Package 'cat2cat'

May 17, 2026

Title Handling an Inconsistently Coded Categorical Variable in a Longitudinal Dataset

Version 0.6.1

Maintainer Maciej Nasinski <nasinski.maciej@gmail.com>

Description Unifying an inconsistently coded categorical variable between two different time points in accordance with a mapping table.
The main rule is to replicate the observation if it could be assigned to a few categories. Then using frequencies or statistical methods to approximate the probabilities of being assigned to each of them.
This procedure was invented and implemented in the paper by 'Nasinski', 'Majchrowska', and 'Broniatowska' (2020) <doi:10.24425/cejeme.2020.134747>.

Depends R (>= 3.6)

License GPL (>= 2) | file LICENSE

URL https://github.com/Polkas/cat2cat, https://polkas.github.io/cat2cat/

BugReports https://github.com/Polkas/cat2cat/issues

Encoding UTF-8

Imports MASS

Suggests caret, dplyr, e1071, fixest, forcats, knitr, magrittr, randomForest, rmarkdown, spelling, testthat (>= 3.0.0), tidyr

LazyData true

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Language en-US

NeedsCompilation no

Author Maciej Nasinski [aut, cre] (ORCID: <https://orcid.org/0000-0001-5236-8040>)

Repository CRAN

Date/Publication 2026-05-17 21:40:02 UTC

Contents

`cat2cat` *Automatic mapping in a panel dataset*

Description

Unifies an inconsistently coded categorical variable in a panel dataset according to a mapping (transition) table. Apply iteratively for each pair of neighboring periods. Use `prune_c2c` to limit growing replications across steps.

Usage

```
cat2cat(
  data = list(old = NULL, new = NULL, time_var = NULL, cat_var = NULL, cat_var_old =
    NULL, cat_var_new = NULL, id_var = NULL, multiplier_var = NULL),
  mappings = list(trans = NULL, direction = NULL, freqs_df = NULL),
  ml = list(data = NULL, cat_var = NULL, method = NULL, features = NULL, args = NULL)
)
```

Arguments

`data`	'named list' with fields 'old', 'new', 'cat_var' (shorthand for 'cat_var_old = cat_var_new = cat_var'), 'time_var' and optional 'id_var', 'multiplier_var'.
`mappings`	'named list' with 3 fields 'trans', 'direction' and optional 'freqs_df'.
`ml`	'named list' (optional) with fields 'data', 'cat_var', 'method', 'features' and optional 'args', 'on_fail', 'fail_warn'.

Details

data args

"old" data.frame older time point in a panel

"new" data.frame more recent time point in a panel

"time_var" character(1) name of the time variable.

"cat_var" character(1) name of the categorical variable. Shorthand: sets both `cat_var_old` and `cat_var_new` to the same value.

"cat_var_old" Optional character(1) name of the categorical variable in the older time point. Default 'cat_var'.

"cat_var_new" Optional character(1) name of the categorical variable in the newer time point. Default 'cat_var'.

"id_var" Optional character(1) name of the unique identifier variable. When specified, subjects observed in both periods are mapped 1-to-1 directly (no replication, `wei_freq_c2c = 1, rep_c2c = 1`). Only subjects absent from the base period enter the replication path. This assumes a subject's true category does not change between adjacent waves. See `vignette("cat2cat_advanced")` for details on when this assumption is and is not satisfied.

"multiplier_var" Optional character(1) name of the multiplier variable - number of replication needed to reproduce the population

"freqs_df" Deprecated - use `mappings$freqs_df` instead.

mappings args

"trans" data.frame with 2 columns - mapping (transition) table - all categories for cat_var in old and new datasets have to be included. First column contains an old encoding and second a new one. The mapping (transition) table should include a candidate for each category in the period being harmonised.

"direction" character(1) direction - "backward" or "forward"

"freqs_df" Optional - data.frame with 2 columns where first one is category name (base period) and second counts. If It is not provided then is assessed automatically. Artificial counts for each variable level in the base period. It is optional nevertheless will be often needed, as gives more control. It will be used to assess the probabilities. The multiplier variable is omitted so sb has to apply it in this table.

Optional ml args

"data" data.frame - dataset with features and the 'cat_var'.

"cat_var" character(1) - the dependent variable name.

"method" character vector - one or a few from "knn", "rf", "lda" and "nb" methods - "knn" k-NearestNeighbors, "lda" Linear Discriminant Analysis, "rf" Random Forest, "nb" Naive Bayes

"features" character vector of features names where all have to be numeric, logical or factor. Factor features are automatically one-hot encoded using the union of levels observed in `ml$data` and the target period.

"args" optional - list parameters: knn: k ; rf: ntree

"on_fail" optional character(1) controlling failed ML weights: `"freq"` (default) uses `wei_freq_c2c`, `"naive"` uses `wei_naive_c2c`, `"na"` leaves failed weights as `NA`, and `"error"` stops when failed weights are detected.

"fail_warn" optional logical(1), default `TRUE`; warn when failed ML weights are replaced or retained as `NA`.

Without the `ml` argument, only frequency-based weights are computed. If an ML model fails, `ml$on_fail` controls whether frequency weights, naive weights, `NA`, or an error are used. The `knn` method is recommended for smaller datasets.

Value

'named list' with 2 fields old and new - 2 data.frames. There will be added additional columns like index_c2c, g_new_c2c, wei_freq_c2c, rep_c2c, wei_(ml method name)_c2c. Additional columns will be informative only for a one data.frame as we always make the changes to one direction. The new columns are added instead of the additional metadata as we are working with new datasets where observations could be replicated. For the transparency the probability and number of replications are part of each observation in the 'data.frame'.

Note

`trans` columns and `cat_var` must be of the same type. The mapping table must include a candidate for every category in the target period. Observations without a matched candidate are dropped; add a `c(NA, NA)` row to `trans` to retain them as `NA`.

See Also

- `cat2cat_ml_run` - validate ML performance before use
- `prune_c2c`, `cross_c2c` - post-processing
- `summary_c2c`, `dummy_c2c` - helpers
- `cat2cat_agg` - for pre-aggregated data
- `vignette("cat2cat")` - identification assumptions and worked examples

Examples

```
## Not run:
data("occup_small", package = "cat2cat")
data("occup", package = "cat2cat")
data("trans", package = "cat2cat")

occup_old <- occup_small[occup_small$year == 2008, ]
occup_new <- occup_small[occup_small$year == 2010, ]

# Adding the dummy level to the mapping table for levels without a candidate
# The best to fill them manually with proper candidates, if possible
# In this case it is only needed for forward mapping, to suppress warnings
trans2 <- rbind(
  trans,
  data.frame(
```

```
    old = "no_cat",
    new = setdiff(c(occup_new$code), trans$new)
  )
)

# default only simple frequencies
occup_simple <- cat2cat(
  data = list(
    old = occup_old, new = occup_new, cat_var = "code", time_var = "year"
  ),
  mappings = list(trans = trans2, direction = "forward")
)

mappings <- list(trans = trans, direction = "backward")

ml_setup <- list(
    data = occup_small[occup_small$year >= 2010, ],
    cat_var = "code",
    method = "knn",
    features = c("age", "sex", "edu", "exp", "parttime", "salary"),
    args = list(k = 10),
    # defaults for failed ML weights
    on_fail = "freq",
    fail_warn = TRUE
)

# ml model performance check
print(cat2cat_ml_run(mappings, ml_setup))

# additional probabilities from knn
occup_ml <- cat2cat(
  data = list(
    old = occup_old, new = occup_new, cat_var = "code", time_var = "year"
  ),
  mappings = mappings,
  ml = ml_setup
)

# strict mode: stop when any ML weights cannot be produced
ml_setup_strict <- ml_setup
ml_setup_strict$on_fail <- "error"

# diagnostic mode: keep failed ML weights as NA and silence warnings
ml_setup_diag <- ml_setup
ml_setup_diag$on_fail <- "na"
ml_setup_diag$fail_warn <- FALSE

## End(Not run)
```

---

cat2cat_agg *Manual mapping for an aggregated panel dataset*

Description

Applies user-defined mapping equations to redistribute aggregated counts (and other numeric columns) when categorical encodings change between time periods. Unlike `cat2cat`, which works on micro-data, this function operates on pre-aggregated data where each row represents a category with associated counts/totals.

Usage

```
cat2cat_agg(
 data = list(old = NULL, new = NULL, cat_var_old = NULL, cat_var_new = NULL, time_var =
    NULL, freq_var = NULL),
  ...
)
```

Arguments

`data`	Named list with 5-6 fields describing the datasets: `old data.frame` - older time period (one row per category). `new data.frame` - newer time period (one row per category). `cat_var character(1)` - (deprecated) category variable name if identical in both periods. Use `cat_var_old`/`cat_var_new` instead. `cat_var_old character(1)` - category variable name in old period. `cat_var_new character(1)` - category variable name in new period. `time_var character(1)` - time variable name. `freq_var character(1)` - frequency/count variable used to compute proportions when splitting one category into many.
`...`	Mapping equations specifying how categories relate across periods. See **Equation syntax** in Details.

Details

Equation syntax

Each equation has the form:

```
OLD_SIDE  DIRECTION  NEW_SIDE
```

where:

- **OLD_SIDE** - one or more old-period category names (use `c(A, B)` for multiple)
- **DIRECTION** - one of:
  - `%>%` or `>` - **forward**: replicates the NEW period, renaming/splitting new categories to match old encoding
  - `%<%` or `<` - **backward**: replicates the OLD period, renaming/splitting old categories to match new encoding
- **NEW_SIDE** - one or more new-period category names

How proportions are calculated

When one category maps to multiple:

- **Backward** (`%<%`): proportions come from `freq_var` in the **new** period (the target encoding)
- **Forward** (`%>%`): proportions come from `freq_var` in the **old** period (the target encoding)

Examples of valid equations

`Automotive %<% c(Automotive1, Automotive2)` Backward: the old "Automotive" row is split into two rows ("Automotive1", "Automotive2") with proportions from new-period counts.

`c(Kids1, Kids2) %>% c(Kids)` Forward: the new "Kids" row is split into two rows ("Kids1", "Kids2") with proportions from old-period counts.

`Home %>% c(Home, Supermarket)` Forward: the new "Home" and "Supermarket" rows are each renamed to "Home" (1-to-many from new perspective; after aggregation they merge).

Typical workflow

1. Call `cat2cat_agg()` with all mapping equations. 2. Bind `$old` and `$new` together. 3. Group by time and the (now unified) category variable. 4. Summarise numeric columns as `sum(value * prop_c2c)`.

Value

A named list with two elements:

`$old data.frame` - old period with `prop_c2c` column added. Categories may be replicated if split by backward equations.

`$new data.frame` - new period with `prop_c2c` column added. Categories may be replicated if split by forward equations.

The `prop_c2c` column contains the proportion (0-1) to apply when aggregating. For rows not affected by any equation, `prop_c2c = 1`. For split categories, proportions sum to 1 within the original category.

Note

- All equations must be valid - unknown category names cause an error.
- Each equation must have exactly one category on one side (the "one" in one-to-many). Many-to-many within a single equation is not allowed.
- Each category in `old` and `new` must appear exactly once (no duplicates allowed before mapping).

See Also

`vignette("cat2cat_advanced")` for a complete workflow example.

Examples

```
data("verticals", package = "cat2cat")
agg_old <- verticals[verticals$v_date == "2020-04-01", ]
agg_new <- verticals[verticals$v_date == "2020-05-01", ]

# cat2cat_agg - can map in both directions at once
# although usually we want to have the old or the new representation

agg <- cat2cat_agg(
  data = list(
    old = agg_old,
    new = agg_new,
    cat_var_old = "vertical",
    cat_var_new = "vertical",
    time_var = "v_date",
    freq_var = "counts"
  ),
  Automotive %<% c(Automotive1, Automotive2),
  c(Kids1, Kids2) %>% c(Kids),
  Home %>% c(Home, Supermarket)
)

## possible processing
library("dplyr")
agg %>%
  bind_rows() %>%
  group_by(v_date, vertical) %>%
  summarise(
    sales = sum(sales * prop_c2c),
    counts = sum(counts * prop_c2c),
    v_date = first(v_date)
  )
```

`cat2cat_ml_run` *Cross-validation diagnostics for cat2cat ML models*

Description

Evaluates whether machine-learning models used in the `ml` argument of `cat2cat` actually improve category assignment over simpler baselines. The function runs a per-group train/test split across every mapping group defined by the transition table.

Usage

```
cat2cat_ml_run(mappings, ml, ...)

## S3 method for class 'cat2cat_ml_run'
print(x, ...)
```

Arguments

mappings	'named list' with 3 fields 'trans', 'direction' and optional 'freqs_df'.
ml	'named list' (optional) with fields 'data', 'cat_var', 'method', 'features' and optional 'args', 'on_fail', 'fail_warn'.
...	other arguments (currently unused).
x	cat2cat_ml_run instance created with cat2cat_ml_run.

Details

For each mapping group (set of candidate categories linked by the transition table) the function:

1. Collects all observations from ml$data whose category belongs to the group.
2. Randomly splits them into training (1 - test_prop) and test (test_prop) sets.
3. Computes two baselines on the test set:
   - **naive**: accuracy of a random guess ($1/k$ where k is the number of candidate categories).
   - **freq**: accuracy of always predicting the most frequent category in the training set.
4. Trains each ML model specified in ml$method on the training set and records its classification accuracy on the test set.

Groups with fewer than 5 observations or only one candidate category are skipped (their accuracy is recorded as NA).

Baseline-Only Diagnostics: To inspect only baseline diagnostics (naive, freq, and their Brier/mean-probability variants), pass empty model and feature vectors: ml$method = character(0) and ml$features = character(0). In this mode, no ML models are trained, but baseline diagnostics are still computed for each mapping group.

Understanding the Metrics: Three complementary metrics evaluate model quality:

Accuracy measures how often the model's top prediction matches the true category. Use this when you only care about the single most likely assignment. Higher is better; theoretical maximum is 1.0.

Mean P(true class) measures the average probability the model assigns to the correct category. This evaluates the full probability distribution, not just the top prediction. For cat2cat, where weights ARE probabilities distributed across candidates, this metric directly measures weight quality. Higher is better; range is $[0, 1]$.

Brier score measures the squared error between predicted probabilities and the one-hot encoded true outcome, normalized to $[0, 1]$. Unlike log-loss, Brier score is bounded and does not explode when P(true) is near zero. Lower is better; 0 means perfect prediction. For k categories, the naive baseline (uniform 1/k) gives Brier = $(1 - 1/k)/2$.

Choosing a Method:

- If accuracy and mean P(true) are similar across methods, prefer simpler methods (freq, lda) over complex ones (rf, knn).
- If ML methods rarely beat the frequency baseline, use wei_freq_c2c — ML adds complexity without benefit.

- If Brier score for ML is similar to or worse than naive, the model is not well-calibrated. Consider `wei_freq_c2c` or a different ML method.
- Use `cross_c2c` to ensemble multiple methods if no single method dominates.

Because the split is random, results will vary between runs. For more stable estimates, call the function several times or use a larger `ml$data` set (e.g. pool multiple survey waves).

Value

An object of class `"cat2cat_ml_run"` (a named list). Each element corresponds to one mapping group and contains:

`naive` `numeric(1)` — random-guess baseline accuracy (1/k where k is number of categories). Theoretical lower bound.

`freq` `numeric(1)` — most-frequent-category baseline accuracy. A simple but often strong baseline.

`acc` Named `numeric` vector — test-set accuracy for each ML method. Higher is better; compare to `freq`.

`brier` Named `numeric` vector — Brier score for each ML method, computed from the full probability vector. Lower is better; range is [0, 1].

`mean_prob` Named `numeric` vector — average probability assigned to the true class. Higher is better. This directly measures the quality of probability weights used by cat2cat.

`naive_brier` `numeric(1)` — Brier score for uniform baseline (= (1 - 1/k) / 2). Serves as a calibration reference.

`naive_mean_prob` `numeric(1)` — mean P(true) for uniform baseline (= 1/k). Equals `naive` by definition.

`freq_brier` `numeric(1)` — Brier score using training set category frequencies as probabilities.

`freq_mean_prob` `numeric(1)` — mean P(true) using training set category frequencies.

The object also carries an `"ml_models"` attribute listing the methods evaluated. Use `print()` for a human-readable summary.

`print` returns `x` invisibly, after printing a summary with average accuracy per method, baseline comparisons, and failure rates.

See Also

`cat2cat` for the main mapping function, `cross_c2c` for ensembling weights from multiple methods.

Examples

```
library("cat2cat")
data("occup", package = "cat2cat")
data("trans", package = "cat2cat")

occup_2006 <- occup[occup$year == 2006, ]
occup_2008 <- occup[occup$year == 2008, ]
```

```
# Forward direction: old encoding -> new encoding
# Use data from OLD encoding periods (2006, 2008)
ml_setup <- list(
  data = rbind(occup_2006, occup_2008),
  cat_var = "code",
  method = c("knn", "rf", "lda", "nb"),
  features = c("age", "sex", "edu", "exp", "parttime", "salary"),
  args = list(k = 10, ntree = 50)
)
mappings <- list(trans = trans, direction = "forward")

set.seed(1234)
res <- cat2cat_ml_run(mappings, ml_setup, test_prop = 0.2)
print(res)

# Typical good results show:
# - ML accuracy > freq baseline (ML adds value)
# - ML Brier < naive (well-calibrated probabilities)
# - ML mean P(true) > freq (better probability weights)
#
# If Brier(ML) >= Brier(naive), the model is poorly calibrated
# and wei_freq_c2c may be safer. Use cross_c2c() to ensemble.

# High failure rate is normal - most groups have <5 observations

# Baseline-only diagnostics (no ML models):
ml_baseline <- list(
  data = rbind(occup_2006, occup_2008),
  cat_var = "code",
  method = character(0),
  features = character(0)
)
baseline_cv <- cat2cat_ml_run(mappings, ml_baseline)
print(baseline_cv)
```

`cat_apply_freq` *Applying frequencies to the object returned by the 'get_mappings' function*

Description

applying frequencies to the object returned by the 'get_mappings' function. We will get a symmetric object to the one returned by the 'get_mappings' function, nevertheless categories are replaced with frequencies. Frequencies for each category/key are sum to 1, so could be interpreted as probabilities.

Usage

```
cat_apply_freq(to_x, freqs)
```

Arguments

to_x	'list' object returned by 'get_mappings'.
freqs	'data.frame' object like the one returned by the 'get_freqs' function.

Value

a 'list' with a structure like 'to_x' object but with probabilities for each category.

Note

'freqs' arg first column (keys) and the to_x arg values have to be of the same type. The uniform distribution (outcomes are equally likely) is assumed for no match for all possible categories.

Examples

```
data("trans", package = "cat2cat")
data("occup", package = "cat2cat")

mappings <- get_mappings(trans)

mappings$to_old[1:4]
mappings$to_new[1:4]

mapp_p <- cat_apply_freq(
  mappings$to_old,
  get_freqs(
    occup$code[occup$year == "2008"],
    occup$multiplier[occup$year == "2008"]
  )
)
head(data.frame(I(mappings$to_old), I(mapp_p)))
mapp_p <- cat_apply_freq(
  mappings$to_new,
  get_freqs(
    occup$code[occup$year == "2010"],
    occup$multiplier[occup$year == "2010"]
  )
)
head(data.frame(I(mappings$to_new), I(mapp_p)))
```

---

cross_c2c *Make a combination of weights from different methods*

Description

adding the additional column which is a mix of weights columns by each row. Ensemble of a few methods usually produces more accurate solutions than a single model would.

Usage

```
cross_c2c(
  df,
  cols = colnames(df)[grepl("^wei_.*_c2c$", colnames(df))],
  weis = rep(1/length(cols), length(cols)),
  na.rm = TRUE
)
```

Arguments

df	‘data.frame‘ like result of the ‘cat2cat‘ function for a specific period.
cols	‘character‘ vector default all columns under the regex "wei_.*_c2c".
weis	‘numeric‘ vector weighs for columns in the ‘cols‘ argument. By default a vector of the same length as ‘cols‘ argument and with equally spaced probability (summing to 1).
na.rm	‘logical(1)‘ if ‘NA‘ values should be omitted, default TRUE.

Value

‘data.frame‘ with the additional column ‘wei_cross_c2c‘.

Examples

```
## Not run:
data("occup_small", package = "cat2cat")
data("occup", package = "cat2cat")
data("trans", package = "cat2cat")

occup_old <- occup_small[occup_small$year == 2008, ]
occup_new <- occup_small[occup_small$year == 2010, ]

# mix of methods - forward direction, try out backward too
occup_mix <- cat2cat(
  data = list(
    old = occup_old, new = occup_new, cat_var = "code", time_var = "year"
  ),
  mappings = list(trans = trans, direction = "backward"),
  ml = list(
    data = occup_new,
    cat_var = "code",
    method = c("knn"),
    features = c("age", "sex", "edu", "exp", "parttime", "salary"),
    args = list(k = 10, ntree = 20)
  )
)
# correlation between ml model
occup_mix_old <- occup_mix$old
cor(
  occup_mix_old[occup_mix_old$rep_c2c != 1, c("wei_knn_c2c", "wei_freq_c2c")]
)
```

```
# cross all methods and subset one highest probability category for each obs
occup_old_highest1_mix <- prune_c2c(cross_c2c(occup_mix$old),
  column = "wei_cross_c2c", method = "highest1"
)

## End(Not run)
```

---

dummy_c2c *Add default cat2cat columns to a 'data.frame'*

Description

a utils function to add default cat2cat columns to a 'data.frame'. It will be useful e.g. for a boarder periods which will not have additional 'cat2cat' columns.

Usage

```
dummy_c2c(df, cat_var, ml = NULL)
```

Arguments

df	'data.frame'.
cat_var	'character(1)' a categorical variable name.
ml	'character' vector of ml models applied, any of 'c("knn", "rf", "lda", "nb")'.

Value

the provided 'data.frame' with additional 'cat2cat' like columns.

Examples

```
## Not run:
dummy_c2c(airquality, "Month")

data("occup_small", package = "cat2cat")
occup_old <- occup_small[occup_small$year == 2008, ]
dummy_c2c(occup_old, "code")
dummy_c2c(occup_old, "code", "knn")

## End(Not run)
```

get_freqs *Getting frequencies from a vector with an optional multiplier*

Description

getting frequencies for a vector with an optional multiplier.

Usage

```
get_freqs(x, multiplier = NULL)
```

Arguments

x	‘vector‘ categorical variable to summarize.
multiplier	‘numeric‘ vector how many times to repeat certain value, additional weights.

Value

‘data.frame‘ with two columns ‘input‘ ‘Freq‘

Note

without multiplier variable it is a basic ‘table‘ function wrapped with the ‘as.data.frame‘ function. The ‘table‘ function is used with the ‘useNA = "ifany"‘ argument.

Examples

```
data("occup", package = "cat2cat")

head(get_freqs(occup$code[occup$year == "2008"]))
head(get_freqs(occup$code[occup$year == "2010"]))

head(
  get_freqs(
    occup$code[occup$year == "2008"],
    occup$multiplier[occup$year == "2008"]
  )
)
head(
  get_freqs(
    occup$code[occup$year == "2010"],
    occup$multiplier[occup$year == "2010"]
  )
)
```

get_mappings *Transforming a mapping (transition) table to two associative lists*

Description

to rearrange the one classification encoding into another, an associative list that maps keys to values is used. More precisely, an association list is used which is a linked list in which each list element consists of a key and value or values. An association list where unique categories codes are keys and matching categories from next or previous time point are values. A mapping (transition) table is used to build such associative lists.

Usage

```
get_mappings(x = data.frame())
```

Arguments

x	'data.frame' or 'matrix' - mapping (transition) table with 2 columns where first column is assumed to be the older encoding.

Value

a list with 2 named lists 'to_old' and 'to_new'.

Examples

```
data("trans", package = "cat2cat")

mappings <- get_mappings(trans)
mappings$to_old[1:4]
mappings$to_new[1:4]
```

occup *Occupational dataset*

Description

Occupational dataset

Usage

```
occup
```

Format

A data frame with around 70000 observations and 12 variables.

id integer id

age numeric age of a subject

sex numeric sex of a subject

edu integer edu level of education of a subject where lower means higher - 1 for at least master degree

exp numeric exp number of experience years for a subject

district integer district

parttime numeric contract type regards time where 1 mean full-time (work a whole week)

salary numeric salary per year

code character code - occupational code

multiplier numeric multiplier for the subject to reproduce a population - how many of such subjects in population

year integer year

code4 character code - occupational code - first 4 digits

Details

occup dataset is an example of unbalance panel dataset. This is a simulated data although there are applied a real world characteristics from national statistical office survey. The original survey is anonymous and take place every two years. It is presenting a characteristics from randomly selected company and then using k step procedure employees are chosen.

occupational dataset

---

occup_panel *Occupational panel dataset with BAEL-style quarterly rotation*

Description

Occupational panel dataset with BAEL-style quarterly rotation

Usage

```
occup_panel
```

Format

A data frame with approximately 3900 observations and 15 variables.

id integer original row identifier from occup

panel_id integer consistent person identifier across quarters

cohort character entry quarter (e.g., "2009Q1")

age numeric age of the worker

sex numeric sex of the worker

edu integer education level (1 = highest)

exp numeric years of experience

district integer district code

parttime numeric contract type (1 = full-time)

salary numeric annual salary

code character occupational code (4-digit pre-2010, 6-digit post-2010)

multiplier numeric survey weight

quarter character survey quarter (e.g., "2009Q1", "2010Q2")

code4 character first 4 digits of occupational code

year integer year extracted from quarter

quarter_num integer quarter number (1-4)

Details

A simulated rotational panel derived from occup, inspired by the Polish BAEL (Badanie Aktywnosci Ekonomicznej Ludnosci - Labour Force Survey). Unlike the main occup dataset (repeated cross-sections), this dataset includes workers observed for 4 consecutive quarters, enabling demonstration of the id_var feature in cat2cat().

Panel design:

- 8 quarters: 2009Q1 through 2010Q4
- Encoding change between 2009Q4 and 2010Q1
- Each cohort enters quarterly and stays for 4 consecutive quarters
- ~150 new subjects enter each quarter (1/4 rotation)
- ~450 subjects observed across the encoding change

For subjects across quarters:

- panel_id is consistent across quarters
- Occupation code is preserved (or mapped via trans at encoding change)
- Age and experience increase annually (every 4 quarters)
- Salary varies slightly between quarters (-1% to +2%)

See Also

occup for the full repeated cross-section dataset, trans for the transition table

Examples

```
data("occup_panel", package = "cat2cat")

# Check panel structure
table(occup_panel$quarter)
table(table(occup_panel$panel_id))  # appearances per subject (target: 4)

# Subjects observed across the encoding change (2009Q4 -> 2010Q1)
panel_2009Q4 <- occup_panel[occup_panel$quarter == "2009Q4", ]
panel_2010Q1 <- occup_panel[occup_panel$quarter == "2010Q1", ]
length(intersect(panel_2009Q4$panel_id, panel_2010Q1$panel_id))
```

---

occup_small *Occupational dataset - small one*

Description

Occupational dataset - small one

Usage

```
occup_small
```

Format

A data frame with around 8000 observations and 12 variables.

id integer id

age numeric age of a subject

sex numeric sex of a subject

edu integer edu level of education of a subject where lower means higher - 1 for at least master degree

exp numeric exp number of experience years for a subject

district integer district

parttime numeric contract type regards time where 1 mean full-time (work a whole week)

salary numeric salary per year

code character code - occupational code

multiplier numeric multiplier for the subject to reproduce a population - how many of such subjects in population

year integer year

code4 character code - occupational code - first 4 digits

Details

occup dataset is an example of unbalance panel dataset. This is a simulated data although there are applied a real world characteristics from national statistical office survey. The original survey is anonymous and take place every two years. It is presenting a characteristics from randomly selected company and then using k step procedure employees are chosen.

occupational dataset

Examples

```
set.seed(1234)
data("occup", package = "cat2cat")
occup_small <- occup[sort(sample(nrow(occup), 8000)), ]
```

`plot_c2c` *Summary plots for cat2cat results*

Description

This function help to understand properties of cat2cat results. It is recommended to run it before further processing, like next iterations.

Usage

```
plot_c2c(data, weis = "wei_freq_c2c", type = c("both", "hist", "bar"))
```

Arguments

`data`	‘data.frame‘ - one of the data.frames returned by the ‘cat2cat‘ function.
`weis`	‘character(1)‘ - name of a certain wei_*_c2c column, added by cat2cat function. Default ‘wei_freq_c2c‘.
`type`	‘character(1)‘ - one of 3 types ‘"both"‘, ‘"hist"‘, ‘"bar"‘.

Value

base plot graphics

Note

It will work only for data.frame produced by cat2cat function.

Examples

```
data("occup_small", package = "cat2cat")
occup_old <- occup_small[occup_small$year == 2008, ]
occup_new <- occup_small[occup_small$year == 2010, ]

occup_2 <- cat2cat(
  data = list(
    old = occup_old, new = occup_new, cat_var = "code", time_var = "year"
  ),
  mappings = list(trans = trans, direction = "backward")
)

plot_c2c(occup_2$old, type = c("both"))
plot_c2c(occup_2$old, type = c("hist"))
plot_c2c(occup_2$old, type = c("bar"))
```

---

prune_c2c *Pruning which could be useful after the mapping process*

Description

user could specify one of four methods to prune replications created in the cat2cat procedure.

Usage

```
prune_c2c(
  df,
  index = "index_c2c",
  column = "wei_freq_c2c",
  method = "nonzero",
  percent = 50
)
```

Arguments

df	‘data.frame‘ like result of the ‘cat2cat‘ function for a specific period.
index	‘character(1)‘ a column name with the ‘cat2cat‘ identifier. Should not be updated in most cases. Default ‘index_c2c‘.
column	‘character(1)‘ a column name with weights, default ‘wei_freq_c2c‘.
method	‘character(1)‘ one of four available methods: "nonzero" (default), "highest", "highest1" or "morethan".
percent	‘integer(1)‘ from 0 to 99

Details

method - specify a method to reduce number of replications

"nonzero" remove nonzero probabilities

"highest" leave only highest probabilities for each subject- accepting ties

"highest1" leave only highest probabilities for each subject - not accepting ties so always one is returned

"morethan" leave rows where a probability is higher than value specify by percent argument

Value

'data.frame' with the same structure and possibly reduced number of rows

Examples

```
## Not run:
data("occup_small", package = "cat2cat")
data("occup", package = "cat2cat")
data("trans", package = "cat2cat")

occup_old <- occup_small[occup_small$year == 2008, ]
occup_new <- occup_small[occup_small$year == 2010, ]

occup_ml <- cat2cat(
  data = list(
    old = occup_old, new = occup_new, cat_var = "code", time_var = "year"
  ),
  mappings = list(trans = trans, direction = "backward"),
  ml = list(
    data = occup_new,
    cat_var = "code",
    method = "knn",
    features = c("age", "sex", "edu", "exp", "parttime", "salary"),
    args = list(k = 10)
  )
)

prune_c2c(occup_ml$old, method = "nonzero")
prune_c2c(occup_ml$old, method = "highest")
prune_c2c(occup_ml$old, method = "highest1")
prune_c2c(occup_ml$old, method = "morethan", percent = 90)

prune_c2c(occup_ml$old, column = "wei_knn_c2c", method = "nonzero")

## End(Not run)
```

summary_c2c *Adjusted summary for regressions on replicated datasets*

Description

adjusting lm/glm object results according to original number of degree of freedom. The standard errors, test statistics and p values have to be adjusted because of replicated observations.

Usage

```
summary_c2c(x, df_old, df_new = x$df.residual)
```

Arguments

x	lm or glm object
df_old	integer number of d.f in original dataset. For bigger datasets 'nrow' should be sufficient.
df_new	integer number of d.f in dataset with replicated rows, Default: x$df.residual

Details

The replication step in `cat2cat` inflates the nominal sample size: the model sees `n_rep` rows but only `n_orig` are independent observations. Naive OLS therefore under-estimates standard errors.

The correction factor is `sqrt(df_new / df_old)`, where `df_new` is the residual d.f. from the replicated model and `df_old` the residual d.f. from the original dataset. Standard errors are multiplied by this factor, test statistics divided by it, and p-values recomputed from the appropriate reference distribution: `t(df_old)` for t-based summaries and standard normal for z-based summaries.

This is a pragmatic d.f. adjustment. It works well when per-subject weights sum to one and no extreme weights dominate.

Note: Ordinary R-squared can be interpreted in neutral replication cases where the response and covariates do not vary across replicated copies and per-observation weights sum to the original observation weight. Adjusted R-squared, AIC, and BIC depend on sample-size and degrees-of-freedom conventions, so the values from the replicated model should not be reported without recomputing them on the intended original-observation scale. If the harmonised category itself enters the model, all fit statistics are conditional on the chosen harmonisation weights.

Value

data.frame with additional columns over a regular summary output, like correct and statistics adjusted by it.

Examples

```
data("occup_small", package = "cat2cat")
data("trans", package = "cat2cat")

occup_old <- occup_small[occup_small$year == 2008, ]
occup_new <- occup_small[occup_small$year == 2010, ]

occup_2 <- cat2cat(
  data = list(
    old = occup_old,
    new = occup_new,
    cat_var = "code",
    time_var = "year"
  ),
  mappings = list(trans = trans, direction = "backward"),
  ml = list(
    data = occup_new,
    cat_var = "code",
    method = "knn",
    features = c("age", "sex", "edu", "exp", "parttime", "salary"),
    args = list(k = 10)
  )
)

# Regression
# we have to adjust size of std as we artificialy enlarge degrees of freedom
lms <- lm(
  formula = I(log(salary)) ~ age + sex + factor(edu) + parttime + exp,
  data = occup_2$old,
  weights = multiplier * wei_freq_c2c
)

summary_c2c(lms, df_old = nrow(occup_old))
```

`trans` *trans dataset containing mappings (transitions) between old (2008) and new (2010) occupational codes. This table could be used to map encodings in both directions.*

Description

trans dataset containing mappings (transitions) between old (2008) and new (2010) occupational codes. This table could be used to map encodings in both directions.

Usage

```
trans
```

Format

A data frame with 2693 observations and 2 variables.

old character an old encoding of a certain occupation

new character a new encoding of a certain occupation

Details

mapping (transition) table for occupations where first column contains old encodings and second one a new encoding

---

`verticals` *verticals dataset*

Description

verticals dataset

Usage

```
verticals
```

Format

A data frame with 21 observations and 4 variables.

vertical character an certain sales vertical

sales numeric a size of sale

counts integer counts size

v_date character Date

Details

random data - aggregate sales across e-commerce verticals

Examples

```
set.seed(1234)
agg_old <- data.frame(
  vertical = c(
    "Electronics", "Kids1", "Kids2", "Automotive", "Books",
    "Clothes", "Home", "Fashion", "Health", "Sport"
  ),
  sales = rnorm(10, 100, 10),
  counts = rgeom(10, 0.0001),
  v_date = rep("2020-04-01", 10), stringsAsFactors = FALSE
)
```

```
agg_new <- data.frame(
  vertical = c(
    "Electronics", "Supermarket", "Kids", "Automotive1",
    "Automotive2", "Books", "Clothes", "Home", "Fashion", "Health", "Sport"
  ),
  sales = rnorm(11, 100, 10),
  counts = rgeom(11, 0.0001),
  v_date = rep("2020-05-01", 11), stringsAsFactors = FALSE
)
verticals <- rbind(agg_old, agg_new)
```

`verticals2` *verticals2 dataset*

Description

verticals2 dataset

Usage

```
verticals2
```

Format

A data frame with 202 observations and 4 variables.

ean product ean

vertical character an certain sales vertical

sales numeric a size of sale

v_date character Date

Details

random data - single products sales across e-commerce verticals

Examples

```
set.seed(1234)
vert_old <- data.frame(
  ean = 90000001:90000020,
  vertical = sample(c(
    "Electronics", "Kids1", "Kids2", "Automotive", "Books",
    "Clothes", "Home", "Fashion", "Health", "Sport"
  ), 20, replace = TRUE),
  sales = rnorm(20, 100, 10),
  v_date = rep("2020-04-01", 20), stringsAsFactors = FALSE
)

vert_old2 <- data.frame(
```

```
  ean = 90000021:90000100,
  vertical = sample(c(
    "Electronics", "Kids1", "Kids2", "Automotive", "Books",
    "Clothes", "Home", "Fashion", "Health", "Sport"
  ), 80, replace = TRUE),
  sales = rnorm(80, 100, 10),
  v_date = rep("2020-04-01", 80), stringsAsFactors = FALSE
)

vert_new <- vert_old2
vert_new$sales <- rnorm(nrow(vert_new), 80, 10)
vert_new$v_date <- "2020-05-01"
vert_new$vertical[vert_new$vertical %in% c("Kids1", "Kids2")] <- "Kids"
vert_new$vertical[vert_new$vertical %in% c("Automotive")] <-
  sample(
    c("Automotive1", "Automotive2"),
    sum(vert_new$vertical %in% c("Automotive")),
    replace = TRUE
  )
vert_new$vertical[vert_new$vertical %in% c("Home")] <-
  sample(
    c("Home", "Supermarket"),
    sum(vert_new$vertical %in% c("Home")),
    replace = TRUE
  )

vert_new2 <- data.frame(
  ean = 90000101:90000120,
  vertical = sample(
    c(
      "Electronics", "Supermarket", "Kids", "Automotive1",
      "Automotive2", "Books", "Clothes", "Home",
      "Fashion", "Health", "Sport"
    ), 20,
    replace = TRUE
  ),
  sales = rnorm(20, 100, 10),
  v_date = rep("2020-05-01", 20), stringsAsFactors = FALSE
)

verticals2 <- rbind(
  rbind(vert_old, vert_old2),
  rbind(vert_new, vert_new2)
)
verticals2$vertical <- as.character(verticals2$vertical)
```

Index