NHGIS - ACS 5-year and 1-year data cleaning pipeline

Shiv Gargé

2024-04-01

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.3     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.3     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
```

Defining Function

1 - Base filter of 5-year ACS (2014 - 2019)

Filter main columns of interest (user indicates additional variables of interest to keep)

```
filter_and_rename_2019 <- function(data, ...) {
  # Base columns to keep
  base_vars <- c("GEOID", "STUSAB", "YEAR")

  # Combine base columns with any additional variables specified
  vars_to_keep <- c(base_vars, ...)

  # Perform the operations
  data %>%
    select(all_of(vars_to_keep)) %>%
    rename(year = YEAR) %>%
    rename(fips = GEOID) %>%
    mutate(year = 2019) %>%
    filter(STUSAB %in% c("CA", "LA", "TX", "NJ", "NY"))
}
```

The 5-year data does not contain a *TL_GEO_ID* which is the main index for the master dataset. The following function takes *GEOID* and mutates it according to pre-defined naming list. Keep in mind, if there are different states you wish to filter, adjust the mutation accordingly. If *TL_GEO_ID* is four digits, there my be a need to drop an extra “0” (see CA for reference)

```
processFips <- function(df) {
  # Loop through each row of the dataframe
  df$fips <- sapply(1:nrow(df), function(i) {
    fips <- df$fips[i]
    stusab <- df$STUSAB[i]

    # Apply specific rules based on STUSAB value
    if (stusab == "LA") {
      fips <- sub("05000US", "", fips) # Remove "G"
    } else if (stusab == "NJ") {
      fips <- sub("05000US", "", fips) # Remove "G"
    } else if (stusab == "NY") {
      fips <- sub("05000US", "", fips) # Remove "G"
    } else if (stusab == "TX") {
      fips <- sub("05000US", "", fips) # Remove "G"
    } else if (stusab == "CA") {
      fips <- sub("05000US", "", fips) # Remove "G"
      fips <- sub("^0", "", fips) # Remove the last 0
    }

    return(fips)
  })

  return(df)
}
```

2 - Base filter of 1-year ACS 2021

Filter main columns of interest (user indicates additional variables of interest to keep)

```
filter_and_rename_2021 <- function(data, ...) {
  # Base columns to keep
  base_vars <- c("TL_GEO_ID", "STUSAB", "YEAR")

  # Combine base columns with any additional variables specified
  vars_to_keep <- c(base_vars, ...)

  # Perform the operations
  data %>%
    select(all_of(vars_to_keep)) %>%
    rename(year = YEAR) %>%
    rename(fips = TL_GEO_ID) %>%
    filter(STUSAB %in% c("CA", "LA", "TX", "NJ", "NY"))

}
```

For 2021 ACS, you don’t need to use *processesFips* since TL_GEO_ID already fits the reduced fips form naming convention.

3 - Data validation

Allows you to define and push variables to specific datatypes.

This is an example for type mapping:

type_mapping <- list(fips = “integer”, STUSAB = “character”, YEAR = “integer”)

Define this before running the function

```
type_check <- function(data, type_mapping) {

  data = data %>%
    select(-STUSAB)
  # Loop through each column in the data frame
  for (col in names(data)) {
    # Check if the column name exists in the type_mapping
    if (col %in% names(type_mapping)) {
      # Get the desired data type for the column
      desired_type <- type_mapping[[col]]

      # Convert the column to the desired data type
      if (desired_type == "character") {
        data[[col]] <- as.character(data[[col]])
      } else if (desired_type == "numeric") {
        data[[col]] <- as.numeric(data[[col]])
      } else if (desired_type == "integer") {
        data[[col]] <- as.integer(data[[col]])
      } else if (desired_type == "factor") {
        data[[col]] <- as.factor(data[[col]])
      } else if (desired_type == "logical") {
        data[[col]] <- as.logical(data[[col]])
      } else if (desired_type == "date") {
        data[[col]] <- as.Date(data[[col]])
      } else {
        warning(paste("Unsupported data type specified for column", col))
      }
    }
  }

  return(data)
}
```

This function renames columns to fall in the same naming convention. The standard used here is to rename variables from 2021 to the 2019 5-year naming convention. Like previously, you will have to define a map as follows:

snap_mapping <- c(“AN0LE001” = “ALXME001”, “AN0LE002” = “ALXME002”, “AN0LE003” = “ALXME003”)

```r
rename_columns <- function(data, column_mapping) {
  # Loop through each old column name and new column name pair
  for (i in seq_along(column_mapping)) {
    old_name <- names(column_mapping)[i]
    new_name <- column_mapping[[i]]

    # Check if the old column name exists in the data frame
    if (old_name %in% names(data)) {
      # Rename the column
      names(data)[names(data) == old_name] <- new_name
    } else {
      warning(paste("Column", old_name, "not found in the data frame"))
    }
  }

  return(data)
}
```

4 - The merges

This function appends the 2019 and 2021 datasets

```r
joint_set <- function(data, data_2) {

  join = bind_rows(data, data_2)

  return(join)

}
```

This function merges the filtered

```
master_merge_with_report <- function(master, data) {
  # Initial row counts
  initial_master_rows <- nrow(master)
  initial_rows <- nrow(data)

  # Merge the datasets
  merged <- inner_join(master, data, by = c("fips", "year"))

  # Calculate dropped rows
  dropped_rows_master <- initial_master_rows - nrow(merged)
  dropped_rows <- initial_rows - nrow(merged)

  # Print the report
  cat("Rows before merge:\n")
  cat(" - Master dataset rows:", initial_master_rows, "\n")
  cat(" - Snap_final dataset rows:", initial_rows, "\n")
  cat("Rows after merge:", nrow(merged), "\n")
  cat("Dropped rows:\n")
  cat(" - From Master dataset:", dropped_rows_master, "\n")
  cat(" - From Snap_final dataset:", dropped_rows, "\n")

  # Identifying conditions for dropped rows can be complex and might require
  # checking which specific rows didn't have a match. This simple report
  # provides a basic overview based on row counts.

  return(merged)
}
```

This function does a final validation to make sure the dataset is balanced for 2019 and 2021

```
processDataset <- function(data) {

  # Identify FIPS with both 2019 and 2021 observations
  valid_fips <- data %>%
    group_by(fips) %>%
    filter(all(c(2019, 2021) %in% year)) %>%
    pull(fips) %>%
    unique()

  # Filter the dataset
  filtered_data <- data %>%
    filter(fips %in% valid_fips)

  # Report the number of unique FIPS
  unique_fips_count <- length(unique(filtered_data$fips))
  cat("Number of unique FIPS with both 2019 and 2021 observations:", unique_fips_count,
"\n")

  # Return the filtered dataset
  return(filtered_data)
}
```

5 Test - Snap/Public Assistance recipients

```
#import 2019 and 2021 data
snap_2019 = read.csv("/Users/shivgarge/Desktop/Root/Research/Thesis/Data/Controls/5year_2019_time_inv/SNAP/nhgis0025_csv/nhgis0025_ds244_20195_county.csv")
snap_2021 = read.csv("/Users/shivgarge/Desktop/Root/Research/Thesis/Data/Controls/ACS_2021/nhgis0024_csv/SNAP/nhgis0027_csv/nhgis0027_ds253_2021_county.csv")
master = read.csv("/Users/shivgarge/Desktop/Root/Research/Thesis/master_w_controls.csv")
```

```
# CHANGE THESE ACCORDING TO REQUIREMENTS

# set the transformation maps

# mapping to standardize estimate names across years
snap_mapping <- c(
  "AN0LE001" = "ALXME001",
  "AN0LE002" = "ALXME002",
  "AN0LE003" = "ALXME003"
)

# mapping to maintain data type
type_mapping <- list(
  fips = "integer",
  YEAR = "integer"
)
```

```
# 2019 #

# Filtering the 2019 dataset
snap_2019 = filter_and_rename_2019(snap_2019, "ALXME001", "ALXME002", "ALXME003") #The variables added here are the estimates of interest

# Apply the mutation to get the same fips naming convention for the merge
snap_2019 = processFips(snap_2019)

# Make sure all the key variables are in the same data type
snap_2019 = type_check(snap_2019, type_mapping)



# 2021 #

# Filtering the 2021 dataset
snap_2021 = filter_and_rename_2021(snap_2021, "AN0LE001", "AN0LE002", "AN0LE003") #The variables added here are the estimates of interest from the 2021 set (Names are not the same across years)

# Make sure all variables are same data type
snap_2021 = type_check(snap_2021, type_mapping)

# Rename estimate variables to 2019 convention (based on snap_mapping)
snap_2021 = rename_columns(snap_2021, snap_mapping)
```

```
#Join (append) the two datasets
snap_join = joint_set(snap_2019, snap_2021)
```

```
# Merge Joint dataset to the master dataset
master = master_merge_with_report(master, snap_join)
```

```
## Rows before merge:
##  - Master dataset rows: 342
##  - Snap_final dataset rows: 631
## Rows after merge: 232
## Dropped rows:
##  - From Master dataset: 110
##  - From Snap_final dataset: 399
```

```
# Check dataset balance.
master = processDataset(master)
```

```
## Number of unique FIPS with both 2019 and 2021 observations: 94
```

Multi-variable table

```
multi_2019 = read.csv("/Users/shivgarge/Desktop/Root/Research/Thesis/Data/Controls/5year_2019_time_inv/Multi/nhgis0028_csv/nhgis0028_ds244_20195_county.csv")
multi_2021 = read.csv("/Users/shivgarge/Desktop/Root/Research/Thesis/Data/Controls/ACS_2021/Multi - unemp_child_worktravel/nhgis0029_csv/nhgis0029_ds253_2021_county.csv")
```

```
multi_mapping <- c(
  # Travel Time to Work (already provided)
  "ANRSE001" = "ALU3E001",
  "ANRSE002" = "ALU3E002",
  "ANRSE003" = "ALU3E003",
  "ANRSE004" = "ALU3E004",
  "ANRSE005" = "ALU3E005",
  "ANRSE006" = "ALU3E006",
  "ANRSE007" = "ALU3E007",
  "ANRSE008" = "ALU3E008",
  "ANRSE009" = "ALU3E009",
  "ANRSE010" = "ALU3E010",
  "ANRSE011" = "ALU3E011",
  "ANRSE012" = "ALU3E012",
  "ANRSE013" = "ALU3E013",

  # Own Children Under 18 Years by Family Type and Age
  "ANSOE001" = "ALU4E001",
  "ANSOE002" = "ALU4E002",
  "ANSOE003" = "ALU4E003",
  "ANSOE004" = "ALU4E004",
  "ANSOE005" = "ALU4E005",
  "ANSOE006" = "ALU4E006",
  "ANSOE007" = "ALU4E007",
  "ANSOE008" = "ALU4E008",
  "ANSOE009" = "ALU4E009",
  "ANSOE010" = "ALU4E010",
  "ANSOE011" = "ALU4E011",
  "ANSOE012" = "ALU4E012",
  "ANSOE013" = "ALU4E013",
  "ANSOE014" = "ALU4E014",
  "ANSOE015" = "ALU4E015",
  "ANSOE016" = "ALU4E016",
  "ANSOE017" = "ALU4E017",
  "ANSOE018" = "ALU4E018",
  "ANSOE019" = "ALU4E019",
  "ANSOE020" = "ALU4E020",

  # Employment Status for the Population 16 Years and Over
  "AN41E001" = "ALY3E001",
  "AN41E002" = "ALY3E002",
  "AN41E003" = "ALY3E003",
  "AN41E004" = "ALY3E004",
  "AN41E005" = "ALY3E005",
  "AN41E006" = "ALY3E006",
  "AN41E007" = "ALY3E007"
)

type_mapping <- list(
  fips = "integer",
  YEAR = "integer"
)
```

```
multi_2021 = rename_columns(multi_2021, multi_mapping)
```

```
multi_2019 = filter_and_rename_2019(multi_2019, "ALU3E001", "ALU3E002", "ALU3E003", "ALU3E004", "ALU3E005", "ALU3E006", "ALU3E007", "ALU3E008", "ALU3E009", "ALU3E010", "ALU3E011", "ALU3E012", "ALU3E013", "ALU4E001", "ALU4E002", "ALU4E003", "ALU4E004", "ALU4E005", "ALU4E006", "ALU4E007", "ALU4E008", "ALU4E009", "ALU4E010", "ALU4E011", "ALU4E012", "ALU4E013", "ALU4E014", "ALU4E015", "ALU4E016", "ALU4E017", "ALU4E018", "ALU4E019", "ALU4E020", "ALY3E001", "ALY3E002", "ALY3E003", "ALY3E004", "ALY3E005", "ALY3E006", "ALY3E007")

multi_2019 = processFips(multi_2019)
multi_2019 = type_check(multi_2019, type_mapping)

multi_2021 = filter_and_rename_2021(multi_2021, "ALU3E001", "ALU3E002", "ALU3E003", "ALU3E004", "ALU3E005", "ALU3E006", "ALU3E007", "ALU3E008", "ALU3E009", "ALU3E010", "ALU3E011", "ALU3E012", "ALU3E013", "ALU4E001", "ALU4E002", "ALU4E003", "ALU4E004", "ALU4E005", "ALU4E006", "ALU4E007", "ALU4E008", "ALU4E009", "ALU4E010", "ALU4E011", "ALU4E012", "ALU4E013", "ALU4E014", "ALU4E015", "ALU4E016", "ALU4E017", "ALU4E018", "ALU4E019", "ALU4E020", "ALY3E001", "ALY3E002", "ALY3E003", "ALY3E004", "ALY3E005", "ALY3E006", "ALY3E007")
multi_2021 = type_check(multi_2021, type_mapping)
```

```
multi_joint = joint_set(multi_2019, multi_2021)
```

```
master = master_merge_with_report(master, multi_joint)
```

```
## Rows before merge:
##  - Master dataset rows: 188
##  - Snap_final dataset rows: 631
## Rows after merge: 188
## Dropped rows:
##  - From Master dataset: 0
##  - From Snap_final dataset: 443
```