

Package ‘mintyr’

October 5, 2026

Title Grouped and Nested Data Pipelines Built on 'data.table'

Version 0.2.0

Description A toolkit for grouped and nested data pipelines built on 'data.table': import many Excel / CSV files (including multi-row headers) into one table, reshape and nest data by trait and group, run reproducible (stratified) k-fold cross-validation inside every group, summarise groups with report-ready descriptive statistics, and write each piece back to its own file or sheet. Developed for animal breeding, where it prepares phenotypic files for 'ASReml-R', 'HIBLUP' or 'DMU', but useful for any multi-group, multi-variable analysis.

License MIT + file LICENSE

URL <https://tony2015116.github.io/mintyr/>,
<https://github.com/tony2015116/mintyr>

BugReports <https://github.com/tony2015116/mintyr/issues>

Depends R (>= 4.1.0)

Imports data.table, parallel, readxl, stats, utils, writexl

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/fusen/version 0.7.2

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

Author Guo Meng [aut, cre],
Guo Meng [cph]

Maintainer Guo Meng <tony2015116@163.com>

Repository CRAN

Date/Publication 2026-10-05 16:30:10 UTC

Contents

c2p_nest	2
desc_stats	4
export_list	8
export_nest	10
export_xlsx	12
format_digits	14
get_path_info	16
import_csv	18
import_xlsx	20
mintyr_example	24
mintyr_examples	25
nest_cv	25
r2p_nest	27
split_cv	29
top_perc	31
w2l_nest	32
w2l_split	34
Index	37

c2p_nest	<i>Column to Pair Nested Transformation</i>
----------	---

Description

Generates combinations of specified columns and creates a nested data structure based on these pairs. Each nested subset renames the combined columns to value1, value2, ... (up to pairs_n) to support uniform iterative analyses such as genetic correlation estimation.

Usage

```
c2p_nest(data, cols, by = NULL, pairs_n = 2L, sep = "-", out_type = "dt")
```

Arguments

data	A data.frame or data.table to be transformed.
cols	A character vector of column names or a numeric vector of column indices to be combined into pairs. Must not overlap with by. Duplicated entries are removed with a warning.
by	A character vector of column names or a numeric vector of column indices to group by. Default is NULL.
pairs_n	A positive integer ≥ 2 indicating the size of each column combination (e.g., 2 for pairwise). Default is 2.
sep	A single character string used as a separator when constructing the pairs identifier column. Default is "-".

out_type A character string specifying the class of each nested object: "dt" (data.table, default) or "df" (data.frame).

Details

The columns specified in `cols` are renamed to `value1`, `value2`, ... within each nested subset. The original column names are preserved in the `pairs` column (e.g., "Sepal.Length-Sepal.Width"), ensuring full traceability for downstream iterative analyses such as genetic correlation estimation.

Columns that belong to neither `cols` nor `by` (referred to internally as "extra columns") are retained inside the nested subsets so that covariates or ID fields remain accessible. Grouping columns (`by`) are *not* duplicated inside the nested data because they are already present as outer key columns in the returned table.

When the number of requested combinations exceeds 500 a message is emitted; above 5000 a warning is raised, as memory usage grows linearly with the combination count.

The names `value1`, `value2`, ..., `pairs` and `data` are reserved: data must not contain other columns with these names. The input object is never modified.

Value

A data.table with columns:

pairs Character. The column-combination identifier, e.g. "Sepal.Length-Sepal.Width".

... Any by grouping columns, one per variable.

data List-column. Each cell holds a data.table (or data.frame when `out_type = "df"`) containing `value1`, `value2`, ..., plus any extra columns that were neither in `cols` nor by.

See Also

[combn](#) for the underlying combination generator.

Examples

```
# Example data preparation: Define column names for combination
col_names <- c("Sepal.Length", "Sepal.Width", "Petal.Length")

# Example 1: Basic column-to-pairs nesting with custom separator
c2p_nest(
  iris,                # Input iris dataset
  cols = col_names,    # Columns to be combined as pairs
  pairs_n = 2,         # Create pairs of 2 columns
  sep = "&"            # Custom separator for pair names
)
# Returns a nested data.table where:
# - pairs: combined column names (e.g., "Sepal.Length&Sepal.Width")
# - data: list column containing data.tables with value1, value2 columns

# Example 2: Column-to-pairs nesting with numeric indices and grouping
c2p_nest(
  iris,                # Input iris dataset
  cols = 1:3,          # First 3 columns to be combined
```

```

  pairs_n = 2,          # Create pairs of 2 columns
  by = 5                # Group by 5th column (Species)
)
# Returns a nested data.table where:
# - pairs: combined column names
# - Species: grouping variable
# - data: list column containing data.tables grouped by Species

```

desc_stats

Descriptive Statistics by Group, with an Optional Total Row

Description

desc_stats() computes descriptive statistics for numeric columns, overall or by group, in one tidy table (one row per group x variable, or one row per group in wide shape). The statistic sets follow rstatix::get_summary_stats(). An optional total row gives the statistics of every variable over all records; results can be returned as numbers or as report-ready text such as "26.66 ± 4.51".

Usage

```

desc_stats(
  data,
  cols = NULL,
  by = NULL,
  type = "common",
  stats = NULL,
  total = FALSE,
  total_label = "Total",
  min_n = 1L,
  probs = c(0, 0.25, 0.5, 0.75, 1),
  conf_level = 0.95,
  na_rm = TRUE,
  digits = NULL,
  fmt = NULL,
  labels = NULL,
  shape = "long",
  sep = "_",
  out_type = "dt"
)

```

Arguments

data	A data.frame or data.table. It is never modified.
cols	Numeric columns to summarise, as names or indices. Default NULL: every numeric column not in by.
by	Grouping column(s), as names or indices. Default NULL. Must not be named variable or value.

type	Preset set of statistics (ignored when stats or fmt is given): "full" n, min, max, median, q1, q3, iqr, mad, mean, sd, se, ci, cv, skew, kurt "common" (default) n, min, max, median, iqr, mean, sd, se, ci "robust" n, median, iqr "five_number" n, min, q1, median, q3, max "mean_sd", "mean_se" n, mean and sd / se "mean_ci" n, mean, ci_low, ci_high "median_iqr", "median_mad" n, median and iqr / mad "quantile" n and the quantiles given by probs "mean", "median" n and the mean / median
stats	Optional character vector of statistics, overriding type: any of "n", "n_miss", "min", "max", "mean", "median", "q1", "q3", "iqr", "mad", "sd", "se", "ci", "ci_low", "ci_high", "cv", "skew", "kurt" and "quantile", in the order wanted.
total	Logical. If TRUE, add totals following one rule: across groups the statistics are recomputed on all records, across variables only counts are added up (different variables are not one quantity). • Total row (with by): an extra group whose by columns hold total_label; every variable is summarised over all records, so its n is the sum of the group n. • Total variable / column (with two or more variables, when n or n_miss is requested or a fmt template uses only counts): an extra variable total_label, last, whose n / n_miss are the sums over the variables in each row; its other statistics are NA, and in wide shape its empty columns are dropped. A count-only wide table thus gets row and column totals, as <code>janitor::adorn_totals()</code>. Without by only the total variable is added. Default FALSE.
total_label	Label of the total row and the total variable. Default "Total". Must not already occur in a by column or as a variable name (or label).
min_n	Minimum number of (non-missing) values a group needs for its statistics to be reported. Groups with fewer values keep n / n_miss but all other statistics are NA, so that e.g. the SD of a farm with two animals is not mistaken for a reliable figure. Default 1.
probs	Probabilities for "quantile"; output columns are named q0, q25, q2.5, ... Default <code>c(0, 0.25, 0.5, 0.75, 1)</code>.
conf_level	Confidence level of ci. Default 0.95.
na_rm	Logical. If TRUE (default) missing values are removed before computing statistics, and n counts the non-missing values. If FALSE, any missing value makes the statistics NA and n counts all values.
digits	NULL (default, no rounding), one non-negative integer for all variables, or a named vector per variable, optionally with one unnamed default: <code>c(2, adg = 0, bf = 1)</code> rounds adg to 0, bf to 1 and every other variable to 2 decimals (variables without a value are not rounded). The counts n / n_miss are integers. With fmt, these are the decimals of placeholders without their own {stat:d} (default 2).

fmt	NULL (default) or a character vector of templates that turn the statistics into report-ready text, e.g. "{mean} ± {sd}". Each template becomes one text column, replacing the numeric statistics; name the elements to name the columns (default: the template without braces, e.g. "mean ± sd"). Placeholders: • {stat} — any statistic listed in stats (except "quantile"), with digits decimals (counts without decimals). • {stat:d} — with d decimals, e.g. {mean:1}. • {stat:d\%} — multiplied by 100 and followed by %, e.g. {cv:1\%}. • {q2.5}, {q97.5}, ... — percentiles ({q1} / {q3} are the quartiles). The statistics needed by the templates are computed automatically; type, stats and probs are then ignored. Missing values are shown as "NA" (e.g. the SD of a single record).
labels	NULL (default) or a named character vector of display names for the variables, e.g. c(adg = "ADG (g)", bf = "Backfat (mm)"). Applied to the variable column (long shape) or the column names (wide shape). Variables without a label keep their name.
shape	"long" (default): one row per group x variable, one column per statistic. "wide": one row per group (total row last), columns <variable><sep><statistic> in variable order, as in a typical report table. With a single statistic or template (e.g. stats = "mean", fmt = "{mean} ± {sd}") the columns are simply named after the variables. Both shapes show the same numbers.
sep	Separator between variable and statistic in wide column names. Default "_".
out_type	"dt" (default) for a data.table, "df" for a data.frame.

Details

Definitions: q1 / q3 and quantile use stats::quantile() (type 7), iqr = q3 - q1, mad is stats::mad() (scaled by 1.4826), se = sd / sqrt(n), ci is the half-width of the t-based confidence interval of the mean (qt((1 + conf_level) / 2, n - 1) * se), and ci_low / ci_high are its bounds (mean -/+ ci), cv = sd / mean as a ratio (shown in percent with fmt = "{cv:1\%}"), skew is the adjusted Fisher-Pearson skewness G1 and kurt the excess kurtosis G2, as in SAS, SPSS and Excel's SKEW() / KURT() (NA with fewer than 3 / 4 values). The CV is only meaningful for strictly positive data and is NA when a group contains values <= 0. Statistics that are undefined for a group (e.g. sd with one value) are NA.

Computation: statistics are computed column by column on the input table (no reshaping to long format, so memory use stays close to the input size). n, mean, sd, min, max and median use data.table's optimised grouped functions, quantiles are read off once-sorted groups, so tens of thousands of groups (sires, litters, pens) are summarised in about a second per million records.

Rows are ordered by variable (in the order of cols), then by group (in the original order of the by values: numeric order, factor levels, alphabetical for text), with the total row last. With a total row, by columns that are not factors or character are returned as character; factors gain the label as their last level.

Value

A data.table (or data.frame). Long shape: the by columns, variable and one column per statistic (or per fmt template, as text). Wide shape: the by columns followed by one column per

variable x statistic (or template).

See Also

[top_perc\(\)](#) for statistics of the top / bottom X% per group.

Examples

```
# Example 1: Common statistics for every numeric column of iris
desc_stats(iris)

# Example 2: Mean and SD by group, with a total row over all records
desc_stats(
  iris,
  by = "Species",           # Grouping column
  type = "mean_sd",        # Preset: n, mean, sd
  total = TRUE,            # n = sum of the groups, mean / sd of all records
  digits = 2               # Round the statistics
)

# Example 3: Count table with row and column totals
# (counts are the only statistic that adds up across variables)
desc_stats(
  iris,
  by = "Species",
  stats = "n",
  total = TRUE,            # Total row and Total column
  shape = "wide"
)

# Without groups the total adds up the counts of all variables
desc_stats(iris, type = "mean_sd", total = TRUE, digits = 2)

# Example 3b: Report table - groups in rows, traits in columns, "mean ± sd"
desc_stats(
  mtcars,
  cols = c("mpg", "hp", "wt"),
  by = "cyl",
  fmt = "{mean} ± {sd}",    # Report-ready text
  total = TRUE,
  shape = "wide"           # One row per group, one column per trait
)

# Example 4: Several templates, per-placeholder decimals, CV in percent and
# percentiles
desc_stats(
  mtcars,
  cols = c("mpg", "wt"),
  by = "am",
  fmt = c(N = "{n}",
    "Mean ± SD" = "{mean:1} ± {sd:1}",
    "CV" = "{cv:1%}",
    "Median [P2.5, P97.5]" = "{median:1} [{q2.5:1}, {q97.5:1}]",
  )
)
```

```
total = TRUE
)

# Example 5: Quantiles, returned as a data.frame
desc_stats(iris, cols = 1:2, type = "quantile",
           probs = c(0.05, 0.5, 0.95), out_type = "df")
```

export_list	<i>Export a List of Data Frames with Hierarchical Directory Management</i>
-------------	--

Description

Exports every element of a named (or unnamed) list of `data.frame` / `data.table` objects to `txt` or `csv` files. Element names may contain forward-slashes (/) to encode arbitrary subdirectory depth, e.g. `"group_a/subject_01/results"` writes `<path>/group_a/subject_01/results.txt`. Un-named elements are automatically labelled `split_<i>`.

Usage

```
export_list(
  data,
  path = tempdir(),
  file_type = "txt",
  na = "NA",
  quote = FALSE,
  ...
)
```

Arguments

data	A non-empty list whose elements are <code>data.frame</code> , <code>data.table</code> , or any object coercible via <code>data.table::as.data.table()</code> .
path	Single character string - the root export directory. Created recursively if absent. Defaults to <code>tempdir()</code> .
file_type	<code>"txt"</code> (tab-separated, default) or <code>"csv"</code> (comma-separated). Case-insensitive.
na	Single string written for missing values. Default <code>"NA"</code> . Use e.g. <code>"-9999"</code> for DMU.
quote	Passed to <code>fwrite</code> . Default <code>FALSE</code> : with a non-empty <code>na</code> , <code>fwrite</code> 's <code>"auto"</code> would quote every header and character field, which command-line breeding programs (HIBLUP, DMU, ...) cannot parse. Use <code>"auto"</code> if values may contain the separator.
...	Further arguments passed to <code>fwrite</code> (e.g. <code>col.names = FALSE</code>).

Details

Performance design:

- All element names are resolved and path components split in a single vectorised pass *before* the write loop, so no string work occurs inside the hot path.
- Unique subdirectories are collected and created in one batch (`k dir.create()` syscalls, where $k \leq n$).
- The field separator is resolved once at function entry.
- `as.data.table()` on an existing `data.table` is a reference-pass (no copy).

Name handling: each `/`-separated component of an element name is sanitised separately (invalid characters become `_`; `..` cannot escape path). If two elements map to the same file (compared case-insensitively), the function stops instead of overwriting.

Error handling: Individual element failures emit a warning and are skipped; the remaining elements continue to be processed.

Value

An invisible named character vector of the file paths written, with length equal to the number of successfully exported elements. The total count is accessible via `length()` on the return value.

See Also

[fwrite](#)

Examples

```
# Example: Export split data to files
out_dir <- file.path(tempdir(), "mintyr_export_list")

# Step 1: Create split data structure
dt_split <- w2l_split(
  data = iris,           # Input iris dataset
  cols = 1:2,           # Columns to be split
  by = "Species"        # Grouping variable
)

# Step 2: Export split data to files
files <- export_list(
  data = dt_split,      # Input list of data.tables
  path = out_dir
)

# Returns (invisibly) a named vector of the written file paths
files

# Clean up
unlink(out_dir, recursive = TRUE)
```

export_nest	<i>Export Nested Data Structures with Hierarchical Directory Organization</i>
-------------	---

Description

Exports list-columns containing `data.frame` or `data.table` objects from a `data.frame/data.table` to `txt` or `csv` files, automatically constructing a hierarchical directory structure from non-nested columns. Exportable nested columns (those holding `data.frame/data.table` elements) are distinguished from non-exportable custom-object columns (e.g. fitted model objects); only the former are written to disk by default.

Usage

```
export_nest(
  data,
  by = NULL,
  cols = NULL,
  path = tempdir(),
  file_type = "txt",
  na = "NA",
  quote = FALSE,
  ...
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> containing at least one nested list-column. Must have one or more rows.
<code>by</code>	Optional character vector of column names used to build the hierarchical output directory structure. When <code>NULL</code> (default), all non-nested columns are used automatically.
<code>cols</code>	Optional character vector of nested column names to export. When <code>NULL</code> (default), all columns whose elements are <code>data.frame/data.table</code> objects are exported automatically; custom-object list-columns are reported and skipped. Specifying a non- <code>data.frame</code> column triggers a warning and that column is skipped.
<code>path</code>	Single character string specifying the root export directory. Defaults to <code>tempdir()</code> . Created recursively if it does not exist.
<code>file_type</code>	Either <code>"txt"</code> (tab-separated, default) or <code>"csv"</code> (comma-separated). Case-insensitive.
<code>na</code>	Single string written for missing values. Default <code>"NA"</code> (<code>fwrite</code> 's own default, an empty field, breaks whitespace-delimited readers). Use e.g. <code>"-9999"</code> for DMU.
<code>quote</code>	Passed to <code>fwrite</code> . Default <code>FALSE</code> : with a non-empty <code>na</code> , <code>fwrite</code> 's <code>"auto"</code> would quote every header and character field, which command-line breeding programs (HIBLUP, DMU, ...) cannot parse. Use <code>"auto"</code> if values may contain the separator.

... Further arguments passed to `fwrite` (e.g. `col.names = FALSE` for software that expects no header).

Details

Nested column classification (mutually exclusive):

- *Exportable* — every element inherits from `data.frame` or `data.table`.
- *Non-exportable* — empty lists or elements of any other class (e.g. model objects). Reported to the console; never written.

Directory layout: `path / <group1_value> / <group2_value> / <nest_col_name>.<file_type>`

Group values are sanitised (characters that are invalid in file names, such as slashes, colons or asterisks, become underscores), so they can never create extra directory levels. The grouping columns must identify every row uniquely; otherwise rows would overwrite each other's files and the function stops with an error listing the duplicated targets.

Performance notes:

- Row data is accessed via `.subset2()` (zero-copy column access) rather than `data[i]`, eliminating per-row `data.table` allocation in the hot loop.
- All output directory paths are pre-computed in a single vectorised `do.call(file.path, ...)` call; only the unique directories are created.
- The field separator and output filenames are computed once before the loop.

Value

An invisible character vector of the files written (length 0 when nothing was written).

See Also

`fwrite`

Examples

```
# Example: Basic nested data export workflow
# A dedicated sub-folder of tempdir() keeps the clean-up safe
out_dir <- file.path(tempdir(), "mintyr_export_nest")

# Step 1: Create nested data structure
dt_nest <- w2l_nest(
  data = iris,           # Input iris dataset
  cols = 1:2,           # Columns to be nested
  by = "Species"        # Grouping variable
)

# Step 2: Export nested data to files
files <- export_nest(
  data = dt_nest,        # Input nested data.table
  cols = "data",         # Column containing nested data
  by = c("name", "Species"), # Columns to create directory structure
```

```

    path = out_dir
  )
  # Returns (invisibly) the paths of the written files
  # Directory structure: out_dir/<name>/<Species>/data.txt
  files

  # Clean up
  unlink(out_dir, recursive = TRUE)

```

export_xlsx

Export Data to XLSX Files

Description

The natural complement to `import_xlsx()`. Accepts either a combined `data.frame` (as produced by `import_xlsx()` with `combine = TRUE`) **or a list of `data.frames`**, and writes the result to disk.

The output **destination** is controlled by a single path argument – there are no separate modes to choose:

- path **ends in** `.xlsx` – write *everything into a single workbook*: one sheet per list element, or (for `data.frame` input) one sheet per `file_col` / `sheet_col` value.
- path **is anything else** – treat it as a *directory* and write *one .xlsx file per list element* or per `file_col` value. For `data.frame` input that also has a `sheet_col` column, each file contains one sheet per `sheet_col` value, which reproduces the original file/sheet layout read by `import_xlsx()`.

List input:

```
res <- list(res1 = data1, res2 = data2, res3 = data3)
```

```

# Directory mode -- writes res1.xlsx, res2.xlsx, res3.xlsx
export_xlsx(res, path = "output/")

```

```

# Single-file mode -- one workbook with sheets res1, res2, res3
export_xlsx(res, path = "output/all.xlsx")

```

Each element must be a `data.frame`, `data.table`, or `tibble`; types may be mixed and column sets may differ. Missing names are filled in as `Sheet1`, `Sheet2`, ...; supplied names must be unique.

Usage

```

export_xlsx(
  data,
  path,
  file_col = "excel_name",
  sheet_col = "sheet_name",
  sheet_name = "Sheet1",

```

```

    drop_cols = TRUE,
    overwrite = TRUE,
    verbose = FALSE
  )

```

Arguments

data	A data.frame / data.table / tibble, or a list of such objects. For list input, names become file names (directory mode) or sheet names (single-file mode).
path	character(1). Output destination. A path ending in .xlsx (case-insensitive) is a single workbook ; anything else is an output directory . Missing directories are created recursively.
file_col	character(1). Column identifying the source file (data.frame input only). Default "excel_name".
sheet_col	character(1). Column identifying the source sheet (data.frame input only). Default "sheet_name".
sheet_name	character(1). Sheet name used when a data.frame has neither tracking column. Default "Sheet1".
drop_cols	logical(1). If TRUE (default), the tracking columns are removed from the exported sheets (data.frame input only).
overwrite	logical(1). Allow overwriting existing files. When FALSE, all targets are checked <i>before</i> anything is written, so a failure never leaves a partial export behind. Default TRUE.
verbose	logical(1). Print a message for every file and sheet written. Default FALSE.

Details

Why writexl?: **writexl** writes .xlsx via a minimal C library with no Java or Perl dependency. It is fast and produces small files, at the cost of no cell formatting, formulas, or styles. For those, use **openxlsx2**.

Name sanitisation: Sheet names are limited to 31 characters, may not contain [] * ? / \ : , and may not start or end with an apostrophe. File names have characters that are invalid on Windows replaced by _. If sanitising or truncating makes two names collide (Excel compares sheet names case-insensitively), suffixes such as _2, _3 are appended.

Missing group values: Rows with NA in file_col or sheet_col are not dropped: they are exported under the label "NA" and a warning is issued.

Directory vs. file dispatch: path is classified purely by its extension. To use a directory whose name ends in .xlsx, append a trailing slash.

Value

Invisibly, a named character vector of written file paths. In directory mode it is named by list element / file_col value; in single-workbook mode it is named by path.

Examples

```
# Example 1: A plain data.frame -> one workbook, one sheet
out_file <- file.path(tempdir(), "mtcars.xlsx")
export_xlsx(mtcars, path = out_file, sheet_name = "mtcars")
invisible(file.remove(out_file))

# Example 2: data.table input works exactly the same way
out_file <- file.path(tempdir(), "mtcars.dt.xlsx")
export_xlsx(data.table::as.data.table(mtcars),
             path = out_file, sheet_name = "mtcars")
invisible(file.remove(out_file))

# Example 3: One sheet per group in a single workbook
# Each Species value becomes a sheet; keep the Species column
out_file <- file.path(tempdir(), "iris_by_species.xlsx")
export_xlsx(iris, path = out_file, file_col = "Species", drop_cols = FALSE)
invisible(file.remove(out_file))

# Example 4: One file per group (directory mode: no .xlsx extension)
out_dir <- file.path(tempdir(), "iris_by_species")
out_files <- export_xlsx(iris, path = out_dir, file_col = "Species")
basename(out_files)
unlink(out_dir, recursive = TRUE)

# Example 5: Round-trip the layout produced by import_xlsx(combine = TRUE)
# Rows are routed back to their original file and sheet
combined <- data.frame(
  excel_name = c("sales", "sales", "costs"),
  sheet_name = c("2024", "2025", "2024"),
  amount     = c(100, 120, 80)
)
out_dir <- file.path(tempdir(), "roundtrip")
out_files <- export_xlsx(combined, path = out_dir) # sales.xlsx, costs.xlsx
basename(out_files)
unlink(out_dir, recursive = TRUE)

# Example 6: Named list -> single workbook, one sheet per element
out_file <- file.path(tempdir(), "combined.xlsx")
res <- list(res1 = iris, res2 = mtcars)
export_xlsx(res, path = out_file)
invisible(file.remove(out_file))

# Example 7: Named list -> directory, one file per element
out_dir <- file.path(tempdir(), "combined")
out_files <- export_xlsx(res, path = out_dir) # res1.xlsx, res2.xlsx
basename(out_files)
unlink(out_dir, recursive = TRUE)
```

Description

Format Numeric Columns to Fixed-Decimal Character Strings

Usage

```
format_digits(
  data,
  cols = NULL,
  digits = 2L,
  percentage = FALSE,
  nan_as_na = FALSE
)
```

Arguments

data	A data.frame or data.table. The input dataset.
cols	A character or integer vector specifying columns to format. If NULL (default), all numeric columns are formatted.
digits	A non-negative integer specifying decimal places. Defaults to 2.
percentage	Logical. If TRUE, values are multiplied by 100 and a "%" sign is appended. Defaults to FALSE.
nan_as_na	Logical. If TRUE, NaN is treated identically to NA and coerced to NA_character_. If FALSE (default), NaN is preserved as the string "NaN".

Details

The function processes columns in the following order:

1. Validates all input parameters with informative error messages.
2. Copies the input only once: data.table inputs are deep-copied via copy(); data.frame inputs are copied implicitly by as.data.table(), avoiding a redundant second copy.
3. Resolves cols to a character vector of valid numeric column names, warning and skipping any non-numeric columns specified.
4. Applies a vectorised formatting function via lapply(.SD, fn) and :=, so all target columns are dispatched in a single data.table call rather than a column-by-column loop.

NA and NaN handling:

- NA_real_ is always returned as NA_character_.
- NaN is returned as "NaN" by default. Set nan_as_na = TRUE to coerce it to NA_character_ instead.

Rounding uses explicit round() before sprintf() to guarantee consistent results across platforms (Windows, Linux, macOS), where the underlying C library's rounding behaviour may otherwise differ.

Value

A data.table with the specified numeric columns formatted as character strings. The original object is never modified.

Note

- data must be a data.frame or data.table.
- Integer column indices in cols are converted to column names internally; duplicates are silently removed.
- digits accepts numeric values such as 2.0 and coerces them to integer; non-integer-valued numbers raise an error.
- The function depends only on data.table and base R.

Examples

```
# Example: Number formatting demonstrations

# Setup test data
dt <- data.table::data.table(
  a = c(0.1234, 0.5678),      # Numeric column 1
  b = c(0.2345, 0.6789),      # Numeric column 2
  c = c("text1", "text2")    # Text column
)

# Example 1: Format all numeric columns
format_digits(
  dt,                          # Input data table
  digits = 2                   # Round to 2 decimal places
)

# Example 2: Format specific column as percentage
format_digits(
  dt,                          # Input data table
  cols = c("a"),               # Only format column 'a'
  digits = 2,                  # Round to 2 decimal places
  percentage = TRUE            # Convert to percentage
)
```

get_path_info

Extract Path Segments or Filenames from File Paths

Description

get_path_info is a merged, upgraded replacement for get_path_segment and get_filename. It operates in two modes:

- **Mode A (when n is specified):** Extract a specific path segment by position, supporting forward indexing, reverse indexing, and range extraction.
- **Mode B (when n = NULL):** Extract the filename, with optional removal of the file extension and/or the directory prefix.

Usage

```
get_path_info(path, n = NULL, rm_extension = TRUE, rm_path = TRUE)
```

Arguments

path	A character vector of file system paths. Supports mixed separators (/ and \\) and Windows drive letters (e.g. C:).
n	A numeric segment index. Defaults to NULL (enters filename mode). • Positive integer: forward index from the path start; 1 = first segment. • Negative integer: reverse index from the path end; -1 = last segment (i.e. the filename segment). • Length-2 vector: extract a contiguous range, e.g. c(2, 4) or c(-3, -1). • 0 is not allowed.
rm_extension	A logical(1) flag controlling extension removal. Defaults to TRUE. • In Mode B (n = NULL): always applied. • In Mode A: only applied when n == -1 (explicitly targeting the filename segment). Has no effect for intermediate directory segments (e.g. n = 2).
rm_path	A logical(1) flag controlling whether the directory prefix is stripped, keeping only the filename. Defaults to TRUE. Only applies in Mode B (n = NULL); ignored when n is specified. With rm_path = FALSE the normalised path keeps its root (/, C:/ or // for UNC paths), so absolute paths stay absolute.

Details**Path normalisation (internal, fully vectorised):**

1. All backslashes and consecutive slashes are collapsed to a single /.
2. Windows drive letter prefixes (C:, D:, etc.) are stripped for segment indexing (and restored in full-path mode).
3. Leading and trailing / characters are removed.
4. Paths that are empty after the above steps (e.g. original inputs "C:/", "/", "") are coerced to NA_character_.

Extension-stripping behaviour (internal .strip_ext helper):

Input	Output	Notes
"report.txt"	"report"	Standard file — last extension removed
"data.tar.gz"	"data.tar"	Compound extension — only last level removed
".bashrc"	".bashrc"	Pure dot-file (no second dot) — unchanged
".report.xlsx"	".report"	Dot-file with extension — extension removed
"no_ext"	"no_ext"	No extension — returned as-is
"file."	"file."	Trailing isolated dot — returned as-is

NA safety: strsplit(NA_character_, ...) returns list(NA) with length 1, not character(0). Consequently, every vapply callback guards against NA paths with an explicit anyNA(x) check rather than length(x) == 0.

Value

A character vector of the same length as path:

- Returns the extracted segment string when the segment exists.
- Returns `NA_character_` when the segment index exceeds the path depth, the input element is `NA`, or the path reduces to empty after normalisation (e.g. `"C:/", "/"`).

See Also

`base::basename()`, `tools::file_path_sans_ext()`

Examples

```
paths <- c("C:/Users/foo/Documents/report.xlsx",
          "/home/user/.bashrc",
          "relative/path/to/data.csv",
          ".hidden.tar.gz",
          NA_character_)

# Mode B: filename only, extension stripped (default)
get_path_info(paths)

# Mode B: filename only, extension preserved
get_path_info(paths, rm_extension = FALSE)

# Mode B: full normalised path, extension stripped
get_path_info(paths, rm_path = FALSE)

# Mode A: extract the 2nd path segment
get_path_info(paths, n = 2)

# Mode A: extract the last segment with extension stripped (n = -1 linkage)
get_path_info(paths, n = -1, rm_extension = TRUE)

# Mode A: range extraction
get_path_info(paths, n = c(2, 3))
```

import_csv

Flexible CSV/TXT File Import via data.table

Description

Reads one or more CSV/TXT files using `fread` as the backend. Supports flexible combination strategies and source-file tracking. All return values are `data.table` objects.

Usage

```
import_csv(
  file,
  combine = TRUE,
  file_col = "_file",
  full_path = FALSE,
  keep_ext = FALSE,
  ...
)
```

Arguments

<code>file</code>	A non-empty character vector of file paths to CSV/TXT files. All paths must point to existing, accessible files.
<code>combine</code>	A logical scalar controlling the combination strategy: • <code>TRUE</code> (default): Combine all files into a single <code>data.table</code>. • <code>FALSE</code>: Return a named list of individual <code>data.table</code> objects.
<code>file_col</code>	A character scalar or <code>NULL</code> specifying the source-tracking column name (default: <code>"_file"</code>). Set to <code>NULL</code> to suppress the source column. Only applies when <code>combine = TRUE</code> .
<code>full_path</code>	A logical scalar controlling path representation in labels: • <code>FALSE</code> (default): Use only the filename (via <code>basename()</code>). • <code>TRUE</code>: Use the full file path.
<code>keep_ext</code>	A logical scalar controlling whether the file extension is retained in labels: • <code>FALSE</code> (default): Strip the file extension (e.g., <code>"data"</code>). • <code>TRUE</code>: Retain the file extension (e.g., <code>"data.csv"</code>).
<code>...</code>	Additional arguments passed directly to <code>fread</code> (e.g., <code>select</code> , <code>drop</code> , <code>na.strings</code> , <code>skip</code> , <code>nThread</code>).

Details

Label generation is controlled by the combination of `full_path` and `keep_ext`:

<code>full_path = FALSE, keep_ext = FALSE</code>	Filename without extension: <code>"data"</code>
<code>full_path = FALSE, keep_ext = TRUE</code>	Filename with extension: <code>"data.csv"</code>
<code>full_path = TRUE, keep_ext = FALSE</code>	Full path without extension: <code>"/path/to/data"</code>
<code>full_path = TRUE, keep_ext = TRUE</code>	Full path with extension: <code>"/path/to/data.csv"</code>

When `combine = TRUE` and `file_col` is not `NULL`, `rbindlist` is called with `idcol = file_col`, which generates the source column directly during the merge step without any intermediate copies.

Value

- `combine = TRUE`: A single `data.table` containing all imported rows. If `file_col` is not `NULL`, the first column contains the source file label for each row.
- `combine = FALSE`: A named list of `data.table` objects. List names are derived from file paths according to `full_path` and `keep_ext` settings.

Note

- All specified files must exist and be readable at call time.
- `combine = TRUE` assumes compatible column structures across files; mismatched columns are automatically aligned via `fill = TRUE`.
- Logical parameters (`combine`, `full_path`, `keep_ext`) reject `NA` values explicitly.
- Only the extension of the file itself is stripped; dots in directory names (e.g. `"v1.2/file"`) are left intact.
- Duplicated labels (same file name in different folders) trigger a warning and are made unique with `make.unique()`.

See Also

[fread](#), [rbindlist](#)

Examples

```
# Example: CSV file import demonstrations

# Setup test files
csv_files <- mintyr_example(
  mintyr_examples("csv_test")    # Get example CSV files
)

# Example 1: Import and combine CSV files using data.table
import_csv(
  csv_files,                    # Input CSV file paths
  combine = TRUE,               # Combine all files into one data.table
  file_col = "_file",          # Column name for file source
  keep_ext = TRUE,              # Include .csv extension in _file column
  full_path = TRUE              # Show complete file paths in _file column
)
```

Description

A high-performance function for importing data from one or multiple Excel files into `data.table` format, with fine-grained control over source tracking columns, sheet selection, row skipping, multi-row headers, and optional parallel reading across (file, sheet) pairs.

Performance characteristics:

- `excel_sheets()` called exactly once per file (cached).
- The real cost of an Excel import is `read_excel()` parsing, which is single-threaded C++ and *not* affected by the `data.table` thread pool. The flat (file x sheet) task list is therefore read in parallel **across processes** when `workers > 1`: a fork pool on Unix/macOS, a PSOCK cluster on Windows. The cluster / fork pool is always torn down on exit.
- `setDT()` converts tibbles in-place — zero vector copies.
- Tracking columns injected via `:=` on small per-sheet tables *before* the single final `rbindlist`.
- The final `rbindlist` uses the ambient `data.table` thread pool; tune it globally with [setDTthreads](#) if needed. Workers are pinned to a single thread during the read phase so parallel processes do not oversubscribe cores.

Usage

```
import_xlsx(
  file,
  combine = TRUE,
  sheet = NULL,
  skip = 0L,
  header_rows = 1L,
  header_sep = "_",
  header_fill = "auto",
  file_col = "excel_name",
  sheet_col = "sheet_name",
  workers = 1L,
  verbose = FALSE,
  on_error = "stop",
  ...
)
```

Arguments

<code>file</code>	Non-empty character vector of paths to existing <code>.xlsx</code> / <code>.xls</code> files.
<code>combine</code>	logical(1). TRUE (default) binds all sheets into a single <code>data.table</code> . FALSE returns a flat named list keyed as " <code><filename>_<sheetname></code> ".
<code>sheet</code>	Positive integer vector, character vector of sheet names, or NULL (default, every sheet). Indices / names must exist in all supplied files.
<code>skip</code>	Non-negative integer(1). Number of rows to skip before the header (e.g. a title row). Default 0L.
<code>header_rows</code>	Positive integer(1). Number of header rows below the skipped rows. Default 1L (ordinary sheets). Values > 1 combine the rows into one name per column (see Details); data start on the next row.

header_sep	character(1). Separator used to join the levels of a multi-row header. Default "_".
header_fill	How merged header cells are resolved when header_rows > 1: "auto" (default) reads the merged ranges stored in .xlsx/.xlsm files and falls back to "right" for .xls; "right" fills upper levels to the right (heuristic, no merge information needed); "none" uses the cells as they are.
file_col	character(1) or NULL. Name of the column recording the source file (extension stripped) when combine = TRUE. Default "excel_name"; NULL omits the column. Ignored when combine = FALSE (provenance is in the list names).
sheet_col	character(1) or NULL. Name of the column recording the source sheet when combine = TRUE. Default "sheet_name"; NULL omits it. The defaults match export_xlsx(), so the result can be written back to the original layout.
workers	integer(1). Number of parallel processes used to read the (file x sheet) tasks. Default 1L (serial). Values > 1 open a fork pool on Unix/macOS or a PSOCK cluster on Windows; the pool is capped at the number of tasks and always shut down on exit. Parallel reading pays off when files are large or numerous; for many tiny sheets the process / serialization overhead can dominate, so leave this at 1L unless the import is heavy.
verbose	logical(1). When TRUE, print one message() per sheet (source file, sheet name, and row x col dimensions, with empty sheets flagged) followed by a summary line. Default FALSE. Output is emitted from the master process, so it surfaces identically in serial and parallel modes.
on_error	"stop" (default) or "warn". With "stop", any file or sheet that cannot be read aborts the import with one error listing <i>all</i> failures. With "warn", failed files/sheets are skipped and listed in a single warning; the import only fails when nothing could be read.
...	Additional arguments forwarded to read_excel (e.g. col_types, na, trim_ws). Do not pass path, sheet, or skip here; use the dedicated parameters above. With header_rows > 1, col_names and range are set internally and must not be passed either; col_types then refers to the data columns.

Details

Multi-row headers. Many hand-made workbooks have a title row and a header spread over several rows with merged cells, e.g.

```
row 1 Growth test records
row 2 ID | Breed | Weight (kg) | Backfat (mm)
row 3   |       | Start | End   | P2 | Loin
row 4 A001 | Duroc | 28.5 | 102.3 | 11.2 | 56.1
```

import_xlsx(f, skip = 1, header_rows = 2) returns the columns ID, Breed, Weight (kg)_Start, Weight (kg)_End, Backfat (mm)_P2, Backfat (mm)_Loin. The rules are:

- Merged cells only store their value in their top-left cell. With header_fill = "auto" the merged ranges are read from the workbook itself (<mergeCell> entries of the sheet XML), and each range is filled with its value, horizontally and vertically. Header cells that are genuinely empty stay empty, so a column whose upper cell is blank gets only its lower label.

- Without merge information (.xls files, or header_fill = "right"), every level except the last is filled to the right, restarting wherever a higher level starts a new label.
- Per column, empty parts and repeated parts (vertically merged cells) are dropped and the remaining parts joined with header_sep; line breaks inside a cell become spaces.
- Columns without any header become col_<j>; duplicates are made unique.
- The header block is read as text, but the data block is read separately, so numbers and dates keep their types (reading the whole sheet as text would turn dates into serial numbers such as "45296").
- Completely empty data rows (spacer or trailing rows) are dropped.

Labels that are only *visually* spread over several columns with "Center Across Selection" (instead of real merging) are not merges; use header_fill = "right" for such sheets. Reading the merge information re-scans the sheet XML once, which costs a fraction of the import time and only happens when header_rows > 1.

Value

combine = TRUE A data.table. Tracking columns excel_name and/or sheet_name are prepended when their respective file_col / sheet_col are not NULL.

combine = FALSE A named list of data.tables, each element named "<filename>_<sheetname>". The list carries a "source_files" attribute with the original file paths.

Note

Files that share the same base name in different folders cannot be told apart by file_col; a warning is issued and the labels are made unique, so a later export_xlsx() round trip does not merge them.

Examples

```
# Example: Excel file import demonstrations

# Setup test files
xlsx_files <- mintyr_example(
  mintyr_examples("xlsx_test") # Get example Excel files
)

# Example 1: Import and combine all sheets from all files
import_xlsx(
  xlsx_files, # Input Excel file paths
  combine = TRUE # Combine all sheets into one data.table
)

# Example 2: Import specific sheets separately
import_xlsx(
  xlsx_files, # Input Excel file paths
  combine = FALSE, # Keep sheets as separate data.tables
  sheet = 2 # Only import the second sheet
)

# Example 3: Multi-row header with merged cells and a title row
```

```
# (row 1 = title, rows 2-3 = header, data from row 4)
mh_file <- mintyr_example("multiheader_test.xlsx")
import_xlsx(
  mh_file,
  skip = 1,                      # Skip the title row
  header_rows = 2                # Combine two header rows into one name
)
# The last column has an empty, unmerged upper cell: it is named "Note".
# The heuristic fill would wrongly call it "Backfat (mm)_Note":
names(import_xlsx(mh_file, sheet = 1, skip = 1, header_rows = 2,
  header_fill = "right"))

# Example 4: Batch import that skips unreadable files instead of stopping
bad_file <- tempfile(fileext = ".xlsx")
writeLines("not a workbook", bad_file)
res <- suppressWarnings(
  import_xlsx(c(mh_file, bad_file), skip = 1, header_rows = 2, on_error = "warn")
)
unique(res$excel_name)
unlink(bad_file)
```

mintyr_example

Get path to mintyr examples

Description

mintyr comes bundled with a number of sample files in its inst/extdata directory. Use `mintyr_example()` to retrieve the full file path to a specific example file.

Usage

```
mintyr_example(path = NULL)
```

Arguments

path	Name of the example file to locate. If NULL or missing, returns the directory path containing the examples.
------	---

Value

Character string containing the full path to the requested example file.

See Also

[mintyr_examples\(\)](#) to list all available example files

Examples

```
# Get path to an example file
mintyr_example("csv_test1.csv")
```

mintyr_examples	<i>List all available example files in mintyr package</i>
-----------------	---

Description

mintyr comes bundled with a number of sample files in its `inst/extdata` directory. This function lists all available example files, optionally filtered by a pattern.

Usage

```
mintyr_examples(pattern = NULL)
```

Arguments

pattern	A regular expression to filter filenames. If <code>NULL</code> (default), all available files are returned.
---------	---

Value

A character vector containing the names of example files. If no files match the pattern or if the example directory is empty, returns a zero-length character vector.

See Also

[mintyr_example\(\)](#) to get the full path of a specific example file

Examples

```
# List all example files
mintyr_examples()
```

nest_cv	<i>Apply V-Fold Cross-Validation to Nested Data</i>
---------	---

Description

`nest_cv` creates v-fold cross-validation splits for each nested data frame of a `nested_data.table` (e.g. the output of [w2l_nest\(\)](#)) and returns one row per fold, with the outer (non-nested) columns broadcast to every fold. The cross-validation itself is performed by [split_cv\(\)](#).

Usage

```
nest_cv(
  data,
  v = 10L,
  repeats = 1L,
  strata = NULL,
  breaks = 4L,
  pool = 0.1,
  seed = NULL,
  materialize = TRUE,
  out_type = "dt"
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> containing at least one nested <code>data.frame</code> / <code>data.table</code> column. A column named <code>data</code> is used if present, otherwise the first nested column.
<code>v</code>	Number of folds. A single integer ≥ 2 . Default 10.
<code>repeats</code>	Number of repeats. A single integer ≥ 1 . Default 1.
<code>strata</code>	NULL (default) or a single column name present in every dataset. Rows are shuffled within each stratum so that every fold receives a proportional share of each stratum.
<code>breaks</code>	Number of quantile bins used when <code>strata</code> is numeric with more than <code>breaks</code> distinct values. Default 4.
<code>pool</code>	Strata holding less than this proportion of the rows are merged with the next-smallest stratum. A number in $[0, 0.5)$. Default 0.1.
<code>seed</code>	NULL (default) or a single number for reproducible folds. The caller's random-number stream is restored on exit.
<code>materialize</code>	Logical. If TRUE (default), <code>train / validate</code> list-columns holding the subsets are added. FALSE keeps only the row indices, which needs far less memory for large data.
<code>out_type</code>	Class of the <code>train / validate</code> subsets: <code>"dt"</code> (<code>data.table</code> , default) or <code>"df"</code> (plain <code>data.frame</code> , e.g. for 'ASReml-R' or other modelling functions that expect a <code>data.frame</code>). Factor levels are kept in full in every subset. Only used when <code>materialize = TRUE</code> ; the returned fold table itself is always a <code>data.table</code> .

Value

A `data.table` with all non-nested columns of data, followed by the columns described in [split_cv\(\)](#) (`id`, `id2`, `train_idx`, `validate_idx` and, if `materialize = TRUE`, `train / validate`, whose class follows `out_type`). Other nested columns are dropped (a message lists them).

See Also

[split_cv\(\)](#)

Examples

```
# Example: Cross-validation for nested data.table demonstrations

# Setup test data
dt_nest <- w2l_nest(
  data = iris,          # Input dataset
  cols = 1:2            # Nest first 2 columns
)

# Example 1: Basic 2-fold cross-validation (reproducible)
nest_cv(
  data = dt_nest,       # Input nested data.table
  v = 2,               # Number of folds (2-fold CV)
  seed = 123           # Reproducible folds
)

# Example 2: Repeated 2-fold CV, keeping only the split objects
nest_cv(
  data = dt_nest,       # Input nested data.table
  v = 2,               # Number of folds (2-fold CV)
  repeats = 2,         # Number of repetitions
  seed = 123,          # Reproducible folds
  materialize = FALSE   # No train/validate copies (saves memory)
)

# Example 3: data.frame subsets, ready for ASReml-R / lm() / glm()
cv_df <- nest_cv(dt_nest, v = 2, seed = 123, out_type = "df")
class(cv_df$train[[1]]) # "data.frame"

# Example 4: masking-style CV (keep all rows, hide validation phenotypes)
cv_idx <- nest_cv(dt_nest, v = 2, seed = 123, materialize = FALSE)
cv_idx[dt_nest, on = "name", full := i.data] # attach the full nested table
masked <- as.data.frame(cv_idx$full[[1]])    # a copy: dt_nest stays intact
masked$value[cv_idx$validate_idx[[1]]] <- NA
sum(is.na(masked$value)) # validation records to be predicted
```

r2p_nest

Row to Pair Nested Transformation

Description

Pivots the levels of one column (names_from, e.g. environment, farm or test station) into separate columns for every trait in cols, aligning records by the identifier columns id (e.g. the animal ID). The result is nested by trait, ready for iterative analyses such as genetic correlations of the same trait across environments.

Usage

```
r2p_nest(data, names_from, cols, out_type = "dt", id = NULL)
```

Arguments

<code>data</code>	Input <code>data.frame</code> or <code>data.table</code> . It is never modified.
<code>names_from</code>	A single column name or index whose levels become columns.
<code>cols</code>	A character vector of column names or numeric indices: the trait columns to pivot.
<code>out_type</code>	Output nesting format (" <code>dt</code> " or " <code>df</code> "). Default " <code>dt</code> ".
<code>id</code>	Identifier column(s) used to align rows across levels of <code>names_from</code> . Default <code>NULL</code> uses all columns that are neither in <code>cols</code> nor <code>names_from</code> (a message lists them); supplying <code>id</code> explicitly is strongly recommended.

Details

The combination of `id` and `names_from` must identify every row uniquely. Otherwise `dcast()` would aggregate duplicated records with `length()` and silently replace trait values by counts; the function stops instead.

Other trait columns in `cols` are never used as identifiers: their values differ between environments and would prevent any pairing.

Value

A nested `data.table` with columns `name` (trait) and `data`. Each nested table holds the `id` columns plus one column per level of `names_from`.

Examples

```
# Example: the same traits recorded on the same animals in two farms
set.seed(1)
growth <- data.frame(
  animal = rep(sprintf("A%02d", 1:6), each = 2),
  farm   = rep(c("farm1", "farm2"), times = 6),
  adg    = round(rnorm(12, 900, 50)),    # average daily gain
  bf     = round(rnorm(12, 11, 1.5), 1)  # backfat
)

# Example 1: column names
r2p_nest(
  growth,
  names_from = "farm",          # levels become columns: farm1, farm2
  cols       = c("adg", "bf"),  # traits to pivot
  id         = "animal"        # aligns records of the same animal
)
# Returns a nested data.table where:
# - name: trait names (adg, bf)
# - data: one row per animal with columns animal, farm1, farm2

# Example 2: numeric indices
r2p_nest(growth, names_from = 2, cols = 3:4, id = 1)
```

split_cv

*Apply V-Fold Cross-Validation to a List of Datasets***Description**

split_cv creates (optionally repeated and stratified) v-fold cross-validation splits for every dataset in a list. It is implemented with base R and data.table only and returns, per dataset, a data.table of fold identifiers, row indices and (optionally) the training / validation subsets.

Usage

```
split_cv(
  data,
  v = 10L,
  repeats = 1L,
  strata = NULL,
  breaks = 4L,
  pool = 0.1,
  seed = NULL,
  materialize = TRUE,
  out_type = "dt"
)
```

Arguments

data	A list whose every element is a data.frame or data.table. Must be non-empty. A single data.frame is not accepted (wrap it in list()).
v	Number of folds. A single integer ≥ 2 . Default 10.
repeats	Number of repeats. A single integer ≥ 1 . Default 1.
strata	NULL (default) or a single column name present in every dataset. Rows are shuffled within each stratum so that every fold receives a proportional share of each stratum.
breaks	Number of quantile bins used when strata is numeric with more than breaks distinct values. Default 4.
pool	Strata holding less than this proportion of the rows are merged with the next-smallest stratum. A number in $[0, 0.5)$. Default 0.1.
seed	NULL (default) or a single number for reproducible folds. The caller's random-number stream is restored on exit.
materialize	Logical. If TRUE (default), train / validate list-columns holding the subsets are added. FALSE keeps only the row indices, which needs far less memory for large data.
out_type	Class of the train / validate subsets: "dt" (data.table, default) or "df" (plain data.frame, e.g. for 'ASReml-R' or other modelling functions that expect a data.frame). Factor levels are kept in full in every subset. Only used when materialize = TRUE; the returned fold table itself is always a data.table.

Details

Fold sizes differ by at most one row. With stratification, rows are shuffled within strata, the strata are laid out one after another and fold labels are dealt out cyclically, so each stratum is spread evenly across folds. The stratification rules (`breaks`, `pool`) follow the same idea as `rsample::vfold_cv()`, but the random fold assignments are not identical to `rsample`'s.

Training and validation subsets can always be rebuilt from the indices: `data[[1]][res[[1]]$train_idx[[k]], ]`.

For mixed-model cross-validation (e.g. genomic prediction with 'ASReml-R') a common alternative to subsetting is masking: keep all records, set the phenotypes of `validate_idx` to NA, fit the model on the full data and correlate the predicted breeding values of the masked individuals with their observations. `materialize = FALSE` is enough for this, since only the indices are needed.

Value

A list of `data.table` objects (one per input dataset, names of data preserved), each with one row per fold and the columns:

- `id` — fold label (`Fold1`, ...), or repeat label (`Repeat1`, ...) when `repeats > 1`.
- `id2` — fold label, only when `repeats > 1`.
- `train_idx` — list-column of training row indices.
- `validate_idx` — list-column of validation row indices.
- `train`, `validate` — list-columns of subsets, `data.table` or `data.frame` according to `out_type` (only when `materialize = TRUE`).

See Also

[nest_cv\(\)](#) for the nested `data.table` variant.

Examples

```
# Prepare example data: Convert first 3 columns of iris dataset to long format and split
dt_split <- w2l_split(data = iris, cols = 1:3)
# dt_split is now a list containing 3 data tables for Sepal.Length, Sepal.Width, and Petal.Length

# Example 1: Single cross-validation (no repeats)
split_cv(
  data = dt_split,      # Input list of split data
  v = 3,                # Set 3-fold cross-validation
  repeats = 1,          # Perform cross-validation once (no repeats)
  seed = 123            # Reproducible folds
)
# Returns a list where each element contains:
# - id: fold labels (Fold1, Fold2, Fold3)
# - train_idx / validate_idx: row indices of each fold
# - train / validate: training and validation subsets

# Example 2: Repeated cross-validation
split_cv(
  data = dt_split,      # Input list of split data
```

```

    v = 3,                # Set 3-fold cross-validation
    repeats = 2,          # Perform cross-validation twice
    seed = 123
  )
  # Returns a list where each element contains:
  # - id: repeat labels (Repeat1, Repeat2)
  # - id2: fold labels (Fold1, Fold2, Fold3)
  # - train_idx / validate_idx, train / validate

  # Example 3: Stratified CV, indices only (memory friendly)
  res <- split_cv(dt_split, v = 5, strata = "Species", seed = 1,
                 materialize = FALSE)
  # Rebuild the training set of fold 1 of the first dataset when needed
  head(dt_split[[1]][res[[1]]$train_idx[[1]], ])

```

top_perc

*Select Top or Bottom Percentage of Data***Description**

Selects the top (largest) or bottom (smallest) percentage of data based on specified traits. Positive percentages extract the largest values; negative percentages extract the smallest values.

Usage

```

top_perc(
  data,
  perc,
  cols,
  by = NULL,
  keep_data = FALSE,
  stats = c("n", "min", "max", "mean", "median", "sd", "se", "cv")
)

```

Arguments

data	A data.frame or data.table.
perc	A numeric vector strictly between -1 and 1 (excluding 0). Positive values (e.g., 0.05) select the top X% of largest values. Negative values (e.g., -0.1) select the bottom X% of smallest values.
cols	Numeric trait column(s) to analyse, as names or indices.
by	Grouping column(s), as names or indices. Default is NULL.
keep_data	Logical. If TRUE, returns a named list where each element contains both stat (summary statistics) and data (the subset rows). If FALSE (default), returns a single combined data.frame of statistics for all perc values.
stats	Statistics of the selected records, computed exactly as in <code>desc_stats()</code> (same names and definitions). Default <code>c("n", "min", "max", "mean", "median", "sd", "se", "cv")</code> ; any of "n", "min", "max", "mean", "median", "q1", "q3", "iqr", "mad", "sd", "se", "ci", "ci_low", "ci_high", "cv", "skew", "kurt".

Details

Ranking uses `ties.method = "min"`, so tied values at the cut-off are all selected and slightly more than the requested share can be returned. Missing values are removed before ranking. Groups with fewer than $1 / \text{abs}(\text{perc})$ non-missing records cannot contribute a single record: they are kept in the summary with $n = 0$ and a warning is issued.

Value

- `keep_data = FALSE`: a `data.frame` with one row per `by` / `cols` / `perc` combination: the `by` columns, `variable`, the requested stats and selection.
- `keep_data = TRUE`: a named list (one element per `perc` value) where each element is a list with `$stat` and `$data`.

See Also

[`desc\_stats\(\)`](#) for statistics of all records.

Examples

```
# Example 1: Basic usage with single trait
# This example selects the top 10% of observations based on Petal.Width
# keep_data=TRUE returns both summary statistics and the filtered data
top_perc(iris,
  perc = 0.1,           # Select top 10%
  cols = c("Petal.Width"), # Column to analyze
  keep_data = TRUE)     # Return both stats and filtered data

# Example 2: Using grouping with 'by' parameter
# This example performs the same analysis but separately for each Species
# Returns a data.frame of summary statistics, one row per Species
top_perc(iris,
  perc = 0.1,           # Select top 10%
  cols = c("Petal.Width"), # Column to analyze
  by = "Species")       # Group by Species
```

w2l_nest

Reshape Wide Data to Long Format and Nest by Specified Columns

Description

`w2l_nest` reshapes a wide-format `data.frame` or `data.table` into long format, then nests the result by name (the pivoted column identifier) and any optional grouping variables supplied via `by`. Each row of the returned table contains a nested `data.table` or `data.frame` in the `data` list-column.

Usage

```
w2l_nest(data, cols = NULL, by = NULL, out_type = "dt")
```

Arguments

<code>data</code>	<code>data.frame</code> or <code>data.table</code> . Wide-format input dataset. The input object is never modified: a <code>data.frame</code> is converted with <code>as.data.table()</code> , a <code>data.table</code> is read as-is.
<code>cols</code>	numeric or character. Columns to pivot from wide to long, specified as integer indices or column names. Default <code>NULL</code> : when <code>NULL</code> , by must be provided and the function performs a pure nest operation (no melting).
<code>by</code>	numeric or character. Additional grouping variables for hierarchical nesting, specified as integer indices or column names. Default <code>NULL</code> .
<code>out_type</code>	character. Class of each nested subset: <code>"dt"</code> for <code>data.table</code> (default) or <code>"df"</code> for <code>data.frame</code> .

Details

Column resolution: both `cols` and `by` accept either integer column positions or character column names. Out-of-bounds indices and unknown names are caught early with informative error messages.

Overlap guard: if any column appears in both `cols` and `by`, the function stops with an error before attempting to melt, preventing silent structural corruption.

Factor-free melting: `melt()` is called with `variable.factor = FALSE` so the name column is always character, avoiding unexpected factor-level ordering in downstream grouping operations.

No side effects: earlier versions called `setDT()` on the argument, which silently turned the caller's `data.frame` into a `data.table`. The input is now left untouched.

Memory efficiency: `.SDcols` restricts `.SD` to non-key columns, so grouping keys are never stored redundantly inside each nested object.

Value

A `data.table` with one row per combination of name (and by levels, if provided). The data list-column holds the corresponding nested `data.table` or `data.frame` for each group. Grouping key columns are never duplicated inside the nested objects.

Note

- Passing an empty table (0 rows) triggers a `warning()` and returns a 0-row table with the regular column structure (name, the by columns and data).
- name and value are reserved for the long format; data must not contain other columns with these names.
- `cols` and `by` must not overlap; overlapping columns will raise an error.
- `out_type` values other than `"dt"` or `"df"` raise an error.

See Also

`tidytable::nest_by()` for a tidyverse-style equivalent.

Examples

```
# Example: Wide to long format nesting demonstrations

# Example 1: Basic nesting by group
w2l_nest(
  data = iris,          # Input dataset
  by = "Species"        # Group by Species column
)

# Example 2: Nest specific columns with numeric indices
w2l_nest(
  data = iris,          # Input dataset
  cols = 1:4,           # Select first 4 columns to nest
  by = "Species"        # Group by Species column
)

# Example 3: Nest specific columns with column names
w2l_nest(
  data = iris,          # Input dataset
  cols = c("Sepal.Length", # Select columns by name
           "Sepal.Width",
           "Petal.Length"),
  by = 5                # Group by column index 5 (Species)
)
# Returns similar structure to Example 2
```

w2l_split

Reshape Wide Data to Long Format and Split into a Named List

Description

w2l_split reshapes a wide-format data.frame or data.table into long format, then splits the result into a named list keyed by the pivoted column identifier (variable) and any optional grouping variables supplied via by. List element names are derived directly from the grouping key combinations produced by split(), guaranteeing name-to-content alignment.

Usage

```
w2l_split(data, cols = NULL, by = NULL, out_type = "dt", sep = "_")
```

Arguments

data	data.frame or data.table. Wide-format input dataset. The input object is never modified.
cols	numeric or character. Columns to pivot from wide to long, specified as integer indices or column names. Default NULL: when NULL, by must be provided and the function splits the data as-is without melting.

by	numeric or character. Additional grouping variables used as secondary split keys, specified as integer indices or column names. Default NULL.
out_type	character. Class of each list element: "dt" for data.table (default) or "df" for data.frame.
sep	character. Separator used when concatenating multiple grouping key values into a single list-element name. Default "_".

Details

Name safety: list names are built from each element's own key values (joined by sep), so names always match contents. This also makes sep work on every data.table version (split(sep =) is only honoured from data.table 1.16.0 on).

Column resolution: both cols and by accept integer column positions or character column names. Out-of-bounds indices and unknown names are caught early with informative error messages.

Overlap guard: columns appearing in both cols and by raise an error before melting to prevent id.vars / measure.vars conflicts.

Factor-free melting: melt() is called with variable.factor = FALSE so the variable column is always character, keeping split() sort order consistent with lexicographic expectations.

Memory efficiency: the elements returned by split() are independent objects, so for out_type = "df" they are converted in place with setDF() and no column vector is duplicated. The input itself is never modified.

Value

A named list of data.table or data.frame objects (controlled by out_type). Names reflect the key combination of variable (and by levels if provided), joined by sep.

- If by is NULL, the list is keyed by the pivoted column names only.
- If by is specified, the list is keyed by variable and all by level combinations.

Note

- An empty input table (0 rows) triggers a warning() and returns an empty list immediately.
- cols and by must not overlap; shared columns raise an error.
- out_type values other than "dt" or "df" raise an error.

See Also

tidytable::group_split() for a tidyverse-style equivalent.

Examples

```
# Example: Wide to long format splitting demonstrations

# Example 1: Basic splitting by Species
w2l_split(
  data = iris,           # Input dataset
  by = "Species"         # Split by Species column
```

```
) |>
  lapply(head)                # Show first 6 rows of each split

# Example 2: Split specific columns using numeric indices
w2l_split(
  data = iris,                # Input dataset
  cols = 1:3,                 # Select first 3 columns to split
  by = 5                      # Split by column index 5 (Species)
) |>
  lapply(head)                # Show first 6 rows of each split

# Example 3: Split specific columns using column names
list_res <- w2l_split(
  data = iris,                # Input dataset
  cols = c("Sepal.Length",    # Select columns by name
           "Sepal.Width"),
  by = "Species"              # Split by Species column
)
lapply(list_res, head)        # Show first 6 rows of each split
# Returns similar structure to Example 2
```

Index

`base::basename()`, [18](#)

`c2p_nest`, [2](#)
`combn`, [3](#)

`desc_stats`, [4](#)
`desc_stats()`, [31](#), [32](#)

`export_list`, [8](#)
`export_nest`, [10](#)
`export_xlsx`, [12](#)

`format_digits`, [14](#)
`fread`, [18–20](#)
`fwrite`, [8–11](#)

`get_path_info`, [16](#)

`import_csv`, [18](#)
`import_xlsx`, [20](#)

`mintyr_example`, [24](#)
`mintyr_example()`, [25](#)
`mintyr_examples`, [25](#)
`mintyr_examples()`, [24](#)

`nest_cv`, [25](#)
`nest_cv()`, [30](#)

`r2p_nest`, [27](#)
`rbindlist`, [19](#), [20](#)
`read_excel`, [22](#)

`setDTthreads`, [21](#)
`split_cv`, [29](#)
`split_cv()`, [25](#), [26](#)

`tools::file_path_sans_ext()`, [18](#)
`top_perc`, [31](#)
`top_perc()`, [7](#)

`w2l_nest`, [32](#)
`w2l_nest()`, [25](#)
`w2l_split`, [34](#)