SSPs Data Processing

2024-04-10

Table of contents

1 Introduction **1**

2 Installation **2**

3 Procedure Overview **2**

3.1 Reading and Transforming Data 2

3.1.1 OECD and IIASA Data 2

3.1.2 WIC Data 3

4 WIC data processing steps **3**

4.1 Joining and Finalizing Data 4

4.1.0.1 Labeling and Cleaning Data for Export 4

1 Introduction

This tutorial demonstrates the utilization of the `gtapssp` package in R for data processing. It covers various steps such as reading, transforming, and analyzing data, making it suitable for both beginners and advanced users.

The package provides optimized and user-friendly functions to **read** data from ZIP files, **interpolate** data using **spline** and **beers** methods. The *gtapssp* functions is accompanied by detailed manual, you can also access this manual by running `?gtapssp` in the R console or pressing `F1` on the function name in RStudio.

2 Installation

To use the *gtapssp* package, it's necessary to have *R* installed on your computer, which can be downloaded from here. Additionally, we recommend downloading *RStudio*, available at here, which provides a user-friendly interface to work with *R*.

You can install the development version of *gtapssp* from GitHub with:

```
# If the devtools package is not already installed, please run the disabled line below.
# install.packages("devtools")
devtools::install_github("tsimonato/gtapssp")
```

3 Procedure Overview

We will go through different stages of data manipulation which include reading data from ZIP files, transforming the data format, performing interpolations, and combining data from different sources.

3.1 Reading and Transforming Data

3.1.1 OECD and IIASA Data

Now, let's read the data from a ZIP file, reshape it, and interpolate it.

```
OECD <- gtapssp::read_csv_from_zip(                                    ①
  zip_dir = "Downloads",
  zip_pattern = "OECD",
  csv_pattern = "ssp_snapshot"
) |>
  tidyr::pivot_longer(                                                 ②
    cols = dplyr::matches(as.character(1500:3000)),
    names_to = "year",
    values_to = "value"
  ) |>
  gtapssp::interpolate_spline(                                         ③
    groups = c("Scenario", "Region"),
    year = "year",
    values = "value",
    method = "fmm"
  )
```

① `read_csv_from_zip` reads CSV files from a ZIP archive, specifying the directory of the files, name file pattern, and name CSV pattern.
② `pivot_longer` transforms `year` columns from wide format to long format.
③ `interpolate_spline` performs data interpolation for missing years using *fmm* spline method.

We process the IIASA data similarly.

```
IIASA <- gtapssp::read_csv_from_zip(
  zip_dir = "Downloads",
  zip_pattern = "IIASA",                                                    ①
  csv_pattern = "ssp_snapshot"
) |>
  tidyr::pivot_longer(
    cols = dplyr::matches(as.character(1500:3000)),
    names_to = "year",
    values_to = "value"
  ) |>
  gtapssp::interpolate_spline(
    groups = c("Scenario", "Region"),
    year = "year",
    values = "value"
  )
```

① The `IIASA` field specifies the pattern to match the ZIP file names.

3.1.2 WIC Data

Additional processing is done for the WIC dataset. This includes transformations and analysis with the `interpolate_beers` function.

4 WIC data processing steps

```
WIC <- gtapssp::read_csv_from_zip(
  zip_dir = "Downloads",
  zip_pattern = "WIC",
  csv_pattern = "ssp_snapshot"
)

WIC <- WIC |>
  tidyr::pivot_longer(
    cols = dplyr::matches(as.character(1500:3000)),
    names_to = "year",
    values_to = "value"
  )

WIC <- WIC |>
  tidyr::separate_wider_delim(
    cols = "Variable",
    names = c("var", "gender_code", "cohort", "education_level"),
    delim = "|",
    too_few = "align_start"
  ) |>
  dplyr::mutate(year = as.integer(year))
```

4.1 Joining and Finalizing Data

The final stage involves merging different datasets and transforming columns to prepare our dataset to be exported.

4.1.0.1 Labeling and Cleaning Data for Export

In this step, we prepare our WIC data for export by labeling and cleaning it to ensure it's in the correct format. This involves reading additional data sets and merging them with our main dataset.

①

```
isoList_dt <- read.csv("data/isoList.csv", na.strings = "")
educDict_dt <- read.csv("data/educDict.csv", na.strings = "")
cohortDict_dt <- read.csv("data/cohortDict.csv", na.strings = "")
genderDict_dt <- read.csv("data/genderDict.csv", na.strings = "")
```

②

```
WIC <-
  WIC |>
  dplyr::left_join(isoList_dt, by = dplyr::join_by(Region)) |>
  dplyr::left_join(educDict_dt, by = dplyr::join_by(education_level)) |>
  dplyr::left_join(cohortDict_dt, by = dplyr::join_by(cohort)) |>
  dplyr::left_join(genderDict_dt, by = dplyr::join_by(gender_code))
```

① Each `read.csv` call reads a different CSV file containing essential data. The `na.strings = ""` parameter treats empty strings as `NA` values.
② The WIC data is joined with the additional datasets using the `dplyr::left_join`.

Finally, `dplyr::transmute` is used to transform and rename columns, resulting in a dataset ready to be exported:

```
WIC <-
  WIC |>
  dplyr::transmute(
    SCE = Scenario,          # SSPs scenarios
    ISO = iso,               # ISO country codes for geographic identification
    EDU = educ,              # Education categories, derived from 'educ'
    GND = gender,            # Gender information
    AGE = age,               # Age groups or categories
    YRS = paste0("Y", year),# Year, formatted with a 'Y' prefix
    POP = value              # Population growth rates
  )
```