Data Preview

Invalid Date

# 1. Overview

In this section, the methodology will be explained thereafter Exploratory Data Analysis will be done.

# 2. Data Acquisition

3 main spectrum of data will be required for this research, namely the Population Data, Master Plan 2019 Subzone boundary and Care Centres.

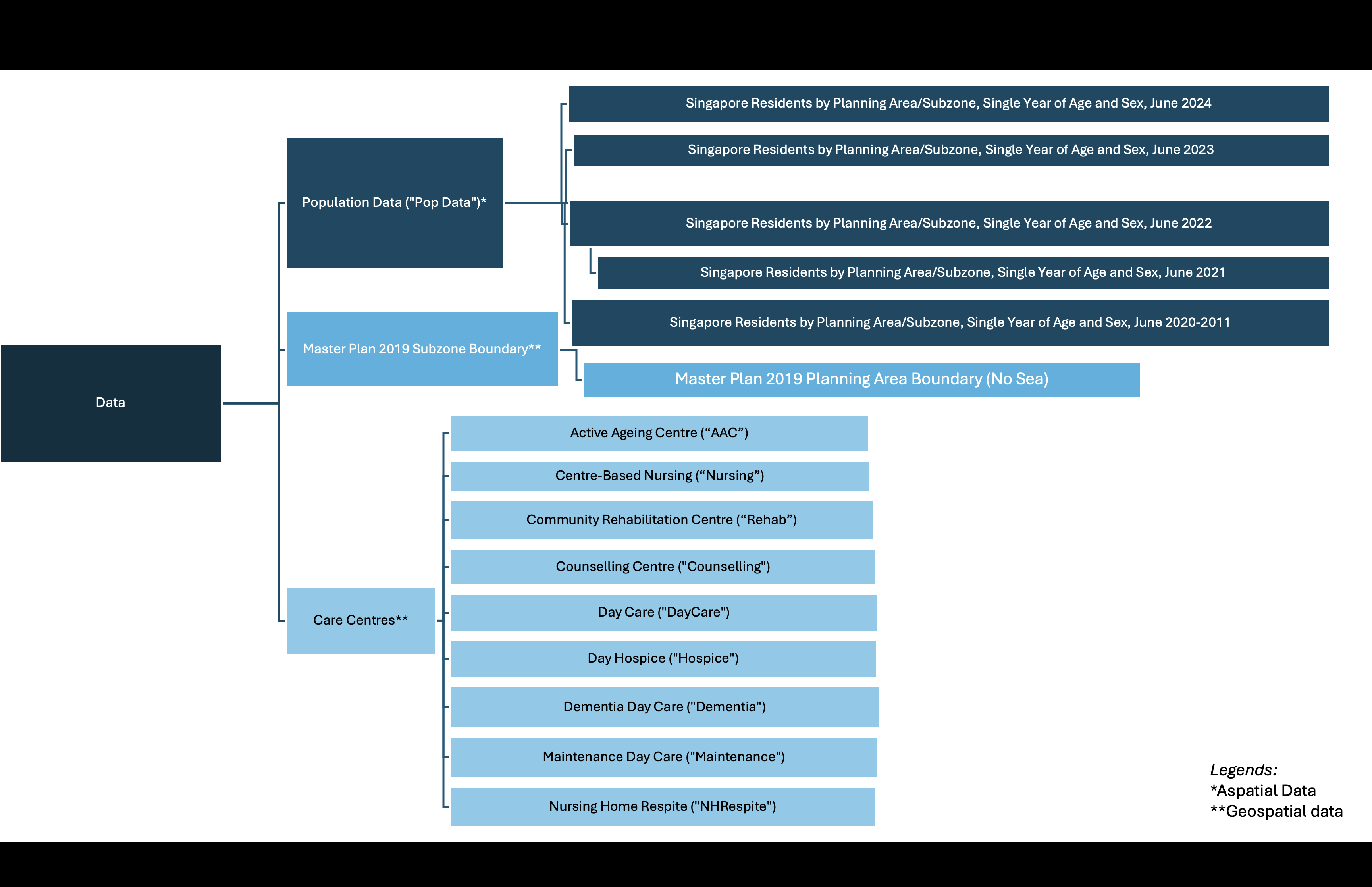


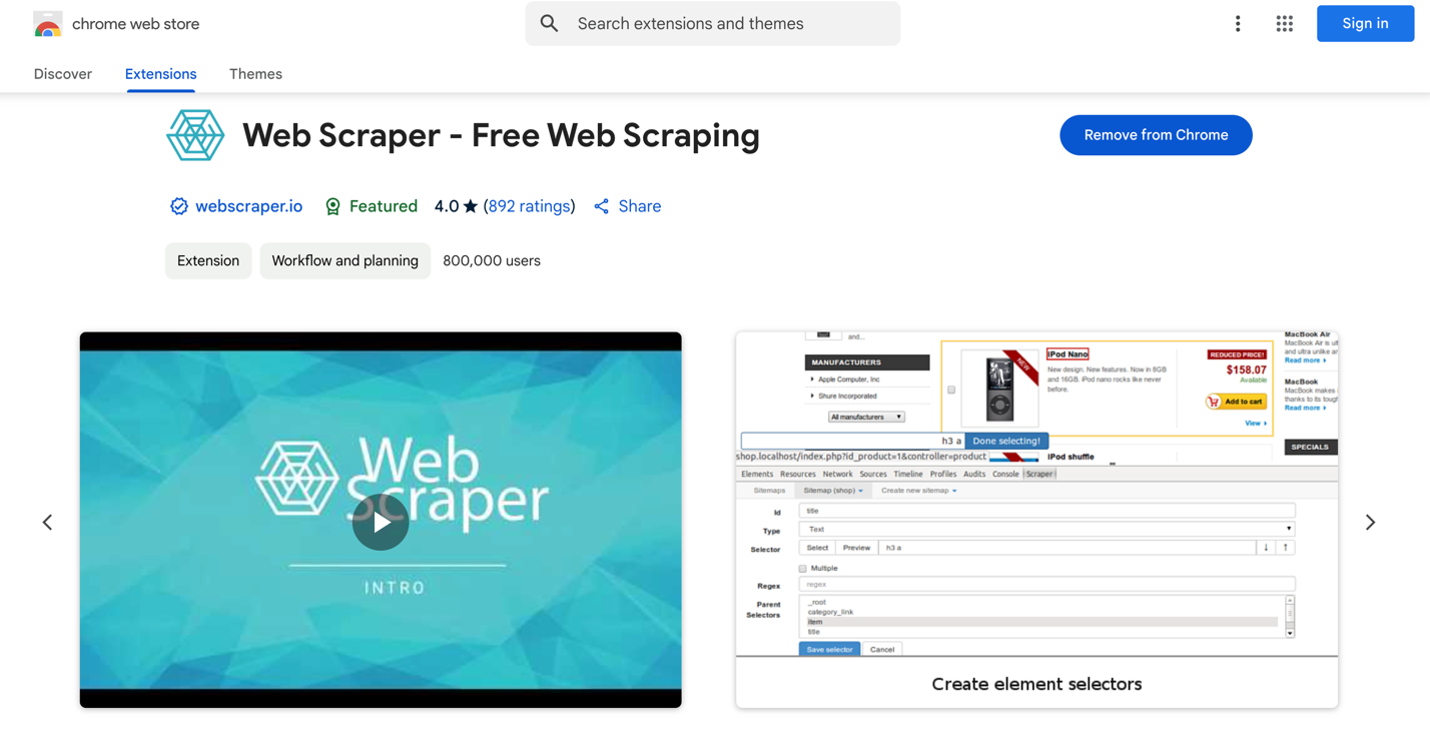
Figure x: Data Overview

### 2.0.1 WebScraping of Care Centres

Due to the lack of a centralised data of all care centres, web scarping is warranted in obtaining the information of the care centres. The geographical locations of the Care Centres alongside the centre names such as Active Ageing Centre, Day Care, Community Rehabilitation Centre, Centre-based Nursing were extracted using a web scraping tool, Web Scraper, available in Chrome web store as Seen in Figure x. As there is no centralised file that consist of the centre names and their locations, the location of each centre has to be manually extracted from the [Care Services](https://www.aic.sg/care-services/) webpage of the Agency of Integrated Care as seen in Figure x.

#### 2.0.1.1 Step 0: Download Web Scraper from Chrome web store

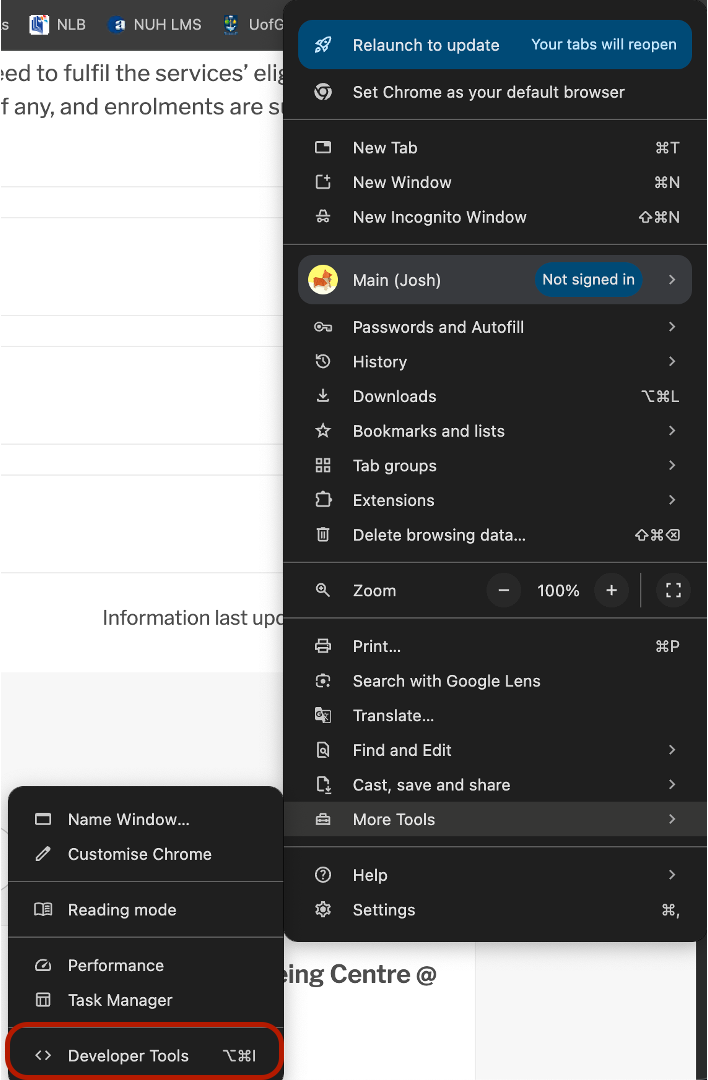
Web Scraper is used as it is free, works reasonably well and available in both Chrome and Firefox web store. In the below steps, Chrome will be the default web browser used.



*Figure x: Web Scraper*

#### 2.0.1.2 Step 1: Navigate to Developer Tools in Chrome Web Browser

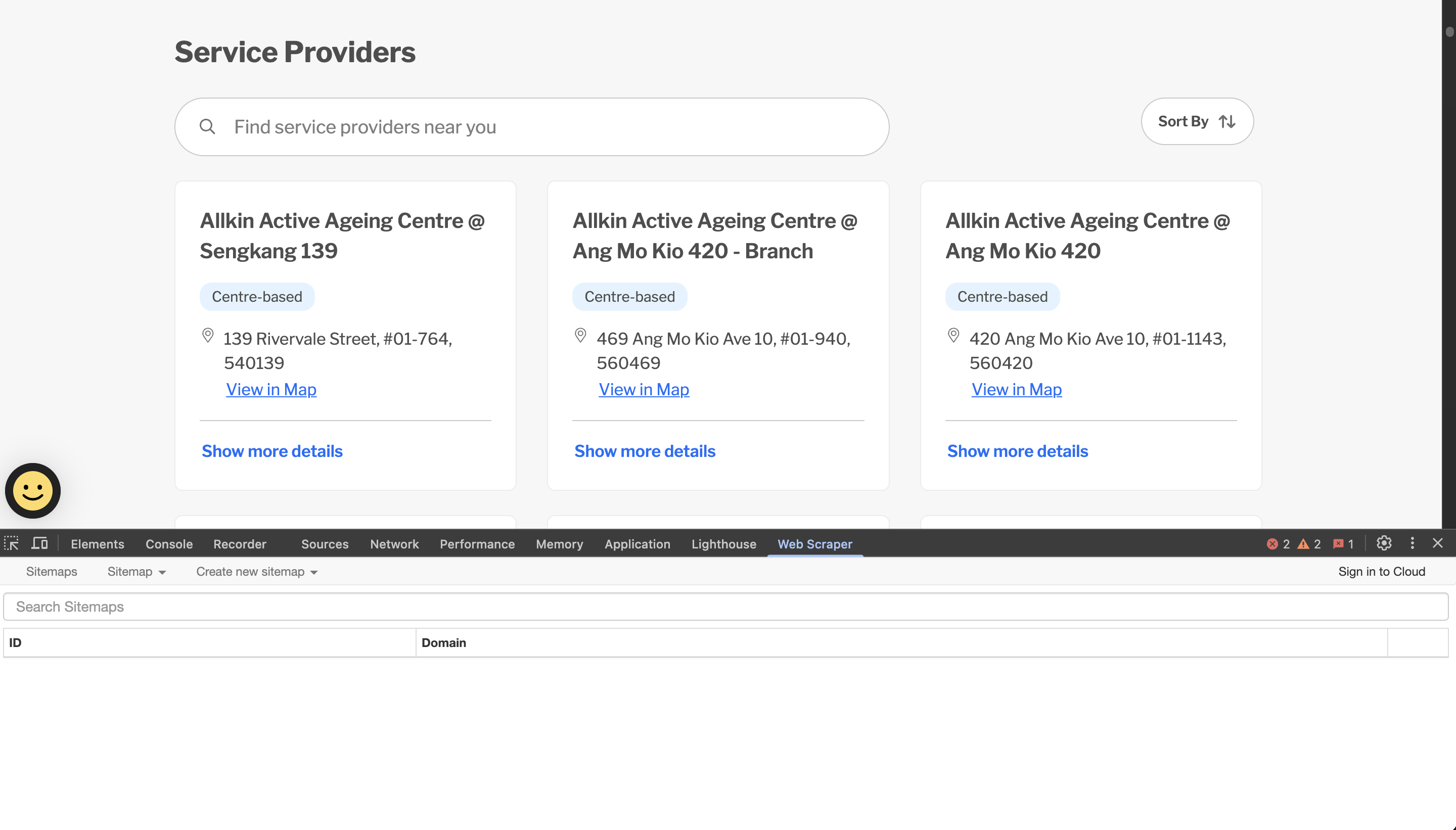
After downloading the extension from Chrome Web Store, press onto the menu bar at the right of the browser and locate Developer Tools while onto the website you would like to scrape information from.



*Figure x: Web Scraper*

#### 2.0.1.3 Step 2: Interface for Webscraper

After clicking onto Developer Tools, click onto the Web Scraper in the menu bar (in black). Following which the below interface will appear.

[](Figure%20x:%20Locate%20Developer%20Tools)

#### 2.0.1.4 Step 3: Create New Sitemap

Click onto “create new sitemap”, thereafter “Create Sitemap”. Sitemap Name will be the overarching term used for these information; in this instance, it will be AAC. The Start URL will be the HTML link that you would like the information to be scraped from.

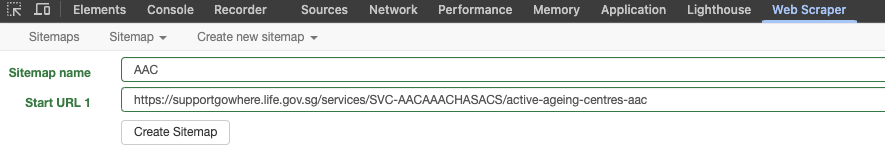


Figure x:

#### 2.0.1.5 Step 4: Add New Selector

After creating a new sitemap, the following interface will appear. Click onto the “Add new selector” to select the information to scrape.

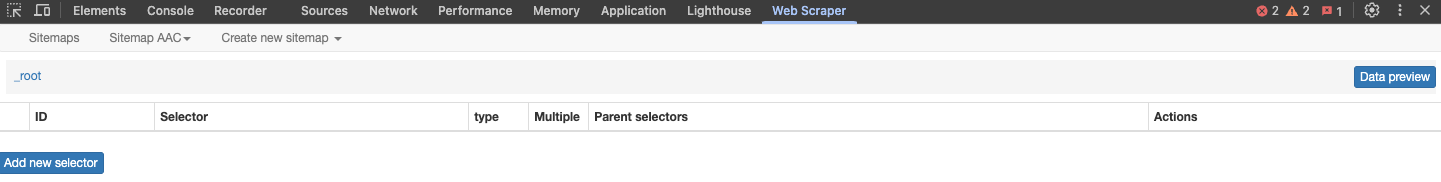


Figure x:

#### 2.0.1.6 Step 5: Selecting Whole Box

Firstly, the id will be the column name. For Type, select Element Attribute from the drop down selection. Thereafter, press on Select under Selector and select two boxes of each centre as seen in the figure below (the remaining boxes will be highlighted through its intelligent function) and press onto Done Selecting in the green box.

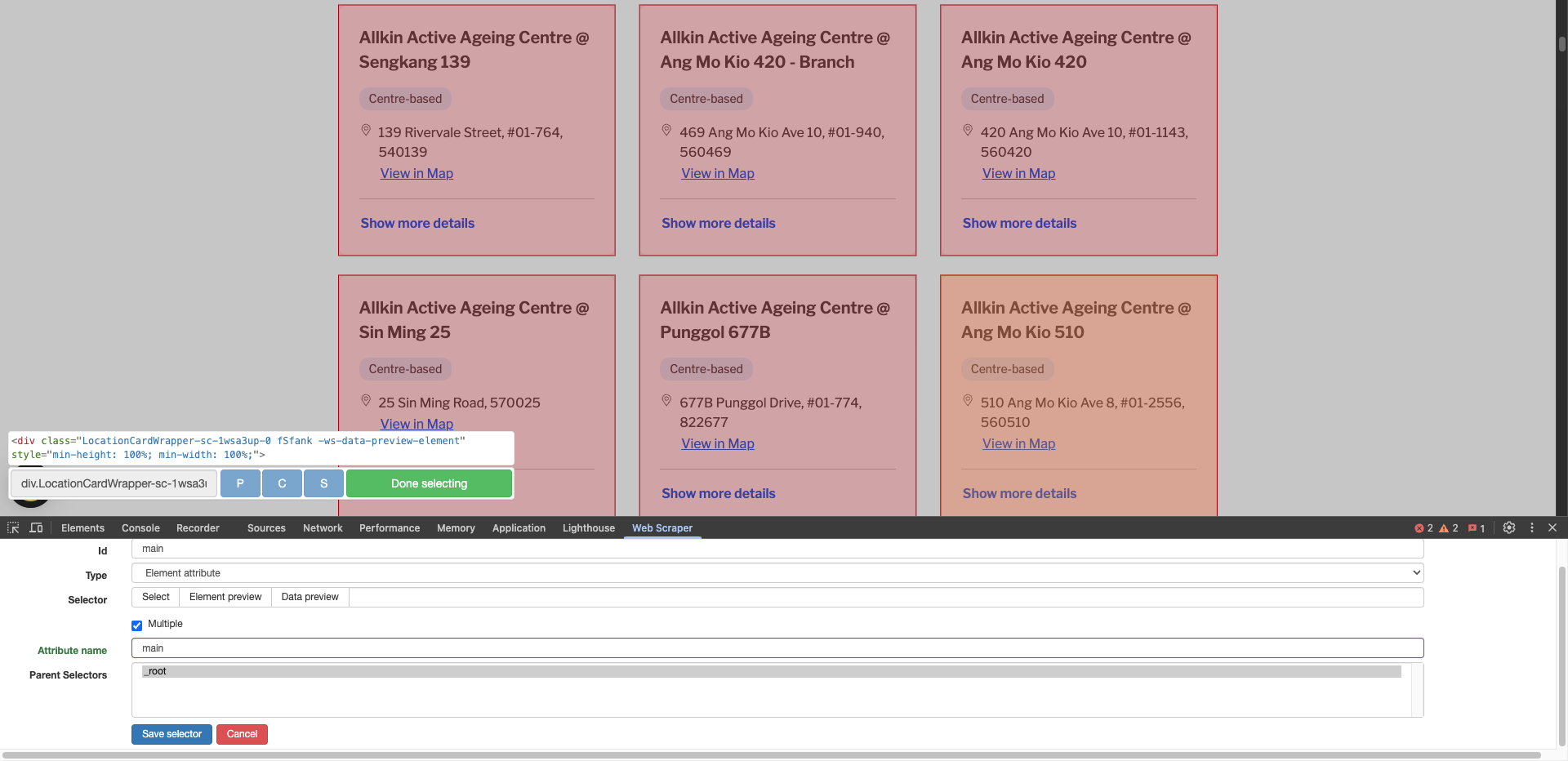


Figure x:

#### 2.0.1.7 Step 6: Sitemap Interface

After adding a new selector, the sitemap page will appear the selector that you’ve inputted.

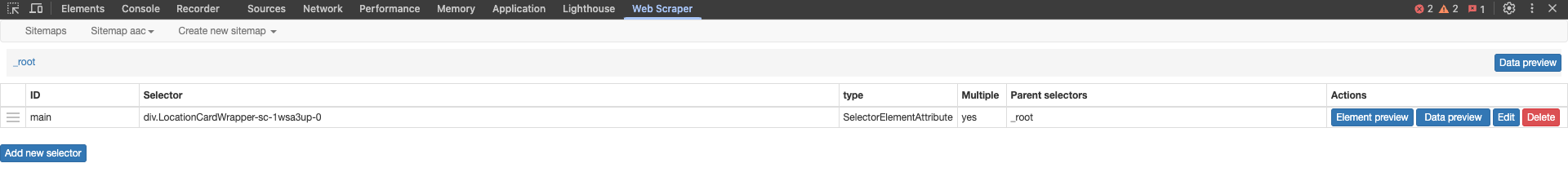


Figure x: Step 6 - Create New Sitemap

#### 2.0.1.8 Step 7: Selecting Name of Care Centre

Firstly, the id will be name (with reference to the name of care centre), serving as the column name. Text will be chosen under Type thereafter press Select under Selector and highlight the first 2 names of the care centres (The remaining care centres will be highlighted through its intelligent function) and press onto Done selecting in the green box. Multiple box will be selected as we would like to scrap multiple names and root parent selector will be root and press onto Save Selector.

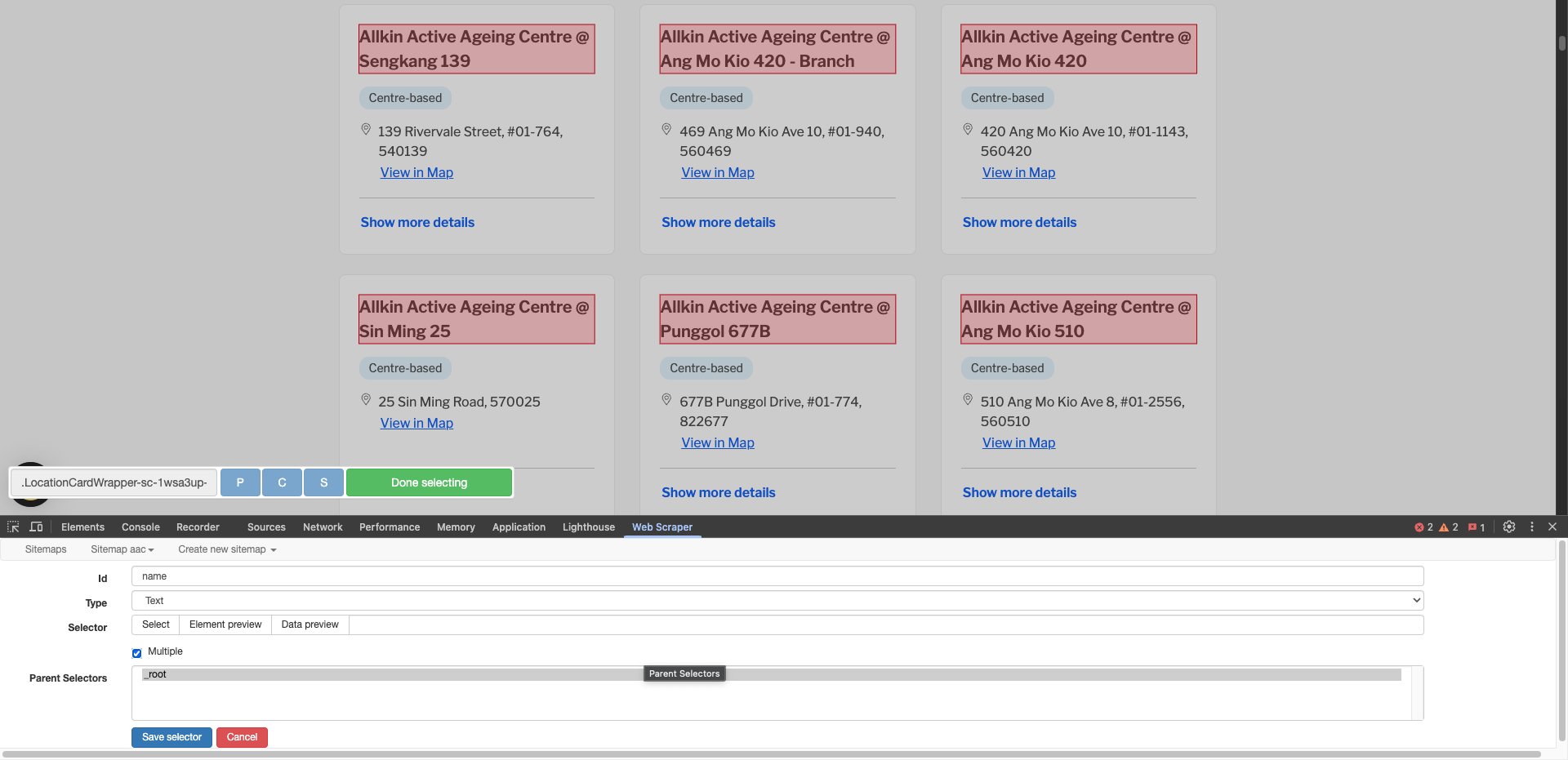


Figure x: Step 7 - Selecting Name of Care Centre

#### 2.0.1.9 Step 8: Create New Sitemap

A popup window will be prompted and Group selectors was selected.



Figure x: Step 8 - Create New Sitemap

#### 2.0.1.10 Step 9: Selecting Address of Care Centre

Similar to Step 7, the id will be address. Text will be chosen under Type thereafter press Select under Selector and highlight the first 2 addresses of the care centres (Remaining addresses will be highlighted through its intelligent function) and press onto Done selecting in the green box. Multiple box will be selected as we would like to scrap multiple addresses and parent selector will be wrapper\_for\_main\_name (as we grouped selectors in step 8) and press onto Save Selector.

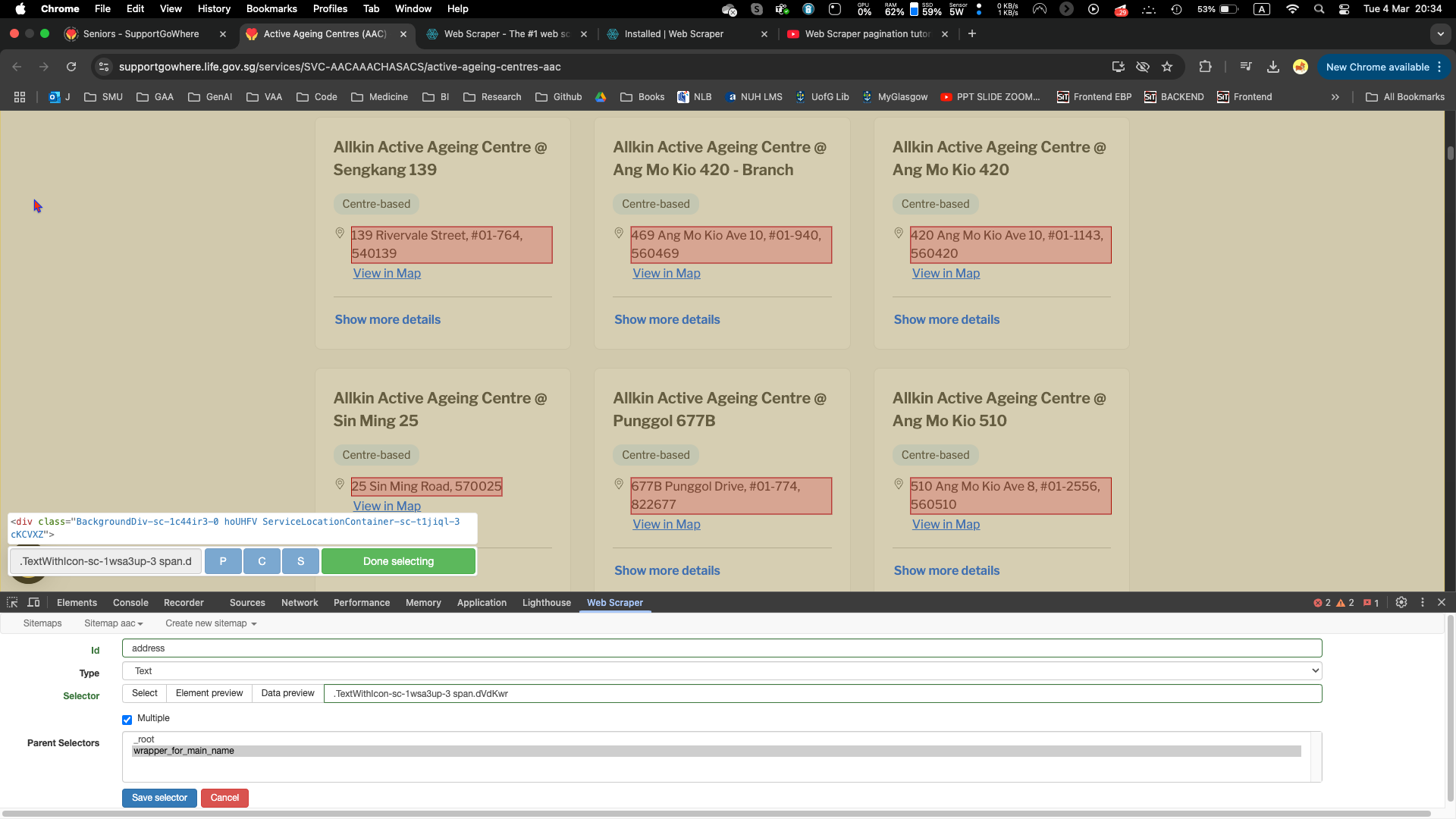


Figure x: Step 9 - Selecting Address of Care Centre

#### 2.0.1.11 Step 10: Data Preview

Prior to data scraping, the data is previewed in ensuring each name of the care centre is correctly tagged to the address using the main website to verify.

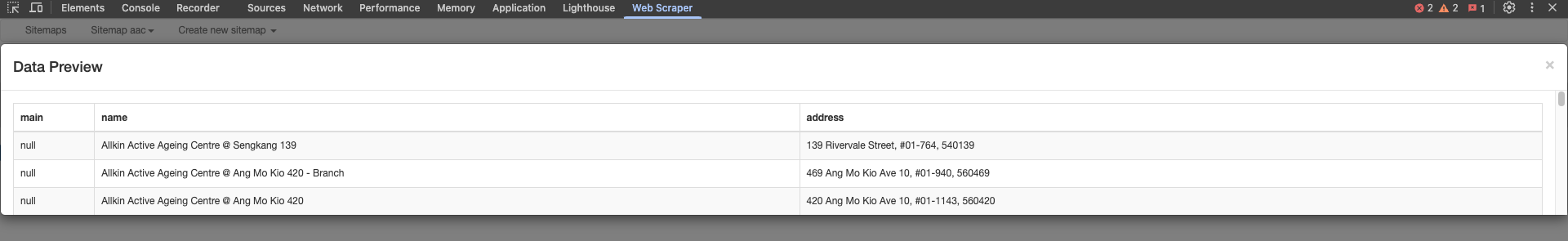


Figure x: Step 10 - Data Preview

#### 2.0.1.12 Step 11: Commence Scraping

Head over to sitemap aac and click onto Scrape. A new browser will appear indicating that it is in process of scraping. It will be closed automatically once the process has ended.

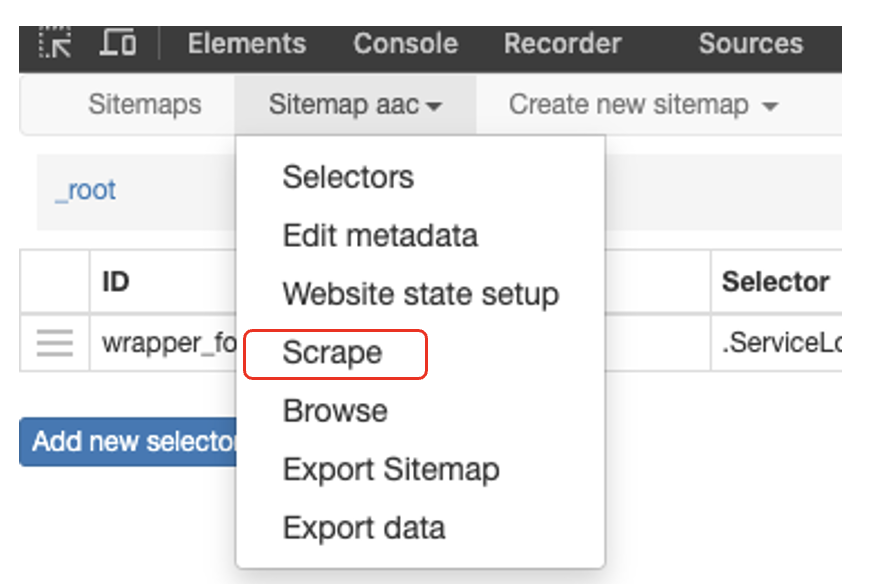


Figure x: Step 11 - Commence Scraping

#### 2.0.1.13 Step 11: Export Data

Export data is selected upon clicking sitemap aac. 2 file options are offered: csv and xlsx. The former was chosen as CSV files are simple and portable which doesn’t complicate data processing. Thereafter the data will be downloaded.

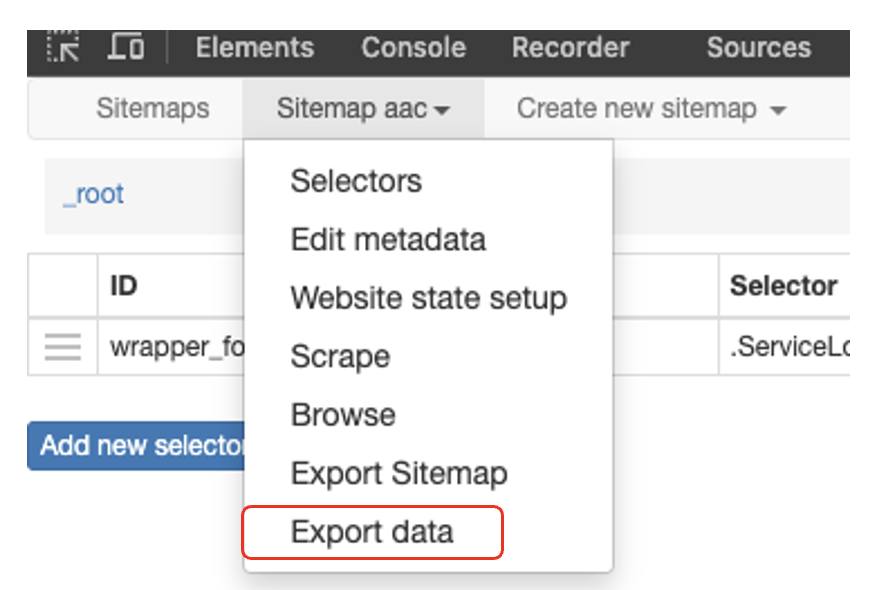


Figure x:

#### 2.0.1.14 Step 11: View CSV File

In ensuring the web scraping successful and accurate, the csv. file is opened and viewed.

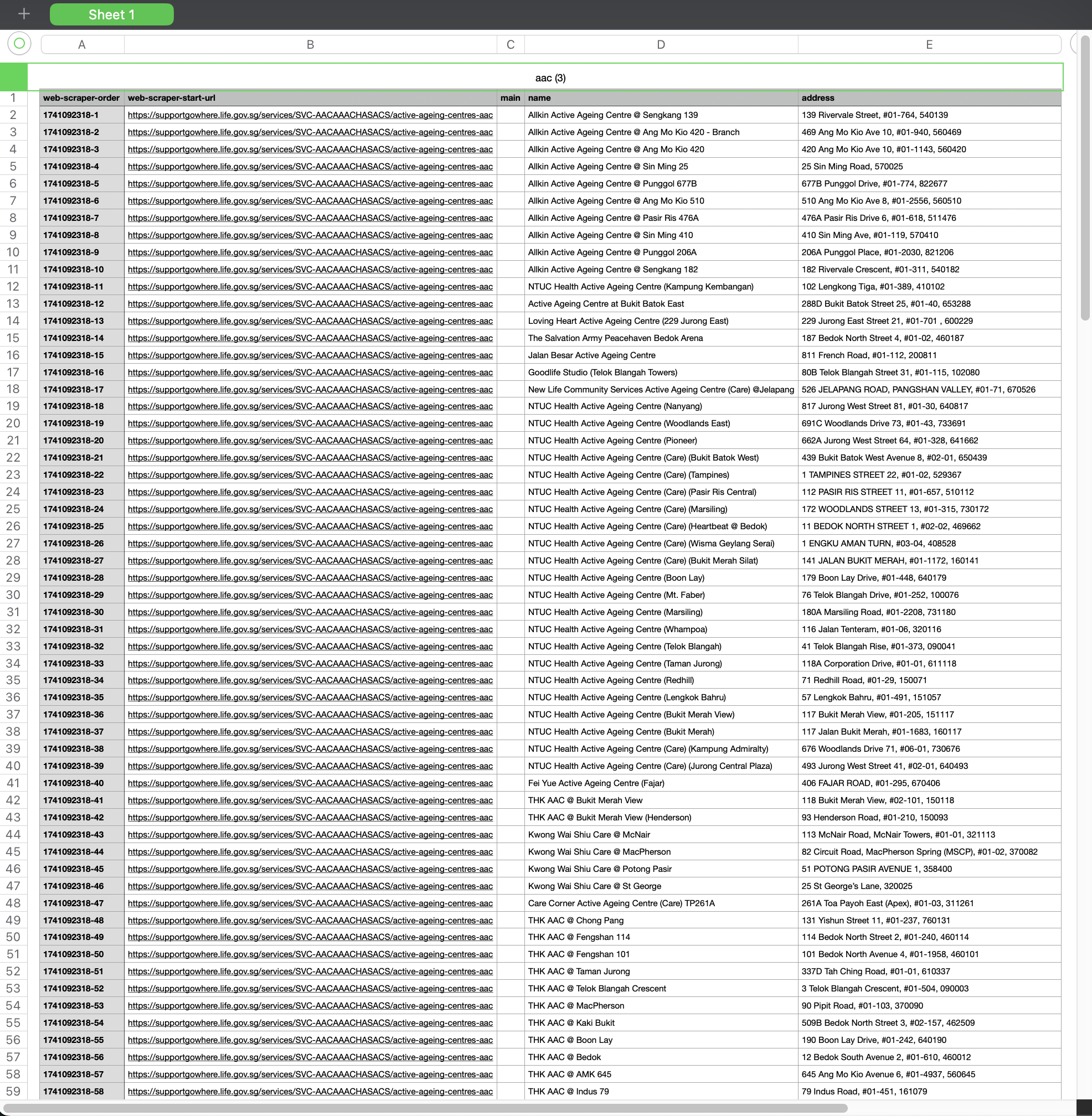


Figure x: Step 11 - View CSV File

The above steps were repeated for each care centre. All of the Care Centre data was extracted on 7th February 2025.

## 2.1 Packages

## Code

The code chunk below, using p\_load function of the [**pacman**](https://cran.r-project.org/web/packages/pacman/pacman.pdf) package, ensures that packages required are installed and loaded in R.

pacman::p\_load(tidyverse, sf, httr,  
 jsonlite, rvest, dplyr, units,  
 lubridate, tmap)  
  
# - Creates a package list containing the necessary R packages  
# - Checks if the R packages in the package list have been installed  
# - If not installed, will install the missing packages & launch into R environment.

## 2.2 Geospatial Data

### 2.2.1 Importing Singapore’s Master Plan 2019 Subzone Boundary

mpsz = st\_read(dsn = "data/subzone/",  
 layer = "MP14\_SUBZONE\_NO\_SEA\_PL")

## 2.3 Care Centre (EXTRACT TRANSFORM LOAD)

aac <- read\_csv("data/carecentre/activeageingcentre.csv")

counselling <- read\_csv("data/carecentre/counselling.csv")

daycare <- read\_csv("data/carecentre/daycare.csv")

dementia <- read\_csv("data/carecentre/dementiadaycare.csv")

hospice <- read\_csv("data/carecentre/dayhospice.csv")

maintenance <- read\_csv("data/carecentre/maintenancedaycare.csv")

nhrespite <- read\_csv("data/carecentre/nhrespite.csv")

nursing <- read\_csv("data/carecentre/centrebasednursing.csv")

rehab <- read\_csv("data/carecentre/communityrehabcentre.csv")

### 2.3.1 Cursory View

Using the glimpse() function, we are able to see that various rows in each data set while sharing the same number of columns. Columns “web-scraper-order” and “web-scraper-start-url” are redundant, thus, will be removed. Additionally, the address includes the postal code and it will seperated from the main street name and block number to facilitate the geospatial mapping thereafter.

glimpse(popdata20)

### 2.3.2 Deleting Unwanted Codes

The following R code is used to remove the columns “web-scraper-order” and “web-scraper-start-url” from multiple datasets: The select() function from the dplyr package is used to select or remove columns from a data frame.

aac <- aac %>% select(-"web-scraper-order", -"web-scraper-start-url")  
counselling <- counselling %>% select(-"web-scraper-order", -"web-scraper-start-url")  
daycare <- daycare %>% select(-"web-scraper-order", -"web-scraper-start-url")  
dementia <- dementia %>% select(-"web-scraper-order", -"web-scraper-start-url")  
hospice <- hospice %>% select(-"web-scraper-order", -"web-scraper-start-url")  
maintenance <- maintenance %>% select(-"web-scraper-order", -"web-scraper-start-url")  
nhrespite <- nhrespite %>% select(-"web-scraper-order", -"web-scraper-start-url")  
nursing <- nursing %>% select(-"web-scraper-order", -"web-scraper-start-url")  
rehab <- rehab %>% select(-"web-scraper-order", -"web-scraper-start-url")

After removing the two columns, each data set has two columns, namely name and address only.

glimpse(aac)

### 2.3.3 Checking for Missing Values

To check for missing or null values in the name and address columns of each dataset, the code uses the summarise() function from the dplyr package. The summarise() function computes summary statistics for the specified columns, which in this case are name and address. The across() function is used to apply the sum(is.na(.)) operation to both columns simultaneously, counting the number of missing (NA) values in each column.

The is.na() function checks whether each value in the name and address columns is missing or null, returning TRUE for missing values and FALSE for non-missing values. The sum() function then counts the number of TRUE values, which corresponds to the number of missing values in each column. This process is applied to each dataset (aac, counselling, daycare, dementia, hospice, maintenance, nhrespite, nursing, and rehab). In conclusion it is able to identify the number of missing values in the name and address columns across all datasets, which helps assess the completeness of the data and highlights any issues that may require cleaning or imputation before further analysis. It returns 0 missing values.

# Checking for missing or null values in 'name' and 'address' columns  
aac\_missing <- aac %>% summarise(across(c(name, address), ~sum(is.na(.))))  
counselling\_missing <- counselling %>% summarise(across(c(name, address), ~sum(is.na(.))))  
daycare\_missing <- daycare %>% summarise(across(c(name, address), ~sum(is.na(.))))  
dementia\_missing <- dementia %>% summarise(across(c(name, address), ~sum(is.na(.))))  
hospice\_missing <- hospice %>% summarise(across(c(name, address), ~sum(is.na(.))))  
maintenance\_missing <- maintenance %>% summarise(across(c(name, address), ~sum(is.na(.))))  
nhrespite\_missing <- nhrespite %>% summarise(across(c(name, address), ~sum(is.na(.))))  
nursing\_missing <- nursing %>% summarise(across(c(name, address), ~sum(is.na(.))))  
rehab\_missing <- rehab %>% summarise(across(c(name, address), ~sum(is.na(.))))

### 2.3.4 Duplicate Check

The code provided checks for duplicate rows in each dataset (aac, counselling, daycare, dementia, hospice, maintenance, nhrespite, nursing, and rehab) by grouping the dataset by all columns using group\_by\_all(). It then filters out the rows that have duplicate combinations of values across all columns using filter(n() > 1). The n() function counts the number of occurrences for each combination of values, and filter(n() > 1) keeps only the rows that appear more than once (i.e., duplicates).

For each dataset, the nrow() function is used to check if there are any rows returned after filtering for duplicates. If there are duplicates (i.e., the number of rows is greater than zero), the dataset with the duplicate rows is returned. However, if no duplicates are found (i.e., nrow() equals zero), the code returns 0 to indicate that there are no duplicates in that dataset.

Thus, the code either returns the rows with duplicate values or 0 if no duplicates are present, providing an indication of whether duplicate entries exist in each dataset.

# Check for duplicates in 'aac'  
aac\_duplicate <- aac %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'counselling'  
counselling\_duplicate <- counselling %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'daycare'  
daycare\_duplicate <- daycare %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'dementia'  
dementia\_duplicate <- dementia %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'hospice'  
hospice\_duplicate <- hospice %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'maintenance'  
maintenance\_duplicate <- maintenance %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'nhrespite'  
nhrespite\_duplicate <- nhrespite %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'nursing'  
nursing\_duplicate <- nursing %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
  
# Check for duplicates in 'rehab'  
rehab\_duplicate <- rehab %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()

## 2.4 Data Manipulation (CC)

### 2.4.1 Seperating postal code from address

The code uses the mutate() function to extract the postal code (last 6 digits) from the address column of the individual dataset and store it in a new column called postal\_code. The postal code is then removed from the address column.

# Active Ageing Centre  
aac <-mutate(aac,  
 postal\_code = str\_extract(address, "[0-9]{6}$"), # Extract postal code  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$") # Remove postal code from address  
 )

# Counselling  
counselling <- mutate(counselling,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# Daycare  
daycare <- mutate(daycare,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# Dementia  
dementia <- mutate(dementia,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# Day Hospice  
hospice <- mutate(hospice,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# Maintenance Daycare  
maintenance <- mutate(maintenance,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# NH Respite  
nhrespite <- mutate(nhrespite,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# Centre Based Nursing  
nursing <- mutate(nursing,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# Community Rehab Centre  
rehab <- mutate(rehab,  
 postal\_code = str\_extract(address, "[0-9]{6}$"),  
 address = str\_remove(address, "[,]?\\s\*[0-9]{6}$")  
)

# Checking for missing or null values in 'name' and 'address' columns  
aac\_missing <- aac %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))  
counselling\_missing <- counselling %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))  
daycare\_missing <- daycare %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))  
dementia\_missing <- dementia %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))  
hospice\_missing <- hospice %>% summarise(across(c(name, address, postal\_code),~sum(is.na(.))))  
maintenance\_missing <- maintenance %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))  
nhrespite\_missing <- nhrespite %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))  
nursing\_missing <- nursing %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))  
rehab\_missing <- rehab %>% summarise(across(c(name, address, postal\_code), ~sum(is.na(.))))

### 2.4.2 Labelling each dataset appropriately

The below code chunk adds a column and naming it as “label” in relation to the name of the dataset. This is done so we are able to combine the dataset together and identify

aac <- aac %>%  
 mutate(label = "aac")

counselling <- counselling %>%  
 mutate(label = "counselling")

daycare <- daycare %>%  
 mutate(label = "daycare")

dementia <- dementia %>%  
 mutate(label = "dementia")

hospice <- hospice %>%  
 mutate(label = "hospice")

maintenance <- maintenance %>%  
 mutate(label = "maintenance")

nhrespite <- nhrespite %>%  
 mutate(label = "nhrespite")

nursing <- nursing %>%  
 mutate(label = "nursing")

rehab <- rehab %>%  
 mutate(label = "rehab")

### 2.4.3 Append all Care Centres into one dataset

The code combines multiple datasets (aac, counselling, daycare, dementia, hospice, maintenance, nhrespite, nursing, and rehab) into a single dataset named c\_data using the bind\_rows() function. This function appends the rows of each dataset, stacking them vertically, to create one consolidated dataset. The resulting c\_data will contain all the rows from the individual datasets, assuming they have the same column structure.

cc\_data <- bind\_rows(  
 aac,   
 counselling,  
 daycare,  
 dementia,  
 hospice,  
 maintenance,  
 nhrespite,  
 nursing,  
 rehab,  
)

### 2.4.4 USE THIS

If yes is 1, no returns 0.

pivoted\_cc\_data <- cc\_data %>%  
 select(-address) %>%  
  
 mutate(present = 1) %>% # Create a column to indicate presence (1)  
  
 pivot\_wider(  
  
 names\_from = label, # Pivot based on the 'label' column  
  
 values\_from = present,  
 values\_fill = list(0)# Use the 'present' column for the values  
   
 )

Grouping by postal\_code: The group\_by(postal\_code) function groups the dataset by the postal\_code column. This ensures that all rows with the same postal\_code are treated as a single group for further operations.

Retaining name and address: The summarise() function is used to retain the name and address columns. For each group (i.e., rows with the same postal\_code), the first(name) and first(address) functions are used to keep the first occurrence of these columns. This ensures that the name and address values are preserved in the final output.

Summarizing Label Columns:The across(aac:rehab, ~ ifelse(any(. == “yes”), “yes”, “no”)) part iterates over each label column (from aac to rehab). For each group, it checks if any row within the group has a “yes” for that label. If at least one “yes” is found, the combined row will have “yes” in that column; otherwise, it will have “no”.

Output:The result is a new dataset (cc\_data\_combined) where rows with the same postal\_code are combined into a single row. The name and address columns are retained, and the label columns are summarized to reflect whether any row in the group had a “yes”.

cc\_data\_t <- cc\_data\_t %>%  
 group\_by(postal\_code) %>%  
 summarise(  
 name = first(name), # Retain the first occurrence of 'name'  
 address = first(address), # Retain the first occurrence of 'address'  
 across(aac:rehab, ~ ifelse(any(. == "yes"), "yes", "no"))  
 )

### 2.4.5 Adding coordinates to care centre

add\_list <- sort(unique(pivoted\_cc\_data$postal\_code)) #parse a list as API cannot read df  
#unique reduces records to pass to portal  
#sort is used to easier to find geo codes

get\_coords <- function(add\_list){  
  
 # Create a data frame to store all retrieved coordinates  
 postal\_coords <- data.frame()  
   
 for (i in add\_list){  
 r <- GET('https://www.onemap.gov.sg/api/common/elastic/search?',  
 query=list(searchVal=i,  
 returnGeom='Y',  
 getAddrDetails='Y'))  
 data <- fromJSON(rawToChar(r$content))  
 found <- data$found  
 res <- data$results  
   
 # Create a new data frame for each address  
 new\_row <- data.frame()  
   
 # If single result, append   
 if (found == 1){  
 postal <- res$POSTAL   
 lat <- res$LATITUDE  
 lng <- res$LONGITUDE  
 new\_row <- data.frame(address = i,   
 postal = postal,   
 latitude\_wgs84 = lat, # renamed to clarify coordinate system  
 longitude\_wgs84 = lng) # renamed to clarify coordinate system  
 }  
   
 # If multiple results, drop NIL and append top 1  
 else if (found > 1){  
 # Remove those with NIL as postal  
 res\_sub <- res[res$POSTAL != "NIL", ]  
   
 # Set as NA first if no Postal  
 if (nrow(res\_sub) == 0) {  
 new\_row <- data.frame(address = i,   
 postal = NA,   
 latitude\_wgs84 = NA,   
 longitude\_wgs84 = NA)  
 }  
 else{  
 top1 <- head(res\_sub, n = 1)  
 postal <- top1$POSTAL   
 lat <- top1$LATITUDE  
 lng <- top1$LONGITUDE  
 new\_row <- data.frame(address = i,   
 postal = postal,   
 latitude\_wgs84 = lat,   
 longitude\_wgs84 = lng)  
 }  
 }  
 else {  
 new\_row <- data.frame(address = i,   
 postal = NA,   
 latitude\_wgs84 = NA,   
 longitude\_wgs84 = NA)  
 }  
   
 # Add the row  
 postal\_coords <- rbind(postal\_coords, new\_row)  
 }  
   
 # Convert to sf object with WGS84 coordinates (EPSG:4326)  
 # Filter out rows with NA coordinates first  
 valid\_coords <- postal\_coords[!is.na(postal\_coords$latitude\_wgs84) &   
 !is.na(postal\_coords$longitude\_wgs84), ]  
   
 if(nrow(valid\_coords) > 0) {  
 coords\_sf <- st\_as\_sf(valid\_coords,   
 coords = c("longitude\_wgs84", "latitude\_wgs84"),  
 crs = 4326)  
   
 # Transform to SVY21 (EPSG:3414)  
 coords\_svy21 <- st\_transform(coords\_sf, 3414)  
   
 # Extract coordinates  
 coords\_matrix <- st\_coordinates(coords\_svy21)  
   
 # Add SVY21 coordinates back to the original dataframe with desired column names  
 valid\_coords$longitude <- coords\_matrix[, 1] # SVY21 X coordinate as longitude  
 valid\_coords$latitude <- coords\_matrix[, 2] # SVY21 Y coordinate as latitude  
   
 # Merge back with rows that had NA coordinates  
 result <- merge(postal\_coords, valid\_coords[c("address", "longitude", "latitude")],   
 by = "address", all.x = TRUE)  
 } else {  
 # If no valid coordinates, add empty SVY21 columns  
 result <- postal\_coords  
 result$longitude <- NA # SVY21 coordinates  
 result$latitude <- NA # SVY21 coordinates  
 }  
   
 return(result)  
}

coords <- get\_coords(add\_list)

The longtitude and latitude is then combined into geometry and the crs has been set to EPSG = 3414.

coords\_sf <- coords %>%  
 st\_as\_sf(coords = c("longitude", "latitude"), crs = 3414, remove = FALSE) %>%  
 select(address, postal, longitude, latitude, latitude\_wgs84, longitude\_wgs84)

### 2.4.6 try this

cc\_data\_final <- cc\_data\_t %>%  
 left\_join(coords\_sf,   
 join\_by(postal\_code = postal)  
)

## 2.5 Population Data

### 2.5.1 Importing Data

[Singapore Residents by Planning Area / Subzone, Single Year of Age and Sex, June 2024](https://www.singstat.gov.sg/-/media/files/find_data/population/statistical_tables/respopagesex2024.ashx)

popdata24 <- read\_csv("data/popdata/respopagesex2024.csv")

[Singapore Residents by Planning Area / Subzone, Single Year of Age and Sex, June 2023](https://www.singstat.gov.sg/-/media/files/find_data/population/statistical_tables/respopagesex2023.ashx)

popdata23 <- read\_csv("data/popdata/respopagesex2023.csv")

[Singapore Residents by Planning Area / Subzone, Single Year of Age and Sex, June 2022](https://www.singstat.gov.sg/-/media/files/find_data/population/statistical_tables/respopagesex2022.ashx)

popdata22 <- read\_csv("data/popdata/respopagesex2022.csv")

|  |
| --- |
| PARSING ERROR\* |
| Warning: One or more parsing issues, call `problems()` on your data frame for details, e.g.:  dat <- vroom(...)  problems(dat)Rows: 60424 Columns: 6── Column specification |

[Singapore Residents by Planning Area / Subzone, Single Year of Age and Sex, June 2021](https://www.singstat.gov.sg/-/media/files/find_data/population/statistical_tables/respopagesex2021.ashx)

popdata21 <- read\_csv("data/popdata/respopagesex2021.csv")

[Singapore Residents by Planning Area / Subzone, Single Year of Age and Sex, June 2011-2020](https://www.singstat.gov.sg/-/media/files/find_data/population/statistical_tables/respopagesex2011to2020.ashx)

popdata20 <- read\_csv("data/popdata/respopagesex2011to2020.csv")

glimpse(popdata20)

popdata24 <- popdata24 %>%  
 mutate(Age = ifelse(Age=="90\_and\_over", "90", Age),   
 Age = parse\_number(Age))

popdata23 <- popdata23 %>%  
 mutate(Age = ifelse(Age=="90\_and\_over", "90", Age),   
 Age = parse\_number(Age))

popdata22 <- popdata22 %>%  
 mutate(Age = ifelse(Age=="90\_and\_over", "90", Age),   
 Age = parse\_number(Age))

popdata21 <- popdata21 %>%  
 mutate(Age = ifelse(Age=="90\_and\_over", "90", Age),   
 Age = parse\_number(Age))

popdata20 <- popdata20 %>%  
 mutate(Age = ifelse(Age=="90\_and\_over", "90", Age),   
 Age = parse\_number(Age))

### 2.5.2 Changing Columns to lowercase

**names(popdata20)**: This part of the code retrieves the current column names of the dataset popdata20. The names() function in R is used to get or set the names of an object, such as the column names of a data frame.

**tolower(names(popdata20))**: The tolower() function is applied to the column names retrieved in the previous step. This function converts all characters in the names to lowercase. For example, if a column name is "PA", it will become "pa".

**names(popdata20) <- ...**: This part assigns the new lowercase column names back to the dataset popdata20. The <- operator is used to update the column names of the dataset with the lowercase versions.

# Assuming your dataset is already loaded as popdata20  
names(popdata20) <- tolower(names(popdata20))  
names(popdata21) <- tolower(names(popdata21))  
names(popdata22) <- tolower(names(popdata22))  
names(popdata23) <- tolower(names(popdata23))  
names(popdata24) <- tolower(names(popdata24))

### 2.5.3 Checking for Missing Values

To check for missing or null values in the name and address columns of each dataset, the code uses the summarise() function from the dplyr package. The summarise() function computes summary statistics for the specified columns, which in this case are name and address. The across() function is used to apply the sum(is.na(.)) operation to both columns simultaneously, counting the number of missing (NA) values in each column.

The is.na() function checks whether each value in the name and address columns is missing or null, returning TRUE for missing values and FALSE for non-missing values. The sum() function then counts the number of TRUE values, which corresponds to the number of missing values in each column. This process is applied to each dataset (aac, counselling, daycare, dementia, hospice, maintenance, nhrespite, nursing, and rehab). In conclusion it is able to identify the number of missing values in the name and address columns across all datasets, which helps assess the completeness of the data and highlights any issues that may require cleaning or imputation before further analysis. It returns 0 missing values.

Results: We noticed that there are 30 missing values popdata22 specifically under the column pop.

popdata20\_missing <- popdata20 %>% summarise(across(c(pa,sz,age,sex,pop,time), ~sum(is.na(.))))  
print(popdata20\_missing)  
  
popdata21\_missing <- popdata21 %>% summarise(across(c(pa,sz,age,sex,pop,time), ~sum(is.na(.))))  
print(popdata21\_missing)  
  
popdata22\_missing <- popdata22 %>% summarise(across(c(pa,sz,age,sex,pop,time), ~sum(is.na(.))))  
print(popdata22\_missing)  
  
popdata23\_missing <- popdata23 %>% summarise(across(c(pa,sz,age,sex,pop,time), ~sum(is.na(.))))  
print(popdata23\_missing)  
  
popdata24\_missing <- popdata24 %>% summarise(across(c(pa,sz,age,sex,pop,time), ~sum(is.na(.))))  
print(popdata24\_missing)

### 2.5.4 Issue with POPDATA22

Using the below code, we are able to see clearly the rows that are affected and in the pop column, it appears as NA. The csv file (respopagesex2022.csv) was opened using excel and each row returned in the below output was then cross checked in excel. There were numbers with comma appeared in excel. This may be because read\_csv() function expects a numeric value (double) in one of the columns, but instead, it found a string (the values in the column are likely formatted with commas, such as “1,020”). This is why the parser is raising an issue earlier on.

na\_rows <- popdata22[is.na(popdata22$pop), ]  
print(na\_rows)

Using problems(), it shows details about any rows or columns that caused problems during the import. The results are a reaffirmation of the explanation above.

problems(popdata22)

Referencing from [Stackoverflow](https://stackoverflow.com/questions/1523126/how-to-read-data-when-some-numbers-contain-commas-as-thousand-separator), the first line of the code is necessary as it defines a new class called "num.with.commas". This class is intended to handle numeric values that are stored as strings with commas (e.g., "1,000"). Thereafter, the second line of the code defines a method to convert a character type to the custom "num.with.commas" class.

* The gsub(",", "", from) function removes commas from the string (e.g., "1,000" becomes "1000")
* The as.numeric() function then converts the cleaned string into a numeric value (e.g., "1000" becomes 1000)

This ensures that numbers with commas are properly converted to numeric values during data import.

setClass("num.with.commas")  
setAs("character", "num.with.commas",   
 function(from) as.numeric(gsub(",", "", from) ) )

The file is then re-imported again and column types were specified.

# Read the CSV file and specify column types  
popdata22 <- read\_csv("data/popdata/respopagesex2022.csv",   
 col\_types = cols(  
 PA = col\_character(),  
 SZ = col\_character(),  
 Age = col\_character(),  
 Sex = col\_character(),  
 Pop = col\_character(),  
 Time = col\_integer() # Adjust if necessary  
 ))

problems(popdata22)

names(popdata22) <- tolower(names(popdata22))  
popdata22\_missing <- popdata22 %>% summarise(across(c(pa,sz,age,sex,pop,time), ~sum(is.na(.))))  
print(popdata22\_missing)

### 2.5.5 Duplicate Check

The code provided checks for duplicate rows in each dataset by grouping the dataset by all columns using group\_by\_all(). It then filters out the rows that have duplicate combinations of values across all columns using filter(n() > 1). The n() function counts the number of occurrences for each combination of values, and filter(n() > 1) keeps only the rows that appear more than once (i.e., duplicates).

For each dataset, the nrow() function is used to check if there are any rows returned after filtering for duplicates. If there are duplicates (i.e., the number of rows is greater than zero), the dataset with the duplicate rows is returned. However, if no duplicates are found (i.e., nrow() equals zero), the code returns 0 to indicate that there are no duplicates in that dataset.

Thus, the code either returns the rows with duplicate values or 0 if no duplicates are present, providing an indication of whether duplicate entries exist in each dataset.

# Check for duplicates in 'aac'  
popdata20\_duplicate <- popdata20 %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
show(popdata20\_duplicate)  
  
  
# Check for duplicates in 'counselling'  
popdata21\_duplicate <- popdata21 %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
show(popdata21\_duplicate)  
  
# Check for duplicates in 'daycare'  
popdata22\_duplicate <- popdata22 %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
show(popdata22\_duplicate)  
  
# Check for duplicates in 'dementia'  
popdata23\_duplicate <- popdata23 %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
show(popdata23\_duplicate)  
  
# Check for duplicates in 'hospice'  
popdata24\_duplicate <- popdata24 %>%   
 group\_by\_all() %>%   
 filter(n() > 1) %>%   
 ungroup()  
show(popdata24\_duplicate)

## 2.6 Changing 90\_and\_over to 90

## 2.7 Calculating Population for Each Age Level

popdata24 <- popdata24 %>%  
 filter(age >= 65) %>% # Filter for age >= 65  
 group\_by(age) %>% # Group by age  
 summarise(total\_pop = sum(pop)) # Summarize the total population for each age group  
  
# View the resulting dataset  
print(popdata24)

## 2.8 Combining Datasets

# Combine popdata23 and popdata24 row-wise  
combined\_data <- bind\_rows(popdata23, popdata24)  
  
# View the combined dataset  
head(combined\_data)

# 3. Exploratory Data Analysis

tmap\_mode("view")  
tm\_shape(mpsz) +  
 tm\_polygons() +  
tm\_shape(CHAS) +  
 tm\_dots()

# 4. References

Tan, K. (2023). Take-home Exercise 1: Geospatial Analytics for Public Good. Retrieved from <https://isss624-kytjy.netlify.app/take-home_ex/take-home_ex1/the1#background>

Urban Redevelopment Authority. (2023). Master Plan 2019 Planning Area Boundary (No Sea) (2024) [Dataset]. data.gov.sg. Retrieved February 23, 2025 from https://data.gov.sg/datasets/d\_6c6d7361dd826d97b91bac914ca6b2ac/view