Package: munch (via r-universe)

October 14, 2024

Type Package

Title Functions for working with the historicized list of communes of Switzerland

Version 0.0.8.9007

Date 2022-05-23

Description Contains historicized municipality data for Switzerland from 1960 onwards, from the ``Historisiertes Gemeindeverzeichnis'' of the Swiss Federal Statistical Office.

License GPL-3

URL https://munch.cynkra.com/, https://github.com/cynkra/munch

BugReports https://github.com/cynkra/munch/issues

Depends R (>= 2.10)

Imports cli, dplyr (>= 0.8.3), dttr2, glue, lifecycle, logging (>= 0.8-104), lubridate, Matrix, purrr (>= 0.3.2), readr, reshape2 (>= 1.4.3), rlang, tibble (>= 1.4.2), tidyr

Suggests daff, gert, knitr, readxl, rmarkdown, rvest, stringr, testthat (>= 2.1.0), usethis (>= 1.4.0), xml2

Remotes edwindj/daff

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

VignetteBuilder knitr

Config/testthat/edition 3

Config/autostyle/scope line_breaks

Config/autostyle/strict true

Config/Needs/website cynkra/cynkratemplate

Repository https://cynkra.r-universe.dev

RemoteUrl https://github.com/cynkra/munch

RemoteRef HEAD

RemoteSha fecb096e5c6b1a004da606f6b72d34d33e8e02f7

Contents

`munch-package` *Functions for working with the historicized list of municipalities of Switzerland*

Description

The mutations of Swiss municipalities are tracked in a machine-readable mutation list. This allows guessing the reference time for any given list of municipality IDs, and to construct mappings between two lists of municipality IDs. The package provides functions to import and work with this mutation list.

Contains historicized municipality data for Switzerland from 1960 onwards, from the "Historisiertes Gemeindeverzeichnis" of the Swiss Federal Statistical Office.

Details

Package:	munch
Type:	Package
Version:	0.0-5
Date:	2014-09-22
License:	GPL-3

Author(s)

Kirill Müller

Maintainer: Kirill Müller <kirill@cynkra.com>

URL: `https://github.com/cynkra/munch`

Issue tracker: `https://github.com/cynkra/munch/issues`

Maintainer: Kirill Müller <`krlmlr+r@mailbox.org`>

Authors:

- Tobias Schieferdecker
- Thomas Knecht

Other contributors:

- Christoph Sax [contributor]
- Swiss Federal Statistical Office (SFSO), Swiss Statistics Web site [copyright holder]

References

`http://www.bfs.admin.ch/bfs/portal/de/index/infothek/nomenklaturen/blank/blank/gem_liste/02.html`

See Also

Useful links:

- `https://munch.cynkra.com/`
- `https://github.com/cynkra/munch`
- Report bugs at `https://github.com/cynkra/munch/issues`

`add_past` *Add the past to original data*

Description

The past is searched by the mHistId.x of the origial file

Usage

```
add_past(x, mutations)
```

Arguments

`x`	The original data as tibble.
`mutations`	Mutation file

Examples

```
mutations <- swcGetMutations()

t <-
  dplyr::filter(mutations, mId.y == 293)

t_1 <- dplyr::filter(t, mHistId.y == max(mHistId.y))

t_past <- add_past(t, mutations)
```

`download_mun_inventory` *download latest municipality inventory*

Description

download latest municipality inventory

Usage

```
download_mun_inventory()
```

`swc_get_mapping` *Compute a matching table between two lists of municipality IDs*

Description

For two lists of Swiss municipality IDs at any two points in time, this function creates a data frame with two columns where each row represents a match between municipality IDs. This can be used as an intermediate table for merging two data sets with municipality identifiers taken at different, possibly unknown, points in time.

Usage

```
swc_get_mapping(ids_from, ids_to)
```

Arguments

`ids_from`	A list of "source" municipality IDs, preferably a factor
`ids_to`	A list of "target" municipality IDs, preferably a factor

Details

It is advisable to use factors as list of municipality IDs. By that, comparisons and merges for municipality IDs are automatically checked for compatibility.

Note that the "from" list must be from an earlier time than the "to" list. Trying to compute the mapping the other way round results in an error. This is intentional: As municipalities are usually merged, it makes sense to use the most recent data set as target for the mapping. This can also be a file with suitable geometries to allow for visualization.

For two lists of municipalities, we construct a mapping from the first list to the second. First, the most probable mutation number in the "municipality mutations" data set is computed.

Value

A data frame with columns prefixed by `from.` and `to` that represents the computed match. The municipality IDs are stored in the columns `from.mId` and `to.mId`. The columns `from.MergeType` and `to.MergeType` contain `valid` if the municipality is contained in both the input and the mapping table, `missing` if the municipality is missing from the input, and `extra` if the municipality is in the input but not in the mapping table; most columns are `NA` for such rows. In addition, the column `MergeType` offers a summary of the "from" and "to" status: Rows with values other than `"valid"` or `"missing"` should be examined.

Examples

```
library(dplyr)
data(SwissPop)
data(SwissBirths)

# Show mismatch of municipality IDs:
ids_from <- with(SwissPop, MunicipalityID)
ids_to <- with(SwissBirths, MunicipalityID)
setdiff(ids_from, ids_to)
setdiff(ids_to, ids_from)

# Compute mapping and count non-matching municipality IDs:
mapping <- swc_get_mapping(ids_from = ids_from, ids_to = ids_to)
with(mapping, sum(mapping$mIdAsNumber.from != mapping$mIdAsNumber.to))

# Communes that are "missing" are mostly lakes and other special communes:
subset(mapping, MatchType == "missing")[, c("mIdAsNumber.from", "mShortName.from")]

# These should be looked at in some detail, and fixed manually:
subset(mapping, !(MatchType %in% c("valid", "missing")))

# Test for injectivity. The result shows that the mapping is almost injective,
# only one "from" commune is mapped to more than one other "to" commune.
# This situation requires further examination.
mapping.dupes <- subset(mapping, duplicated(mIdAsNumber.from))
(noninjective.mapping <- subset(
  mapping, mIdAsNumber.from %in% mapping.dupes$mIdAsNumber.from
))
```

```
# Simple treatment (just for this example): Remove duplicates, and use only
# valid matches:
cleaned.mapping <- subset(
  mapping,
  !duplicated(mIdAsNumber.from) & MatchType == "valid"
)

# Now merge the two datasets based on the mapping table:
SwissPop.1970 <- subset(SwissPop, Year == "1970")
SwissPopMapping.1970 <- merge(SwissPop.1970,
  cleaned.mapping[, c("mId.from", "mId.to")],
  by.x = "MunicipalityID", by.y = "mId.from"
)

# Datasets from the "from" table must be suitably aggregated.  For the given
# case of population totals we use the sum.
SwissPopMapping.1970.agg <- group_by(
  SwissPopMapping.1970,
  mId.to,
  HouseholdSize
) %>%
  summarize(Households = sum(Households))
with(SwissPopMapping.1970.agg, stopifnot(
  length(unique(mId.to)) * length(levels(HouseholdSize)) ==
    length(mId.to)
))

# The aggregated "from" dataset now can be merged with the "to" dataset:
SwissBirths.1970 <- subset(SwissBirths, Year == "1970")
SwissPopBirths.1970 <- merge(SwissPopMapping.1970.agg, SwissBirths.1970,
  by.x = "mId.to", by.y = "MunicipalityID"
)

# Some more communes are still missing from the 1970 statistics, although
# the matches are valid:
subset(mapping, mIdAsNumber.to %in% setdiff(
  SwissPopMapping.1970.agg$mId.to, SwissBirths.1970$MunicipalityID
))[
  ,
  c("mId.from", "mShortName.from", "MatchType")
]

# The "from" list must be from an earlier time than the "to" list.
try(swc_get_mapping(ids_from = ids_to, ids_to = ids_from))
```

swc_get_merger_mapping_table

Create mapping table for a certain time interval

Description

Produces a mapping table that can be joined to your data. Municipalities that were merged to another municipality during the given time period are mapped to that municipality. Filtering by canton is supported.

Usage

```
swc_get_merger_mapping_table(
  start_year,
  end_year,
  canton = NULL,
  type = "flat"
)
```

Arguments

`start_year`	First year of time interval (integer)
`end_year`	Last year of time interval (integer)
`canton`	Canton abbreviation as character (e.g. "GE", "ZH", "TI", etc.) to focus on. If left ‘NULL‘ (default) all cantons are considered.
`type`	Two options: - "flat" (default) returns the table with one row per year per mapping - "compact" returns a more compact table with one row per mapping, containing the time interval it is valid for

Value

Mapping table for the given time interval in the specified canton

Examples

```
swc_get_merger_mapping_table(2005, 2010)
swc_get_merger_mapping_table(2015, 2019, canton = "ZH", type = "compact")
```

`swc_get_municipality_state` *Existing municipalities in a specified year*

Description

This function produces a tibble containing the names and BFS-numbers (official municipality numbers) of all existing municipalities in a given year. Filtering by canton is supported.

Usage

```
swc_get_municipality_state(year, canton = NULL)
```

Arguments

`year`	Year of interest (integer).
`canton`	Canton abbreviation as character (e.g. "GE", "ZH", "TI", etc.) to focus on. If left ‘NULL‘ (default) all cantons are considered.

Value

Tibble containing municipality numbers (BFS-numbers) and names of existing municipalities in the given year.

Examples

```
swc_get_municipality_state(1987)
swc_get_municipality_state(2000, "ZH")
```

`swc_get_mun_history` *Get municipality mutation history*

Description

The history goes back as far as the mutation number becocmes 1000 (first time registration)

Usage

```
swc_get_mun_history(munId)
```

Arguments

`munId`	Municipality Id as integer

Examples

```
waedenswil_history <- swc_get_mun_history(293)
```

`swc_get_mun_merges` *Get municipality merges for a certain year and cantona*

Description

Get municipality merges for a certain year and cantona

Usage

```
swc_get_mun_merges(year = NULL, canton = NULL)
```

Arguments

`year`	The year where the merges took place
`canton`	The abbreviation as characger of a canton from which the merges should be returned. If NULL, all merges are returned

Examples

```
zuerich_merges <- swc_get_mun_merges(year = 2019, canton = "ZH")
```

`swc_get_mutations` *Create a list of municipality mutations*

Description

Municipality mutation lists are required to compute matching tables for municipalities between different points in time.

Usage

```
swc_get_mutations(mids = NULL, canton = NULL)
```

Arguments

`mids`	A list of municipality id's (BFS-numbers) of which the mutations should be retrieved.
`canton`	Canton abbreviation as character (e.g. "GE", "ZH", "TI", etc.) to focus on. If left ‘NULL‘ (default) all cantons are considered.

Details

The municipality dataset of the Swiss historicized communes data is a list of snapshots of municipality states. Each state is valid during a certain period of time, adjoining states are linked by admission and abolition numbers: The abolition number of a former state is the same as the admission number of a subsequent state. The states can be thought of as nodes in a graph, where the edges are *mutations* – transformations from one state to the next.

This function performs a self-merge on the municipality data: Each abolition number is matched with its corresponding admission number (if available). If no corresponding admission or abolition number is found, the record is included with NAs instead of matched values. Records without admission or abolition number are excluded. The result is a list of mutations, i.e., a list of edges in the graph of municipality state snapshots.

Value

A data frame that represents mutations.

Examples

```
head(swc_get_mutations(), 20)
head(subset(swc_get_mutations(), !is.na(mHistId.x)), 20)
```

`swc_read_data` *Download municipality mutation data*

Description

Download municipality mutation data

Usage

```
swc_read_data()
```

`SwissBirths` *Total births per municipality in Switzerland between 1969 and 2012*

Description

Each row contains the number of births for a municipality and a year. Not all municipalities are present for each year due to changes in the definition of the municipalities.

Usage

```
SwissBirths
```

Format

A data frame with 105058 observations of 4 variables:

```
 $ Year            : Factor w/ 44 levels "1969","1970",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ MunicipalityID  : Factor w/ 2485 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ MunicipalityName: Factor w/ 2485 levels "Aeugst am Albis",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Births          : num  13 174 22 36 20 13 17 4 32 48 ...
```

Author(s)

Swiss Federal Statistical Office

Source

STAT-TAB, section 01.2 "Bevölkerung/Bevölkerungsstand und -bewegung", item "Lebendgeburten nach institutionellen Gliederungen, Geschlecht und Staatsangehörigkeit des Kindes und nach Altersklasse der Mutter".

STAT-TAB URL: http://www.pxweb.bfs.admin.ch/Database/German_01%20-%20Bev%C3%B6lkerung/01.2%20-%20Bev%C3%B6lkerungsstand%20und%20-bewegung/01.2%20-%20Bev%C3%B6lkerungsstand%20und%20-bewegung.asp?lang=1&prod=01&secprod=2&openChild=true.

File link: http://www.pxweb.bfs.admin.ch/Database/German_01%20-%20Bev%C3%B6lkerung/01.2%20-%20Bev%C3%B6lkerungsstand%20und%20-bewegung/px-d-01-2D02.px.

SwissPop *Census population by household size per commune in Switzerland between 1970 and 2000*

Description

Each row contains the number of households for a municipality, a year and a household size.

Usage

```
SwissPop
```

Author(s)

Swiss Federal Statistical Office

Source

STAT-TAB, section 40.1 "Eidgenössische Volkszählung/1970–2000", item "Privathaushalte nach Haushaltsgrösse und Region".

STAT-TAB URL: http://www.pxweb.bfs.admin.ch/Database/German_40%20-%20Eidgen%C3%B6ssische%20Volksz%C3%A4hlung/40.1%20-%201970-2000/40.1%20-%201970-2000.asp?lang=1&prod=40&secprod=1&openChild=true.

File link: http://www.pxweb.bfs.admin.ch/Database/German_40%20-%20Eidgen%C3%B6ssische%20Volksz%C3%A4hlung/40.1%20-%201970-2000/px-d-40-1A01.px.

Index