Package 'dataprep'

October 1, 2026

Type Package

Title Fast, Efficient, and Versatile Data Preprocessing and Reshaping with 'C++', 'OpenMP' & 'SIMD'

Version 0.1.8

Description Fast, efficient, and versatile preprocessing and reshaping of tabular and time-series data. Most heavy routines are implemented in 'C++' via 'Rcpp', with optional 'OpenMP' parallelization and 'SIMD' acceleration ('AVX2' / 'AVX-512') on supported hardware. The 0.1.8 release rewrites the cleaning routines in 'C++' and delivers a 1.1–1146× speedup over 0.1.5. The 'melt()' and 'dcast()' reshaping functions achieve a 0.6×–1628.9× speedup for 'melt()' and a 1.9×–799.8× speedup for 'dcast()' relative to every one of the seven major alternatives in the R and Python ecosystems, at every tested scale (from 1,000 to 100,000,000 rows), and produce output identical to 'reshape2', 'data.table', 'tidyr', 'pandas', 'polars', 'dask', and 'duckdb'. Core preprocessing steps include variable deletion by missing fraction, observation deletion by consecutive missing runs, point-by-point weighted outlier removal via conditional extremum, traditional percentile-based outlier removal, and linear interpolation within short time periods. The package also provides fast reshaping, descriptive statistics, missing-value diagnosis, multiple imputation strategies, winsorization, several outlier detection methods (IQR, MAD, percentile), data transformation and standardization, categorical encoding, duplicate removal, data validation, data quality reporting, and stratified sampling. Feature-engineering helpers cover binning, high-correlation and low-variance filtering, and string cleaning. Time-series tools cover detrending, diurnal-cycle removal, rolling statistics, lag creation, resampling, simple decomposition, day/night and season flags, log returns, drift detection, and panel balancing. Fit/transform-style machine-learning interfaces prevent data leakage during preprocessing. Methods are based on, and improved from: Liang, C.-S., Wu, H., Li, H.-Y., Zhang, Q., Li, Z. & He, K.-B. (2020) <doi:10.1016/j.scitotenv.2020.140923>. This work was supported by the National Natural Science Foundation of China (No. 12301674).

Depends R (>= 4.0.0)

ByteCompile TRUE

Imports Rcpp (>= 1.0.10), ggplot2, stats, parallel

LinkingTo Rcpp (>= 1.0.10)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), data.table, reshape2, tidyr, dplyr, microbenchmark, reticulate

VignetteBuilder knitr, rmarkdown

SystemRequirements C++17, optional OpenMP

License GPL (>= 2)

Encoding UTF-8

NeedsCompilation yes

URL https://github.com/chunshengliang/dataprep, https://chunshengliang.github.io/dataprep/

BugReports https://github.com/chunshengliang/dataprep/issues

Config/testthat/edition 3

LazyData true

Config/roxygen2/version 8.1.0

Author Chun-Sheng Liang [aut, cre] (ORCID: <https://orcid.org/0000-0002-7386-1958>), Hao Wu [aut], Hai-Yan Li [aut], Qiang Zhang [aut], Zhanqing Li [aut], Ke-Bin He [aut]

Maintainer Chun-Sheng Liang <chun-shengliang@qq.com>

Repository CRAN

Date/Publication 2026-10-01 15:41:00 UTC

Contents

dataprep-package 3
balance_panel 5
bin_data 6
clean_strings 7
condextr 9
create_lags 10
data 11
data1 14
dataprep 15
data_report 17
day_night_flag 18

dcast . . . 19
decompose_ts . . . 23
deduplicate . . . 24
descdata . . . 25
descplot . . . 26
detect_outliers . . . 27
detrend_ts . . . 28
drift_detect . . . 29
dry_run . . . 30
encode_categorical . . . 31
filter_high_cor . . . 32
filter_low_var . . . 33
impute_missing . . . 34
log_returns . . . 35
melt . . . 35
na_diagnose . . . 39
obsedele . . . 40
optisolu . . . 42
percdata . . . 43
percoutl . . . 44
percplot . . . 45
phys_filter . . . 46
prep_fit . . . 47
prep_transform . . . 49
remove_diurnal_cycle . . . 50
resample_time . . . 50
roll_apply . . . 51
sample_data . . . 52
season_flag . . . 53
shorvalu . . . 54
transform_data . . . 55
validate_data . . . 56
varidele . . . 56
winsorize . . . 57
zerona . . . 58

Index **59**

`dataprep-package` *dataprep: Fast, Efficient, and Versatile Data Preprocessing and Reshaping with 'C++', 'OpenMP' & 'SIMD'*

Description

Fast, efficient, and versatile preprocessing and reshaping of tabular and time-series data. Most heavy routines are implemented in 'C++' via 'Rcpp', with optional 'OpenMP' parallelization and 'SIMD' acceleration ('AVX2' / 'AVX-512') on supported hardware. The 0.1.8 release rewrites the cleaning routines in 'C++' and delivers a 1.1–1146× speedup over 0.1.5. The 'melt()' and 'dcast()' reshaping functions achieve a 0.6×–1628.9× speedup for 'melt()' and a 1.9×–799.8× speedup for 'dcast()' relative to every one of the seven major alternatives in the R and Python ecosystems, at every tested scale (from 1,000 to 100,000,000 rows), and produce output identical to 'reshape2', 'data.table', 'tidyr', 'pandas', 'polars', 'dask', and 'duckdb'. Core preprocessing steps include variable deletion by missing fraction, observation deletion by consecutive missing runs, point-by-point weighted outlier removal via conditional extremum, traditional percentile-based outlier removal, and linear interpolation within short time periods. The package also provides fast reshaping, descriptive statistics, missing-value diagnosis, multiple imputation strategies, winsorization, several outlier detection methods (IQR, MAD, percentile), data transformation and standardization, categorical encoding, duplicate removal, data validation, data quality reporting, and stratified sampling. Feature-engineering helpers cover binning, high-correlation and low-variance filtering, and string cleaning. Time-series tools cover detrending, diurnal-cycle removal, rolling statistics, lag creation, resampling, simple decomposition, day/night and season flags, log returns, drift detection, and panel balancing. Fit/transform-style machine-learning interfaces prevent data leakage during preprocessing. Methods are based on, and improved from: Liang, C.-S., Wu, H., Li, H.-Y., Zhang, Q., Li, Z. & He, K.-B. (2020) doi:10.1016/j.scitotenv.2020.140923. This work was supported by the National Natural Science Foundation of China (No. 12301674).

Details

The package is organised around a four-step cleaning pipeline:

1. **Variable deletion** ([varidele()]) – drop columns whose missing fraction exceeds a threshold.
2. **Observation deletion** ([obsedele()]) – drop rows with a consecutive missing run longer than 'half' minutes on both sides.
3. **Outlier removal** ([condextr()]) – point-by-point weighted conditional extremum detection.
4. **Short-period interpolation** ([shorvalu()]) – fill remaining short gaps from nearby valid anchors.

Steps 1-4 are wrapped by [dataprep()] for one-call use. The design reasoning behind the ordering is documented in 'vignette("dataprep-philosophy")'.

The package also provides fast wide-to-long and long-to-wide reshaping ([melt()], [dcast()]), a full set of descriptive and diagnostic helpers ([descdata()], [descplot()], [percdata()], [percplot()], [na_diagnose()], [data_report()]), and fit/transform-style preprocessing plans that prevent data leakage ([prep_fit()], [prep_transform()]).

This work was supported by the National Natural Science Foundation of China (No. 12301674).

Author(s)

Maintainer: Chun-Sheng Liang <chun-shengliang@qq.com> (ORCID)

Authors:

- Chun-Sheng Liang <chun-shengliang@qq.com> (ORCID)
- Hao Wu
- Hai-Yan Li
- Qiang Zhang
- Zhanqing Li
- Ke-Bin He

See Also

Core pipeline: [dataprep()], [varidele()], [obsedele()], [condextr()], [shorvalu()].

Reshaping: [melt()], [dcast()].

Leakage-free workflow: [prep_fit()], [prep_transform()].

Diagnostics and reporting: [descdata()], [descplot()], [percdata()], [percplot()], [na_diagnose()], [data_report()].

`balance_panel` *Balance panel data*

Description

Transforms an unbalanced panel dataset into a balanced one by either keeping only complete units or filling missing time points with a specified value.

Usage

```
balance_panel(data, unit_col = NULL, time_col = NULL,
              fill = NA_real_, method = c("fill", "complete"),
              verbose = FALSE)
```

Arguments

`data`	A data frame containing panel data.
`unit_col`	Column index or name identifying the individual unit (e.g., station ID, subject ID). If `NULL`, the first column is assumed.
`time_col`	Column index or name identifying the time point. If `NULL`, the first column matching `date` is used.
`fill`	Value used to fill missing combinations when `method = "fill"`.
`method`	Balancing method: `"fill"` adds rows for missing unit-time combinations and fills **numeric** non-key columns with `fill`; non-numeric columns of the added rows are left as `NA`. `"complete"` keeps only units that have observations at every time point.
`verbose`	Logical; if `TRUE`, prints progress message.

Details

The function assumes that the time column is sorted and contains no duplicates within a unit. If duplicates exist, they should be handled beforehand.

Value

A balanced data frame.

Examples

```
balance_panel(data[1:200, c(1, 4, 17:19)], unit_col = 2, time_col = 1, method = "complete")
```

---

bin_data *Discretize continuous variables into bins*

Description

Converts numeric columns into categorical factors using equal-width, equal-frequency, or custom breakpoints. This is a common preprocessing step for transforming continuous variables for modeling or visualization.

Usage

```
bin_data(data, cols = NULL, method = "equal_width",
         bins = 10, breaks = NULL, include_lowest = TRUE,
         labels = NULL, verbose = FALSE)
```

Arguments

`data`	A data frame containing numeric columns to bin.
`cols`	Column indices or names to bin. If `NULL`, all numeric columns are used.
`method`	Binning method. One of `"equal_width"`, `"equal_freq"`, or `"custom"`.
`bins`	Number of bins for equal-width and equal-frequency methods.
`breaks`	Numeric vector of breakpoints for custom binning. Required when `method = "custom"`.
`labels`	Optional character vector of labels for the bins.
`include_lowest`	Logical; if `TRUE`, the lowest break value is included in the first bin.
`verbose`	Logical; if `TRUE`, prints a timing message.

Details

For `"equal_width"`, bins are created by dividing the range of the data into equal-width intervals. For `"equal_freq"`, bins are created by quantiles so that each bin contains approximately the same number of observations. The `"custom"` method uses user-supplied breakpoints.

Missing values are preserved as `NA` in the output.

Value

If data is a data frame, a data frame with the selected columns replaced by factors. If data is a numeric vector, a factor vector.

Examples

```
data <- data.frame(x = rnorm(100), y = runif(100))
# Equal-width binning into 5 bins
bin_data(data, cols = 1:2, bins = 5)

# Equal-frequency binning
bin_data(data, cols = "x", method = "equal_freq", bins = 4)

# Custom breaks
bin_data(data, cols = "x", method = "custom", breaks = c(-Inf, 0, Inf), labels = c("neg", "pos"))
```

---

clean_strings *Clean and standardize character columns*

Description

Applies common string cleaning operations to character or factor columns: trimming whitespace, changing case, and applying regular expression substitutions. Numeric and other non-character columns are left untouched.

Usage

```
clean_strings(
  data,
  cols = NULL,
  trim = FALSE,
  tolower = FALSE,
  toupper = FALSE,
  pattern = NULL,
  replacement = NULL,
  verbose = FALSE
)
```

Arguments

data	A data frame, matrix, or character vector.
cols	Columns to clean. If NULL (default), all character and factor columns are selected. Columns that are neither character nor factor are skipped, so the numeric and logical columns of data are never silently coerced to character.
trim	Logical; if TRUE, leading and trailing whitespace is removed with trimws.
tolower	Logical; if TRUE, converted to lowercase.

toupper	Logical; if TRUE, converted to uppercase.
pattern	Optional regular expression passed to gsub.
replacement	Replacement string for pattern. Defaults to "" (deletion) when pattern is supplied and replacement is NULL.
verbose	Logical; if TRUE, prints timing message.

Details

The operations are applied in a fixed order:

1. trim (via trimws),
2. tolower then toupper,
3. pattern replacement (via gsub).

When both tolower and toupper are TRUE, the uppercase conversion wins (it is applied last). In practice only one of the two should be set.

For factor columns, the underlying integer codes are dropped and the column becomes a character vector. If you need to keep the factor type, convert the cleaned values back with factor().

Value

A data frame (or character vector, if the input was a vector) with the selected columns cleaned.

Examples

```
df <- data.frame(
  id   = 1:3,
  name = c("  Alice ", "BOB", "Charlie "),
  city = c("New York", "london", "Paris"),
  stringsAsFactors = FALSE
)

# Only the character columns are touched by default
clean_strings(df, trim = TRUE, tolower = TRUE)

# Explicit column selection
clean_strings(df, cols = "name", trim = TRUE, toupper = TRUE)

# Regex replacement
clean_strings(df, cols = "city",
              pattern = "\\s+", replacement = "_")

# Vector input
clean_strings(c(" A ", " b ", "C"), trim = TRUE, tolower = TRUE)
```

condextr *Remove outliers using point-by-point weighed outlier removal by conditional extremum*

Description

An outlier removal method that considers each potential outlier individually, using conditional thresholds based on top and bottom percentiles and error margins. It is a more refined alternative to simple percentile-based removal.

Usage

```
condextr(data, cols = NULL, group = NULL, top = 0.995,
top.error = 0.1, top.magnitude = 0.2, bottom = 0.0025, bottom.error = 0.2,
bottom.magnitude = 0.4, interval = 10, by = "min", half = 30,
times = 10, date_col = NULL, cores = NULL, verbose = FALSE)
```

Arguments

`data`	A data frame, matrix, or numeric vector. If a vector, outlier removal is applied directly.
`cols`	Column indices or names of numeric variables.
`group`	Grouping column index or name. Outlier removal is performed within each group.
`top`, `top.error`, `top.magnitude`	Parameters for the top threshold.
`bottom`, `bottom.error`, `bottom.magnitude`	Parameters for the bottom threshold.
`interval`	Number of outlier marking iterations between observation deletions.
`by`	Time unit for observation deletion. An invalid unit string raises an error.
`half`	Half window size in minutes for observation deletion.
`times`	Number of observation deletions.
`date_col`	Time column index or name.
`cores`	Number of CPU cores passed to the C++ backend. `NULL` (default) lets the backend choose based on data size.
`verbose`	Logical; if `TRUE`, prints timing and deletion counts.

Details

The method marks values as outliers if they are the maximum or minimum and exceed a threshold that depends on a high percentile plus an error margin. After a specified number of marking steps, observation deletion is applied to remove rows with excessive missing values created by outlier removal.

Value

A data frame with outliers removed.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
condextr(obsedele(data[1:500, c(1, 4, 17:19)], cols = 3:5, group = 2),
         cols = 3:5, group = 2)
```

---

create_lags *Create lagged variables*

Description

Generates lagged (or lead) versions of selected numeric columns. Positive lags shift values backward in time (past values), negative lags shift forward. Useful for time series modeling.

Usage

```
create_lags(data, cols = NULL, lags = 1, prefix = "lag_",
            group = NULL, date_col = NULL, verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector.
cols	Column indices or names to create lags for.
lags	Vector of lag orders (positive for past, negative for future).
prefix	Prefix for new column names.
group	Optional grouping column for group-specific lags.
date_col	Currently ignored. Data are assumed to be in the correct time order.
verbose	Logical; if TRUE, prints number of lag columns created.

Value

A data frame or matrix with additional lag columns.

Examples

```
create_lags(data[1:100, c(1, 4, 17:19)], cols = 3:5, lags = c(1, 2))
```

`data` *Example data (particle number concentrations in SMEAR I Varrio forest)*

Description

Particle number size distribution (PNSD) and auxiliary variables measured at the SMEAR I Varrio forest station in Finland. The raw data were downloaded from https://smear.avaa.csc.fi/download and subset to two months (January and July 2020) for package size reasons.

Usage

```
data
```

Format

A data frame with 7,640 observations on the following 65 variables.

`date` a POSIXct vector, sampling time (10-minute resolution).

`tconc` a numeric vector, total number concentration.

`TPNC` a numeric vector, total particle number concentration.

`monthyear` a character vector, e.g. `"January 2020"`.

`1` a numeric vector, particle diameter in nm (1 nm).

`1.12` a numeric vector, particle diameter in nm.

`1.26` a numeric vector, particle diameter in nm.

`1.41` a numeric vector, particle diameter in nm.

`1.58` a numeric vector, particle diameter in nm.

`1.78` a numeric vector, particle diameter in nm.

`2` a numeric vector, particle diameter in nm.

`2.24` a numeric vector, particle diameter in nm.

`2.51` a numeric vector, particle diameter in nm.

`2.82` a numeric vector, particle diameter in nm.

`3.16` a numeric vector, particle diameter in nm.

`3.55` a numeric vector, particle diameter in nm.

`3.98` a numeric vector, particle diameter in nm.

`4.47` a numeric vector, particle diameter in nm.

`5.01` a numeric vector, particle diameter in nm.

`5.62` a numeric vector, particle diameter in nm.

`6.31` a numeric vector, particle diameter in nm.

`7.08` a numeric vector, particle diameter in nm.

`7.94` a numeric vector, particle diameter in nm.

`8.91` a numeric vector, particle diameter in nm.

`10` a numeric vector, particle diameter in nm.

`11.2` a numeric vector, particle diameter in nm.

`12.6` a numeric vector, particle diameter in nm.

`14.1` a numeric vector, particle diameter in nm.

`15.8` a numeric vector, particle diameter in nm.

`17.8` a numeric vector, particle diameter in nm.

`20` a numeric vector, particle diameter in nm.

`22.4` a numeric vector, particle diameter in nm.

`25.1` a numeric vector, particle diameter in nm.

`28.2` a numeric vector, particle diameter in nm.

`31.6` a numeric vector, particle diameter in nm.

`35.5` a numeric vector, particle diameter in nm.

`39.8` a numeric vector, particle diameter in nm.

`44.7` a numeric vector, particle diameter in nm.

`50.1` a numeric vector, particle diameter in nm.

`56.2` a numeric vector, particle diameter in nm.

`63.1` a numeric vector, particle diameter in nm.

`70.8` a numeric vector, particle diameter in nm.

`79.4` a numeric vector, particle diameter in nm.

`89.1` a numeric vector, particle diameter in nm.

`100` a numeric vector, particle diameter in nm.

`112` a numeric vector, particle diameter in nm.

`126` a numeric vector, particle diameter in nm.

`141` a numeric vector, particle diameter in nm.

`158` a numeric vector, particle diameter in nm.

`178` a numeric vector, particle diameter in nm.

`200` a numeric vector, particle diameter in nm.

`224` a numeric vector, particle diameter in nm.

`251` a numeric vector, particle diameter in nm.

`282` a numeric vector, particle diameter in nm.

`316` a numeric vector, particle diameter in nm.

`355` a numeric vector, particle diameter in nm.

`398` a numeric vector, particle diameter in nm.

`447` a numeric vector, particle diameter in nm.

`501` a numeric vector, particle diameter in nm.

`562` a numeric vector, particle diameter in nm.

`631` a numeric vector, particle diameter in nm.

`708` a numeric vector, particle diameter in nm.

`794` a numeric vector, particle diameter in nm.

`891` a numeric vector, particle diameter in nm.

`1000` a numeric vector, particle diameter in nm.

Details

The 61 numeric channels between `1` and `1000` are size bins of the differential particle number size distribution, approximately logarithmically spaced. Column names are the geometric mean diameter in nanometres, written in the shortest form that R accepts as a name (so `1` rather than `1.00`, and `10` rather than `10.0`). Values are number concentrations in cm^{-3}. `tconc` and `TPNC` are integrals of the size distribution; `monthyear` is the month label used for group-wise cleaning.

The smallest seven size bins (`1` through `2`) and the largest three (`794`, `891`, `1000`) are entirely `NA` in this two-month subset, so `varidele()` will drop them even at a permissive `fraction`.

The data are the raw input for the cleaning pipeline described in `vignette("dataprep-cleaning")`. They contain:

- size bins with substantial missing fractions (especially the smallest and largest bins),
- observations with long consecutive missing runs,
- extreme values that are not always genuine outliers.

After running `dataprep` on this table, a smaller, cleaner table is returned. The `data1` dataset is the already-aggregated seven-column version; it has no long gaps and no obvious outliers, and is not a useful input for the cleaning pipeline.

Source

https://smear.avaa.csc.fi/download

References

Liang, C.-S., Wu, H., Li, H.-Y., Zhang, Q., Li, Z. & He, K.-B. (2020). Efficient data preprocessing, episode classification, and source apportionment of particle number concentrations. *Science of the Total Environment*, 741, 140923. doi:10.1016/j.scitotenv.2020.140923

Examples

```
data

dim(data)
names(data)[1:6]

# Per-column missing fraction, ordered by severity.
# The smallest and largest size bins have the most missing data.
na_frac <- sort(colMeans(is.na(data)), decreasing = TRUE)
head(na_frac, 10)
```

```
# The 61 size bins run from column 5 to column 65.
size_bins <- 5:65
summary(data[, size_bins[1:3]])
```

`data1` *Example data (aggregated particle number concentrations, SMEAR I Varrio forest)*

Description

An already-aggregated seven-column version of `data`. The 61 size bins of the raw size distribution have been collapsed into three log-normal modes (nucleation, Aitken, accumulation) plus two total concentrations (`tconc`, `TPNC`). The dataset covers the same two-month window as `data` (January and July 2020) and is intended for read-only demos.

Usage

```
data1
```

Format

A data frame with 7,640 observations on the following 7 variables.

`date` a POSIXct vector, sampling time (10-minute resolution).

`monthyear` a character vector, e.g. `"January 2020"`.

`Nucleation` a numeric vector, number concentration in the nucleation mode (nominally 3–25 nm).

`Aitken` a numeric vector, number concentration in the Aitken mode (nominally 25–100 nm).

`Accumulation` a numeric vector, number concentration in the accumulation mode (nominally 100–1000 nm).

`tconc` a numeric vector, total number concentration.

`TPNC` a numeric vector, total particle number concentration.

Details

Unlike `data`, which is the raw size-resolved table, `data1` contains only mode-integrated values. The three integrated values can remain non-zero even when the smallest raw size bins (`1` through `2`) are entirely `NA` in the two-month subset. It has:

- very few missing values,
- no long consecutive missing runs,
- no obvious outliers.

Because of this, `data1` is **not** a useful input for the cleaning pipeline (`varidele`, `obsedele`, `condextr`, `shorvalu`, `dataprep`): those steps have nothing to remove on an already-clean table. It is intended for demos that only read the data:

- descdata / descplot — descriptive statistics and plots,
- percdata / percplot — percentile summaries,
- na_diagnose — missing-value diagnostics,
- data_report — compact quality report.

Use data for anything that modifies values.

Source

https://smear.avaa.csc.fi/download

References

Liang, C.-S., Wu, H., Li, H.-Y., Zhang, Q., Li, Z. & He, K.-B. (2020). Efficient data preprocessing, episode classification, and source apportionment of particle number concentrations. *Science of the Total Environment*, 741, 140923. doi:10.1016/j.scitotenv.2020.140923

Examples

```
data1

dim(data1)
names(data1)

# The three log-normal modes and the two total concentrations
summary(data1[, 3:7])

# A quick look at how the modes relate to the totals
cor(data1[, 3:7], use = "complete.obs")

# data1 has very few missing values compared with the raw size bins
colMeans(is.na(data1))

# For a read-only descriptive plot
if (interactive()) {
  descplot(data1, cols = 3:7)
}
```

---

dataprep *Data preprocessing with multiple steps in one function*

Description

Performs variable deletion (varidele), observation deletion (obsedele), conditional extremum outlier removal (condextr), and short-period interpolation (shorvalu) in sequence.

Usage

```
dataprep(data, cols = NULL, group = NULL, optimal = FALSE,
interval = 10, times = 10, fraction = 0.25,
top = 0.995, top.error = 0.1, top.magnitude = 0.2,
bottom = 0.0025, bottom.error = 0.2, bottom.magnitude = 0.4, by = "min",
half = 30, intervals = 30, date_col = NULL, cores = NULL, verbose = FALSE)
```

Arguments

data	A data frame containing numeric columns and optionally a grouping column.
cols	Column indices or names of numeric variables to process.
group	Grouping column index or name.
optimal	Logical; if TRUE, optisolu is used to find optimal interval and times.
interval	Number of outlier-marking rounds between two observation-deletion rounds inside condextr.
times	Number of observation-deletion rounds in condextr.
fraction	Missing proportion threshold for variable deletion.
top, top.error, top.magnitude, bottom, bottom.error, bottom.magnitude	Outlier removal parameters.
by	Time unit for observation deletion. An invalid unit string raises an error.
half	Half window size in minutes for observation deletion.
intervals	Time gap for interpolation periods.
date_col	Time column index or name. If NULL, automatically detected.
cores	Number of CPU cores passed to obsedele(), condextr(), and optisolu(). NULL (default) lets each backend choose based on data size.
verbose	Logical; if TRUE, prints timing and deletion/interpolation counts.

Value

A preprocessed data frame.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
dataprep(data[1:60, c(1, 4, 18:19)], cols = 3:4, group = 2, interval = 2, times = 1, cores = 1)
```

`data_report` *Generate a simple data quality report*

Description

Computes a data quality summary and returns it invisibly as a list. When `verbose = TRUE`, the summary is also printed to the console.

Usage

```
data_report(data, cols = NULL, date_col = NULL, verbose = FALSE)
```

Arguments

`data`	A data frame.
`cols`	Optional column indices or names for the numeric columns to include in the report. If NULL, all numeric columns are used.
`date_col`	Reserved for future use; currently ignored (passed through to `na_diagnose`).
`verbose`	Logical; if `TRUE`, prints progress and timing messages.

Value

Invisibly returns a list containing data dimensions, variable types, missing value diagnosis, and descriptive statistics.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
data_report(data[1:200, c(1, 4, 17:19)], cols = 3:5, date_col = 1)
```

day_night_flag *Day/night flag*

Description

Creates a day/night indicator for each time point. If latitude and longitude are not provided, local time is used (daylight defined by an hour threshold). Otherwise, an approximate solar zenith angle is computed to determine whether the sun is above the horizon.

Usage

```
day_night_flag(data, date_col = NULL, lat = NULL, lon = NULL,
               threshold = 6, type = c("binary", "sun", "shade"),
               local_tz = "UTC", verbose = FALSE)
```

Arguments

data	A data frame with a time column.
date_col	Time column index or name. If NULL, the first column matching date is used.
lat	Latitude of the location(s). Can be a scalar or a vector of length equal to the number of rows.
lon	Longitude of the location(s). Same behavior as lat.
threshold	Hour threshold used for local-time based classification (default 6). Only used if latitude/longitude are missing.
type	Output type: "binary" returns 1/0, "sun" returns "day"/"night", "shade" returns "day"/"shade".
local_tz	Time zone for local time calculation (default UTC).
verbose	Logical; if TRUE, prints progress message.

Details

When latitude/longitude are supplied, the function computes the solar zenith angle using a simplified astronomical formula. Values with a solar zenith angle below 90 degrees are considered day.

Value

A vector of day/night flags.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
day_night_flag(data[1:100, c(1, 4, 17:19)], date_col = 1, lat = 40, lon = -105)
```

---

dcast *Cast a long-format data.frame into a wide format*

Description

Structural inverse of `melt`: a long-format data frame with one row per `(id, variable)` pair is reshaped into a wide-format data frame with one row per `id` combination and one column per `variable` level. The C++ backend builds compact integer lookup tables for both the row keys (id columns) and the column keys (variable column), then writes values by output column in strictly sequential order.

Usage

```
dcast(data, id = NULL, formula = NULL,
      variable = NULL, value = NULL, value.var = NULL,
      fill = NA_real_, fun.aggregate = NULL,
      na.rm = FALSE, cores = 0L, verbose = FALSE)
```

Arguments

`data`	A data frame in long format. It must contain one column identifying the variable (e.g. `"variable"`), one column holding the value (e.g. `"value"`), and one or more id columns.
`id`	Identifier columns. Character names, integer indices, a logical mask, or `NULL`. If `NULL`, all columns other than the variable and value columns are used as ids.
`formula`	Optional formula of the form `id1 + id2 ~ variable`. When supplied, the left-hand side overrides `id` and the right-hand side overrides `variable`.
`variable`	Name (or index) of the column identifying the variable. If `NULL`, a column named `"variable"`, `"variables"`, `"Variable"`, `"Variables"`, `"VARIABLE"`, or `"VARIABLES"` is used when exactly one such column exists.
`value`	Name (or index) of the column holding the values. If `NULL`, a column named `"value"`, `"values"`, `"Value"`, `"Values"`, `"VALUE"`, or `"VALUES"` is used when exactly one such column exists.
`value.var`	Alias for `value`. Used only when `value` is `NULL`; if both are supplied, `value` takes precedence.
`fill`	Value used to fill cells for `(id, variable)` pairs that do not appear in the input. Cells that appear in the input with value `NA` keep `NA` unless `na.rm = TRUE`. Default `NA_real_`.
`fun.aggregate`	Optional function used to reduce duplicate `(id, variable)` pairs. Called with a single argument (the vector of values for that pair) plus `na.rm = TRUE`. Without it, the last occurrence wins.

na.rm	Logical; if TRUE, rows whose value is NA or NaN are skipped during scatter, so the corresponding output cell keeps the fill value. Default FALSE.
cores	Number of OpenMP threads. 0 (default) lets the C++ backend choose based on data size.
verbose	Logical; if TRUE, prints timing information.

Details

dcast() is the structural inverse of melt. The C++ backend detects canonical melt() output automatically (the variable column is periodic and every id column is constant within one period) and switches to a *block-path tile transpose*: each tile of TILE x period doubles is read contiguously into an L1 buffer, transposed in place, and written contiguously to the output columns. This keeps both reads and writes sequential and enables OpenMP parallelisation. The cost per tile is O(TILE * period), independent of the number of levels, which is why wide-level tables scale well.

When the input is not block-aligned, dcast() builds an open-addressing hash of 64-bit packed row keys. If the combined bit budget of the id columns exceeds 64, it falls back to a 96-bit fingerprint (uint64_t + uint32_t) computed by a 4-way parallel FNV-1a and stored in a 16-byte slot table. For shuffle-friendly input, a 4-pass LSD radix sort over the packed keys replaces the hash table entirely; and when the block path is taken and the id column is a permutation of 1..n_blocks, a direct-index shortcut bypasses both.

The output is identical to reshape2::dcast, data.table::dcast, tidyr::pivot_wider, pandas.pivot, polars.pivot, duckdb PIVOT, and dask on every tested shape, within tol = 1e-12. See vignette("dataprep-melt-dcast for the implementation notes and vignette("dataprep-performance") for the benchmark tables.

On the canonical long-to-wide shape, dcast() is faster than every tested alternative. The speed-up relative to reshape2, data.table, tidyr, pandas, polars, dask, and duckdb spans 1.9x (against reshape2 on 1e3 rows and 10 levels, Ubuntu 25.10) to 799.8x (against duckdb on 1e8 rows and 100 levels, Windows 11 Pro for Workstations). The median across all tested cells and all competitors is 46.5x on Ubuntu 25.10 and 41.4x on Windows 11 Pro for Workstations; the mean is 90.2x and 76.6x, respectively.

Value

A wide-format data frame with one row per unique id combination and one column per unique value of the variable column, plus the id columns. Column names are the string representation of the variable values.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

Liang, C.-S., Wu, H., Li, H.-Y., Zhang, Q., Li, Z. & He, K.-B. (2020). Efficient data preprocessing, episode classification, and source apportionment of particle number concentrations. *Science of the Total Environment*, 741, 140923. doi:10.1016/j.scitotenv.2020.140923

See Also

melt for the inverse operation. vignette("dataprep-melt-dcast") for usage notes and implementation.

Examples

```
## --- Basic usage -------------------------------------------------

long <- data.frame(
  id       = rep(1:3, each = 2),
  variable = rep(c("x", "y"), 3),
  value    = c(1, 2, 3, 4, 5, 6)
)
dcast(long, id = "id", variable = "variable", value = "value")


## --- Formula interface -------------------------------------------

dcast(long, formula = id ~ variable)


## --- Multiple id columns -----------------------------------------

long2 <- data.frame(
  year     = rep(2020:2021, each = 4),
  city     = rep(c("A", "B"), each = 2, times = 2),
  variable = rep(c("temp", "rain"), 4),
  value    = c(15.2, 210, 14.8, 180, 16.1, 230, 15.5, 195)
)

# Explicit id columns
dcast(long2, id = c("year", "city"),
      variable = "variable", value = "value")

# Formula form (equivalent)
dcast(long2, formula = year + city ~ variable)


## --- Fill missing cells ------------------------------------------

# (2, "y") is missing from the input; fill = 0 gives it 0.
long3 <- data.frame(
  id       = c(1, 1, 2),
  variable = c("x", "y", "x"),
  value    = c(1, 2, 3)
)
dcast(long3, id = "id",
      variable = "variable", value = "value",
      fill = 0)


## --- Aggregate duplicate (id, variable) pairs --------------------
```

```
# id = 1 appears twice with variable = "x"; the mean is 1.5.
long4 <- data.frame(
  id       = c(1, 1, 2),
  variable = c("x", "x", "x"),
  value    = c(1, 2, 3)
)
dcast(long4, id = "id",
      variable = "variable", value = "value",
      fun.aggregate = mean)

# sum and length work the same way
dcast(long4, id = "id",
      variable = "variable", value = "value",
      fun.aggregate = sum)


## --- Skip NA values during scatter ---------------------------------

long5 <- data.frame(
  id       = c(1, 1, 2, 2),
  variable = c("x", "y", "x", "y"),
  value    = c(1, NA, 3, 4)
)

# Default: NA stays as NA
dcast(long5, id = "id",
      variable = "variable", value = "value")

# na.rm = TRUE: (1, "y") is skipped, cell keeps the fill value
dcast(long5, id = "id",
      variable = "variable", value = "value",
      na.rm = TRUE, fill = -1)


## --- Round-trip with melt() ----------------------------------------

wide <- data.frame(id = 1:3, a = c(1.5, 2.5, 3.5), b = c(4.5, 5.5, 6.5))
back <- dcast(melt(wide, id.vars = "id"),
              id = "id", variable = "variable", value = "value")
back[order(back$id), ]


## --- Larger example: 1000 rows, 50 levels -------------------------

set.seed(1)
n_rows   <- 1000L
n_levels <- 50L
long_big <- data.frame(
  id       = rep(seq_len(n_rows / n_levels), each = n_levels),
  variable = rep(sprintf("v%02d", seq_len(n_levels)),
                 times = n_rows / n_levels),
  value    = rnorm(n_rows)
```

```
)
wide_big <- dcast(long_big, id = "id",
                  variable = "variable", value = "value")
dim(wide_big)
```

`decompose_ts` *Simple time series decomposition*

Description

Decomposes a time series into trend, seasonal, and residual components using simple moving averages and group means. The function returns the residual component (or the original series with trend removed) to simplify analysis.

Usage

```
decompose_ts(data, cols = NULL, date_col = NULL,
             period = "month", method = "additive", verbose = FALSE)
```

Arguments

`data`	A data frame with a time column.
`cols`	Column indices or names to decompose.
`date_col`	Time column index or name.
`period`	Seasonal period: `"month"`, `"day"`, or `"hour"`.
`method`	Decomposition method: `"additive"` or `"multiplicative"`.
`verbose`	Logical; if `TRUE`, prints progress message.

Value

A data frame with the residual component.

Examples

```
decompose_ts(data[1:500, c(1, 4, 17:19)], cols = 3:5, date_col = 1, period = "month")
```

`deduplicate` *Remove duplicate observations*

Description

Identifies and removes duplicate rows. Supports exact duplication on all or selected columns, and fuzzy duplication for numeric columns using a rounding tolerance.

Usage

```
deduplicate(data, cols = NULL, method = "exact",
            key_cols = NULL, tol = 1e-8, max_dist = 1,
            verbose = FALSE)
```

Arguments

`data`	A data frame.
`cols`	The column indices or names used for duplication detection. If NULL and `method = "exact"`, all columns are used. For `method = "fuzzy"`, must be numeric columns.
`method`	Duplicate detection method. One of `"exact"` or `"fuzzy"`.
`key_cols`	Deprecated. Use `cols` instead.
`tol`	Tolerance for fuzzy matching. Values are rounded to the number of decimal places indicated by `tol` before duplicate detection.
`max_dist`	Not used in current implementation.
`verbose`	Logical; if `TRUE`, prints number of duplicates removed.

Details

Fuzzy matching works by rounding numeric columns to a sufficient number of decimal places (derived from `tol`) and then applying exact duplicate detection.

Value

A data frame with duplicates removed.

Examples

```
# Exact duplicates on all columns
deduplicate(data[1:200, c(1, 4, 17:19)])
# Exact duplicates on selected columns
deduplicate(data[1:200, c(1, 4, 17:19)], cols = 3:5)
# Fuzzy duplicates with tolerance 0.01
deduplicate(data[1:200, c(1, 4, 17:19)], cols = 3:5, method = "fuzzy", tol = 0.01)
```

descdata *Fast descriptive statistics*

Description

Computes descriptive statistics for selected numeric columns using efficient C++ routines.

Usage

```
descdata(data, cols = NULL, stats = 1:9, first = "variables",
         cores = NULL, verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector. If a vector is supplied, it is treated as a single variable and the output's first column is named "value".
cols	Column indices or names of numeric variables to describe. If NULL, all numeric columns are used.
stats	Statistic selection from the nine available: n, na, mean, sd, median, trimmed, min, max, IQR. Can be a numeric vector or a character vector of statistic names.
first	Name for the first column of the output, which contains the variable identifiers.
cores	Number of CPU cores. NULL (default) means single-threaded execution; pass an integer > 0 to enable OpenMP when nrow(x) * ncol(x) > 1e5.
verbose	Logical; if TRUE, prints timing message.

Details

The function accepts vectors, matrices, and data frames. Numeric columns are required. The calculation is performed entirely in C++ via Rcpp for speed.

Value

A data frame with descriptive statistics for each variable.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
descdata(data, cols = 5:65)
descdata(data, cols = 5:65, stats = c(2, 7:9))
descdata(data, cols = 5:65, stats = c("na", "min", "max", "IQR"))
```

descplot *View descriptive statistics via plot*

Description

Applies descdata to the selected columns and produces a plot showing the descriptive statistics. The plot is generated using ggplot2 and a suitable melt function.

Usage

```
descplot(data, cols = NULL, stats = 1:9, first = "variables",
         ncol = NULL, num_xaxis = "log", verbose = FALSE)
```

Arguments

data	A data frame containing numeric columns to describe.
cols	Column indices or names of numeric variables. If NULL, all numeric columns are used.
stats	Statistics to plot, as in descdata.
first	Name for the variable identifier column (passed to descdata).
ncol	Number of columns in the facet layout (passed to facet_wrap).
num_xaxis	How to treat numeric column names on the x-axis. Options: "log" (default): if all selected column names are numeric, convert to numeric and use log10 scale; "numeric": convert to numeric but use linear scale; "character", "factor", "keep", or FALSE: keep as character/factor (no conversion).
verbose	Logical; if TRUE, prints timing message.

Details

This function first calls descdata and then melts the result into a long format suitable for ggplot2. If the column names of the original data are numeric, the plot uses a logarithmic x-axis and line geometry; otherwise, a bar plot is produced. The num_xaxis parameter allows overriding the default log transform.

Value

A ggplot object displaying the descriptive statistics.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest. 2. Wickham, H. 2016. ggplot2: elegant graphics for data analysis. Springer-Verlag New York.

Examples

```
descplot(data, cols = 5:65)
descplot(data, cols = 5:65, stats = c("min","max","IQR"), num_xaxis = "numeric")
```

`detect_outliers` *Detect outliers using multiple methods*

Description

Identifies outliers in selected columns using one of several methods: IQR-based, Median Absolute Deviation (MAD), or percentile-based. The function can either return a logical mask indicating outlier positions or replace outliers with NA.

Usage

```
detect_outliers(data, cols = NULL, method = "iqr",
                top = 0.995, bottom = 0.0025, coef = 1.5,
                group = NULL, mask_only = TRUE, verbose = FALSE)
```

Arguments

`data`	A data frame, matrix, or numeric vector.
`cols`	The column indices or names of selected variables. If NULL, all numeric columns are used.
`method`	Detection method. One of "iqr" (default), "mad", "percentile".
`top`	The top percentile threshold for percentile method.
`bottom`	The bottom percentile threshold for percentile method.
`coef`	The coefficient for IQR or MAD method. For IQR, values beyond Q1 - coef*IQR and Q3 + coef*IQR are outliers. For MAD, values with |z| > coef are outliers.
`group`	Optional grouping column for group-wise detection.
`mask_only`	Logical. If TRUE (default), returns a logical matrix of outlier positions. If FALSE, returns data with outliers replaced by NA.
`verbose`	Logical; if `TRUE`, prints progress message.

Details

The IQR method uses Tukey's fences: values outside [Q1 - coef*IQR, Q3 + coef*IQR] are considered outliers. The MAD method uses robust z-scores: \|0.6745*(x - median)/MAD\| > coef. The percentile method flags values above the top percentile or below the bottom percentile.

Value

If mask_only = TRUE, a logical matrix with TRUE indicating outliers. If mask_only = FALSE, a data frame with outliers set to NA.

Examples

```
# Return mask
mask <- detect_outliers(data[1:100, c(1, 4, 17:19)], cols = 3:5, method = "iqr")
# Replace outliers with NA
cleaned <- detect_outliers(data[1:100, c(1, 4, 17:19)], cols = 3:5,
                           method = "mad", mask_only = FALSE)
```

---

detrend_ts *Remove linear trend from time series*

Description

Fits a linear trend (intercept + slope) to each selected numeric column (or vector) and returns the residuals. This is useful for removing long-term trends before correlation analysis.

Usage

```
detrend_ts(data, cols = NULL, date_col = NULL, verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector.
cols	Column indices or names to detrend. If NULL, all numeric columns are used.
date_col	Optional time column (not strictly required for linear detrending).
verbose	Logical; if TRUE, prints progress message.

Value

A data frame or vector with the linear trend removed.

Examples

```
detrend_ts(data[1:100, c(1, 4, 17:19)], cols = 3:5)
```

`drift_detect` *Sensor drift detection*

Description

Detects potential sensor drift by comparing rolling window statistics (mean or standard deviation) to their long-term baseline. Returns a logical matrix indicating time points where the rolling statistic deviates beyond a threshold.

Usage

```
drift_detect(data, cols = NULL, window = 30, threshold = 3,
             method = c("mean", "sd"), group = NULL,
             date_col = NULL, verbose = FALSE)
```

Arguments

`data`	A data frame, matrix, or numeric vector.
`cols`	Column indices or names of numeric variables. If `NULL`, all numeric columns are used.
`window`	Window size for rolling statistics.
`threshold`	Threshold in units of standard deviation. Values with absolute z-score above this are flagged as drift.
`method`	Rolling statistic to use: `"mean"` or `"sd"`.
`group`	Optional grouping column for group-wise detection.
`date_col`	Optional time column (not required for calculation).
`verbose`	Logical; if `TRUE`, prints progress message.

Value

A logical matrix with rows corresponding to observations and columns to the selected variables. `TRUE` indicates suspected drift.

Examples

```
drift_detect(data[1:500, c(1, 4, 17:19)], cols = 3:5, window = 50, threshold = 3)
```

dry_run *Simulate preprocessing and report changes without modifying data*

Description

Simulates the effect of one or more preprocessing steps on a dataset and returns a report describing which variables would be removed, how many observations might be deleted, and other changes. The input data are not altered.

Usage

```
dry_run(data, steps = c("varidele", "obsedele", "outlier"),
        cols = NULL, group = NULL, date_col = NULL,
        fraction = 0.25, top = 0.995, bottom = 0.0025,
        by = "min", half = 30, method_outlier = "iqr", coef = 1.5,
        verbose = FALSE)
```

Arguments

data	A data frame to be simulated.
steps	Character vector of steps to simulate. Currently supported: "varidele", "obsedele", and "outlier".
cols	Column indices or names of numeric variables to consider. If NULL, all numeric columns are used.
group	Optional grouping column for outlier detection.
date_col	Optional time column for observation deletion.
fraction	Missing fraction threshold for variable deletion.
top	Top percentile for outlier detection (percentile method).
bottom	Bottom percentile for outlier detection (percentile method).
by	Time unit for consecutive missing deletion (see obsedele).
half	Half window size in minutes for consecutive missing deletion.
method_outlier	Outlier detection method: "iqr", "mad", or "percentile".
coef	Coefficient for IQR or MAD outlier detection.
verbose	Logical; if TRUE, prints progress message.

Details

This function is useful for understanding the potential impact of preprocessing without changing the dataset. It actually runs varidele, obsedele, and detect_outliers on a copy of the input, and returns precise per-step before/after counts. The caller's data frame is never modified.

Value

A list with the following components:

- `original_n`: number of rows in the original data.
- `original_ncol`: number of columns in the original data.
- `varidele`: a list containing removed column names and count (if step included).
- `obsedele`: a list with `rows_before`, `rows_after`, and `removed`.
- `outlier`: a list with `na_before`, `na_after`, and `added`.
- `final_n`: number of rows remaining after simulation, reflecting both variable deletion and observation deletion.
- `final_ncol`: number of columns remaining after simulation.

Examples

```
dry_run(data[1:200, c(1, 4, 17:19)], cols = 3:5, steps = c("varidele", "obsedele", "outlier"))
```

---

encode_categorical *Encode categorical variables*

Description

Converts factor or character columns into numeric representations using label encoding, frequency encoding, or one-hot encoding. One-hot encoding creates binary indicator columns for each category.

Usage

```
encode_categorical(data, cols = NULL, method = "label",
                   group = NULL, prefix = "enc_", verbose = FALSE)
```

Arguments

`data`	A data frame containing categorical variables.
`cols`	The column indices or names of selected categorical variables. If `NULL` (default), all factor and character columns are selected automatically. An error is raised if any explicitly selected column is not categorical.
`method`	Encoding method. One of `"label"` (integer labels), `"frequency"` (frequency of each category), or `"onehot"` (binary indicator columns).
`group`	Optional grouping column for grouped frequency encoding.
`prefix`	Prefix for one-hot encoded column names (used only when `method = "onehot"`).
`verbose`	Logical; if `TRUE`, prints progress message.

Value

A data frame with encoded variables. For one-hot encoding, original categorical columns are replaced by binary columns.

Examples

```
# Label encoding
encode_categorical(data[1:100, c(1, 4)], cols = 2, method = "label")
# One-hot encoding
encode_categorical(data[1:100, c(1, 4)], cols = 2, method = "onehot")
```

---

filter_high_cor *Remove highly correlated variables*

Description

Identifies and removes numeric columns that have an absolute Pearson correlation above a given threshold. Pairs are evaluated on the subset of rows where both columns are non-missing.

Usage

```
filter_high_cor(data, cols = NULL, cutoff = 0.9,
                method = "pearson", keep = "first",
                verbose = FALSE)
```

Arguments

data	A data frame.
cols	Column indices or names of numeric variables to check. If NULL, all numeric columns are used.
cutoff	Absolute correlation threshold. Variables with correlation > cutoff are candidates for removal.
method	Correlation method. Only "pearson" is currently implemented.
keep	Strategy for keeping one variable from each correlated pair: "first" keeps the first occurrence; "highest_var" keeps the one with higher variance.
verbose	Logical; if TRUE, prints removed column names.

Details

The function computes the pairwise-complete Pearson correlation matrix for the selected numeric columns. For each pair with absolute correlation above cutoff, the column to remove is determined by the keep strategy.

Value

A data frame with highly correlated variables removed.

Examples

```
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- x1 * 0.9 + rnorm(n, sd = 0.2)
x4 <- x1 * 0.2 + rnorm(n)
df <- data.frame(x1, x2, x3, x4)
filter_high_cor(df, cutoff = 0.8)
```

---

filter_low_var *Remove low-variance (near-constant) variables*

Description

Removes numeric columns whose variance (or standard deviation) falls below a specified threshold.

Usage

```
filter_low_var(data, cols = NULL, cutoff = 0.01,
               method = "var", verbose = FALSE)
```

Arguments

data	A data frame.
cols	Column indices or names of numeric variables to check. If NULL, all numeric columns are used.
cutoff	Variance (or standard deviation) threshold. Columns with statistic <= cutoff are removed.
method	Statistic to use: "var" for variance, "sd" for standard deviation.
verbose	Logical; if TRUE, prints removed column names and their variance.

Details

Variance is computed with NA values removed. For method = "sd", the standard deviation is compared **directly** against cutoff; no internal squaring is performed.

Value

A data frame with low-variance columns removed.

Examples

```
data <- data.frame(
  id = 1:100,
  const = rep(5, 100),
  noise = rnorm(100, sd = 0.005)
)
filter_low_var(data, cutoff = 0.001)
```

`impute_missing` *Impute missing values*

Description

Impute missing values

Usage

```
impute_missing(
  data,
  cols = NULL,
  method = "linear",
  group = NULL,
  date_col = NULL,
  max_gap = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A data frame or matrix.
`cols`	Columns to impute. If `NULL`, all numeric columns are used.
`method`	"linear", "locf", "nocb", "mean", or "median".
`group`	Optional grouping column.
`date_col`	Time column.
`max_gap`	Not implemented; ignored with a warning.
`verbose`	Logical.

Value

A data frame with imputed values.

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
impute_missing(data[1:100, c(1, 4, 17:19)], cols = 3:5, method = "locf")
```

log_returns *Logarithmic returns for financial time series*

Description

Computes the logarithmic return $\log(x_t/x_{t-1})$ for selected numeric columns. Used extensively in financial time series analysis. Group-wise computation is supported.

Usage

```
log_returns(data, cols = NULL, group = NULL, verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector.
cols	Column indices or names of numeric variables. If NULL, all numeric columns are used.
group	Optional grouping column for group-specific returns.
verbose	Logical; if TRUE, prints progress message.

Details

The function only computes a return when both the current and previous values are positive. Otherwise, the result is NA.

Value

A vector or data frame with logarithmic returns. The first observation of each group (or the vector) is NA.

Examples

```
log_returns(data[1:200, c(1, 4, 17:19)], cols = 3:5)
```

melt *Fast wide-to-long data reshaping with flexible ID/measure specification*

Description

Transforms a wide-format data frame into a long-format data frame using a C++ backend with SIMD and optional OpenMP parallelism. Supports automatic inference of ID columns, user-specified ID or measure variables, custom column names, and two storage layouts (row-major or column-major).

Usage

```
melt(data, id = NULL, measure.vars = NULL,
     variable.name = "variable", value.name = "value",
     na.rm = FALSE, cores = NULL, major = NULL,
     as.factor = NULL,
     verbose = FALSE, parallel_threshold = 5e6, id.vars = NULL)
```

Arguments

data	A data frame to reshape. Numeric, integer, logical, character, and factor columns are accepted as id columns. Factor columns used as measure.vars cannot be cast to numeric, so their cells are written as NA in the value column; pass measure.vars as character or numeric to avoid this.
id	Identifier columns. Character names, integer indices, a logical mask, or NULL. If NULL and measure.vars is also NULL, ID columns are inferred automatically: non-numeric, non-integer, non-logical columns and factors are treated as IDs.
measure.vars	Measure columns. Character names, integer indices, a logical mask, or NULL. If supplied, all other columns become IDs. Only one of id and measure.vars should be supplied.
variable.name	Name of the new column that stores the original column names of the measure variables. Default "variable". See as.factor for its type.
value.name	Name of the new numeric column that stores the melted values. Default "value".
na.rm	Logical; if TRUE, rows where the value column is NA are removed. Default FALSE.
cores	Number of OpenMP threads. NULL (default) lets the C++ backend pick a value based on data size. The global option dataprep.cores overrides the automatic choice.
major	Memory layout. NULL (default) is equivalent to "col": column-major, identical to reshape2::melt (rows grouped by variable). "row" produces a row-major result matching the row order of tidyr::pivot_longer (rows grouped by id). There is no automatic switching based on the input shape.
as.factor	Whether the variable column is a factor. NULL (default) uses TRUE when major is "col" and FALSE when major is "row". Explicit TRUE / FALSE overrides that default. The return value is always a plain data.frame; no tibble attributes are attached.
verbose	Logical; if TRUE, prints timing messages.
parallel_threshold	Minimum number of output elements, i.e.\ nrow(data) * length(measure.vars), before automatic parallelism is enabled. Default 5e6.
id.vars	Alias of id, provided for reshape2 / data.table compatibility. Supplying both is an error.

Details

melt() is the fast, drop-in replacement for reshape2::melt and tidyr::pivot_longer. The C++ backend (melt_cpp) has been benchmarked against **all seven major alternatives** in the R and

Python ecosystems: reshape2, data.table, tidyr, pandas, polars, dask, and duckdb. Every cell is measured with a C++ steady-clock timer and an adaptive times rule.

Speed-up relative to each competitor spans 0.6x to 1628.9x. The median across all tested cells and all competitors is 11.3x on Ubuntu 25.10 and 5.6x on Windows 11 Pro for Workstations; the mean is 67.8x and 46.6x, respectively. The largest gaps appear on wide tables (many value columns, few rows); the smallest gaps appear on small tables where initialization overhead dominates.

Representative cells (means; numbers in parentheses are the speed-up of melt() relative to that competitor) taken from the Ubuntu 25.10 host:

- **1e6 rows, 1 id, 9 value columns** — 3.474 ms vs 9.646 ms (data.table, 2.8x), 80.014 ms (tidyr, 23.0x), 648.895 ms (duckdb, 186.8x).
- **1e7 rows, 1 id, 9 value columns** — 33.412 ms vs 365.065 ms (data.table, 10.9x), 1083.987 ms (tidyr, 32.4x), 6389.564 ms (duckdb, 191.2x).
- **1e3 rows, 1 id, 10000 value columns** — 2.295 ms vs 12.415 ms (data.table, 5.4x), 19.408 ms (polars, 8.5x), 3737.866 ms (dask, 1628.9x).
- **1e3 rows, 10 id, 10000 value columns** — 25.009 ms vs 71.605 ms (polars, 2.9x), 182.019 ms (data.table, 7.3x), 23777.051 ms (dask, 950.7x).

The speed-up comes from three design choices:

1. **Two layout paths, selected by major.** The default is column-major. On the column-major path, each output column is written as one contiguous memcpy of the input column — the fastest possible pattern. On the row-major path, tiles of eight rows are transposed in registers with _mm512_shuffle_f64x2.
2. **SIMD streaming stores.** For large outputs, _mm512_stream_pd and _mm256_stream_si256 write directly to memory, bypassing the CPU cache and avoiding the cache pollution that would otherwise evict useful input data. A _mm_sfence() is issued at the end of each streaming region.
3. **Hugepage hint for large outputs.** Output vectors larger than 512 KB are allocated through Rf_allocVector() and then hinted with madvise(MADV_HUGEPAGE), so the kernel can back them with 2 MB pages. This reduces first-touch page faults on the 1e8-row case. There is no custom allocator and no free pool.

Two fast paths for small inputs are used. Tiny inputs (n <= 2048, n_meas <= 64, n_id <= 8) route to melt_tiny_cpp; small inputs (total <= 131072, n_meas <= 256) route to melt_small_cpp. Both skip hugepage hinting, OpenMP setup, and thread-cap detection, which keeps melt() the fastest engine even on tables of a few thousand rows.

The output is identical to reshape2::melt, data.table::melt, tidyr::pivot_longer, pandas.melt, polars.unpivot, dask, and duckdb UNPIVOT on every tested shape, within tol = 1e-12. See vignette("dataprep-melt-dcast") for the implementation notes and vignette("dataprep-performance") for the full benchmark tables, including mean, median, and the per-competitor gradient at every tested scale.

Value

A data frame in long format. When `na.rm = FALSE`, it has `nrow(data) * length(measure.vars)` rows. With `na.rm = TRUE`, rows whose value is `NA` or `NaN` are dropped, so the row count is smaller and not known in advance. Columns are the ID columns, a column named `variable.name` (a factor when `as.factor` resolves to `TRUE`, otherwise a character vector), and a numeric column named `value.name`.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. 2. Eddelbuettel, D. and Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, 40(8), 1–18. 3. Liang, C.-S., Wu, H., Li, H.-Y., Zhang, Q., Li, Z. & He, K.-B. (2020). Efficient data preprocessing, episode classification, and source apportionment of particle number concentrations. *Science of the Total Environment*, 741, 140923. doi:10.1016/j.scitotenv.2020.140923

See Also

`dcast` for the inverse operation. `vignette("dataprep-melt-dcast")` for usage notes and implementation. `vignette("dataprep-performance")` for benchmark tables.

Examples

```
## --- Basic usage ------------------------------------------------

df <- data.frame(id = 1:5,
                 category = factor(letters[1:5]),
                 v1 = rnorm(5), v2 = rnorm(5))

# Automatic ID inference: `id` and `category` are non-numeric
melt(df)

# Explicit ID columns
melt(df, id.vars = c("id", "category"))

# Explicit measure columns
melt(df, measure.vars = c("v1", "v2"))


## --- Custom column names and na.rm ------------------------------

df2 <- data.frame(id = 1:3, x = c(1, NA, 3), y = c(4, 5, NA))

melt(df2, id.vars = "id",
     variable.name = "var", value.name = "val",
     na.rm = TRUE)
```

```
## --- Two memory layouts ----------------------------------------

# major = NULL (default): column-major (reshape2-compatible)
# major = "col" : column-major, reshape2-compatible
# major = "row" : row-major,    tidyr-compatible
df_wide <- data.frame(id = 1:100,
                      matrix(rnorm(100 * 50), ncol = 50))

res_default <- melt(df_wide, id.vars = "id")
res_col     <- melt(df_wide, id.vars = "id", major = "col")
res_row     <- melt(df_wide, id.vars = "id", major = "row")

# The two layouts produce the same content in a different order
identical(sort(res_col$value), sort(res_row$value))

## --- as.factor controls the variable column type ------------------

# Default: factor for "col", character for "row"
is.factor(res_col$variable)                     # TRUE
is.character(res_row$variable)                  # TRUE

# Explicit override
res_row_fac <- melt(df_wide, id.vars = "id", major = "row",
                    as.factor = TRUE)
is.factor(res_row_fac$variable)                 # TRUE
```

`na_diagnose` *Diagnose missing value patterns in data*

Description

Provides a summary of missing values for selected columns, including the number and fraction of missing values, the number of consecutive missing runs, and the longest consecutive missing run length (in number of observations).

Usage

```
na_diagnose(data, cols = NULL, date_col = NULL, verbose = FALSE)
```

Arguments

`data`	A data frame or matrix containing variables to diagnose.
`cols`	The column indices or names of selected variables. If `NULL`, all numeric columns are used.
`date_col`	Reserved for future use; currently ignored.
`verbose`	Logical; if `TRUE`, prints progress messages.

Value

A data frame with columns: `variable`, `n` (total observations), `na` (number of missing values), `na_frac` (missing fraction), `na_runs` (number of consecutive missing runs), `max_run` (longest consecutive missing run length in number of observations).

Examples

```
na_diagnose(data[1:500, c(1, 4, 17:19)], cols = 3:5)
```

`obsedele` *Delete observations with excessive consecutive missing values*

Description

Delete observations with excessive consecutive missing values

Usage

```
obsedele(
  data,
  cols = NULL,
  group = NULL,
  by = "min",
  half = 30,
  date_col = NULL,
  cores = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A data frame.
`cols`	Columns to check. If `NULL`, all numeric columns are used.
`group`	Optional grouping column.
`by`	Time unit used only to validate the internal `step_sec`. The 0.1.8 anchor-based scan does not use a regular grid, so this argument does not affect the result.
`half`	Half window size in minutes.
`date_col`	Time column.
`cores`	Number of CPU cores.
`verbose`	Logical.

Details

For every missing value in each selected column, the C++ backend computes the time distance to the nearest non-missing anchor on the left and on the right. A row is deleted when **any** selected column has **both** distances exceed `half` minutes. When a run touches the series boundary, the missing side is treated as `+Inf`, so boundary rows are only deleted when the surviving side is also too far away.

This is a change from dataprep 0.1.5, which collapsed all selected columns into one long vector before computing missing runs. The old approach merged NA runs across columns and over-deleted boundary rows. See `vignette("dataprep-migration")` for the upgrade guide.

Value

A data frame with rows removed.

Boundary behaviour

The comparison is inclusive: if an anchor is exactly `half` minutes away, the row is retained. On the SMEAR I Varrio 2025 full-year dataset this rule retains three rows that 0.1.5 removed.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

See Also

varidele, condextr, na_diagnose, dataprep.

Examples

```
df <- data.frame(
  date  = as.POSIXct("2024-01-01 00:00:00", tz = "UTC") + 0:9 * 600,
  group = rep(1L, 10),
  x     = c(1, NA, NA, NA, 5, NA, NA, 2, NA, 3)
)
obsedele(df, cols = "x", group = "group", half = 30)
```

optisolu *Find optimal combination of interval and times for condextr*

Description

Searches over a grid of `interval` and `times` values to find the combination that yields better outlier removal than the traditional percentile method, based on sample deletion ratio (SDR), outlier removal ratio (ORR), and signal-to-noise ratio (SNR).

Usage

```
optisolu(data, cols = NULL, group = NULL, interval = 35, times = 10,
top = 0.995, top.error = 0.1, top.magnitude = 0.2,
bottom = 0.0025, bottom.error = 0.2, bottom.magnitude = 0.4,
by = "min", half = 30, date_col = NULL, cores = NULL,
verbose = FALSE)
```

Arguments

`data`	A data frame containing numeric columns and optionally a grouping column.
`cols`	Column indices or names of numeric variables.
`group`	Grouping column index or name.
`interval`	Maximum interval value to test (from 1 to `interval`).
`times`	Maximum times value to test (from 1 to `times`).
`top`	Top percentile threshold.
`top.error`	Error margin for the top threshold.
`top.magnitude`	Magnitude margin for the top threshold.
`bottom`	Bottom percentile threshold.
`bottom.error`	Error margin for the bottom threshold.
`bottom.magnitude`	Magnitude margin for the bottom threshold.
`by`	Time unit used when computing observation deletion.
`half`	Half window size in minutes for observation deletion.
`date_col`	Time column index or name.
`cores`	Number of CPU cores.
`verbose`	Logical; if `TRUE`, prints progress message.

Details

This function can be computationally intensive. It compares `condextr` results with `percoutl` results.

Value

A data frame with columns: `case`, `interval`, `times`, `sdr`, `orr`, `snr`, `index`, `relaindex`, and `optimal` (logical).

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
optisolu(data[1:50, c(1, 4, 18:19)], cols = 3:4, group = 2,
         interval = 2, times = 1)
```

`percdata` *Calculate top and bottom percentiles of selected variables*

Description

Computes percentiles (quantiles) for selected numeric columns. By default, six bottom percentiles (0th to 0.5th) and six top percentiles (99.5th to 100th) are calculated. The function supports grouping.

Usage

```
percdata(data, cols = NULL, group = NULL, diff = 0.1,
        part = "both", na.rm = TRUE, verbose = FALSE)
```

Arguments

`data`	A data frame, matrix, or numeric vector. If a vector, a data frame is created internally.
`cols`	Column indices or names of numeric variables. If `NULL`, all numeric columns are used.
`group`	Optional grouping column index or name. If provided, percentiles are computed separately for each group.
`diff`	Common difference between quantile probabilities. Default is 0.1.
`part`	Which parts to return: `"both"` (default), `"bottom"`, or `"top"`. For backward compatibility, `part` also accepts the integer codes `2` (`"both"`), `0` (`"bottom"`), and `1` (`"top"`).
`na.rm`	Logical; if `TRUE`, NA values are removed before quantile computation.
`verbose`	Logical; if `TRUE`, prints timing message.

Details

The function uses `stats::quantile()` on each selected column. For grouped data, percentiles are computed within each group.

Value

A data frame with rows corresponding to percentiles and columns corresponding to variables (plus a `percentile` column). If grouped, an additional grouping column is present.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
percdata(data, cols = 27:61, group = 4)
```

---

percoutl *Traditional percentile-based outlier removal*

Description

Removes values above a top percentile and below a bottom percentile. Thresholds are computed from quantiles. Optionally, observation deletion based on consecutive missing values can be performed after outlier removal.

Usage

```
percoutl(data, cols = NULL, group = NULL, top = 0.995,
         bottom = 0.0025, by = "min", half = 30,
         date_col = NULL, cores = NULL, verbose = FALSE)
```

Arguments

`data`	A data frame or matrix. If a numeric vector is supplied, only outlier marking (no observation deletion) is performed because there is no time axis.
`cols`	Column indices or names of numeric variables. If `NULL`, all numeric columns are used.
`group`	Optional grouping column for group-wise outlier removal.
`top`	Top percentile threshold. Values above this quantile are removed.
`bottom`	Bottom percentile threshold. Values below this quantile are removed.

by	Time unit for observation deletion (see obsedele).
half	Half window size, in minutes, for consecutive missing deletion.
date_col	Time column index or name. If NULL, automatically detected.
cores	Number of OpenMP threads. Passed to obsedele.
verbose	Logical; if TRUE, prints timing and deletion counts.

Details

This method is a one-size-fits-all approach and may remove non-outliers or fail to remove some outliers. It is provided for comparison with condextr, which uses a point-by-point weighted conditional extremum criterion.

Value

A data frame with outliers removed.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
percoutl(obsedele(data[1:500, c(1, 4, 17:19)], cols = 3:5, group = 2),
         cols = 3:5, group = 2)
```

---

percplot *Plot top and bottom percentiles of selected variables*

Description

Plots percentiles computed by percdata using ggplot2.

Usage

```
percplot(data, cols = NULL, group = NULL, diff = 0.1,
         part = "both", ncol = NULL, num_xaxis = "auto",
         verbose = FALSE)
```

Arguments

data	A data frame.
cols	Column indices or names of numeric variables.
group	Grouping column.
diff	Difference between quantile probabilities.
part	Which part to plot: "both", "bottom", or "top". For backward compatibility, part also accepts the integer codes 2 ("both"), 0 ("bottom"), and 1 ("top").
ncol	Number of columns in facet layout.
num_xaxis	How to treat numeric column names on the x-axis. "auto": use "log" if column names are numeric and evenly spaced on a log scale with a range ratio >= 1000, otherwise keep them as factor levels. "log" or "numeric": force the corresponding scale. "character", "factor", "keep", or FALSE: keep as character/factor.
verbose	Logical; if TRUE, prints timing message.

Value

A ggplot object.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest. 2. Wickham, H. 2016. ggplot2: elegant graphics for data analysis. Springer-Verlag New York.

Examples

```
percplot(data, cols = 5:65, group = 4)
percplot(data, cols = 5:65, group = 4, num_xaxis = "numeric")
```

---

phys_filter *Physical limit filtering*

Description

Sets values outside user-specified physical limits to NA. This is commonly used to remove impossible measurements, such as relative humidity outside 0–100% or negative concentrations. The function supports per-column limits and grouping.

Usage

```
phys_filter(data, cols = NULL, min_val = NULL, max_val = NULL,
            group = NULL, date_col = NULL, verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector.
cols	Column indices or names of numeric variables to filter. If NULL, all numeric columns are used.
min_val	Lower limit(s). If a single value is given, it is applied to all selected columns. Otherwise, provide a vector with length equal to the number of selected columns.
max_val	Upper limit(s). Same behavior as min_val.
group	Optional grouping column (currently not used but reserved for consistency).
date_col	Optional time column (not used).
verbose	Logical; if TRUE, prints progress message.

Details

Values strictly less than min_val or strictly greater than max_val are set to NA. If min_val or max_val is NULL, that bound is not checked.

Value

A data frame or vector with out-of-range values set to NA.

Examples

```
phys_filter(data[1:100, c(1, 4, 17:19)], cols = 3:5, min_val = 0, max_val = 1000)
```

---

prep_fit *Build a preprocessing plan on training data to prevent data leakage*

Description

Computes and stores all preprocessing parameters (such as missing fractions, outlier thresholds, imputation methods, and scaling statistics) using only the training data. The returned plan object can then be applied to new data via prep_transform without re-estimating any parameters, ensuring that information from the test set does not leak into the preprocessing steps.

Usage

```
prep_fit(data, steps = c("varidele", "obsedele", "outlier", "impute", "scale"),
         cols = NULL, group = NULL, date_col = NULL,
         fraction = 0.25, top = 0.995, bottom = 0.0025,
         by = "min", half = 30, method_outlier = "iqr",
         coef = 1.5, method_impute = "linear",
         scale_method = "zscore", verbose = FALSE)
```

Arguments

data	A data frame containing the training data. All preprocessing parameters are estimated from this data only.
steps	A character vector specifying the preprocessing steps to include. Available steps are "varidele", "obsedele", "outlier", "impute", and "scale". The default is all five steps in that order.
cols	Column indices or names of numeric variables to be processed. If NULL, all numeric columns are selected.
group	Optional grouping column index or name used for grouped outlier detection, imputation, or scaling.
date_col	Optional time column index or name used by obsedele for time-aware observation deletion.
fraction	Missing fraction threshold for variable deletion in the "varidele" step.
top	Top percentile used by the percentile outlier detection method.
bottom	Bottom percentile used by the percentile outlier detection method.
by	Time extension unit for observation deletion (see obsedele).
half	Half window size in minutes for consecutive missing value deletion (see obsedele).
method_outlier	Outlier detection method: "iqr", "mad", or "percentile".
coef	Coefficient for IQR or MAD outlier detection.
method_impute	Missing value imputation method: "linear", "locf", "nocb", "mean", or "median".
scale_method	Scaling method used in the "scale" step: "zscore", "minmax", or "robust". Other values fall back to identity scaling (center = 0, scale = 1).
verbose	Logical; if TRUE, prints progress messages.

Details

The function follows the *fit-transform* paradigm. All thresholds, means, standard deviations, and group-wise statistics are computed solely from data (training set). Applying these saved parameters to test data with prep_transform guarantees that no information from the test set influences the preprocessing, thereby preventing data leakage.

Value

A list of class prep_plan containing all parameters learned from the training data. The list includes:

- steps: the ordered preprocessing steps.
- params: a list of parameters such as missing fractions, outlier thresholds, imputation method, and scaling center/scale values.
- data_info: original column names, selected column indices, and other metadata.
- final_data: the fully preprocessed training data (optional, useful for inspection).

Examples

```
# Build a preprocessing plan using the first 100 rows as training data
plan <- prep_fit(data[1:100, c(1, 4, 47:49)], cols = 3:5, group = 2)
# Apply the plan to new data
newdata <- data[101:200, c(1, 4, 47:49)]
cleaned <- prep_transform(plan, newdata)
```

---

prep_transform *Apply a preprocessing plan to new data*

Description

Uses a preprocessing plan created by `prep_fit` to transform new data. All parameters (thresholds, statistics, etc.) are taken from the plan and are not re-estimated, thus ensuring that the same preprocessing logic is applied consistently and without data leakage.

Usage

```
prep_transform(plan, newdata, verbose = FALSE)
```

Arguments

`plan`	A preprocessing plan object returned by `prep_fit`.
`newdata`	A data frame containing the new data (e.g., test set) to be transformed. It must contain the same columns as the original training data used to build the plan.
`verbose`	Logical; if `TRUE`, prints progress messages.

Details

The function iterates through the steps stored in `plan` and applies the corresponding operations using the saved parameters. For example, if the `"outlier"` step used group-specific thresholds, those exact thresholds are applied to the new data grouped in the same way.

Value

A data frame after applying the preprocessing steps specified in `plan`. The output contains the same variables as the training data after preprocessing.

Examples

```
# Build plan
plan <- prep_fit(data[1:100, c(1, 4, 47:49)], cols = 3:5, group = 2)
# Transform new data
newdata <- data[101:200, c(1, 4, 47:49)]
cleaned <- prep_transform(plan, newdata)
```

remove_diurnal_cycle *Remove diurnal cycle*

Description

Removes the mean diurnal (or other periodic) cycle by subtracting the average value for each time period (hour, month, or day). The grouping is based on a time column.

Usage

```
remove_diurnal_cycle(data, cols = NULL, date_col = NULL,
                     by = "hour", verbose = FALSE)
```

Arguments

data	A data frame with a time column.
cols	Column indices or names to adjust.
date_col	Time column index or name.
by	Period for cycle removal: "hour", "month", or "day".
verbose	Logical; if TRUE, prints progress message.

Value

A data frame with diurnal cycle removed.

Examples

```
remove_diurnal_cycle(data[1:200, c(1, 4, 17:19)], cols = 3:5, date_col = 1, by = "hour")
```

resample_time *Resample time series to a coarser period*

Description

Aggregates time series data to a specified period (hour, day, or month) using a chosen summary statistic. The function requires a time column.

Usage

```
resample_time(data, cols = NULL, date_col = NULL,
              period = "hour", fun = "mean", na.rm = TRUE,
              verbose = FALSE)
```

Arguments

data	A data frame with a time column.
cols	Column indices or names to aggregate.
date_col	Time column index or name.
period	Target period: "hour", "day", or "month".
fun	Aggregation function: "mean", "sum", "min", "max", "median", or "sd".
na.rm	Logical; whether to remove NA values before aggregation.
verbose	Logical; if TRUE, prints progress message.

Value

A data frame with aggregated time series.

Examples

```
resample_time(data[1:500, c(1, 4, 17:19)], cols = 3:5, date_col = 1, period = "hour", fun = "mean")
```

---

roll_apply *Apply rolling window statistics*

Description

Computes rolling statistics (mean, sd, median, sum, min, max, var) over a moving window of fixed length. Supports group-wise computation and various alignments.

Usage

```
roll_apply(data, cols = NULL, window = 3, method = "mean",
           align = "right", group = NULL, date_col = NULL,
           verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector.
cols	Column indices or names to process.
window	Window size (number of observations).
method	Statistic to compute: "mean", "sd", "median", "sum", "min", "max", or "var".
align	Alignment of window: "right", "left", or "center".
group	Optional grouping column for group-wise rolling computation.
date_col	Not used. Present for interface consistency.
verbose	Logical; if TRUE, prints progress message.

Value

A data frame or vector with rolling statistics.

Examples

```
roll_apply(data[1:100, c(1, 4, 17:19)], cols = 3:5, window = 5, method = "mean")
```

`sample_data` *Random sampling with optional stratification*

Description

Random sampling with optional stratification

Usage

```
sample_data(
  data,
  group = NULL,
  size = NULL,
  frac = NULL,
  replace = FALSE,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A data frame.
`group`	Grouping column.
`size`	Sample size per group (see Details).
`frac`	Sampling fraction per group (see Details).
`replace`	Sample with replacement.
`seed`	Random seed. If `NULL`, the current RNG state is used; if supplied, `set.seed(seed)` is called before sampling.
`verbose`	Logical.

Details

If `group` is `NULL`, either `size` (absolute number of rows) or `frac` (fraction of `nrow(data)`) must be supplied.

If `group` is provided, sampling is performed separately within each group. With `size`, every group contributes the same absolute number of rows (capped at the group's own size unless `replace = TRUE`). With `frac`, every group contributes its own `round(frac * group_size)` rows (at least 1), so groups of different sizes are sampled proportionally. If `group` is provided but both `size` and `frac` are `NULL`, each group contributes exactly one row.

Value

A sampled data frame.

Examples

```
sample_data(mtcars, frac = 0.5)
sample_data(mtcars, size = 3, group = "cyl", seed = 123)

# frac is per-group: large groups contribute more rows
n_by_cyl <- function(d) table(d$cyl)
n_by_cyl(sample_data(mtcars, frac = 0.5, group = "cyl", seed = 1))
```

---

season_flag *Season flag*

Description

Creates season, month, or quarter indicator from a time column. Useful for grouped analysis and preprocessing by season.

Usage

```
season_flag(data, date_col = NULL, type = c("season", "month", "quarter"),
            verbose = FALSE)
```

Arguments

data	A data frame with a time column.
date_col	Time column index or name. If NULL, the first column matching date is used.
type	Type of flag: "season" (winter, spring, summer, autumn), "month" (abbreviated month), or "quarter" (Q1-Q4).
verbose	Logical; if TRUE, prints progress message.

Value

A character vector or factor containing the requested flag.

Examples

```
season_flag(data[1:200, c(1, 4, 17:19)], date_col = 1, type = "season")
```

shorvalu *Interpolation with values to refer to within short periods*

Description

Performs linear interpolation of missing values within short time segments, constrained by the availability of observed values in the same segment. This method ensures that interpolation is based on nearby valid data.

Usage

```
shorvalu(data, cols = NULL, intervals = 30, units = "mins",
         date_col = NULL, cores = NULL, verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector. If a vector, simple linear interpolation is applied.
cols	Column indices or names of numeric variables to interpolate. Must be continuous.
intervals	Time gap used to define short periods.
units	Time unit for intervals: "secs", "mins", "hours", "days", "weeks".
date_col	Time column index or name. If NULL, automatically detected.
cores	Number of CPU cores for parallel computation (passed to C++).
verbose	Logical; if TRUE, prints timing and interpolation counts.

Value

A data frame with missing values filled by linear interpolation within each period.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
shorvalu(condextr(obsedele(data[1:250, c(1, 4, 17:19)], cols = 3:5, group = 2),
                  cols = 3:5, group = 2),
         cols = 3:5)
```

`transform_data` *Transform and standardize numeric variables*

Description

Applies a transformation (log, sqrt, inverse, Box-Cox, Yeo-Johnson) or standardization (z-score, centering, scaling, min-max, robust scaling) to selected numeric columns. Group-wise transformation is supported.

Usage

```
transform_data(data, cols = NULL, method = "log", group = NULL,
               lambda = 1.0, verbose = FALSE)
```

Arguments

`data`	A data frame or matrix containing numeric variables.
`cols`	The column indices or names of selected variables. If NULL, all columns are used.
`method`	Transformation method. One of `"log"`, `"log1p"`, `"sqrt"`, `"inverse"`, `"boxcox"`, `"yeojohnson"`, `"zscore"`, `"center"`, `"scale"`, `"minmax"`, `"robust"`.
`group`	Optional grouping column for group-wise transformation.
`lambda`	Parameter for Box-Cox and Yeo-Johnson transformations (default 1.0).
`verbose`	Logical; if `TRUE`, prints progress message.

Details

For `"boxcox"`, values must be positive. For `"yeojohnson"`, both positive and negative values are allowed. Standardization methods ignore NA values.

Value

A data frame with transformed variables.

Examples

```
# Log transformation
transform_data(data[1:100, c(1, 4, 17:19)], cols = 3:5, method = "log")
# Z-score standardization by group
transform_data(data[1:100, c(1, 4, 17:19)], cols = 3:5, method = "zscore", group = 2)
```

validate_data *Validate data against a set of rules*

Description

Checks whether data conforms to user-specified rules such as column type, value range, uniqueness, and presence of missing values. Returns a report indicating which rules passed and which failed.

Usage

```
validate_data(data, rules, verbose = FALSE)
```

Arguments

data	A data frame to validate.
rules	A list of rules. Each rule is a list with components: column (column name), type (one of "numeric", "integer", "character", "factor"), min, max, unique (logical), and na_allowed (logical).
verbose	Logical; if TRUE, prints progress message.

Value

A data frame with columns: rule, column, passed (logical), and message. If a rule passes, message is "OK"; otherwise it contains a description of the failure.

Examples

```
rules <- list(
  list(column = "3.98", type = "numeric", min = 0, na_allowed = TRUE),
  list(column = "monthyear", type = "character", unique = FALSE)
)
validate_data(data[1:50, c(1, 4, 17:19)], rules)
```

varidele *Delete variables containing too many missing values*

Description

Removes any numeric column whose missing value proportion exceeds a given threshold.

Usage

```
varidele(data, cols = NULL, fraction = 0.25, verbose = FALSE)
```

Arguments

`data`	A data frame or matrix containing numeric columns.
`cols`	Column indices or names of the numeric columns to consider. If `NULL`, all columns are used.
`fraction`	Missing proportion threshold. Columns with a missing fraction `>= fraction` are deleted. Default 0.25.
`verbose`	Logical; if `TRUE`, prints timing and deleted variables.

Details

The missing fraction is computed after excluding rows where all selected variables are NA. Any column whose missing fraction reaches the threshold is removed, regardless of its position in the selected block.

Value

A data frame with the retained columns.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

References

1. Example data is from https://smear.avaa.csc.fi/download. It includes particle number concentrations in SMEAR I Varrio forest.

Examples

```
varidele(data, cols = 5:65)
```

`winsorize` *Winsorize outliers by capping extreme values*

Description

Replaces extreme values in selected columns with the specified top and bottom quantile thresholds, rather than deleting them. This is useful for reducing the influence of outliers while preserving sample size.

Usage

```
winsorize(data, cols = NULL, top = 0.995, bottom = 0.0025,
          group = NULL, verbose = FALSE)
```

Arguments

data	A data frame, matrix, or numeric vector.
cols	The column indices or names of selected variables. If NULL, all columns are used.
top	The top quantile threshold. Values above this are capped at the threshold.
bottom	The bottom quantile threshold. Values below this are capped at the threshold.
group	Optional grouping column for group-wise winsorization.
verbose	Logical; if TRUE, prints progress message.

Value

A data frame with extreme values capped.

Examples

```
winsorize(data[1:100, c(1, 4, 17:19)], cols = 3:5)
```

---

zerona *Turn zeros to missing values*

Description

Converts zero values to NA. Useful when zero represents an invalid measurement or when a logarithmic scale is desired.

Usage

```
zerona(x)
```

Arguments

x	A data frame, matrix, or vector containing zeros.

Value

An object of the same class with zeros replaced by NA.

Author(s)

Chun-Sheng Liang <chun-shengliang@qq.com>

Examples

```
zerona(0:5)
zerona(cbind(a = 0:5, b = c(6:10, 0)))
```

Index

∗ **datasets**
  data, 11
  data1, 14
∗ **manip**
  dcast, 19
  melt, 35
∗ **package**
  dataprep-package, 3

balance_panel, 5
bin_data, 6

clean_strings, 7
condextr, 9, *14*, *41*, *42*, *45*
create_lags, 10

data, 11, *14*, *15*
data1, *13*, 14
data_report, *15*, 17
dataprep, *13*, *14*, 15, *41*
dataprep-package, 3
day_night_flag, 18
dcast, 19, *38*
decompose_ts, 23
deduplicate, 24
descdata, *15*, 25
descplot, *15*, 26
detect_outliers, 27
detrend_ts, 28
drift_detect, 29
dry_run, 30

encode_categorical, 31

filter_high_cor, 32
filter_low_var, 33

gsub, *8*

impute_missing, 34

log_returns, 35

melt, *19–21*, 35

na_diagnose, *15*, *17*, 39, *41*

obsedele, *14*, 40
optisolu, 42

percdata, *15*, 43
percoutl, *42*, 44
percplot, *15*, 45
phys_filter, 46
prep_fit, 47
prep_transform, 49

remove_diurnal_cycle, 50
resample_time, 50
roll_apply, 51

sample_data, 52
season_flag, 53
shorvalu, *14*, 54

transform_data, 55
trimws, *7*

validate_data, 56
varidele, *14*, *41*, 56

winsorize, 57

zerona, 58