DOCUMENTATION - ASGS Temporal Correspondence code

Owen Forbes

11/05/2023

Temporal Correspondence for Datasets in ASGS 2016 Boundaries

This R code is designed to apply temporal correspondence to different datasets and bring them into the Australian Statistical Geography Standard (ASGS) 2016 boundaries. The purpose of this script is to allow users to harmonize and analyze data from different time periods and geographies consistently.

The code uses a function named `temporal_concordance_census_fn` that takes several input parameters to process datasets from different years and geographies. This function achieves this by following these main steps:

1. Load and process correspondence files.
2. Load and process datasets for different years.
3. Apply correspondence to the datasets.
4. Combine datasets from different years.
5. Perform formatting consistency checks on the final assembled dataset.
6. Save the output to a CSV file.

A parallel process is included for Census data and National data in `{r} state_stack_fn()` which does not do any temporal correspondence, but applies parallel steps of data formatting and processing, and joining data from multiple years together.

NB there is a custom correspondence code file for "Internet connection type" - since it only has years 2006-2016 (no 2021) - it is otherwise identical to the code for all other datasets

1. Load and process correspondence files

The correspondence files are loaded and processed for each year (2006, 2011, 2016, and 2021). Quality column ("Poor", "Acceptable", "Good" is joined on to the correspondence sheet from another sheet in each excel file.)

2 - 3. Load datasets + apply temporal correspondence

Datasets for different years (2006, 2011, 2016, and 2021) are loaded and processed with the following steps:

- Read the dataset.
- OUTER MERGE the dataset with the correspondence sheet, joined by the target geography
- Calculate the corresponded values by multiplying the input variable and the correspondence ratio (this needs to be the inverse for 2021 –> 2016, since the ratio describes moving from 2016 –> 2021)
- Group the dataset by the target geography column and other filter variables.

- Sum the corresponded values within each group (summing over situations where a single target area occurs multiple times in the target column, corresponding to multiple origin areas).

The below code chunk demonstrates (in this case for 2011 data) the key steps of outer merge, multiplying by correspondence ratio, grouping by relevant columns, and calculating sum to generate the new corresponded values. These are the key steps to be applied in other temporal correspondence applications.

```r
# Outer join original data frame with relevant
# correspondence sheet, joined on the relevant geography
# e.g. SA2_CODE_2011
merging_df_2011 <- merge(df_2011, df_corr_2011, by = {
    {
        paste0(GEO_TYPE, "_CODE_2011")
    }
}, all = T)

# multiply original values by correspondence ratio
merging_df_2011$new_vals_raw <- merging_df_2011[, VAR_NAME] *
    merging_df_2011$RATIO

# group by filter variables and new_geography, find new
# correspondence calculated vals as sum over multiple
# entries for target geom (grouped by filter vars and
# target geom)
merging_df_2011 <- merging_df_2011 %>%
    group_by(filters_combo, .data[[GEO_TO]]) %>%
    mutate(new_vals = round(sum(new_vals_raw)))
```

This data frame can then be ungrouped, the relevant columns can be retained, and only unique rows are kept with distinct() - to get rid of duplicate entries after the group_by –> sum step.

```r
# Rename uncertainty indicator

uncertainty_colname <- paste0(VAR_NAME, "_uncertainty_correspondence")
merging_df_2011[[uncertainty_colname]] <- merging_df_2011$QI_INDICATOR


# ungroup
merging_df_2011 <- merging_df_2011 %>%
    ungroup()

# Keep only necessary columns
if (length(FILTER_VARS) == 1) {
    out_df_2011 <- merging_df_2011 %>%
        dplyr::select(.data[[FILTER_VARS[1]]], .data[[GEO_TO]],
            new_vals, .data[[uncertainty_colname]], calendar_year)
} else if (length(FILTER_VARS) == 2) {
    out_df_2011 <- merging_df_2011 %>%
        dplyr::select(.data[[FILTER_VARS[1]]], .data[[FILTER_VARS[2]]],
            .data[[GEO_TO]], new_vals, .data[[uncertainty_colname]],
            calendar_year)
} else if (length(FILTER_VARS) == 3) {
    # If there is an additional filter column in the data,
    # use the following line instead
    out_df_2011 <- merging_df_2011 %>%
        dplyr::select(.data[[FILTER_VARS[1]]], .data[[FILTER_VARS[2]]],
            .data[[FILTER_VARS[3]]], .data[[GEO_TO]], new_vals,
            .data[[uncertainty_colname]], calendar_year)
} else if (length(FILTER_VARS) == 4) {
    # If there is an additional filter column in the data,
    # use the following line instead
    out_df_2011 <- merging_df_2011 %>%
        dplyr::select(.data[[FILTER_VARS[1]]], .data[[FILTER_VARS[2]]],
            .data[[FILTER_VARS[3]]], .data[[FILTER_VARS[4]]],
            .data[[GEO_TO]], new_vals, .data[[uncertainty_colname]],
            calendar_year)
} else {
    print("check number of filter variables in FILTER_VARS argument")
}


# rename values column after temporal correspondence
out_df_2011 <- rename(out_df_2011, {
    {
        VAR_NAME
    }
} := new_vals)

# Remove duplicate rows and NA for geom
out_df_2011 <- distinct(out_df_2011, .keep_all = T) %>%
    drop_na(.data[[GEO_TO]])
```

4. Combine datasets from different years

Datasets from all years are combined using the `rbind` function, and then sorted by the calendar year and other filter variables.

```
out_df_all_years <- rbind(out_df_2006, out_df_2011, out_df_2016,
    out_df_2021) %>%
    arrange(calendar_year, .data[[GEO_TO]], age_group, sex)
```

5. Perform formatting consistency checks on the final assembled dataset

The final assembled dataset is checked for formatting consistency. Some of the key checks include:

- Reordering columns.
- Formatting the geography column to match the standard `_CODE16` format.
- Converting all column names to snake case (except the geography column).
- Removing "total" rows and "years" from the age_group column.

Example user inputs for state_stack_fn() and temporal_concordance_census_fn():

```r
data_file_base <- "census_year12"  # Base file name for datasets e.g. census_year12
VAR_NAME <- "completed_year12"  # Name of the input variable column.
GEO_TO <- "SA4_CODE_2016"  # Target geography column
FILTER_VARS <- c("age_group", "sex")  # Name of original data set filter variable(s).
GEO_TYPE <- "SA4"  # Type of target geometry (used for looping over list of correspondence files)
GEO_TYPE_2006 <- "SD"  # 2006 Type of geometry (SLA, SSD, SD, LGA)


# Example usage
temporal_concordance_census_fn(origin_folder_path_base = origin_folder_path_base,
    destination_folder_path_base = destination_folder_path_base,
    data_file_base = data_file_base, VAR_NAME = VAR_NAME, GEO_TO = GEO_TO,
    FILTER_VARS = FILTER_VARS, GEO_TYPE = GEO_TYPE, GEO_TYPE_2006 = GEO_TYPE_2006)
```