Task 1: Data Extraction

James Mbewu

13/06/2021

Overview

This R Markdown document serves as the report and code for Task 1: Data Extraction for the City of Cape Town Data Science Unit Code Challenge. The code by itself can be found in the file data_extraction.R.

Brief

Use the AWS S3 SELECT command to read in the H3 resolution 8 data from `city-hex-polygons-8-10.geojson`. Use the `city-hex-polygons-8.geojson` file to validate your work.

Please log the time taken to perform the operations described, and within reason, try to optimise latency and computational resources used.

Report

The crux of this task is to use AWS S3 SELECT to query and read in the H3 resolution data of level 8 from a larger dataset in a bucket on AWS S3 that contains H3 resolution data of levels 8, 9 and 10. For this I used the R package paws.

The query was performed and the dataset extracted was validated by comparing H3 resolution indexes with an existing dataset that contained only H3 resolution data of level 8. The time to complete the task and subtasks was logged.

AWS credentials

We first load the AWS S3 credentials from the link provided and set the appropriate environmental variables that paws will use to access files in the AWS S3 bucket. For security reasons it is important to explicitly write these kinds of credentials directly in the file. Then we establish the connection with AWS S3 using paws.

```r
rm(list = ls())

start <- Sys.time()

# Setup AWS S3 access
aws_credentials_url <- "https://cct-ds-code-challenge-input-data.s3.af-south-1.amazonaws.com/ds_code_cha
secrets <- fromJSON(aws_credentials_url)

Sys.setenv(
  "AWS_ACCESS_KEY_ID" = secrets$s3$access_key,
  "AWS_SECRET_ACCESS_KEY" = secrets$s3$secret_key,
  "AWS_DEFAULT_REGION" = "af-south-1",
  "AWS_REGION" = "af-south-1"
)
```

```r
# Setup connection to AWS S3
s3 <- paws::s3()
```

Extracting the data

It is much more efficient to extract only the data we require than downloading all the data and filtering it locally. To extract the level 8 data we need to query the features array within the .geojson file and extract all the attributes from features that represent level 8 H3 hexes. The response is passed back in json format and then parsed into an R data frame.

```r
# EXTRACT RES 8 FEATURES ================================

# Setup query to select only data from features with resolution 8
query <- "SELECT d.type, d.properties, d.geometry FROM  S3Object[*].features[*] d WHERE d.properties.res

# Perform query
extracted_h3_res8 <- s3$select_object_content(
  Bucket = "cct-ds-code-challenge-input-data",
  Key = "city-hex-polygons-8-10.geojson",
  Expression = query,
  ExpressionType = "SQL",
  RequestProgress = list(
    Enabled = TRUE
  ),
  InputSerialization = list(
    JSON = list(
      Type = "DOCUMENT"
    )
  ),
  OutputSerialization = list(
    JSON = list(
      RecordDelimiter = ","
    )
  )
)

# Parse query response to be in json array format and convert to df
extracted_h3_res8 <- extracted_h3_res8$Payload$Records$Payload %>%
  str_sub(1,-1L -1)
extracted_h3_res8 <- paste0("[",extracted_h3_res8,"]")

extracted_h3_res8_df <- extracted_h3_res8 %>%
  jsonlite::fromJSON()


end_extraction <- Sys.time()
extraction_time <- difftime(end_extraction, start)
print(paste("extraction_time =",extraction_time))
```

```
## [1] "extraction_time = 5.48149657249451"
```

Validation

To validate that we extracted all the correct features, we compare the H3 level 8 indexes of the extracted data with a dataset that only includes the H3 level 8 data and was provided to us. If they are all the same then the validation test has passed. In this case we get the file directly from the AWS S3 bucket and convert it to an R dataframe.

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

start_validation <- Sys.time()

# Get city-hex-polygons-8.geojson for validation
h3_res8_obj <- s3$get_object(Bucket = "cct-ds-code-challenge-input-data",
              Key = "city-hex-polygons-8.geojson")

# Convert from raw to df
h3_res8_df <- h3_res8_obj$Body %>%
  rawToChar() %>%
  jsonlite::fromJSON()

# Compare indexes to make sure they are the same (assume in same order)
indexes_from_8_10_data <- extracted_h3_res8_df$properties$index
indexes_from_8_data <- h3_res8_df$features$properties$index

validation_result <- all(indexes_from_8_data == indexes_from_8_10_data)
if(validation_result) {
  print("Validation passed! :)")
} else
{
  print("Validation failed! :(")
}
```

```
## [1] "Validation passed! :)"
```

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

```
## [1] "validation_time = 5.82794761657715"
```

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

```
## [1] "total_time = 11.3235890865326"
```

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(extraction_time,validation_time,total_time)
log_times_df <- data.frame(Times = log_times)
rownames(log_times_df) <- c("Extraction","Validation","Total")
kable(log_times_df, caption = "Log times for Data Extraction Task")
```

Table 1: Log times for Data Extraction Task

	Times
Extraction	5.481497 secs
Validation	5.827948 secs

	Times
Total	11.323589 secs

Conclusions

We have successfully queried the H3 resolution data and extracted only the H3 level 8 data from the AWS S3 bucket. The extracted data was validated with an existing file to test if the extraction was successful.

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).

```r
# 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.

```r
# 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
```

```r
# 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.g
  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.

Task 5: Further Data Transformations

James Mbewu

13/06/2021

Overview

This R Markdown document serves as the report and code for Task 5: Further Data Transformations for the City of Cape Town Data Science Unit Code Challenge. The code by itself can be found in the file further_data_transformations.R.

Brief

Write a script which anonymises the `sr_hex.csv` file, but preserves the following resolutions (You may use H3 indexes or lat/lon coordinates for your spatial data):

- location accuracy to within approximately 500m
- temporal accuracy to within 6 hours
- scrubs any columns which may contain personally identifiable information.

We expect in the accompanying report that follows you will justify as to why this data is now anonymised. Please limit this commentary to less than 500 words. If your code is written in a code notebook such as Jupyter notebook or Rmarkdown, you can include this commentary in your notebook.

Report

The goal of this task is to anonymise the service request dataset `sr_hex.csv` while retaining a degree of spatial and temporal accuracy. We need to decide how to anonymise the different types of data that are present in the dataset.

Data Anonymisation

Data anonymisation is the task of mutating data so it contains less information than you started. It is important for a number of reasons such as personal data protection, removing bias in use of the data and removing bias in statistical analyses. In general it has become an important issue with the rise of online advertising and tracking, high profile data hacks, and the increased use of machine learning models in all spheres of life. It is important to anonymise carefully so that you don't remove all of the information in the data and render the data useless. You also don't always want the data to look like it has been anonymised and encourage further digging. There are many techniques that can be used to anonymise data, some of which we will apply here to this dataset.

The dataset `sr_hex.csv` contains data on service requests that the City of Cape Town has processed. The reasons it should be anonymised depend on the use case:

- We could want it to be anonymised so that any personal information (such as names, addresses, location, phone numbers, medical conditions etc.) that may have slipped into it can't be hacked or used improperly by people with access to the data. Some data might seem to not contain much information, but combined with other data it could prove more revealing.

- We could want it to be anonymised so that users of the data aren't biased in any way, for example towards or against a particular suburb or perhaps a type of service request like fixing a playground might be deemed less important.

- We might also want to anonymise it for statistical analysis so that inferences are not made with unconscious bias.

Here we detail how and why we have anonymised different attributes in the dataset. As stated above, there are many reasons to anonymise data, and what you anonymise may vary:

- **Code** - *Removed*:
  - this attribute contains more detailed human entered information that has the potential to contain personally identifiable information. It will be removed.

- **Latitude** and **Longitude** - *Perturbed*:
  - this spatial data could be used to pinpoint the exact location of a service request and should be changed so it can't be used for nefarious purposes. We still want to retain some location data so we will perturb it by a maximum of 500m.

- **h3_level_8_index** - *Removed*:
  - this could either be removed or recalculated for the perturbed location. If it was kept the same it could be used to triangulate a more accurate location than we want to allow.

- **SubCouncil2016**,**Wards2016** and **OfficialSuburbs** - *Removed*:
  - this attribute contains location information that may bias use of the data or enable linking with other datasets and used improperly. They can always be approximated from the perturbed location if necessary.

- **CreationTimestamp**,**CreationDate**,**CompletionTimestamp**,**CompletionDate**, **ModificationTimestamp** and **Duration** - *Perturbed*:
  - this temporal data could be used for example to infer things like when CCT workers are in the area. We still want to retain some of the data so we will perturb them by a maximum of 6 hours. Duration will be recalculated from this so that it remains consistent.

- **CodeGroup**,**directorate**,**department** - *Shuffled together*:
  - this set of data contains information on the departments that dealt with the service request. They are unlikely to contain personal information, but could bias the use of the data. We will shuffle the order of them, while grouping them together, to prevent this bias while still retaining some information.

- **NotificationNumber** - *Shuffled*:
  - this data could probably be used to link to other datasets and be used to infer more information. They will be shuffled.

- **Open**,**NotificationType** - *Retained*:
  - this data is not likely to contain any personal information or enable linking as it doesn't change much. It will be retained

Load Data

First we download the dataset if it isn't present already and then read it into an R data frame.

```r
rm(list = ls())
```

```r
num_lines <- Inf
# LOAD DATA ==========================================================

start <- Sys.time()

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

sr_data_filename <- "data/sr.csv"
sr_gz_data_filename <- "data/sr.csv.gz"
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))
```

```
## [1] "download_time = 0.00710678100585938"
```

```r
start_load <- Sys.time()

# Read in data
# 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())
```

```
## Warning: Missing column names filled in: 'X1' [1]

## Warning: 41636 parsing failures.
##  row                   col   expected actual              file
## 2099 ModificationTimestamp date like     NaT 'data/sr_hex.csv'
## 2182 ModificationTimestamp date like     NaT 'data/sr_hex.csv'
## 2230 ModificationTimestamp date like     NaT 'data/sr_hex.csv'
## 2415 ModificationTimestamp date like     NaT 'data/sr_hex.csv'
## 2721 ModificationTimestamp date like     NaT 'data/sr_hex.csv'
## .... ..................... .......... ...... .................
## See problems(...) for more details.
```

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

```
## [1] "load_data_time = 29.8501281738281"
```

Data Anonymisation

Next we perform the data anonymisation on the data frame. We first remove and then shuffle any columns that need to be shuffled.

```r
# DATA ANONYMISATION ==========================================================



start_anonymisation <- Sys.time()

```

```r
# Remove SubCouncil2016, Wards2016, OfficialSuburbs, Code and h3_level8_index
# NotificationType and NotificationNumber
drop_cols <- c("SubCouncil2016","Wards2016","OfficialSuburbs","Code",
               "h3_level8_index")
sr_hex_data <- sr_hex_data[ , !(names(sr_hex_data) %in% drop_cols)]

# Shuffle CodeGroup and directorate and department together
shuffle_cols_dep <- c("CodeGroup","directorate","department")
sr_hex_data[, shuffle_cols_dep] <- sr_hex_data[sample(1:nrow(sr_hex_data)), shuffle_cols_dep]

# Shuffle NotificationNumber
shuffle_cols_not <- c("NotificationNumber")
sr_hex_data[, shuffle_cols_not] <- sr_hex_data[sample(1:nrow(sr_hex_data)), shuffle_cols_not]
```

Then we do the random perturbations of the spatial data. We want the resulting location to be a maximum of about 500m away from the original location. This corresponds to a change in latitude and longitude of 350m. Around Cape Town, 350m in latitude is approximately 0.00314 degrees, while 350m in longitude is approximately 0.00382 degrees.

```r
# Do numerical transforms

# Spatial transforms
# add/subtract max 350m from the latitude and longitude
# 350m ~ 0.00314 deg lat and 0.00382 deg lon around Cape Town
set.seed(NULL)
lat_350 <- 0.00314
lon_350 <- 0.00382
# check it gives different random numbers
sr_hex_data <- sr_hex_data %>% mutate(Latitude = Latitude + runif(n(),-lat_350,lat_350),
                                      Longitude = Longitude + runif(n(),-lon_350,lon_350))
```

Next we do the random perturbations of the temporal data. CreationTimestamp and CompletionTimestamp are perturbed by a maximum of 3 hours so that the resulting Duration is only perturbed by a maximum of 6 hours. We also make sure the resulting Duration cannot be negative. The ModificationTimestamp is perturbed by the same amount as the CompletionTimestamp as they appear to be linked in the data.

```r
# Temporal transforms
# convert CompletionTimestamp to POSIXct
sr_hex_data <- sr_hex_data %>% mutate(CompletionTimestamp = str_sub(CompletionTimestamp,end=-7),
                                      CompletionTimestamp = as.POSIXct(CompletionTimestamp,format = "%Y
                                      CompletionTimestamp = CompletionTimestamp - 2*60*60)

# Add/subtract 3hrs from CreationTimestamp and CompletionTimeStamp/ModificationTimestamp
#  and adjust Duration and CreationDate and CompletionDate accordingly.
# Make sure Duration doesn't go negative by only allowing CreationTimestamp to
#  move forward 1/2 Duration and CompletionTimestamp only move back 1/2 Duration
sr_hex_data <- sr_hex_data %>% mutate(CreationTimestamp = CreationTimestamp +
                                        as.integer(round(runif(n(),-3*60*60,min(3*60*60,Duration/2*60*6
                                      CompletionPerturbation = as.integer(round(runif(n(),max(-3*60*60,
                                      CompletionTimestamp = CompletionTimestamp + CompletionPerturbatio

                                      ModificationTimestamp = ModificationTimestamp + CompletionPerturba
                                      Duration = (as.integer(CompletionTimestamp) - as.integer(Creation
                                      CreationDate = as.Date(CreationTimestamp),
                                      CompletionDate = as.Date(CompletionTimestamp)) %>%
                              select(-CompletionPerturbation)


end_anonymisation <- Sys.time()
anonymisation_time <- difftime(end_anonymisation, start_anonymisation)
print(paste("anonymisation_time =",anonymisation_time))
```

```
## [1] "anonymisation_time = 16.9904906749725"
```

Finally, we write the resulting dataframe to file.

```
# WRITE TO FILE ==========================================================

start_write <- Sys.time()

# Format CreationTimestamp, CompletionTimestamp, ModificationTimestamp for output
timestamp_format <- "%Y-%m-%d %H:%M:%S%z"
sr_hex_data <- sr_hex_data %>%
  mutate(CreationTimestamp = format(CreationTimestamp,format = timestamp_format),
         CompletionTimestamp = format(CompletionTimestamp,format = timestamp_format),
         ModificationTimestamp = format(ModificationTimestamp,format = timestamp_format))

# Write to file
col_names <- colnames(sr_hex_data)
col_names[1] <- ""
write.table(sr_hex_data,file="data/sr_hex_anon.csv",
            row.names = FALSE, col.names = col_names, sep=",",quote=FALSE)

end_write <- Sys.time()
write_time <- difftime(end_write, start_write)
print(paste("write_time =",write_time))
```

```
## [1] "write_time = 1.69600864648819"
```

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

```
## [1] "total_time = 2.47744425535202"
```

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,anonymisation_time,
               write_time,total_time)
log_times_df <- data.frame(Times = log_times)
colnames(log_times_df) <- c("Times")
rownames(log_times_df) <- c("Download","Load","Anonymisation",
                            "Write","Total")
kable(log_times_df, caption = "Log times for Further Data Transformations Task")
```

Table 1: Log times for Further Data Transformations Task

	Times
Download	0.0071068 secs
Load	29.8501282 secs
Anonymisation	16.9904907 secs
Write	101.7605188 secs
Total	148.6466553 secs

Conclusions

We have now successfully anonymised the dataset `sr_hex.csv`.

Task 6: Data Loading Tasks

James Mbewu

14/06/2021

Overview

This R Markdown document serves as the report and code for Task 6: Data Loading Tasks for the City of Cape Town Data Science Unit Code Challenge. The code by itself can be found in the file further_data_transformations.R.

Brief

Select a subset of columns (including the H3 index column) from the `sr_hex.csv` or the anonymised file created in the task above, and write it to the write-only S3 bucket.

Be sure to name your output file something that is recognisable as your work, and unlikely to collide with the names of others.

Report

The goal of this task is to upload a file containing a subset of the columns of the `sr_hex.csv` file to the AWS S3 bucket provided. To do this we again use the package paws. We will also verify that the upload has completed successfully by comparing the eTag returned from AWS S3 with the MD5 checksum of the file we uploaded.

Load Credentials

We first load the credentials we will use later for the upload.

```r
rm(list = ls())


# Setup AWS S3 access
aws_credentials_url <- "https://cct-ds-code-challenge-input-data.s3.af-south-1.amazonaws.com/ds_code_cha
secrets <- fromJSON(aws_credentials_url)

Sys.setenv(
  "AWS_ACCESS_KEY_ID" = secrets$s3$access_key,
  "AWS_SECRET_ACCESS_KEY" = secrets$s3$secret_key,
  "AWS_DEFAULT_REGION" = "af-south-1",
  "AWS_S3_ENDPOINT" = "",
  "AWS_REGION" = "af-south-1"
)
```

Prepare Data

Next we load the data from file and choose a subset of the columns (NotificationNumber, h3_level_8_index, Latitude and Longitude) to upload. The resulting data frame is then saved to file.

```r
# PREPARE DATA ==========================================================

start <- Sys.time()

# Download data if not present
# TODO: using local data for now

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
sr_hex_data <- read_csv("data/sr_hex.csv",n_max = 1000,skip = 0, col_types = cols())

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


start_save_subset <- Sys.time()

# Choose a subset of the columns
cols <- c("NotificationNumber","h3_level8_index","Latitude","Longitude")
#cols <- c("NotificationNumber")
sr_hex_data <- sr_hex_data[ , cols]

# Write the data to file
sr_sub_filename <- "data/sr_hex_sub_james_mbewu.csv"
write.table(sr_hex_data,file=sr_sub_filename,
            row.names = FALSE, sep=",",quote=FALSE)

end_save_subset <- Sys.time()
save_subset_time <- difftime(end_save_subset, start_save_subset)
print(paste("save_subset_time =",save_subset_time))
```

Upload data

Next we upload the file using paws and give it the name `sr_hex_sub_james_mbewu.csv` in the S3 bucket.

```r
# UPLOAD DATA ===========================================================

start_upload <- Sys.time()

# Setup connection to AWS S3
s3 <- paws::s3()

# # Upload file
# result <- s3$put_object(
#   Body = sr_sub_filename,
#   Bucket = "testbucketmbewu",
#   Key = "sr_hex_sub_james_mbewu.csv"
# )

result <- s3$put_object(
  Body = sr_sub_filename,
  Bucket = "cct-ds-code-challenge-input-data",
  Key = "sr_hex_sub_james_mbewu.csv"
)

end_upload <- Sys.time()
upload_time <- difftime(end_upload, start_upload)
print(paste("upload_time =",upload_time))
```

Verification

Finally we verify that the file uploaded was the file we sent to the S3 bucket. This can be done by comparing the eTag returned by S3 with the MD5 checksum of the file. If the upload was not a multipart upload, then they should be identical. If the upload was a multipart upload, then we would need to do some calculations to figure out the MD5 checksum of the merged file in the S3 bucket and verify it is indeed the same as what we sent.

```r
# VERIFICATION ==================================================================

start_verification <- Sys.time()
# compare the eTag and the md5sum
eTag <- str_extract(result$ETag,"[:alnum:]+")
num_parts <- as.numeric(str_match(string = eTag,
                        pattern = '-(\\d+)$')[2])

# Check it wasn't a multipart upload
if(is.na(num_parts)){
  md5 <- as.character(md5sum(sr_sub_filename))
  if(md5 == eTag) {
    print("File successfully uploaded. :)")
  }
  else{
    print("File was NOT successfully uploaded. :(")
  }
} else{
  print("Multipart upload verification not supported yet.")
}


end_verification <- Sys.time()
verification_time <- difftime(end_verification, start_verification)
print(paste("verification_time =",verification_time))


total_time <- difftime(end_verification, 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,save_subset_time,
               upload_time,verification_time,total_time)
log_times_df <- data.frame(Times = log_times)
colnames(log_times_df) <- c("Times")
rownames(log_times_df) <- c("Download","Load","Save Subset",
                            "Upload","Verification","Total")
kable(log_times_df, caption = "Log times for Data Loading Tasks Task")
```

Conclusions

Unfortunately, when I ran this on Monday morning I got a `Error: AccessDenied (HTTP 403). Access Denied` error. I'm not too sure what the issue was because I was able to successfully query data in the bucket. So it isn't a problem of connecting to it. In addition I was able to upload data when testing it with my own bucket.

In any case, when testing on my own bucket the I could verify that the file in the bucket is indeed the same file that we sent and the upload was successful.