Package 'xmap'

August 23, 2026

Type Package

Title Transforming Data Between Statistical Classifications

Version 0.2.0

Description Provides support for transformations of numeric aggregates between statistical classifications (e.g. occupation or industry categorisations) using the 'Crossmaps' framework.
Implements classes for representing transformations between a source and target classification as graph structures, and methods for validating and applying crossmaps to transform data collected under the source classification into data indexed using the target classification codes.
Documentation about the 'Crossmaps' framework is provided in the included vignettes and in Huang (2024, <doi:10.48550/arXiv.2406.14163>).

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Maintainer Cynthia A. Huang <cynthiahqy@gmail.com>

URL https://github.com/cynthiahqy/xmap, https://cynthiahqy.github.io/xmap/

BugReports https://github.com/cynthiahqy/xmap/issues

LazyData true

Depends R (>= 4.1)

Imports cli (>= 3.4.0), dplyr (>= 1.1.0), methods, pillar (>= 1.6.0), rlang (>= 1.0.0), tibble, tidyr, tidyselect, vctrs (>= 0.6.0)

Suggests forcats, ggalluvial, ggforce, ggplot2, ggrepel, glue, knitr, purrr, RColorBrewer, rmarkdown, scales, stringr, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Cynthia A. Huang [aut, cre] (ORCID:
<https://orcid.org/0000-0002-9218-987X>),
Laura Puzzello [aut, fnd]

Repository CRAN

Date/Publication 2026-08-23 15:50:02 UTC

Contents

`apply_xmap` *Apply Crossmap Transformation to Conformable Data*

Description

This function applies crossmap transformation to a dataset, transforming data based on specified mapping rules.

Usage

```
apply_xmap(.data, .xmap, values_from, keys_from = names(.xmap$.from), ...)

diagnose_apply_xmap(
  .data,
  .xmap,
  values_from,
  keys_from = names(.xmap$.from),
  ...
)
```

Arguments

`.data`	The dataset to transform.
`.xmap`	An `xmap_tbl` object.
`values_from`	A `tidyselect` expression of columns in `.data` with values to transform
`keys_from`	A `tidyselect` expression specifies the column in `.data` to match with `.xmap$from`
`...`	(reserved)

Details

diagnose_apply_xmap() checks whether .data is conformable with .xmap – the same two conditions apply_xmap() checks – and returns detail on any offending rows, to help resolve the specific issue rather than just knowing something's wrong. The returned xmap_diagnosis's details has one entry per condition (NULL where that check passed):

- not_covered: rows of .data whose keys_from key has no matching link in .xmap$.from
- missing_values: rows of .data with a missing value in one or more values_from columns

Value

A tibble with transformed data.

diagnose_apply_xmap() returns an xmap_diagnosis object: a list with valid (a scalar logical) and details (a named list of tibbles of offending rows, one per check, NULL where that check passed). Printing the result shows a readable pass/fail report; see new_xmap_diagnosis().

Functions

- diagnose_apply_xmap(): Returns an xmap_diagnosis object diagnosing why .data fails apply_xmap()'s conformability checks.

Examples

```
abc_xmap <- demo$abc_links |>
  as_xmap_tbl(from = "lower", to = "upper", weight_by = "share")
abc_data <- tibble::tibble(
  lower = unique(demo$abc_links$lower),
  count = runif(length(unique(demo$abc_links$lower)), min = 100, max = 500)
)
apply_xmap(
  .data = abc_data,
  .xmap = abc_xmap,
  values_from = count
)
```

---

as_xmap_tbl *Coerce links into a crossmap tibble*

Description

Converts an object of links into an xmap_tbl. Methods exist for data.frame and matrix — see their respective sections below for how from/to/weight_by are interpreted by each. Aborts with a message pointing at the offending condition if the links aren't a valid crossmap — the same conditions validate_as_xmap() checks, though currently implemented independently rather than by calling it (except for the matrix method, which does call validate_as_xmap() directly).

Usage

```
as_xmap_tbl(x, ...)

## S3 method for class 'data.frame'
as_xmap_tbl(x, from, to, weight_by, ..., tol = .Machine$double.eps^0.5)

## S3 method for class 'matrix'
as_xmap_tbl(
  x,
  ...,
  from = NULL,
  to = NULL,
  weight_by = NULL,
  tol = .Machine$double.eps^0.5
)

diagnose_as_xmap_tbl(
  x,
  from,
  to,
  weight_by,
  ...,
  tol = .Machine$double.eps^0.5
)
```

Arguments

x	An object with links to coerce. Methods exist for `data.frame` and `matrix`.
...	(reserved) Additional arguments passed to methods.
from	Identifies the 'from' nodes. For the `data.frame` method, the column in `x` that specifies them (tidyselect). For the `matrix` method, see the Matrix method section below.
to	Identifies the 'to' nodes. For the `data.frame` method, the column in `x` that specifies them (tidyselect). For the `matrix` method, see the Matrix method section below.
weight_by	Identifies the weight of the links. For the `data.frame` method, the column in `x` that specifies it (tidyselect). For the `matrix` method, see the Matrix method section below.
tol	Tolerance of comparison.

Details

`diagnose_as_xmap_tbl()` checks whether `x`'s links form a valid crossmap — the same conditions `validate_as_xmap()` checks, though currently implemented independently rather than by calling it — and returns detail on any offending rows, to help resolve the specific issue rather than just knowing something's wrong. The returned `xmap_diagnosis`'s `details` has one entry per condition ('NULL' where that check passed):

- bad_dups: rows sharing a .from-.to pair with another row
- miss_from, miss_to, miss_weight_by: rows with a missing .from, .to, or .weight_by value, respectively
- nonpositive_weights: rows whose .weight_by is zero or negative
- bad_froms: for each .from whose outgoing weights don't sum to (near enough) one, that .from and its actual weight sum

Value

Returns an xmap tibble object.

diagnose_as_xmap_tbl() returns an xmap_diagnosis object: a list with valid (a scalar logical) and details (a named list of tibbles of offending rows, one per check, NULL where that check passed). Printing the result shows a readable pass/fail report; see new_xmap_diagnosis().

Data frame method

as_xmap_tbl.data.frame() takes a data.frame-like object and converts it into an xmap_tbl based on specified columns for from, to, and weight_by.

Matrix method

as_xmap_tbl.matrix() takes an adjacency matrix (rows = .from, columns = .to, cells = .weight_by, per validate_as_xmap()'s .matrix method) and reshapes it into an xmap_tbl, dropping zero-weight cells (non-links). It checks matrix validity with validate_as_xmap() *before* reshaping — checking only after would let an all-zero row (a .from with no outgoing links) disappear silently, since dropping its only cells removes the row from the reshaped table before anything could flag it.

from/to/weight_by here are optional strings naming the resulting columns, since a matrix (unlike a data frame) has no columns to select from — identity comes from dimnames() instead. They default to names(dimnames(x)) when set, falling back to "rowname"/"colname"/"cell" (named after where each value is actually pulled from) when x has no named dimnames.

Examples

```
demo$abc_links |>
  as_xmap_tbl(from = lower, to = upper, weight_by = share)
abc_matrix <- demo$abc_links |>
  tidyr::pivot_wider(names_from = upper, values_from = share, values_fill = 0) |>
  tibble::column_to_rownames("lower") |>
  as.matrix()
as_xmap_tbl(abc_matrix)
```

compose_xmap *Compose Two Crossmaps Through a Shared Intermediate Classification*

Description

Given xmap1 (S -> M) and xmap2 (M -> T) sharing intermediate key set M, chains them into a single crossmap S -> T without materialising M-level values. Composed weights sum, over every shared m, the product of xmap1's weight onto m and xmap2's weight from m:

$$w(s,t) = \sum_{m \in M} w_1(s,m)\, w_2(m,t)$$

Usage

```
compose_xmap(xmap1, xmap2, ..., tol = .Machine$double.eps^0.5)
```

Arguments

xmap1	An xmap_tbl, S -> M.
xmap2	An xmap_tbl, M -> T. Every value in xmap1's .to must appear in xmap2's .from; the reverse isn't required – xmap2 may hold .from values xmap1 never uses.
...	(reserved)
tol	Tolerance of comparison.

Details

Re-checks that both inputs are actually valid crossmaps, not just correctly classed, and aborts otherwise.

Only takes two crossmaps at a time. Matrix multiplication is associative, so chain longer sequences with Reduce() instead of a dedicated variadic interface – see the example below. Grouped composition (e.g. one xmap1 per group, composed against a shared xmap2) is likewise left to the caller via dplyr::group_map().

Known limitation: composing two individually-tol-valid crossmaps can produce a composed crossmap that fails that same tol. Composed weights are sums of products of the input weights, which amplifies floating-point drift relative to either input alone – and compounds further across a Reduce()-chained sequence. Widening tol on the compose_xmap() call (or on the final Reduce() step) works around this in practice, but the underlying cause is .weight_by being plain double rather than a representation with an exact sum-to-1 guarantee (see #27).

Value

An xmap_tbl, S -> T.

Examples

```
abc_xmap <- demo$abc_links |>
  as_xmap_tbl(from = lower, to = upper, weight_by = share)
top_xmap <- tibble::tibble(
  upper = c("AA", "BB", "CC", "DD", "EE"),
  top = c("AAA", "AAA", "BBB", "BBB", "BBB"),
  weight = 1
) |>
  as_xmap_tbl(from = upper, to = top, weight_by = weight)
compose_xmap(abc_xmap, top_xmap)

# chaining more than two crossmaps: reduce pairwise composition over a
# list, e.g. lower -> upper -> top -> region
region_xmap <- tibble::tibble(
  top = c("AAA", "BBB"),
  region = c("north", "south"),
  weight = 1
) |>
  as_xmap_tbl(from = top, to = region, weight_by = weight)
Reduce(compose_xmap, list(abc_xmap, top_xmap, region_xmap))
```

---

demo *Demo objects for the* xmap *package*

Description

A collection of demo inputs for experimenting with functions in the xmap package. _pairs objects are tibbles with just source-target *pairs* (no weights) _links objects are tibbles with weighted source-target *links*.

Usage

```
demo
```

Format

demo**:**

A list with:

ctr_iso3c_pairs named vector with 249 elements. Names are ISO-3 country codes, values are ISO English country names. Retrieved from countrycode package: https://github.com/vincentarelbundock/countrycode

anzsco22_isco8_crosswalk tibble with 10 rows and 5 columns. Subset of crosswalk between ANZSCO22 and ISCO8 Occupation Code Standards published by The AUstralian Bureau of Statistics

anzsco22_stats tibble with 10 rows and 2 columns. Stylised Occupation Counts

simple_links tibble with 10 rows and 3 columns. specifying links xcode->alphacode by weight

abc_links tibble with 6 rows and 3 columns, specifying links lower->upper by share

aus_state_pairs named list with 1 element named "AUS" containing codes for the Australian states

aus_state_pop_df tibble containing 2022 population figures for Australia by state. Retrieved from: https://www.abs.gov.au/statistics/people/population/national-state-and-territory-populati jun-2022

Examples

```
demo$abc_links
```

`indstat` *UNIDO INDSTAT4 industrial statistics (masked), with country lookup*

Description

A subset of UNIDO's INDSTAT4 industrial-statistics database, with the reported output `value` masked to a constant, bundled together with a small country-code lookup table since the two are relationally paired (`indstat$masked_sample$country` joins onto `indstat$country_lookup$code`). Used in `vignette("extract-validate-existing")` (Case 2) to demonstrate grouped crossmap validation across `country`/`year`. Some `isic` industry codes are reported only in combination (`isiccomb`), with a single `value` covering several `isic` codes at once – the vignette splits these back out.

Usage

```
indstat
```

Format

`indstat`**:**

A list with:

masked_sample tibble with 17,365 rows and 11 columns:

- **ctable** table code; `14` (the only value in this subset) denotes the OUTPUT dimension of INDSTAT4
- **country** three-digit UN M49 country code (joins onto `country_lookup$code`); 133 distinct countries in the full INDSTAT4 Rev.3 dataset, 8 in this subset
- **year** observation year (1990-2013)
- **isic** 3- or 4-digit ISIC Rev.3 industry code; every 4-digit code nests inside the 3-digit code given by its first three digits
- **isiccomb** ISIC code as originally reported – either the same as `isic`, or a combined code (containing a letter, e.g. `"151A"`) covering several `isic` codes at once
- **value** reported output value in USD, masked to `1000` in this dataset (real values are not shipped); `NA` for `isic` codes with no directly reported value (i.e. covered only by another row's `isiccomb`)

utable output valuation methodology, consistent within a country/year but variable across countries: 11 = basic prices, 12 = factor prices, 13 = producers' prices, 14 = valuation not defined

source reporting-status flag (0-3) – exact code meanings are undocumented upstream, not just unconfirmed here (see reference below)

unit value unit; always "$" (USD) in INDSTAT4, no national-currency variants

country_iso3c ISO-3c country code, joined from country_lookup

country_name country name, joined from country_lookup

country_lookup tibble with 8 rows and 4 columns, a small lookup table of the 8 countries included in masked_sample, used to join ISO-3c codes and country names onto it:

code three-digit UN M49 country code, joins onto masked_sample$country

name country name

iso3c ISO-3c country code

income_group World Bank income group classification, 2006 vintage: "H" = high income, "UM" = upper-middle income, "LM" = lower-middle income, "L" = low income. All four groups are represented in this subset

isic_rev3_lookup tibble with 529 rows and 4 columns, the full ISIC Rev. 3 classification hierarchy, giving a label for every isic/isiccomb code number used in masked_sample:

code ISIC Rev. 3 code: a single letter for "section" (17 rows), otherwise 2/3/4 digits for "division"/"group"/"class"

level one of "section", "division", "group", "class", determined by the length of code

label English description of the code

parent_code code of the immediate numeric parent – a "class" code's first three digits (its "group"), or a "group" code's first two digits (its "division"). NA for "section" and "division", since sections cover ranges of divisions rather than sharing a numeric prefix with them

Details

The 8 reporters are five large economies (BRA, CHN, DEU, JPN, USA) plus three chosen for structurally distinct splitting behaviour once the split is re-aggregated to 3-digit ISIC: Colombia (splits are entirely reconvergent – imputed at 4 digits, exact at 3), Romania (the deepest sustained convergence in the source extract) and Yemen (~95% of isic values sit in splits that cross a 3-digit boundary).

Source

masked_sample: downloaded and parsed from the UNIDO INDSTAT4 website (Rev.3, 2019 vintage). See https://cynthiahqy.github.io/indstat-TPP/001-clean_INDSTAT.html for the cleaning pipeline this subset was derived from.

country_lookup: read by data-raw/indstat.R from data-raw/indstat-country-lookup.csv, exported alongside masked_sample by the same upstream script. income_group is the 2006 column of the World Bank's historical income classification workbook ("Country Analytical History" sheet of OGHIST.xlsx); current download at https://datahelpdesk.worldbank.org/knowledgebase/articles/906519-world-bank-country-and-lending-groups

isic_rev3_lookup: UN Statistics Division classifications registry, ISIC Rev. 3 English structure file, downloaded from https://unstats.un.org/unsd/classifications/Econ/Download/In%

20Text/ISIC_Rev_3_english_structure.Txt into data-raw/isic_rev3_structure.txt. See #34.

---

timor_occupn *Timor-Leste census occupation codes*

Description

A ~1% sample of individual-level records from the Timor-Leste Population and Housing Census 2015, prepared for the occupation-categorisation analysis in Mata Dalan Institute (2020) – see @source below. Used in vignette("extract-validate-existing") (Case 1) to demonstrate recovering an implicit occupation-recoding script as an explicit crossmap.

Usage

```
timor_occupn
```

Format

A tibble with 11,775 rows and 5 columns:

houseid household identifier (5,508 distinct households)

pno person number within the household

p3p3_sex sex of the individual: "1. Male" or "2. Female"

p3p4_age age in years (0-98)

occupn original occupation code (161 distinct non-missing values, ranging 110-9999). NA where no occupation code was recorded – these rows skew toward younger ages (median 12 vs. 39.5 for rows with a code) but the two groups overlap, so age alone doesn't fully explain which rows are missing

Source

Individual-level extract of the Timor-Leste Population and Housing Census 2015 microdata, prepared for the occupation-category analysis (Figures 1-2) in: Mata Dalan Institute (2020), "The Informal Sector in Timor-Leste in the Midst of COVID-19", August 2020, with support from Oxfam and Professor Brett Inder (Monash University). https://oi-files-cng-v2-prod.s3.eu-west-2.amazonaws.com/asia.oxfam.org/s3fs-public/file_attachments/MDI_COVID-19_Informal%20sector%20Research_Aug%2020_Final_English.pdf

timor_occupn is a ~1% sample of the full 1,179,654-row individual-level census extract, grouped by occupn and sampled with dplyr – so the set of occupation codes present is closer to fully represented than a plain random sample of individuals would give. See data-raw/occupation.R.

validate_apply_xmap *Cheaply check whether* .data *is conformable with an* xmap_tbl

Description

validate_apply_xmap() checks the same two conditions apply_xmap() requires before transforming .data – every keys_from key has a matching .xmap$.from link, and no values_from column has a missing value – and returns a single logical, without building the offending-rows/columns detail objects that diagnose_apply_xmap() does. It's the primitive to reach for when you only need a pass/fail answer – e.g. checking many .data/.xmap group pairs with dplyr::mutate() or purrr::map2_lgl() before applying any of them. Reach for diagnose_apply_xmap() once validate_apply_xmap() says something failed and you need to know why; apply_xmap() checks the same conditions at transform time and aborts with a message pointing at the offending condition.

Usage

```
validate_apply_xmap(
  .data,
  .xmap,
  values_from,
  keys_from = names(.xmap$.from),
  ...
)
```

Arguments

.data	The dataset to transform.
.xmap	An xmap_tbl object.
values_from	A tidyselect expression of columns in .data with values to transform
keys_from	A tidyselect expression specifies the column in .data to match with .xmap$from
...	(reserved)

Value

A single logical.

Examples

```
abc_xmap <- demo$abc_links |>
  as_xmap_tbl(from = "lower", to = "upper", weight_by = "share")
abc_data <- tibble::tibble(
  lower = unique(demo$abc_links$lower),
  count = runif(length(unique(demo$abc_links$lower)), min = 100, max = 500)
)
validate_apply_xmap(abc_data, abc_xmap, values_from = count)
```

validate_as_xmap *Cheaply check whether links form a valid crossmap*

Description

A valid crossmap's links must satisfy four conditions:

Usage

```
validate_as_xmap(x, ..., tol = .Machine$double.eps^0.5)

## S3 method for class 'data.frame'
validate_as_xmap(x, from, to, weight_by, ..., tol = .Machine$double.eps^0.5)

## S3 method for class 'matrix'
validate_as_xmap(x, ..., tol = .Machine$double.eps^0.5)
```

Arguments

x	An object with links to validate. Methods exist for data.frame and matrix.
...	Passed to methods.
tol	Tolerance of comparison.
from	The column in x that specifies the 'from' nodes.
to	The column in x that specifies the 'to' nodes.
weight_by	The column in x that specifies the weight of the links.

Details

- every link has a non-missing .from, .to, and .weight_by
- no two links share the same .from-.to pair (data-frame representations only — see the .matrix method for why this doesn't carry over to a matrix representation)
- every .weight_by is strictly positive — a weight of zero or less means the pair isn't a valid link at all, not a degenerate one
- for each .from, the .weight_by values of its outgoing links sum to (approximately) one — this is what guarantees the total mass before and after a transformation stays the same

validate_as_xmap() checks these conditions and returns a single logical, without building the offending-rows detail objects that diagnose_as_xmap_tbl() does. It's the primitive to reach for when you only need a pass/fail answer — e.g. inside dplyr::filter() or dplyr::group_map() over many groups. Reach for diagnose_as_xmap_tbl() once validate_as_xmap() says something failed and you need to know why; xmap_tbl()/as_xmap_tbl() check the same conditions at construction time and abort with a message pointing at the offending condition.

Value

A single logical.

Conditions for matrices

A matrix represents `.from`/`.to` identity through `dimnames()` (rows = `.from`, columns = `.to`) rather than per-link values, so the three conditions above translate differently:

- non-missing `.from`/`.to` becomes "`rownames()`/`colnames()` are non-`NULL`, with no repeated names"; non-missing `.weight_by` becomes "no `NA` cells"
- the no-duplicate-pairs check does not carry over as-is: a single cell can't encode a duplicate pair (each is already a unique row x column intersection). What can still happen — and is checked above as a `.from`/`.to`-identity condition, not a pairs condition — is repeated `dimnames()`: base R places no uniqueness constraint on them, e.g. `matrix(1:4, 2, 2, dimnames = list(c("a", "a"), c("x", "y")))` is a valid matrix with a repeated row name. A repeated row name would mean the same `.from` key has more than one, independently-checked set of outgoing weights; a repeated column name would mean weights for the same `.to` key are split across columns, invisible to `rowSums()`. Both are rejected by the row/column name uniqueness check
- weights summing to one becomes a row-sum check; a row summing to exactly zero (a `.from` with no outgoing links) fails here too, since 0 is never near enough to 1

Examples

```
demo$abc_links |>
  validate_as_xmap(from = lower, to = upper, weight_by = share)
abc_matrix <- demo$abc_links |>
  tidyr::pivot_wider(names_from = upper, values_from = share, values_fill = 0) |>
  tibble::column_to_rownames("lower") |>
  as.matrix()
validate_as_xmap(abc_matrix)
```

Index