index <- point_to_h3(sfc_point, res = 8)
    }

    return(h3_level8_index)
  })


  # Exit if error threshold is reached
  num_attempts <- end_idx
  num_fail <- num_fail +
    sum(!is.na(as.numeric(sr_data$h3_level8_index[start_idx:end_idx])))

  if(num_fail/num_attempts > join_error_threshold) {
    percent_fail <- as.integer(num_fail/num_attempts * 100)
    stop(paste0("Too many service requests (",percent_fail,
    "%) have invalid location data. Exiting."))
  }

  # Progress indicator
  if(i %% as.integer(length(start_idxs)/10) == 0) {
    portion_complete <<- portion_complete + 1
    print(paste0(10*portion_complete,"% complete..."))
  }


}

print(paste0(num_fail," out of ",nrow(sr_data)," records failed to join."))

end_calc_h3 <- Sys.time()
calc_h3_time <- difftime(end_calc_h3, start_calc_h3)
print(paste("calc_h3_time =",calc_h3_time))

start_join_data <- Sys.time()
# Join the tables by h3_level_8_index
hex_8_data$features$h3_level8_index <- hex_8_data$features$properties$index
```

```
sr_data_combined <- sr_data %>% left_join(hex_8_data$features,by="h3_level8_index")

end_join_data <- Sys.time()
join_data_time <- difftime(end_join_data, start_join_data)
print(paste("join_data_time =",join_data_time))
```

### Validation

To validate the data we simply have to download a provided dataset that includes service request data matched with H3 level 8 indexes and compare it with the joined dataset that we created above. If all the indexes match then we have successfully joined the datasets.

```
# VALIDATION ======================================================================

start_validation <- Sys.time()
# Download validation data if not present

sr_hex_data_filename <- "data/sr_hex.csv"
sr_hex_gz_data_filename <- "data/sr_hex.csv.gz"
options(timeout=180)
if(!file.exists(sr_hex_data_filename)) {
  sr_hex_data_url <- "https://cct-ds-code-challenge-input-data.s3.af-south-1.amazonaws.com/sr_hex.csv.gz
  download.file(sr_hex_data_url, sr_hex_gz_data_filename)
  gunzip(sr_hex_gz_data_filename, remove=FALSE)
}


# Load validation dataset
# For testing purposes we are only going to read in a subset of the data
sr_hex_data <- read_csv("data/sr_hex.csv",n_max = num_lines,skip = 0, col_types = cols());

# Compare indexes to make sure they are the same (assume in same order for now)
h3_indexes_true <- sr_hex_data$h3_level8_index
h3_indexes_computed <- sr_data_combined$h3_level8_index

validation_result <- all(h3_indexes_true == h3_indexes_computed)
if(validation_result) {
  print("Validation passed! :)")
} else
{
  print("Validation failed! :(")
}


end_validation <- Sys.time()
validation_time <- difftime(end_validation, start_validation)
print(paste("validation_time =",validation_time))


total_time <- difftime(end_validation, start)
print(paste("total_time =",total_time))
```

### Performance

The time taken to perform these operations was logged to the command line and is presented in the table below.

```
# Table showing the logged times for each subsection and the total
log_times <- c(download_time,load_time,calc_h3_time,
               join_data_time,validation_time,total_time)
log_times_df <- data.frame(Times = log_times)
colnames(log_times_df) <- c("Times")
rownames(log_times_df) <- c("Download","Load","Calc H3 Index",
                            "Join Data","Validation","Total")
kable(log_times_df, caption = "Log times for Initial Data Transformation Task")
```

### Conclusions

We have now successfully joined the service request dataset and the H3 level 8 dataset. The number of failed joins is output to the user at the end. If the location data is not valid for a percentage of the data read so far then the program exits. From minimal testing, the parallelisation seems to speed up the program.

The joined dataset was validated using a provided dataset and comparing H3 level 8 indexes.