Package 'mdiharmonize'

November 4, 2026

Type Package

Title Harmonize Confidential Firm-Level Microdata into a Common Format

Version 0.1.0

License GPL-3

URL https://github.com/Secretariat-CompNet/mdiharmonize

BugReports https://github.com/Secretariat-CompNet/mdiharmonize/issues

Description Harmonizes confidential firm-level microdata held at national statistical institutes into one common structure, so that the same analysis code runs unchanged in every country. Metadata tables describe each institute's raw files and map their variables to the common ones; the functions read the raw files, apply the mapping (revaluing, recoding, reclassifying, redefining and remapping), make industry, product and regional classifications consistent over time, switch between enterprise, firm and plant identifiers, and add capital stocks computed with the perpetual inventory method. Quality checks on the harmonized data report duplicate keys, type conversion losses, breaks in yearly statistics and jumps in missing values, and outputs pass statistical disclosure control before they leave the institute. Builds on the analysis tools in 'mditools'. Capital stocks via the perpetual inventory method follow OECD (2009) <doi:10.1787/9789264068476-en>; disclosure control applies the minimum-frequency and dominance rules with secondary suppression described in Hundepool et al. (2012) <doi:10.1002/9781118348239>.

Encoding UTF-8

Imports data.table, dplyr, grDevices, haven, igraph, mditools (>= 0.1.1), openxlsx, stats, stringr, tools, utils

Suggests askpass, bit64, ggplot2, gridExtra, readr, RODBC, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Daniele Aglio [aut],
Eric Bartelsman [aut],
Julián Díaz-Acosta [aut],
Mirja Hälbig [aut],
Marco Miorandi [aut],
Johanna Weiss [aut, cre],
José Alberto Ferreira [ctb],
Elena Sponga [ctb],
Reetuparna Vishwanath [ctb],
Chengzi Yi [ctb],
Alessandro Zona Mattioli [ctb],
European Union [fnd] (Technical Support Instrument (TSI), project ProdTool, TSI-2025-MULTIMSPROD-IBA, TSI Project Number 101261159)

Maintainer Johanna Weiss <johanna.weiss@iwhesl.onmicrosoft.com>

Repository CRAN

Date/Publication 2026-11-04 20:30:02 UTC

Contents

hier_col . . . 3
log_warnings . . . 4
mdi_build_pim_capital . . . 5
mdi_check_harmonized . . . 6
mdi_classify_conc . . . 8
mdi_coerce_md_types . . . 9
mdi_concord_nace . . . 10
mdi_config . . . 11
mdi_disclose . . . 13
mdi_disclosure_config . . . 15
mdi_export . . . 16
mdi_harmonize . . . 18
mdi_harmonize_codes_over_time . . . 20
mdi_key_id_switch . . . 21
mdi_load_deflators . . . 23
mdi_load_md_metadata . . . 24
mdi_load_nsi_metadata . . . 25
mdi_plot_breaks . . . 26
mdi_plot_na_share . . . 27
mdi_post_harmonization_checks . . . 27
mdi_preprocess . . . 29
mdi_run . . . 31

Index **33**

`hier_col` *The column holding a given level of a hierarchy file*

Description

Hierarchy files (`MD_nace_hier.csv`, `MD_nuts_hier.csv`, `MD_kic_hier.csv`) name their columns `h_0`, `h_1`, ... — the number is the level index, finest first. Nothing about what a level *means* is encoded in that name; `MD_hier_levels.csv` declares it. `hier_col()` answers "which column holds NUTS 3", so module code asks for the level it wants instead of hard-coding `h_1`.

Usage

```
hier_col(hier, hlevels, level_name = NULL, md_varname = NULL)
```

Arguments

`hier`	Character, the hierarchy's name as it appears in `MD_hier_levels.csv`'s `hier` column and as the loaded object is named — e.g. `"MD_nuts_hier"`.
`hlevels`	The declaration table, `MD_hier_levels.csv` as loaded: in a module, `MD_hier_levels`.
`level_name`	Character, the level's stable short name — `"nuts2"`, `"division2"`, `"total"`. Give exactly one of `level_name` or `md_varname`.
`md_varname`	Character, an MD variable whose codes sit at the wanted level — `"nuts3"`. Use this when the level is decided by which variable the data happens to carry.

Value

One column name, e.g. `"h_1"`.

See Also

`hier_levels()` for a file's columns in level order.

Examples

```
# in a run, MD_hier_levels is the loaded metadata file
levels <- data.table::data.table(
  hier = "MD_nuts_hier", level = c("h_0", "h_1", "h_2"),
  level_name = c("lau", "nuts3", "nuts2"), md_varname = c("lau", "nuts3", "nuts2"))
hier_col("MD_nuts_hier", level_name = "nuts2", hlevels = levels)   # "h_2"
hier_col("MD_nuts_hier", md_varname = "nuts3", hlevels = levels)   # "h_1"
```

`log_warnings` *Write a message to a log file*

Description

Writes (or appends) a message to a text file, for warnings, diagnostics and other information a run should keep, instead of printing to the console. `WarningsDescription.txt` in the output folder is the run's default log; the checkers and the post-harmonization report write their own files.

Usage

```
log_warnings(
  message,
  path,
  append = TRUE,
  file_name = "WarningsDescription.txt"
)
```

Arguments

`message`	character, the message; a vector is written one element per line.
`path`	character(1), the folder the file is in, ending in `/` (e.g. `cfg$dirs$output`).
`append`	logical(1), add to the file (`TRUE`) or start it anew.
`file_name`	character(1), the file's name.

Value

`NULL`, invisibly.

Examples

```
dir <- paste0(tempfile("log_"), "/")
dir.create(dir)
log_warnings("first line", dir, append = FALSE)
log_warnings("second line", dir)
readLines(paste0(dir, "WarningsDescription.txt"))
```

`mdi_build_pim_capital` *Add a Perpetual Inventory Method capital stock to SBS*

Description

Gives each SBS firm-year its NACE 2-digit industry from BR (by default the time-consistent `MDnace`, see `nace_var`), starts each firm's capital stock from its industry's capital per employee (EU KLEMS), accumulates it with `mdi_pim_capital()` from tangible investment (`ni_tan`) and the industry depreciation rate, and returns SBS with a `K_pim` column added.

Usage

```
mdi_build_pim_capital(
  sbs,
  br,
  capital_stocks,
  nace_hier,
  id_sbs,
  id_br = id_sbs,
  conctab = NULL,
  br_varnames = NULL,
  emp_var = "persons",
  nace_var = "MDnace"
)
```

Arguments

`sbs`	A `data.table`, the harmonized SBS MD dataset: `id_sbs`, `year`, `ni_tan` and `emp_var`.
`br`	A `data.table`, the harmonized BR MD dataset: `id_br`, `year` and the industry column `nace_var`.
`capital_stocks`	A `data.table` in the long format of `rocket/auxdata/capital_stocks_<country>.csv`: `year`, `nace` (EU KLEMS codes, e.g. `"A01"`), `cc`, `variable` and `value`. Rows with `variable` `"K_GFCF_per_emp"` (capital per employee) and `"d_GFCF"` (depreciation rate) are used.
`nace_hier`	A `data.table`, `MD_nace_hier`: `h_0` (4-digit), `h_1` (3-digit) and `h_2` (2-digit) NACE codes.
`id_sbs`, `id_br`	Character, the firm identifier column of SBS and of BR (e.g. `"firmid"`, `"plantid"`). When they differ, BR is brought to SBS's level with `mdi_key_id_switch()` first.
`conctab`	A `data.table`, the concordance between `id_br` and `id_sbs` (needed only when they differ).
`br_varnames`	A `data.table`, `MD_varnames` filtered to BR (needed only when `id_br` and `id_sbs` differ).

`emp_var` Character, SBS's employment column used to scale the initial capital. Default `"persons"`.

`nace_var` Character, BR's industry column. Default `"MDnace"`, the time-consistent code from `mdi_concord_nace()`: a firm whose raw `nace` changes for classification reasons (e.g. the 2008 NACE revision) keeps one industry, so its depreciation rate doesn't jump. `"nace"` uses the code as recorded each year.

Value

`sbs` with a `K_pim` column added, every row kept: a firm-year with no BR match (so no industry) gets `K_pim = NA`. When BR's NACE codes are all 3-digit, industries are matched at `h_1` instead of `h_0`. A `K_pim` already in `sbs` (from an earlier run) is replaced. `sbs` itself is not modified.

Examples

```
yrs <- 2010:2012
sbs <- data.table::data.table(firmid = rep(c("f1", "f2"), each = 3), year = rep(yrs, 2),
                              persons = 10L, ni_tan = 1:6)
br <- data.table::data.table(firmid = rep(c("f1", "f2"), each = 3), year = rep(yrs, 2),
                             MDnace = rep(c("0111", "1011"), each = 3))
capital_stocks <- data.table::data.table(
  year = rep(yrs, 4), nace = rep(c("A01", "A01", "C10", "C10"), each = 3), cc = "XX",
  variable = rep(c("K_GFCF_per_emp", "d_GFCF"), each = 3, times = 2),
  value = rep(c(100, 0.1, 200, 0.2), each = 3)
)
nace_hier <- data.table::data.table(h_0 = c("0111", "1011"), h_1 = c("011", "101"),
                                    h_2 = c("01", "10"))
mdi_build_pim_capital(sbs, br, capital_stocks, nace_hier, id_sbs = "firmid")
```

---

mdi_check_harmonized *Check a harmonized MD dataset against its metadata*

Description

The post-harmonization quality checks for one MD dataset: whether its key variables identify rows uniquely, whether every row has a `year`, whether each variable has the class (and, for dates, the format) `md_catalogue` declares, which dates the data actually covers, and whether yearly statistics (p25, p50, p75, mean, sd) of its numeric variables change by more than `threshold` from one year to the next, a possible break in the series, and whether the share of `NA` values of any variable changes by more than `na_threshold` from one year to the next or is 100% in a year (the statistics only see the non-`NA` values, so a loss of coverage doesn't show in them).

It checks the data against the metadata, assuming the metadata itself is well-formed; that is checked separately, on the metadata files alone (MDI's metadata checker).

Usage

```
mdi_check_harmonized(
  dt,
  dataset,
  md_catalogue,
  country,
  break_vars = NULL,
  threshold = 0.25,
  na_threshold = 0.1
)
```

Arguments

dt	data.table, the harmonized dataset.
dataset	character(1), its MD dataset name, e.g. "BR".
md_catalogue	data.table, the MD catalogue: MD_dataset, MD_varname, is_key, class, domain (the date format, for date variables) and a coverage column per country.
country	character(1), the country code; md_catalogue[[country]] is the expected date range reported next to the actual one.
break_vars	character or NULL: the numeric variables to check for breaks. NULL checks every variable md_catalogue declares numeric for dataset; variables not declared numeric are skipped either way.
threshold	numeric(1), the relative year-on-year change above which a statistic is flagged (0.25 = 25%; on real firm-level data, 10% flags about a third of all year-on-year changes). From a previous value of 0 there is no relative change: p25, p50 and p75 aren't flagged then (they jump between 0 and small values for variables with many zeros), a mean or sd that leaves 0 is, with from_zero = TRUE and growth NA.
na_threshold	numeric(1), the year-on-year change in a variable's share of NA values above which it is flagged (0.1 = 10 percentage points).

Value

A list:

duplicates one row: key variables, whether and how many rows are duplicated.

years one row: whether there is a year column, and how many rows have no year.

classes expected and actual class of each non-date variable.

date_formats expected and actual class and format of each date variable.

date_ranges detected format, first and last date, and expected range of each date variable.

breaks the flagged statistics: dataset, variable, stat, year, growth, from_zero.

na_shares the flagged years: dataset, variable, year, na_share, change (from the year before), all_na.

series named list, per numeric variable flagged by either check, of its yearly statistics (input to mdi_plot_breaks()).

na_series named list, per variable flagged by either check, of its yearly share of NA values: year, na_share (input to mdi_plot_na_share()).

A table is NULL when no variable applies (e.g. breaks without a year column or without any break). Key variables are left out of the NA check.

Examples

```
dt <- data.table::data.table(firmid = c("a", "a", "b"), year = c(2020, 2020, 2021),
                             emp = c(10, 12, 20))
catalogue <- data.table::data.table(
  MD_dataset = "SBS", MD_varname = c("firmid", "year", "emp"),
  is_key = c(1L, 1L, 0L), class = c("character", "numeric", "numeric"),
  domain = "", FI = "2020 - 2021")
res <- mdi_check_harmonized(dt, "SBS", catalogue, "FI")
res$duplicates
res$breaks
```

---

mdi_classify_conc *Classify the concordance rows (edges)*

Description

This tool classifies concordance edges into type 1:1, 1:m, m:1, or m:m. Optionally, the tool also creates numbered 'components' of the m:m edges. The input is a concordance data.table, with left and right columns and the function returns a data.table, possibly merged with the initial table, with conctype (and optional compnum).

Usage

```
mdi_classify_conc(
  DTconc,
  left = "left",
  right = "right",
  mm_groups = TRUE,
  mergeconc = FALSE
)
```

Arguments

DTconc	a concordance table (data.table) with two columns: given by parameters left and right.
left	character ('left') is the name of the left side of the concordance
right	character ('right') is the name of the right side of the concordance
mm_groups	logical (TRUE) if m:m grouping (block diagonalization) is required.
mergeconc	logical (FALSE) returns the conc merged with the original input

Value

Returns a data.table with columns `left`, `right`, `conctype`, `outdegree`, `indegree` (and `compnum`) where conctype gives type of match ("1:1","m:1","1:m","m:m"); outdegree gives number of right nodes to which a left node goes; indegree gives number of left nodes that go to a right node; if mm_groups=TRUE, then `compnum` gives component number. If mergeconc=TRUE then mdi_classify_conc() returns the result merged with the original DTconc input.

Examples

```
concx <- data.table::data.table(
  leftx = as.character(c(1, 2, 2, 3, 4, 5, 6, 6, 7, 8, 8)),
  rightx = c("A", "B", "C", "D", "D", "F", "E", "F", "H", "G", "H")
)
classified <- mdi_classify_conc(concx, left = "leftx", right = "rightx",
                                mm_groups = TRUE, mergeconc = TRUE)
```

---

mdi_coerce_md_types *Convert a harmonized MD dataset's variables to their declared classes*

Description

Converts every variable of `dt` to the class `md_vars` declares for it, so module code sees the same types in every country, whatever types came out of import and harmonization. Conversions that fail on a value don't error: `as.numeric()`, `as.Date()` etc. turn it into `NA`. The returned log counts those, per variable, and also counts numeric values other than 0/1 converted to logical, which `as.logical()` turns into `TRUE` without an `NA` to show for it.

Classes: `character`, `integer`, `numeric`, `logical` (from text: `true`/`t`/`1` and `false`/`f`/`0`, anything else `NA`), `date` (with the format in `domain`, if any) and `factor`. `factor` is only applied where `domain` is `MD_codebook`, the variables whose values recode/reclass already checked against the codebook.

Usage

```
mdi_coerce_md_types(dt, dataset, md_vars, verbose = FALSE)
```

Arguments

`dt`	data.table, a harmonized MD dataset. Modified by reference.
`dataset`	character(1), its MD dataset name, for the log.
`md_vars`	data.table, the variables to convert: `MD_varname`, `class`, `domain`. Variables not in `dt`, or without a class, are skipped.
`verbose`	logical(1), print each conversion.

Value

A list: data, dt converted; log, one row per converted variable: MD_dataset, MD_varname, desired_class, na_before, na_after, na_introduced, logical_out_of_domain.

Examples

```
dt <- data.table::data.table(emp = c("1", "x"), exporter = c(0, 2))
vars <- data.table::data.table(MD_varname = c("emp", "exporter"),
                               class = c("numeric", "logical"), domain = NA)
res <- suppressWarnings(mdi_coerce_md_types(dt, "SBS", vars))
res$log
```

---

mdi_concord_nace *NACE classification harmonization*

Description

This tool harmonizes NSI NACE classification over time. It simplify the harmonization process using the mode NACE code in the post break year as MD NACE code, and harmonizing accordingly previous year codes. It defines the break year as the year with the most changes in NACE codes.

Usage

```
mdi_concord_nace(
  DT,
  id_col,
  weight_emp,
  cumsh_del,
  sh_del,
  weight_col = "persons_br",
  seed = 1L
)
```

Arguments

DT	the MD dataset (e.g. BR) to harmonize, already loaded (was loaded internally from dirTMPSAVE given a dataset_name).
id_col	character, the name of DT's row-identifier column for this country (e.g. entid, plantid) – was looked up internally from MD_idInfo/CountryCode given a dataset_name.
weight_emp	logical to decide whether or not to weight code matches by employment (instead of number of firms).
cumsh_del	number indicating the cumulative residual share of firms deleted for the construction of year by year concordances.
sh_del	number indicating the share of firms deleted for the construction of year by year concordances.

weight_col	character, the name of DT's employment-weight column, used only when weight_emp is TRUE.
seed	integer, the random seed for the firms whose code is assigned by a random draw (weighted by how other firms' codes moved), so a rerun gives the same MDnace. The caller's random state is restored afterwards. NULL uses the current random state instead.

Value

the original MD dataset with the old NSI NACE code and the new MD NACE.

Examples

```
library(data.table)
# four firms whose NACE codes change with the 2008 revision
BR <- data.table(
  entid = rep(c("A", "B", "C", "D"), each = 4),
  year  = rep(2006:2009, times = 4),
  nace  = c("1511", "1511", "1011", "1011",
            "1511", "1511", "1011", "1011",
            "1513", "1513", "1013", "1013",
            "5211", "5211", "4711", "4711")
)
br_harmonized <- mdi_concord_nace(BR, id_col = "entid", weight_emp = FALSE,
                                  cumsh_del = 0.95, sh_del = 0.02)
br_harmonized[entid == "D"]  # pre-2008 "5211" mapped to "4711"
```

---

mdi_config *Build the MDI pipeline configuration object*

Description

Constructs the cfg object the pipeline functions take: the run's directories, flags and versions. Derives the rocket/payload/metadata directory paths from mdi_dir/country/launch_version, scans the payload for the launch's onboarded countries and research modules, and reads the rocket/payload version stamps. In MDI, the settings come from the countdown script (their names there in brackets below).

Usage

```
mdi_config(
  mdi_dir,
  country,
  launch_version,
  input_dir,
  output_dir,
  tmp_dir,
```

```
  import = TRUE,
  clean_tmp = FALSE,
  phc_run = TRUE,
  module_run = FALSE,
  debug = FALSE,
  filtered_harmonization = FALSE,
  db_user = NULL,
  datasets = NULL,
  modules = NULL,
  modules_dir = NULL
)
```

Arguments

mdi_dir	Character, the root MDI installation directory (dirMDI).
country	Character, 2-3 letter NSI country code (CountryCode).
launch_version	Character, e.g. "v2.4" (MDI_launch_version).
input_dir	Character, directory holding the NSI/mock firm-level data files (dirINPUTDATA).
output_dir	Character, directory generated files are exported to (dirOUTPUT).
tmp_dir	Character, directory for temporary MDI virtual dataset storage (dirTMPSAVE), used as given (MDI's countdown includes the country, so switching country can't mix two countries' datasets).
import	Logical, import raw NSI data files if TRUE; if FALSE, read the already-harmonized MDI virtual dataset from tmp_dir instead (MDIimportFlag).
clean_tmp	Logical, clean tmp_dir before running (MDIcleanTMP).
phc_run	Logical, FALSE to skip the post-harmonization report (mdi_post_harmonization_checks()), e.g. for mock data or CI (MDIphcRUN). Default TRUE.
module_run	Logical, TRUE once post-harmonization checks have passed and modules are ready to run; FALSE when running a launch for the first time (MDImoduleRUN).
debug	Logical, TRUE to leave logs/warnings/errors unsuppressed (MDIdebug).
filtered_harmonization	Logical, harmonize only the variables listed in the current launch's MDnames_select file (filteredHarmonization).
db_user	Character or NULL, database user for NSI datafiles with format == "oracle" (db_user, if set).
datasets	Character vector or NULL, the MD datasets to harmonize (MDIdatasets, if set), e.g. c("SBS", "PRODCOM") to redo only those after a run that stopped there; the other datasets already in tmp_dir stay as they are. NULL (default): all. Checked against the country's concordance by mdi_harmonize_pipeline().
modules	Character vector or NULL, the research modules to run (MDImodules, if set), e.g. "MOD0". NULL (default): every module of the launch. With filtered_harmonization, only the chosen modules' variables are harmonized.
modules_dir	Character or NULL, a folder of research modules to use instead of the launch's Rmodules/ (dirMODULES, if set), e.g. a workshop participant's own module

next to a shared, read-only MDI installation. Each 4-letter-named subfolder is a module. NULL (default): the launch's modules. The modules are run by the launch's Rmodules/main.R either way.

Value

A list (cfg) with elements country, launch_version, dirs (all derived paths: mdi, rocket, control, payload, nsimeta, mdmeta, progs, output, tmpsave, inputdata), flags (import, clean_tmp, phc_run, module_run, debug, filtered_harmonization), db_user, datasets (as given), countries (onboarded country codes found under the launch's MDmetadata/), modules (the research modules to run: those found under the launch's Rmodules/ — a 4-letter-named subfolder is a module — narrowed to modules if given), rocket_version, payload_version. dirs$progs is the module folder (modules_dir if given), dirs$main the launch's Rmodules/main.R.

Examples

```
mdi_dir <- paste0(tempdir(), "/")
dir.create(paste0(mdi_dir, "rocket"), recursive = TRUE)
dir.create(paste0(mdi_dir, "payload/Launch_v2.4/MDmetadata/FI"), recursive = TRUE)
dir.create(paste0(mdi_dir, "payload/Launch_v2.4/Rmodules/CPNT"), recursive = TRUE)
writeLines("commit\nabc123", paste0(mdi_dir, "rocket/rocket_github_commit.csv"))
writeLines("commit\ndef456", paste0(mdi_dir, "payload/Launch_v2.4/payload_github_commit.csv"))
cfg <- mdi_config(
  mdi_dir = mdi_dir, country = "FI", launch_version = "v2.4",
  input_dir = mdi_dir, output_dir = mdi_dir, tmp_dir = mdi_dir,
  phc_run = FALSE
)
cfg$countries
cfg$modules
```

---

mdi_disclose *Disclosure Check for Aggregated Data*

Description

Applies disclosure control rules to aggregated data based on dominance and minimum-observation criteria. This function is typically called in mdi_export() for export_type = "sum_stat".

Two steps are performed:

- **Primary disclosure:** Suppresses (-999) all aggregate cells that fail disclosure rules: (i) do not satisfy the dominance criterion (domPerc), or (ii) have fewer than minNumObs observations. Flags are recorded as discflag1_* and discflag1.
- **Secondary disclosure:** If hierarchical aggregation (hier = TRUE) is used and exactly one child cell is suppressed within a parent group, the smallest remaining cell in that group is also suppressed (-999). Flags are recorded as discflag2.

Usage

```
mdi_disclose(
  DT,
  hhfile = NULL,
  hier = FALSE,
  minNumObs = 5,
  domSh = 0.85,
  domAllCols = 0,
  dom_direction = c("high", "low")
)
```

Arguments

DT	A data.table containing aggregated statistics and disclosure variables (e.g. from mdi_aggregate()). **Must be keyed** on every grouping/label column (see Details).
hhfile	A data.table containing the hierarchy structure, with one column per aggregation level. Only used when hier = TRUE. Default NULL.
hier	Logical. Set to TRUE if multiple aggregation levels (child/parent nodes) are present in the dataset. Default: FALSE.
minNumObs	Numeric. Minimum number of observations required per cell. Country-specific; set to MDIminNumObs in module/pipeline code. Default: 5.
domSh	Numeric. Threshold for maximum allowed dominance share. Country-specific; set to MDIdomSh in module/pipeline code. Default: 0.85.
domAllCols	Numeric 0/1, country-specific (MDIdomAllCols in MD_disclosure_info.csv, defaulting to 0 when absent). Controls how far the dominance rule reaches: • 0 (default): sum-columns only. Reproduces the historical behaviour. • 1: every statistic derived from the dominance variable — domPerc_<var> covers all <agg>_<var> columns and a bare domPerc covers every statistic column, so weighted means, medians and quantiles are protected too.
dom_direction	Character. Direction of the dominance suppression rule. "high" (default) suppresses cells where domPerc >= domSh (standard rule). "low" suppresses cells where domPerc < domSh (inverted rule, used for Germany where the dominance formula is defined differently).

Details

- **The input must be keyed** (setkeyv) on all of its grouping/label columns. The key is the only signal separating label columns from statistic columns; without it every label would be suppressed to -999 along with the numbers. mdi_aggregate() sets the key, but rbindlist() drops it, so a table assembled from several aggregations must be re-keyed before export. Unkeyed input is an error, not silent corruption.
- A table carrying mdi_hier_apply()'s node column must be passed with hier = TRUE (and its hhfile); a table with no node column must be passed with hier = FALSE. Either mismatch is an error — silently they would leave a suppressed child recoverable from its parent total, or return an empty table.

- Dominance targets are resolved from the domPerc column *name* (domPerc_<var> -> <agg>_<var> columns, matched by longest suffix so emp and tot_emp do not collide), not paired positionally with sum-columns. A bare domPerc (produced when the dominance variable is a fixed column such as emp rather than "var") reaches every sum-column.
- dom_direction = "high" suppresses when domPerc >= domSh; dom_direction = "low" suppresses when domPerc < domSh (Germany).
- Suppression affects both the target aggregate columns and, when triggered, the NumObs column. Secondary disclosure ensures that no single cell can be back-calculated from its parent total.

Value

A copy of DT (the input is left unchanged), with:

- Suppressed cells (-999) where disclosure criteria are not met.
- Flags discflag1_* and discflag1 for primary disclosure.
- Flag discflag2 for secondary disclosure (if applicable).

Examples

```
library(data.table)
DT <- data.table(
  nace        = c("A", "B", "C"),
  sum_emp     = c(100, 200, 40),
  mean_emp    = c(5, 8, 2),
  domPerc_emp = c(0.50, 0.92, 0.20),
  NumObs      = c(10L, 10L, 3L)
)
setkeyv(DT, "nace")
# domAllCols = 0: dominance masks sum_emp only; C is below minNumObs
mdi_disclose(DT, hier = FALSE, minNumObs = 5, domSh = 0.85, domAllCols = 0)
# domAllCols = 1: dominance also masks mean_emp
mdi_disclose(DT, hier = FALSE, minNumObs = 5, domSh = 0.85, domAllCols = 1)
```

---

mdi_disclosure_config *Build the disclosure-control configuration for one country*

Description

Reads MD_disclosure_info.csv's disclosure_variable column and the country's own column, and returns them as a named list. Every value is a number, except MDIdomVar (a variable name) and MDIdomDirection ("high" or "low"), kept as text.

Usage

```
mdi_disclosure_config(mdmeta_dir, country)
```

Arguments

mdmeta_dir	Character, the launch's MDmetadata/ directory.
country	Character, the NSI country code.

Value

A named list, one element per row of MD_disclosure_info.csv, under the file's own disclosure_variable names verbatim (currently MDIminNumObs, MDIdomVar, MDIdomSh, MDIdomNr, MDIdomAllCols, MDIdomDirection, show_domPerc, show_values). Numbers, except MDIdomVar and MDIdomDirection ("high", or "low" for DE).

Examples

```
dir <- tempfile()
dir.create(dir)
writeLines(c(
  "disclosure_variable,FI",
  "MDIminNumObs,3",
  "MDIdomVar,var"
), paste0(dir, "/MD_disclosure_info.csv"))
mdi_disclosure_config(paste0(dir, "/"), "FI")
```

---

mdi_export *Export Data and Log Output Description*

Description

Exports a given data table to a specified format (csv, RDS, txt, dta, xlsx, sas) and appends an entry to an output description file.

Usage

```
mdi_export(
  DT,
  format,
  output_name,
  datasets_used,
  purpose,
  share_0_1,
  zeroes,
  selection,
  dirOUTPUT,
  hier = FALSE,
  hhfile = NULL,
  rel_other_output = "",
  export_type = "sum_stat",
  description = " ",
```

```
  CountryCode = "XX",
  res_group = "",
  show_values = 0,
  show_domPerc = 1,
  minNumObs = 5L,
  domSh = 0.85,
  domAllCols = 0,
  dom_direction = c("high", "low"),
  disc_method = c("obs_df", "firm_count")
)
```

Arguments

DT	A data.table to be exported.
format	Character. Export format: "csv", "RDS", "txt", "dta", "xlsx", or "sas".
output_name	Character. File name to create, without extension or country code prefix.
datasets_used	Character vector. Name(s) of the MD datasets used.
purpose	Character. Research purpose of the analysis.
share_0_1	Character. Explanation of any shares equal to 0 or 1.
zeroes	Character. Explanation of any zero values, or justification for suppressing them.
selection	Character. Whether results cover the full sample or a selection.
dirOUTPUT	Character. Path to the output directory (must end with "/").
hier	Logical. TRUE if the dataset has multiple aggregation levels (hierarchical structure). Default FALSE.
hhfile	A data.table hierarchy file, required when hier = TRUE. Default NULL.
rel_other_output	Character. How this output relates to other exported files. Default "".
export_type	Character. Type of output: "sum_stat", "reg_tab", or "other". Default "sum_stat".
description	Character. Additional explanation of the output file. Default " ".
CountryCode	Character. Two-letter ISO country code used in file naming and the output description log. Default "XX".
res_group	Character. Research group identifier used in file naming. Default "".
show_values	Numeric. 1 to show disclosed values, 0 to mask them. Default 0.
show_domPerc	Numeric. 1 to include dominance percentile columns in output, 0 to remove them. Default 1.
minNumObs	Integer. Minimum number of observations for disclosure checks, country-specific (MDIminNumObs). Passed to mdi_disclose() for "sum_stat" and mdi_disclose_reg_tab() for "reg_tab". Default 5L.
domSh	Numeric. Dominance-share threshold for "sum_stat" exports, country-specific (MDIdomSh). Passed to mdi_disclose(). Default 0.85.
domAllCols	Numeric 0/1 for "sum_stat" exports, country-specific (MDIdomAllCols). 0 (default) masks sum-columns only; 1 masks every statistic derived from the dominance variable. Passed to mdi_disclose().

dom_direction — Character. Dominance suppression direction for "sum_stat" exports. "high" (default) suppresses when domPerc >= domSh; "low" suppresses when domPerc < domSh. Passed to mdi_disclose().

disc_method — Character. Disclosure rule for "reg_tab" exports. "obs_df" (default) uses df and NumObs; "firm_count" uses NumFirms and NumEnt. Passed to mdi_disclose_reg_tab().

Value

Writes files to disk; returns NULL invisibly.

Examples

```
# sums of employment per industry, with their dominance shares and
# observation counts
emp_by_nace <- data.table::data.table(nace = c("A", "B"), sum_emp = c(100, 5),
                                      domPerc_emp = c(0.5, 0.5), NumObs = c(10L, 2L))
data.table::setattr(emp_by_nace, "sumvars", "emp")
data.table::setkeyv(emp_by_nace, "nace")
out_dir <- paste0(tempfile("mdi_export_"), "/")
dir.create(out_dir)
mdi_export(
  DT            = emp_by_nace,
  format        = "csv",
  output_name   = "Employment_by_industry",
  datasets_used = "SBS",
  purpose       = "Employment by industry.",
  share_0_1     = "None",
  zeroes        = "None",
  selection     = "Full sample",
  export_type   = "sum_stat",
  CountryCode   = "XX",
  res_group     = "ABCD",
  dirOUTPUT     = out_dir
)
# industry B has fewer than minNumObs = 5 observations: its sum is masked
data.table::fread(paste0(out_dir, "XX_ABCD_Employment_by_industry.csv"))
readLines(paste0(out_dir, "OutputDescription.csv"))
```

---

mdi_harmonize *Harmonize an NSI data file to MD specification*

Description

Applies a data file's rows of the NSI-to-MD concordance, method by method:

revalue detail is an expression in x, the NSI variable, e.g. log(x), x*1000 or format(as.Date(x, "%Y%m%d"), "%Y").

recode / reclass `detail` names the code table (codeconc or classconc) that maps the variable's values, `left` to `right`.

redefine builds an MD variable from several NSI variables: `detail` is `+` or `-` (NA-safe sum or difference) or `fn(...)`.

remap renames the NSI variable to its MD name.

It then keeps only the MD variables and sets `year`: from the file's year variable if it has one (a panel), otherwise from the concordance.

Usage

```
mdi_harmonize(
  DT,
  concordance,
  country,
  code_tables = list(),
  year_var = NA_character_,
  verbose = FALSE
)
```

Arguments

`DT`	data.table, one NSI data file. Modified by reference.
`concordance`	data.table, the concordance rows for this file: `NSI_dataset`, `NSI_varname`, `MD_varname`, `method`, `detail`, `year`.
`country`	character(1), the country code. A `detail` starting with it names an NSI metadata table, which `mdi_load_nsi_metadata()` loads under an `NSI` prefix instead (`FI_MD_codeconc` -> `NSI_MD_codeconc`).
`code_tables`	named list of the code tables recode/reclass rows can name: the NSI and MD metadata, e.g. `c(cfg$nsi, cfg$md)`.
`year_var`	character(1), the file's year variable, or `NA`/`""` if it has none.
`verbose`	logical(1), report progress with messages.

Value

`DT`, harmonized.

Examples

```
raw <- data.table::data.table(TURN = c(1, 2), EMP = c(3, 4))
conc <- data.table::data.table(
  NSI_dataset = "sbs", year = 2020,
  NSI_varname = c("TURN", "TURN", "EMP"),
  MD_varname = c("turnover", "turnover", "emp"),
  method = c("revalue", "remap", "remap"),
  detail = c("x*1000", NA, NA)
)
mdi_harmonize(raw, conc, "FI")
```

mdi_harmonize_codes_over_time *Harmonize a classification code over time*

Description

Classification codes (e.g. CN8 product codes, PRODCOM codes, NUTS regions) change from year to year. Given a year-on-year concordance of old to new codes, builds the mapping from each year's codes to the codes of the last year with `mdi_make_conc()`, and applies it to `dt`: the harmonized code replaces `code_name`, and the original is kept as `NSI_<code_name>`.

Usage

```
mdi_harmonize_codes_over_time(dt, conc, code_name, data_years)
```

Arguments

`dt`	A `data.table` with a `year` column and a `code_name` column.
`conc`	A `data.table`, the year-on-year concordance (`<country>_<code>t0_<code>t1_conc.csv`): `left` (old code), `right` (new code), `year`.
`code_name`	Character, the code column to harmonize (e.g. `"cn08"`).
`data_years`	Numeric, length 2: the first and last year the raw data covers. Harmonization runs over the years both the concordance and the data cover (from the year after the data's first year, since the first year has no earlier year to map from).

Value

A list: `data`, `dt` with the harmonized `code_name` and the original as `NSI_<code_name>`; and `codes`, the mapping table `mdi_make_conc()` produced (`year`, `code_name`, `<code_name>_harmonized`). `dt` itself is not modified. `NULL`, with a warning, when the concordance's years and the data's don't overlap.

Examples

```
# code A (2010) became B in 2011, which became C in 2012; X never changed
conc <- data.table::data.table(left = c("A", "X", "B", "X"), right = c("B", "X", "C", "X"),
                               year = c(2011, 2011, 2012, 2012))
dt <- data.table::data.table(year = c(2010, 2011, 2012, 2012), cn08 = c("A", "B", "C", "X"))
out <- mdi_harmonize_codes_over_time(dt, conc, "cn08", data_years = c(2010, 2012))
out$data
```

mdi_key_id_switch *Switch Identifier Level via Aggregation or Disaggregation*

Description

Performs aggregation or disaggregation of a harmonized dataset across hierarchical identifier levels using concordance tables and metadata-driven rules.

The supported identifier hierarchy is:

```
plantid -> firmid -> entid -> entgrp
```

The function relies on MD_varnames to identify keys, variable structure, and — via its aggregation_method/disaggregatio columns — how each variable should be transformed, and on concordance tables (*_conc.csv) to map between identifier levels.

Usage

```
mdi_key_id_switch(
  DT,
  conctab,
  md_varnames,
  method,
  obj_id_type,
  cols = NULL,
  key_id = "firmid",
  weight_data = NULL
)
```

Arguments

DT	A data.table containing the harmonized dataset to transform. Must include exactly one firm identifier column (plantid, firmid, entid, or entgrp) and a year column.
conctab	A data.table concordance table mapping source identifier to target identifier, with columns for both id levels and year.
md_varnames	A data.table, MD_varnames already filtered to the current dataset (e.g. MD_varnames[MD_dataset == "BR"]). Passing unfiltered MD_varnames (spanning more than one MD_dataset) errors.
method	Character. "agg" to aggregate from lower to higher identifier level, or "disagg" to disaggregate from higher to lower.
obj_id_type	Character. Target identifier level: "to_firmid", "to_entid", "to_entgrp", or "to_plantid".
cols	Optional character vector of columns to process. Default NULL (all columns).
key_id	Character. Name of the source identifier column in DT (e.g. "plantid", "firmid"). Default "firmid".

`weight_data` Named list of `data.table`s, the MD datasets the run has (e.g. `list(SBS = sbs, BR = br, PRODCOM = prodcom)`), for the weights of `"weighted_alloc"` disaggregation. `NULL` (default): every `"weighted_alloc"` variable is split equally, with a warning. Only the target id, `year` and the weight columns are used; the tables are not modified.

Details

mdi_key_id_switch

Variable-specific transformation rules are looked up from the `aggregation_method`/`disaggregation_method` columns in `MD_varnames`.

For aggregation (`method = "agg"`), multiple lower-level units are combined into a higher-level unit, for example from plant to firm. Each variable is aggregated according to its `aggregation_method`.

Supported aggregation methods include `"sum"`, `"weighted_avg"`, `"weighted_mode"`, `"mode"`, `"any"`, `"all"`, `"min"`, and `"max"`.

For disaggregation (`method = "disagg"`), higher-level observations are split into lower-level units using concordance mappings, for example from firm to plant. Each variable is transformed according to its `disaggregation_method`.

Supported disaggregation methods include `"replicate"` and `"weighted_alloc"`.

Aggregation preserves totals for additive variables such as those using `"sum"`. Disaggregation with `"replicate"` does not preserve totals. Disaggregation with `"weighted_alloc"` approximately preserves totals. Factor variables are internally converted to character to avoid unintended integer behavior.

Rows whose `key_id` the concordance has no target unit for are dropped, with a message counting them per year. In aggregation, a group whose values are all `NA` stays `NA` (not 0 for `"sum"`, or `FALSE`/`TRUE` for `"any"`/`"all"`).

Survey-weight variables (`weights_<dataset>_nsi`) are dropped before the switch, with a warning: they belong to the sample, not to the firm.

If an aggregation rule names a weight column (`"weighted_avg:w"`, `"weighted_mode:w1|w2"`) and none of the named column(s) are present in `DT`, a warning is raised once upfront and the variable falls back to an unweighted mean/mode.

A `"weighted_alloc"` rule lists its weights in order, each as `<MD_dataset>.<MD_varname>`, e.g. `"weighted_alloc:SBS.emp|PRODCOM.rev|BR.persons_br|equal"`. The chain is in `MD_varnames` so every module splits a variable the same way. A weight is that column of `weight_data[[<MD_dataset>]]`, summed per target unit and year; it can't be a column of `DT` itself, which holds the source unit's value on every target row. For each source unit and year, the first candidate with weights that aren't all missing or zero for its target units wins; if none does, the value is split equally. A candidate whose dataset isn't in `weight_data`, isn't at the target level or lacks the column is skipped, with one warning naming it. A message reports, per variable, how many source units used each candidate.

Value

A `data.table` with variables aggregated or disaggregated to the requested identifier level.

Examples

```
# how each variable is aggregated and disaggregated, as in MD_varnames
md_varnames <- data.table::data.table(
  MD_dataset = "SBS",
  MD_varname = c("plantid", "firmid", "year", "emp", "sales"),
  is_key = c(1, 1, 1, 0, 0),
  aggregation_method = c(NA, NA, NA, "sum", "sum"),
  disaggregation_method = c(NA, NA, NA, "replicate",
                            "weighted_alloc:BR.persons_br|equal")
)
# which plants belong to which firm, per year
plant_firm <- data.table::data.table(plantid = c("P1", "P2", "P3"),
                                     firmid = c("F1", "F1", "F2"), year = 2020)

# plants to firms: F1 sums P1 and P2
plants <- data.table::data.table(plantid = c("P1", "P2", "P3"), year = 2020,
                                 emp = c(10, 30, 5), sales = c(100, 300, 50))
mdi_key_id_switch(DT = plants, conctab = plant_firm, md_varnames = md_varnames,
                  method = "agg", obj_id_type = "to_firmid", key_id = "plantid")

# firms to plants: F1's sales split by its plants' employment in BR, emp
# copied to each plant
firms <- data.table::data.table(firmid = c("F1", "F2"), year = 2020,
                                emp = c(40, 5), sales = c(400, 50))
br <- data.table::data.table(plantid = c("P1", "P2", "P3"), year = 2020,
                             persons_br = c(10, 30, 5))
mdi_key_id_switch(DT = firms, conctab = plant_firm, md_varnames = md_varnames,
                  method = "disagg", obj_id_type = "to_plantid", key_id = "firmid",
                  weight_data = list(BR = br))
```

`mdi_load_deflators` *Load deflators and the NACE-to-deflator-industry concordance*

Description

Reads `auxdata/deflators.csv`, filtered to the country's own rows. Every column except `cc`/`DEFind`/`nace` is coerced to numeric — `deflators.csv` is measurement data (deflator index values), the opposite default from the classification-file loaders (`mdi_read_meta_dir()`/`mdi_load_md_metadata()`/`mdi_load_nsi_metadata()`) so this does not reuse that primitive. Also reads `MD_nace_DEFind_conc.csv`, the concordance mapping NACE codes to the deflator's own industry grouping (`DEFind`).

Deliberately country-agnostic: `deflators.csv` carries a real row for every country code that needs one, including variants of a country code used for a second firm-level sample of the same country (their deflators are the country's own – deflators are national-accounts data, independent of the firm-level sample). No mapping between codes is hardcoded here.

Usage

```
mdi_load_deflators(rocket_dir, mdmeta_dir, country)
```

Arguments

rocket_dir	Character, the rocket/ directory holding auxdata/deflators.csv.
mdmeta_dir	Character, the launch's MDmetadata/ directory, holding MD_nace_DEFind_conc.csv.
country	Character, the NSI country code.

Value

A list with elements deflators and nace_DEFind (the names modules know them by as globals).

Examples

```
rocket_dir <- paste0(tempfile("rocket_"), "/")
mdmeta_dir <- paste0(tempfile("mdmeta_"), "/")
dir.create(paste0(rocket_dir, "auxdata"), recursive = TRUE)
dir.create(mdmeta_dir)
writeLines(c("cc,year,DEFind,pnv", "FI,1995,A,1.1", "PT,1995,A,2.2"),
           paste0(rocket_dir, "auxdata/deflators.csv"))
writeLines(c("nace,DEFind", "\"0111\",A01"), paste0(mdmeta_dir, "MD_nace_DEFind_conc.csv"))
mdi_load_deflators(rocket_dir, mdmeta_dir, "FI")
```

---

mdi_load_md_metadata *Load MD metadata for the current launch*

Description

Reads every MD classification/lookup file (_codebook, _varnames, _catalogue, _idInfo, _hier_levels, _classconc, _hier, _class, _indentedclass) via mdi_read_meta_dir(). MD_datafiles.csv is read separately, plainly (no column of it needs character coercion, and the equivalent NSI-side file has numeric year_start/year_end columns that must not be forced to character, so both are kept off the shared rule for consistency). MD_disclosure_info.csv is not included here — see mdi_disclosure_config().

Usage

```
mdi_load_md_metadata(mdmeta_dir)
```

Arguments

mdmeta_dir	Character, the launch's MDmetadata/ directory.

Value

A named list of data.tables, one per matched file, named after the file's own basename without extension (e.g. MD_nace_hier.csv -> MD_nace_hier), plus MD_datafiles if present.

Examples

```
dir <- tempfile()
dir.create(dir)
writeLines(c("code,description", "1,Active"), paste0(dir, "/MD_codebook.csv"))
writeLines(c("name,description"), paste0(dir, "/MD_datafiles.csv"))
md <- mdi_load_md_metadata(paste0(dir, "/"))
names(md)
```

---

mdi_load_nsi_metadata *Load NSI metadata for one country*

Description

Reads the country's stacked NSI files (_datafiles.csv, _varnames.csv) from nsimeta_dir, and the NSI-to-MD concordance/classification files (_conc, _classconc, _codeconc, plus the NSI-level and MD-level _class/_classconc family) from nsimeta_dir and mdmeta_dir/<country>/, via mdi_read_meta_dir(), renamed from their <country>_... filenames to NSI_...-prefixed names. Per-classification files are included (e.g. FI_cn08_class.csv -> NSI_cn08_class, FI_NSI_cn08_MD_cn08_classco -> NSI_NSI_cn08_MD_cn08_classconc): mdi_harmonize()'s reclass step looks them up by exactly these names. NSI_datafiles.csv is read plainly (it has year_start/year_end numeric columns that must not be forced to character). Its path column is relative to the input data folder as given (for mock data with one subfolder per country, that folder is the country's).

Usage

```
mdi_load_nsi_metadata(nsimeta_dir, mdmeta_dir, country)
```

Arguments

nsimeta_dir	Character, the country's rocket/NSImetadata/<CC>/ directory.
mdmeta_dir	Character, the launch's MDmetadata/ directory — the country's NSI_MD_conc-family files live under mdmeta_dir/<country>/.
country	Character, the NSI country code.

Value

A named list of data.tables: NSI_datafiles, NSI_varnames, plus every matched NSI-MD concordance/classification file (e.g. FI_MD_conc.csv -> NSI_MD_conc, FI_classconc.csv -> NSI_classconc).

Examples

```
nsimeta_dir <- paste0(tempfile("nsimeta_"), "/")
mdmeta_dir <- paste0(tempfile("mdmeta_"), "/")
dir.create(nsimeta_dir)
dir.create(paste0(mdmeta_dir, "FI"), recursive = TRUE)
writeLines(c("NSI_datafile,path,year_start,year_end", "bd2003,bd/,2003,2003"),
```

```
          paste0(nsimeta_dir, "FI_datafiles.csv"))
writeLines(c("NSI_datafile,NSI_varname,is_key", "bd2003,firmid,1"),
          paste0(nsimeta_dir, "FI_varnames.csv"))
writeLines(c("NSI_dataset,NSI_varname,MD_varname,method,detail,year",
             "bd,acq,acq,revalue,as.logical(x),2003"),
          paste0(mdmeta_dir, "FI/FI_MD_conc.csv"))
nsi <- mdi_load_nsi_metadata(nsimeta_dir, mdmeta_dir, "FI")
names(nsi)
```

`mdi_plot_breaks` *Plot the yearly statistics of a variable, marking its breaks*

Description

Plots p25, p50, p75 and mean of a numeric variable per year on the left axis and sd (dashed) on the right, and labels the years `mdi_check_harmonized()` flagged as breaks. Needs the ggplot2 package.

Usage

```
mdi_plot_breaks(series, breaks, title = NULL)
```

Arguments

`series`	data.table, one variable's yearly statistics: `year`, `p25`, `p50`, `p75`, `mean`, `sd` (an element of `mdi_check_harmonized()`'s `series`).
`breaks`	data.table, that variable's rows of `mdi_check_harmonized()`'s `breaks` (`dataset`, `variable`, `stat`, `year`); may have no rows.
`title`	character(1) or `NULL`: the plot's title. `NULL`: `paste("Break Detection in Aggregates:", dataset, "-", variable)`, from `breaks`.

Value

A ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  dt <- data.table::data.table(firmid = rep(letters[1:4], 3),
                               year = rep(2019:2021, each = 4),
                               emp = c(1:4, 1:4, 11:14))
  catalogue <- data.table::data.table(
    MD_dataset = "SBS", MD_varname = c("firmid", "year", "emp"),
    is_key = c(1L, 1L, 0L), class = c("character", "numeric", "numeric"),
    domain = "", FI = "2019 - 2021")
  res <- mdi_check_harmonized(dt, "SBS", catalogue, "FI")
  mdi_plot_breaks(res$series$emp, res$breaks)
}
```

mdi_plot_na_share *Plot the yearly share of NA values of a variable, marking flagged years*

Description

Plots the share of NA values of a variable per year, and labels the years mdi_check_harmonized() flagged (a change above na_threshold, or every value NA). Needs the ggplot2 package.

Usage

```
mdi_plot_na_share(na_series, na_shares, title = "Share of NA values")
```

Arguments

na_series	data.table, one variable's yearly share of NA values: year, na_share (an element of mdi_check_harmonized()'s na_series).
na_shares	data.table, that variable's rows of mdi_check_harmonized()'s na_shares (year); may have no rows.
title	character(1), the plot's title.

Value

A ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  s <- data.table::data.table(year = 2019:2021, na_share = c(0, 0, 0.3))
  mdi_plot_na_share(s, data.table::data.table(year = 2021), "SBS - emp")
}
```

mdi_post_harmonization_checks *Write the post-harmonization quality report*

Description

The review gate between harmonization and the modules: checks every harmonized MD dataset in cfg$dirs$tmpsave with mdi_check_harmonized() (duplicate keys, rows without a year, classes and date formats, date ranges, breaks in yearly statistics, jumps in the yearly share of NA values) and adds the coercion fallout log (NAs introduced, or out-of-domain values force-cast to logical, by the type conversion). Writes, in cfg$dirs$tmpsave:

- <CC>_phc_results.txt, all results;

- `breaks_report.pdf`, a page per variable either check flags: its yearly statistics (`mdi_plot_breaks()`, numeric variables) above its yearly share of `NA` values (`mdi_plot_na_share()`); needs ggplot2 and gridExtra.

Every numeric variable a dataset has is checked, including the ones the pipeline computes (`K_pim`), so run this after the whole pipeline. Someone at the NSI reviews these; once they look fine, `MDImoduleRUN = TRUE` lets the modules run.

Usage

```
mdi_post_harmonization_checks(
  cfg,
  threshold = 0.25,
  na_threshold = 0.1,
  verbose = FALSE
)
```

Arguments

`cfg`	config list from `mdi_config()`, with the MD metadata (`cfg$md`) and the selected NSI metadata (`cfg$nsi`) loaded.
`threshold`	numeric(1), the relative year-on-year change flagged as a break (see `mdi_check_harmonized()`).
`na_threshold`	numeric(1), the year-on-year change in the share of `NA` values flagged (see `mdi_check_harmonized()`).
`verbose`	logical(1), report where the results are with a message.

Value

The per-dataset results of `mdi_check_harmonized()`, invisibly.

Examples

```
# a harmonized SBS with a duplicated firm-year and a row without a year
tmpsave <- paste0(tempfile("phc_"), "/")
dir.create(tmpsave)
saveRDS(data.table::data.table(firmid = c("a", "a", "b"), year = c(2020L, 2020L, NA),
                               emp = c(1, 1, 1)),
        paste0(tmpsave, "SBS.RDS"))
catalogue <- data.table::data.table(
  MD_dataset = "SBS", MD_varname = c("firmid", "year", "emp"), is_key = c(1L, 1L, 0L),
  class = c("character", "integer", "numeric"), domain = "", FI = "2020 - 2021")
cfg <- list(dirs = list(tmpsave = tmpsave), country = "FI",
            md = list(MD_catalogue = catalogue,
                      MD_varnames = catalogue[, c("MD_dataset", "MD_varname")]),
            nsi = list())
res <- mdi_post_harmonization_checks(cfg)
res$SBS$duplicates
readLines(paste0(tmpsave, "FI_phc_results.txt"))[1:8]
```

`mdi_preprocess` *Preprocessing Tool for Raw Data Harmonization*

Description

This function applies one or more preprocessing operations to a `data.table` object. It is designed to prepare raw microdata files for harmonization by applying a flexible set of transformations through a structured instruction string.

Usage

```
mdi_preprocess(preprocessing_string, dt, read_datafile = NULL)
```

Arguments

`preprocessing_string`	A character string defining one or more preprocessing instructions
`dt`	A `data.table` containing the raw input dataset to be processed
`read_datafile`	A function `function(datafile, cols, text_cols)` returning the columns `cols` of the raw file `datafile` as a `data.table`, with `text_cols` read as text. Only required when `preprocessing_string` contains a `mergefrom` instruction; the pipeline reads the file with its own `NSI_datafiles` row (path, format, encoding, Oracle) and its own declared classes.

Details

The input `preprocessing_string` is a single string where:

- Multiple operations are separated by double pipes (`||`)
- Parameters within each operation are separated by pipes (`|`)
- When a parameter contains multiple elements (e.g., multiple columns), they are separated by a hash symbol (#)

IMPORTANT: When performing more than one operation, **order matters**. The result of each step is fed into the next.

Supported operations:

- `dedup|<id_col1#id_col2#...>|<method>|[dedup_col]` Removes duplicates based on identifier columns. Methods:
  - `na`: keep the row with the fewest missing values
  - `max` / `min`: keep the row with the max/min value of the specified `dedup_col`
  - `meanmode`: aggregate numeric columns using the mean, and categorical columns using the mode
  - `random`: keep one row among duplicates at random (the same row on every run)
- `filter|<col>|<operator>|<value>` Filters rows based on a logical condition. Operators:
  - `eq`, `neq`: equal / not equal

  - gt, gte, lt, lte: greater/less than (or equal to)
  - in, nin: value in (or not in) a list, separated by # (e.g., US#IT#DE)
  - isna, notna: keep rows where the column is / is not missing. These take **no value**, so the instruction has three parts, not four (e.g., filter|FIRM_ID|notna to drop rows with no firm identifier).
- agg|<group_col1#group_col2#...>|<var1:func#var2:func#...> Aggregates rows to a lower dimension by grouping. Functions supported: sum, mean, min, max, median, sd, mode, pickmaxby-refcol
- restruct|<col_to_remove>|<col1#col2#...> Drops a column that introduces duplication and deduplicates on a set of remaining columns.
- reshape|<id_col1#id_col2#...>|<names_from>|<values_from> Reshapes data from long to wide format using data.table::dcast.
- reshapelong|<id_col1#id_col2#...>|<measure_col1#measure_col2#...>|<names_to>|<values_to> Reshapes data from wide to long format using data.table::melt: each measure column becomes a row. names_to is the new column holding the original column name, values_to the new column holding its value. Useful when a raw file stores one record per category in separate columns but the MD schema expects one row per category.

  Example (trade data with one column per direction, to one row per direction, then drop the empty ones): reshapelong|FIRM_ID#PRODUCT#COUNTRY|import#export|FLOW|VALUE||filter|VALUE|notna
- derive|<new_col>|condmap|<cond1>:<val1>#<cond2>:<val2>#...|default:<default_val> Creates a new column based on a sequence of conditional expressions.
  - Conditions are evaluated in order and the first match is used.
  - The condition (condX) is written in standard R syntax (e.g., colA == 1 & colB == 'X').
  - The value (valX) can be a column name or a static value.
  - A fallback value must be provided via default:<value>.

  Example: derive|new_col|condmap|type == 2:colA#year == 2008 & type == 1:colB|default:NA
- scaleif|<condition_col>|<val1:factor1#val2:factor2#...>|<col1#col2#...> Conditionally scales one or more columns by a numeric factor depending on the value of a categorical column.
  - The second argument is a mapping of value:factor pairs (e.g., 1:1#2:1000#5:100).
  - The third argument lists the columns to be scaled when the condition is met.

  Example: scaleif|unit_code|1:1#2:1000#3:0.001|value#quantity
- trimchars|<col1#col2#...>|<n> Trims the last n characters from each specified character column. Useful for standardizing identifier or code formats.

  Example: trimchars|prodcom_code|2
- parsedate|<col>|<informat>|[lang]|[outformat] Parses a date column given its layout in strptime tokens (e.g. %d-%b-%y), independently of the system locale. Month names (%b abbreviated / %B full) are matched through readr's bundled CLDR month names: lang is a two-letter language code (it, de, fr, ... - see readr::date_names_langs() for the ~185 available); English if omitted. Case is normalized first, so "31-OTT-12" and "31-ott-12" both parse. By default the column is replaced by a Date; an optional outformat (e.g. %Y%m%d) returns a formatted string instead. Values that do not parse are set to NA with a warning; if no non-empty value parses the operation stops.

  Example (balance-sheet closing dates with Italian month names like "31-OTT-12", then keep the latest balance sheet per firm): parsedate|CLOSING_DATE|%d-%b-%y|it||dedup|FIRM_ID|max|CLOSING_DATE

- `mergefrom|<datafile_name>|<join_key1#join_key2#...>|<col1#col2#...>|[join_type]`
  Reads the join keys and the listed columns of another raw file, `datafile_name` (an `NSI_datafile` of `NSI_datafiles`), with `read_datafile` (see below), and merges the columns into the current dataset on the join keys. The other file's own preprocessing is not run.
  The join type is optional and defaults to a **left join**. Supported join types:
  - `left`: keep all rows from the current dataset (default)
  - `inner`: keep only rows with matching keys in both datasets
  - `full`: full outer join; keep all rows from both datasets

  The join keys must uniquely identify rows in **both** datasets, and be text on both sides or numbers on both sides. If a key combination is missing on either side, the corresponding values are set to `NA`.
  Example: `mergefrom|sbs_2020|FIRM_ID#Jahr|NACE|left`

Value

A `data.table` with the specified preprocessing operations applied

Examples

```
dt <- data.table::data.table(firm_id = c(1, 1, 2), year = 2020,
                             country = c("IT", "IT", "FR"), emp = c(5, 6, 7))
# one row per firm and year (the one with most employment), Italian firms only
mdi_preprocess("dedup|firm_id#year|max|emp||filter|country|eq|IT", dt)
```

`mdi_run` *Run MDI up to the modules*

Description

One run at an NSI, up to the research modules: starts `OutputDescription.csv` in `cfg$dirs$output` (launch, rocket and payload versions, then a row per exported file), harmonizes the data (`mdi_harmonize_pipeline()`), writes the post-harmonization report (`mdi_post_harmonization_checks()`) unless the modules are to run (`cfg$flags$module_run`) or the report is switched off (`cfg$flags$phc_run`, e.g. for mock data), and starts `WarningsDescription.txt`.

The modules themselves run in the global environment, which a package function doesn't touch: the calling script does that after this.

Usage

```
mdi_run(cfg, oracle_channel = NULL, verbose = TRUE)
```

Arguments

`cfg`	config list from `mdi_config()`, with the MD metadata (`cfg$md`) loaded.
`oracle_channel`	passed on to `mdi_harmonize_pipeline()`.
`verbose`	logical(1), report the run's progress with messages, including where the post-harmonization report is and what to do next.

Value

`cfg`, as `mdi_harmonize_pipeline()` returns it, invisibly.

Index

hier_col, 3
hier_levels(), *3*

log_warnings, 4

mdi_aggregate(), *14*
mdi_build_pim_capital, 5
mdi_check_harmonized, 6
mdi_check_harmonized(), *26–28*
mdi_classify_conc, 8
mdi_coerce_md_types, 9
mdi_concord_nace, 10
mdi_concord_nace(), *6*
mdi_config, 11
mdi_config(), *28*, *31*
mdi_disclose, 13
mdi_disclose(), *17, 18*
mdi_disclose_reg_tab(), *17, 18*
mdi_disclosure_config, 15
mdi_export, 16
mdi_export(), *13*
mdi_harmonize, 18
mdi_harmonize_codes_over_time, 20
mdi_harmonize_pipeline(), *12*, *31, 32*
mdi_key_id_switch, 21
mdi_key_id_switch(), *5*
mdi_load_deflators, 23
mdi_load_md_metadata, 24
mdi_load_nsi_metadata, 25
mdi_load_nsi_metadata(), *19*
mdi_make_conc(), *20*
mdi_pim_capital(), *5*
mdi_plot_breaks, 26
mdi_plot_breaks(), *7*, *28*
mdi_plot_na_share, 27
mdi_plot_na_share(), *8*, *28*
mdi_post_harmonization_checks, 27
mdi_post_harmonization_checks(), *12*, *31*
mdi_preprocess, 29
mdi_run, 31