Package 'scrutr'

April 25, 2026

Title Scrutinizing Collections of Structured Datasets

Version 0.3.1

Description Provides a coherent interface for exploring and transforming multiple related data frames that share a common structure. Complements single-dataset inspection tools by operating across an entire collection at once. Also includes lightweight utilities for related file and folder management tasks.

License MIT + file LICENSE

URL https://github.com/danielrak/scrutr

BugReports https://github.com/danielrak/scrutr/issues

Depends R (>= 4.1.0)

Imports dplyr, glue, lubridate, magrittr, purrr, rio, stats, stringr, tibble, tools, utils, writexl

Suggests knitr, readxl, rmarkdown, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Daniel Rakotomalala [aut, cre]

Maintainer Daniel Rakotomalala <rakdanielh@gmail.com>

Repository CRAN

Date/Publication 2026-04-25 09:40:08 UTC

Contents

`chars_structure` *Get :alpha: / :digit: patterns from each symbol of character vector*

Description

Get :alpha: / :digit: patterns from each symbol of character vector

Usage

```
chars_structure(input_vector, unique = TRUE, named_output = TRUE)
```

Arguments

`input_vector`	Character. Vector to process
`unique`	Logical 1L. If TRUE, the result is reduced to unique values.
`named_output`	Logical 1L. If TRUE, output vector is named after corresponding input values.

Value

Character. Vector describing structure of each element of input_vector, see example.

Examples

```
library(magrittr)
input <- c("ABC123", "DE4F56", "789GHI", "ABC123")

# Default values of unique and named_output:
chars_structure(input_vector = input, unique = TRUE, named_output = TRUE)

# unique is set to default value TRUE and named_output is set to FALSE:
chars_structure(input_vector = input, unique = TRUE, named_output = FALSE)

# unique is set to FALSE and named_output to FALSE:
chars_structure(input_vector = input, unique = FALSE, named_output = FALSE)

# unique is set to FALSE and named_output to defalut value TRUE:
chars_structure(input_vector = input, unique = FALSE, named_output = TRUE)
```

---

chars_structure_general

Generalized characters structure

Description

Get any user-defined patterns from each symbol of character vector

Usage

```
chars_structure_general(
  input_vector,
  split = "",
  patterns_and_replacements,
  unique = TRUE,
  named_output = TRUE
)
```

Arguments

input_vector	Character. Vector to process
split	Character 1L. Symbol separator
patterns_and_replacements	Character. Named character vector of patterns (names) and replacements (values)
unique	Logical 1L. If TRUE, the result is reduced to unique values
named_output	Logical 1L. If TRUE, output vector is named after corresponding input values.

Value

A character vector describing the generalized structure of each element of `input_vector`. If `named_output` is `TRUE`, the vector is named after the corresponding input values. If `unique` is `TRUE`, only structures for unique input values are returned; otherwise one structure per element of `input_vector`.

Examples

```
input <- c("ABC123", "DE4F56", "789GHI", "ABC123")

# Default values of unique and named_output:
chars_structure_general(input_vector = input, split = "",
                        patterns_and_replacements = c("[:alpha:]" = "[letter]",
                                                      "[:digit:]" = "[number]"),
                        unique = TRUE, named_output = TRUE)

# unique is set to default value TRUE and named_output is set to FALSE:
chars_structure_general(input_vector = input, split = "",
                        patterns_and_replacements = c("[:alpha:]" = "[letter]",
                                                      "[:digit:]" = "[number]"),
                        unique = TRUE, named_output = FALSE)

# unique is set to FALSE and named_output to FALSE:
chars_structure_general(input_vector = input, split = "",
                        patterns_and_replacements = c("[:alpha:]" = "[letter]",
                                                      "[:digit:]" = "[number]"),
                        unique = FALSE, named_output = FALSE)

# unique is set to FALSE and named_output to defalut value TRUE:
chars_structure_general(input_vector = input, split = "",
                        patterns_and_replacements = c("[:alpha:]" = "[letter]",
                                                      "[:digit:]" = "[number]"),
                        unique = FALSE, named_output = TRUE)
```

---

convert_all *Convert all datasets within a folder to a given file format*

Description

Scan a folder for files matching specified extensions, then convert each one to the target format. Void character values are replaced with `NA` and character whitespace is trimmed.

Usage

```
convert_all(
  input_folderpath,
  considered_extensions,
  to,
  output_folderpath = input_folderpath
)
```

Arguments

`input_folderpath`
: Character. Folder path containing datasets to convert.

`considered_extensions`
: Character vector. File extensions to include (must be supported by **rio**).

`to`
: Character. The target output format (must be supported by **rio**).

`output_folderpath`
: Character. Folder path for converted files (defaults to `input_folderpath`).

Value

Invisibly returns a character vector of output file paths.

Examples

```
mydir <- system.file("permadir_examples_and_tests/convert_all", package = "scrutr")
outdir <- tempdir()

convert_all(input_folderpath = mydir,
            considered_extensions = c("rds"),
            to = "csv",
            output_folderpath = outdir)

list.files(outdir, pattern = "csv$")
```

`convert_r` *Batch dataset format conversion using an Excel mask*

Description

Read an Excel mask describing files to convert, then import, clean, and re-export each file in the desired format. Cleaning includes replacing void character values (`""`) with `NA` and trimming whitespace.

Usage

```
convert_r(mask_filepath, output_path)
```

Arguments

`mask_filepath`
: Character. Full file path to the Excel mask. The mask must contain columns: `folder_path`, `file`, `converted_file`, `to_convert` (1 to convert, 0 to skip).

`output_path`
: Character. Folder path where converted files will be placed.

Value

Invisibly returns a character vector of output file paths.

Examples

```
mydir <- system.file("permadir_examples_and_tests/convert_r", package = "scrutr")

mask <- data.frame(
  folder_path = rep(mydir, 2),
  file = c("original_cars.rds", "original_mtcars.csv"),
  converted_file = c("converted_cars.csv", "converted_mtcars.csv"),
  to_convert = rep(1, 2)
)


mask_path <- file.path(tempdir(), "mask_convert_r.xlsx")
writexl::write_xlsx(mask, mask_path)

convert_r(
  mask_filepath = mask_path,
  output_path = tempdir()
)

# Clean up:
file.remove(file.path(tempdir(),
  c("converted_cars.csv", "converted_mtcars.csv", "mask_convert_r.xlsx")))
```

`detect_chars_structure` *Detect character structure Detect if values within a character variable match at least one of defined patterns*

Description

Detect character structure Detect if values within a character variable match at least one of defined patterns

Usage

```
detect_chars_structure(vector, patterns, verbose = FALSE)
```

Arguments

`vector`	Character. Input vector to detect pattern from
`patterns`	Character. Patterns to detect within vector. Regex is supported
`verbose`	Logical 1L. If TRUE, additional details related to the pattern detection are provided

Value

Logical 1L. If verbose is set to TRUE, the function returns a list with the following elements in order:

- "Any defined structure" : Logical 1L. TRUE if the pattern is detected anywhere from the input vector
- "Which" : Character. Unique values of input vector matching the defined patterns
- "Where" : Integer. Indexes of values from input vector matching the defined patterns

Examples

```
detect_chars_structure(
  vector = c("ABCD1234", "4567EF", "89GHIJ10"),
  patterns = "[:alpha:]{4}" # detect four consecutive alphabetic values
)

detect_chars_structure(
  vector = c("ABCD1234", "4567EF", "89GHIJ10"),
  patterns = "[:alpha:]{4}",
  verbose = TRUE
)
```

---

detect_chars_structure_datasets

Detect character structure from datasets

Description

Detect character structure from datasets

Usage

```
detect_chars_structure_datasets(
  datasets_folderpath,
  considered_extensions,
  patterns,
 output_filepath = file.path(datasets_folderpath, paste0("detect_chars_structure_",
    basename(datasets_folderpath), ".rds")),
  get_output_in_session = TRUE
)
```

Arguments

`datasets_folderpath`
: Character 1L. Folder path of datasets to process. These datasets must be at the root of the path

considered_extensions
: Character. Datasets file extensions to consider. Extensions must be one supported by the rio:: package

patterns
: Character. Patterns to detect across the datasets variables. Regex is supported

output_filepath
: Character 1L. Output folder path.

get_output_in_session
: Logical 1L. If TRUE, the function return a list, such that each element element corresponds to pattern detection details for each considered dataset

Value

If get_output_in_session is TRUE, a named list of data frames (one per dataset file), each with columns var (variable name), any_defined_structure (logical), and examples (character). The list is also saved as an RDS file at output_filepath. If get_output_in_session is FALSE, the function returns NULL invisibly and is called for its side effect of writing the RDS file.

Examples

```
mydir <- system.file("detect_chars_structure_datasets", package = "scrutr")
outfile <- file.path(tempdir(), "detect_college.rds")

detect <- detect_chars_structure_datasets(
  datasets_folderpath = mydir,
  considered_extensions = "xlsx",
  patterns = "(?i)college",
  output_filepath = outfile,
  get_output_in_session = TRUE)

# head(lapply(detect, head))

file.exists(outfile)
unlink(outfile)
```

---

dupl_show *Show observations of all duplicated values of a variable or a combination of variables*

Description

Show observations of all duplicated values of a variable or a combination of variables

Usage

```
dupl_show(data_frame, vars)
```

Arguments

data_frame	Data.frame. Input data frame. Must be in the Global Environment and has a data.frame class
vars	Character. Vector of variable or combination of variables from which duplicates are checked

Value

Data.frame. The part of inputted data frame with all observations of duplicated values of indicated variable or combination of variables

Examples

```
# A fictional data with duplicated values:
df <- data.frame("person_id" = c(1, 1, 2, 3,
                                 2, 4, 5, 5 ,1),
                 "person_age" = c(25, 25, 21, 32,
                                  21, 48, 50, 50, 52),
                 "survey_month" = c("jan", "feb", "mar", "apr",
                                    "apr", "may", "jun", "jul", "jan"),
                 "survey_answer" = c("no", "yes", "no", "yes",
                                     "yes", "yes", "no", "yes", NA))

# Shuffling observations and columns to make duplicates difficult to see:
set.seed(1)
df <- df[sample(1:nrow(df)),
         sample(1:ncol(df))]
df

# See all of the rows where person_id has more than an unique possible value:
dupl_show(data = df, var = "person_id")

# See all of the rows where the combination of person_id and survey_month variables has
# more than an unique possible value :
dupl_show(data = df, var = c("person_id", "survey_month"))
```

---

dupl_sources *Illustrate sources of all duplicated values of a variable or a combination of variables*

Description

Illustrate sources of all duplicated values of a variable or a combination of variables

Usage

```
dupl_sources(data_frame, vars, output_as_df = FALSE)
```

Arguments

data_frame	Data.frame. Input data frame. Must be in the Global Environment and has a data.frame class
vars	Character. Vector of variable or combination of variables from which duplicates are checked
output_as_df	Logical 1L. If TRUE, output is rendered as a data.frame.

Value

List or data.frame. For each duplicated row regarding to vars, different values of the same variable are shown, separated by AND

Examples

```
# A fictional data with duplicated values:
df <- data.frame("person_id" = c(1, 1, 2, 3,
                                 2, 4, 5, 5 ,1),
                 "person_age" = c(25, 25, 21, 32,
                                  21, 48, 50, 50, 52),
                 "survey_month" = c("jan", "feb", "mar", "apr",
                                    "apr", "may", "jun", "jul", "jan"),
                 "survey_answer" = c("no", "yes", "no", "yes",
                                     "yes", "yes", "no", "yes", NA))

# Shuffling observations and columns to make duplicates difficult to see:
set.seed(1)
df <- df[sample(1:nrow(df)),
         sample(1:ncol(df))]
df

dupl_sources(data_frame = df, vars = "person_id")
dupl_sources(data_frame = df, vars = "person_id", output_as_df = TRUE)
```

---

folder_structure_replicate

Replicate the folder structure of a given directory

Description

Replicate the folder structure of a given directory

Usage

```
folder_structure_replicate(dir, to)
```

Arguments

dir	Character 1L. Path of directory which structure will be replicated
to	Character 1L. Path of an output directory in which replicated structured will be placed

Value

No return value, called for side effects. The directory structure of dir is recreated inside the to directory.

Examples

```
library(magrittr)

temp_dir_to_replicate <- tempfile()
dir.create(temp_dir_to_replicate)

dir.create(file.path(temp_dir_to_replicate, "dir1"))
dir.create(file.path(temp_dir_to_replicate, "dir2"))

temp_dir_out <- tempfile()
dir.create(temp_dir_out)

folder_structure_replicate(
  dir = temp_dir_to_replicate,
  to = temp_dir_out)

unlink(temp_dir_to_replicate)
unlink(temp_dir_out)
```

---

inspect *Inspect a data frame*

Description

Compute a summary of each variable in a data frame: class, distinct values, missing values, void strings, character lengths, and sample modalities.

Usage

```
inspect(data_frame, nrow = FALSE)
```

Arguments

data_frame	A data frame to inspect.
nrow	Logical. If TRUE, the number of observations is printed as a message in addition to the returned table.

Value

A tibble with one row per variable and columns: `variables`, `class`, `nb_distinct`, `prop_distinct`, `nb_na`, `prop_na`, `nb_void`, `prop_void`, `nchars`, `modalities`.

Examples

```
inspect(CO2)
```

`inspect_vars` *Inspect a collection of datasets*

Description

Read all datasets from a folder matching the specified extensions, inspect each one, and write a comprehensive Excel report covering variable presence, types, and per-dataset summaries.

Usage

```
inspect_vars(
  input_path,
  output_path,
  output_label,
  considered_extensions,
  encoding = NULL
)
```

Arguments

`input_path`	Character. Folder path containing datasets to explore.
`output_path`	Character. Folder path where the Excel output will be written.
`output_label`	Character. A concise label for the output file name.
`considered_extensions`	Character vector. File extensions to include (without the dot, e.g. `"rds"` not `".rds"`).
`encoding`	Character 1L or `NULL`. Encoding passed to `rio::import()` for text-based formats (CSV, TSV). Must be one of `"unknown"`, `"UTF-8"`, or `"Latin-1"`. Defaults to `NULL` (let `rio` / `data.table` pick the default). Ignored for binary formats (e.g. RDS, xlsx).

Value

Invisibly returns the path to the written Excel file. The file contains sheets: dims, inspect_tot, one per dataset inspection, vars_detect, vars_detect_everywhere, vars_detect_not_everywhere, vars_compclasses, vars_compclasses_allsame, vars_compclasses_not_allsame.

Examples

```
mydir <- file.path(tempdir(), "inspect_vars_example")
dir.create(mydir)

saveRDS(cars, file.path(mydir, "cars1.rds"))
saveRDS(mtcars, file.path(mydir, "cars2.rds"))

inspect_vars(input_path = mydir, output_path = mydir,
             output_label = "cardata",
             considered_extensions = "rds")

# Read back the 10 sheets:
purrr::map(1:10, function(x)
           rio::import(file.path(mydir, "inspect_vars_cardata.xlsx"),
                       sheet = x))

unlink(mydir, recursive = TRUE)
```

---

inspect_write *Inspect a data frame and write the output to an Excel file*

Description

Inspect a data frame and write the output to an Excel file

Usage

```
inspect_write(data_frame, output_path, output_label = NULL)
```

Arguments

data_frame	A data frame to inspect.
output_path	Character. Folder path where the Excel output will be written.
output_label	Character. Optional label for the output file. If NULL, defaults to the deparsed name of data_frame.

Value

Invisibly returns the path to the written Excel file. The file contains the inspection table with prepended observation and variable counts.

Examples

```
mydir <- file.path(tempdir(), "inspect_write_example")
dir.create(mydir)
inspect_write(data_frame = cars,
              output_path = mydir,
              output_label = "cars")
readxl::read_xlsx(file.path(mydir, "inspect_cars.xlsx"))
unlink(mydir, recursive = TRUE)
```

`ljoin_checks` *Perform a classical dplyr::left_join() and add check information related to join*

Description

Perform a classical dplyr::left_join() and add check information related to join

Usage

```
ljoin_checks(ltable, rtable, ...)
```

Arguments

`ltable`	Data.frame. Left data frame in the join
`rtable`	Data.frame. Right data frame in the join
`...`	Any other arguments passed to `dplyr::left_join()`.

Value

Data.frame. Output of dplyr::left_join() with messages on number of observations in left, right and joined data frames and list of common variables between ltable and rtable

Examples

```
left_table <- data.frame("person_id" = c(1, 1, 2, 3,
                                 2, 4, 5, 5 ,1),
                 "person_age" = c(25, 25, 21, 32,
                                  21, 48, 50, 50, 52),
                 "survey_month" = c("jan", "feb", "mar", "apr",
                                    "apr", "may", "jun", "jul", "jan"),
                 "survey_answer" = c("no", "yes", "no", "yes",
                                     "yes", "yes", "no", "yes", NA))

right_table <- data.frame("person_id" = c(2, 5, 4, 3, 1),
                          "person_name" = c("John", "Marie", "Pierre", "Marc", "Jimmy"))

list("left_table" = left_table,
     "right_table" = right_table)

ljoin_checks(left_table, right_table, by = "person_id")
```

mask_convert_r *Create an excel mask compatible with the convert_r() function*

Description

Create an excel mask compatible with the convert_r() function

Usage

```
mask_convert_r(output_path, output_filename = "mask_convert_r.xlsx")
```

Arguments

output_path	Character 1L. Folder path where the mask will be created
output_filename	Character 1L. File name (with extension) of the mask

Value

No return value, called for side effects. An Excel file (`.xlsx`) is written to `file.path(output_path, output_filename)` containing an empty template with columns `folder_path`, `file`, `converted_file`, and `to_convert`.

Examples

```
mydir <- file.path(tempdir(), "convert_r_tests_examples")
dir.create(mydir)

mask_convert_r(output_path = mydir)

list.files(mydir)

unlink(mydir, recursive = TRUE)
```

mask_rename_r *Create an excel mask compatible with the rename_r() function.*

Description

Create an excel mask compatible with the rename_r() function. This must be used on a collection of files, i.e stored within the same folder.

Usage

```
mask_rename_r(input_path, output_filename = "mask_rename_r.xlsx")
```

Arguments

input_path	Character 1L. Folder containing the set of files to rename
output_filename	Character 1L. File name of the excel mask.

Value

No return value, called for side effects. An Excel mask file is written to input_path.

Examples

```
library(magrittr)
data(cars)
data(mtcars)

mydir <- tempfile()
dir.create(mydir)

saveRDS(cars, file.path(mydir, "cars.rds"))
saveRDS(mtcars, file.path(mydir, "mtcars.rds"))

list.files(mydir)

mask_rename_r(input_path = mydir)

list.files(mydir)

readxl::read_xlsx(file.path(mydir, "mask_rename_r.xlsx"))

unlink(mydir, recursive = TRUE)
```

---

path_move *Move through paths*

Description

Move through paths

Usage

```
path_move(path_vector, path_separator = "/", move)
```

Arguments

path_vector	Character. Vector of paths with equal number of levels.
path_separator	Character. Path separator (adapted to your OS, e.g. "/").
move	Integer. If positive, outputs path up to the specified level. If negative, removes the last specified level(s).

Value

Character vector of transformed paths.

Examples

```
pvector <- c(
  "level_1/level_2/level_3/file_1.ext",
  "level_1/level_2/level_3/file_2.ext"
)

path_move(path_vector = pvector,
          path_separator = "/",
          move = 1)

path_move(path_vector = pvector,
          path_separator = "/",
          move = 2)

path_move(path_vector = pvector,
          path_separator = "/",
          move = -1)

path_move(path_vector = pvector,
          path_separator = "/",
          move = -2)
```

---

rename_r *Industrialized file renaming*

Description

Industrialized file renaming

Usage

```
rename_r(mask_filepath)
```

Arguments

mask_filepath	Character 1L. Entire file path of the excel mask

Value

No return value, called for side effects. Files are renamed on disk according to the instructions in the Excel mask.

Examples

```
library(magrittr)
data(cars)
data(mtcars)

mydir <- tempfile()
dir.create(mydir)

# Two example files to rename:
saveRDS(cars, file.path(mydir, "cars.rds"))
saveRDS(mtcars, file.path(mydir, "mtcars.rds"))
list.files(mydir)

# Create the mask:
mask_rename_r(input_path = mydir)

# Fill the mask (in practice you can do it manually):
mask <- rio::import(file.path(mydir, "mask_rename_r.xlsx"))
mask[["renamed_file"]] <- c("cars_renamed.rds", "mtcars_renamed.rds")
mask[["to_rename"]] <- rep(1, 2)
writexl::write_xlsx(mask, file.path(mydir, "mask_rename_r.xlsx"))

# Apply the rename function:
rename_r(mask_filepath = file.path(mydir, "mask_rename_r.xlsx"))

# See the renamed files:
list.files(mydir)

# Clean tempdir:
unlink(mydir, recursive = TRUE)
```

---

replace_multiple *Replace character vector values using a correspondence approach*

Description

Names of `replacements` are matched literally (not as regular expressions). Elements of `input_vector` that match none of the patterns are returned unchanged.

Usage

```
replace_multiple(input_vector, replacements, replace_all = FALSE)
```

Arguments

`input_vector`	Character. Character vector on which replacements take place.
`replacements`	Character. Named character vector defining replacement correspondences (names = patterns, values = replacements). Names are treated as literal strings (regex metacharacters such as `.`, `(`, `+` are not interpreted).
`replace_all`	Logical. If `TRUE`, `stringr::str_replace_all()` is used instead of `stringr::str_replace()`.

Value

Character vector with replacements applied.

Overlapping patterns

When several patterns can match the same element, the first match found in `input_vector` (scanned left to right) is used. When patterns start at the same position, the one listed first in `replacements` wins, not the longest. Order `replacements` accordingly if some patterns are prefixes of others.

Examples

```
input <- c("one-one", "two-two-one", "three-three-two")

replace_multiple(input,
                 replacements =
                   c("one" = "1", "two" = "2",
                     "three" = "3"))

replace_multiple(input,
                 replacements =
                   c("one" = "1", "two" = "2",
                     "three" = "3"),
                 replace_all = TRUE)

# Unmatched elements are returned as-is:
replace_multiple(c("one", "unmatched"), c("one" = "1"))

# Regex metacharacters are matched literally:
replace_multiple(c("a.b", "aXb"), c("." = "DOT"))
```

---

table_prop *Frequencies and proportions in one output*

Description

Combines base::table() and base::prop.table() outputs in a single one

Usage

```
table_prop(..., margin = NULL, round = 3, noquote = FALSE)
```

Arguments

`...`	Arguments passed to `base::table()`.
`margin`	Integer 1L. The same argument as in base::prop.table()
`round`	Integer 1L. Number of digits after decimal in base::prop.table() output
`noquote`	Logical 1L. If TRUE, return an object of class noquote that provides better view of the output

Value

Matrix or noquote matrix. Frequencies with proportions in brackets, within a matrix

Examples

```
df <- data.frame(
                  "variable_1" = c("v1_1", "v1_1",
                                   "v1_2", "v1_2", "v1_2", "v1_2"),
                  "variable_2" = c("v2_1", "v2_1", "v2_1", "v2_1",
                                   "v2_2", "v2_2")
                                   )
table_prop(df$variable_1)
table_prop(df$variable_1, df$variable_2)
table_prop(df$variable_1, df$variable_2, margin = 2, noquote = TRUE)
df <- data.frame("person_id" = c(1, 1, 2, 3,
                                 2, 4, 5, 5 ,1),
                  "person_age" = c(25, 25, 21, 32,
                                   21, 48, 50, 50, 52),
                  "survey_month" = c("jan", "feb", "mar", "apr",
                                     "apr", "may", "jun", "jul", "jan"),
                  "survey_answer" = c("no", "yes", "no", "yes",
                                      "yes", "yes", "no", "yes", NA))

table_prop(df$survey_month)
table_prop(df$survey_month, df$survey_answer)
table_prop(df$survey_month, df$survey_answer,
           margin = 2, round = 4)
```

---

vars_compclasses *Collection-level variables types comparison*

Description

Compare the class of each variable across a collection of datasets.

Usage

```
vars_compclasses(data_frames)
```

Arguments

data_frames	A named list of data frames to compare.

Value

A data frame with a `vars_union` column and one column per dataset showing the class of each variable (`"-"` if absent from that dataset).

Examples

```
data_list <- list(cars = cars, mtcars = mtcars)
vars_compclasses(data_list)
```

---

vars_compclasses_allsame

Variable class comparison - consistent types across all datasets

---

Description

Variable class comparison - consistent types across all datasets

Usage

```
vars_compclasses_allsame(vars_compclasses_table)
```

Arguments

vars_compclasses_table	Output of the `vars_compclasses()` function.

Value

A subset of `vars_compclasses_table` containing only variables with the same class across all datasets where they appear.

Examples

```
data_list <- list(cars = cars, mtcars = mtcars)
vcompclasses <- vars_compclasses(data_list)
vars_compclasses_allsame(vcompclasses)
```

---

vars_compclasses_not_allsame

Variable class comparison - inconsistent types across datasets

---

Description

Variable class comparison - inconsistent types across datasets

Usage

```
vars_compclasses_not_allsame(vars_compclasses_table)
```

Arguments

`vars_compclasses_table`
: Output of the `vars_compclasses()` function.

Value

A subset of `vars_compclasses_table` containing only variables with different classes across datasets where they appear.

Examples

```
data_list <- list(cars = cars, mtcars = mtcars)
vcompclasses <- vars_compclasses(data_list)
vars_compclasses_not_allsame(vcompclasses)
```

`vars_detect` *Variable detection patterns*

Description

Detect the presence or absence of each variable across a collection of datasets.

Usage

```
vars_detect(data_frames)
```

Arguments

`data_frames`
: A named list of data frames to compare.

Value

A data frame with a `vars_union` column listing all variables, and one column per dataset indicating `"ok"` (present) or `"-"` (absent). Rows are sorted to highlight presence/absence patterns.

Examples

```
data_list <- list(cars = cars, mtcars = mtcars)
vars_detect(data_list)
```

vars_detect_everywhere

Variable detection - presence across all datasets

Description

Variable detection - presence across all datasets

Usage

```
vars_detect_everywhere(vars_detect_table)
```

Arguments

vars_detect_table
: Output of the `vars_detect()` function.

Value

A subset of `vars_detect_table` containing only variables present in all datasets.

Examples

```
data_list <- list(cars = cars, mtcars = mtcars)
vdetect_table <- vars_detect(data_list)
vars_detect_everywhere(vdetect_table)
```

vars_detect_not_everywhere

Variable detection - inconsistent patterns

Description

Variable detection - inconsistent patterns

Usage

```
vars_detect_not_everywhere(vars_detect_table)
```

Arguments

vars_detect_table
: Output of the `vars_detect()` function.

Value

A subset of `vars_detect_table` containing only variables that are not present in every dataset.

Examples

```
data_list <- list(cars = cars, mtcars = mtcars)
vdetect_table <- vars_detect(data_list)
vars_detect_not_everywhere(vdetect_table)
```

Index