Package 'basetable'

September 14, 2026

Title Fast and Memory-Efficient Base R Table Manipulation

Version 1.4.1

Description A tabular data manipulation, exploration and validation toolkit with a base R-style interface (subset, transform, aggregate, merge, split) and no external computation dependency. Grouping, joins, ordering, filtering, reshaping and delimited-file reading run in a bundled 'C++' engine that uses multiple threads for the heavier operations. Grouped reducers accumulate in compiled code without materialising intermediate columns, so grouped aggregation and counting allocate close to nothing. Results are returned as an ordinary data frame with a light 'basetable' class.

License MIT + file LICENSE

URL https://github.com/ielbadisy/basetable

BugReports https://github.com/ielbadisy/basetable/issues

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.2.0)

Imports parallel, stats, utils

Suggests bench, data.table, dplyr, ggplot2, knitr, rmarkdown, scales, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements C++17

NeedsCompilation yes

Author Imad El Badisy [aut, cre]

Maintainer Imad El Badisy <elbadisyimad@gmail.com>

Repository CRAN

Date/Publication 2026-09-13 23:20:13 UTC

Contents

`basetable-package` *Base-Faithful Tabular Data Tools*

Description

The **basetable** package provides a compact set of base-style data manipulation and exploratory data analysis helpers backed by a native C++ table engine.

Details

Core design rules:

- exported data verbs take the main data object as their first argument
- grouping is explicit with `by =`
- outputs are plain data frames or lists in most cases
- interfaces are intended to work naturally in nested calls and native `|>` pipelines

---

`adddays` *Add days to a date*

Description

Add days to a date

Usage

```
adddays(x, n)
```

Arguments

`x`	An atomic vector.
`n`	Integer count.

Value

A Date vector.

addedrows *Rows present in* new *but not in* old

Description

Rows present in new but not in old

Usage

```
addedrows(old, new, by = NULL)
```

Arguments

old	Baseline data.frame ("before" state).
new	Updated data.frame ("after" state), or replacement values when used for value substitution.
by	Character vector of column names identifying groups or join keys.

Value

The added rows.

addmonths *Add months to a date*

Description

Add months to a date

Usage

```
addmonths(x, n, invalid = c("previous", "next", "missing", "error"))
```

Arguments

x	An atomic vector.
n	Integer count.
invalid	How to handle an invalid resulting date.

Value

A Date vector.

addweeks *Add weeks to a date*

Description

Add weeks to a date

Usage

```
addweeks(x, n)
```

Arguments

x	An atomic vector.
n	Integer count.

Value

A Date vector.

addyears *Add years to a date*

Description

Add years to a date

Usage

```
addyears(x, n, invalid = c("previous", "next", "missing", "error"))
```

Arguments

x	An atomic vector.
n	Integer count.
invalid	How to handle an invalid resulting date.

Value

A Date vector.

aggregate *Aggregate values by group*

Description

Compute grouped summaries for one or more value columns. When `data` is a single file path, the file is scanned once and only the grouping and value fields are extracted, without materialising the other columns.

Usage

```
aggregate(data, by, value = NULL, fun, ..., na.rm = FALSE, sort = TRUE)
```

Arguments

`data`	A data frame, or a single path to a delimited text file for the fused one-pass file mode.
`by`	Character vector of grouping columns.
`value`	Optional character vector of value columns to summarize.
`fun`	Summary function applied to each value column. In file mode this is one or more of `"sum"`, `"mean"`, `"var"`, `"sd"`, `"min"`, `"max"`, `"n"` and defaults to `"sum"`.
`...`	Additional arguments passed to `fun`; in file mode, reader options such as `where`, `delim`, `n_threads`.
`na.rm`	Whether to remove missing values before summarizing (defaults to `TRUE` in file mode).
`sort`	Whether to sort output rows by group.

Value

A basetable containing one row per group.

antimerge *Anti-join two tables*

Description

Keep rows in `x` whose keys do not exist in `y`.

Usage

```
antimerge(x, y, by)
```

Arguments

x, y	Data frames or data tables to compare.
by	Character vector of join columns.

Value

A basetable containing rows from x without matches in y.

---

applyby *Apply a function to each group*

Description

Apply a function to each group

Usage

```
applyby(data, by, fun, ..., bind = FALSE, id = ".group")
```

Arguments

data	A data.frame.
by	Character vector of column names identifying groups or join keys.
fun	Function applied to each element, column, or group.
...	Additional arguments (unused, or passed through depending on the function).
bind	Combine the per-group results into a single table.
id	Optional name for a source identifier column.

Value

A list of per-group results, or a combined table when bind = TRUE.

applycols *Apply a function to selected columns*

Description

Apply a function to selected columns

Usage

```
applycols(data, cols, fun, ...)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
fun	Function applied to each element, column, or group.
...	Additional arguments (unused, or passed through depending on the function).

Value

data with the selected columns transformed.

assertcols *Assert that columns exist (alias)*

Description

Assert that columns exist (alias)

Usage

```
assertcols(data, cols)
```

Arguments

data	A data.frame.
cols	Character vector of column names.

Value

See assert_cols().

assertcomplete *Assert that there are no missing values*

Description

Assert that there are no missing values

Usage

```
assertcomplete(data, cols = NULL)
```

Arguments

data	A data.frame.
cols	Character vector of column names.

Value

data, invisibly, if the assertion passes.

assertkey *Assert that a key is unique (alias)*

Description

Assert that a key is unique (alias)

Usage

```
assertkey(data, by)
```

Arguments

data	A data.frame.
by	Character vector of column names identifying groups or join keys.

Value

See assert_key().

assertnames *Assert that required column names are present*

Description

Assert that required column names are present

Usage

```
assertnames(data, names)
```

Arguments

data	A data.frame.
names	Required column names.

Value

data, invisibly, if the assertion passes.

assertrange *Assert that a column falls within a numeric range*

Description

Assert that a column falls within a numeric range

Usage

```
assertrange(data, column, lower, upper)
```

Arguments

data	A data.frame.
column	Name of a single column.
lower	Lower bound.
upper	Upper bound.

Value

data, invisibly, if the assertion passes.

assertrows *Assert that a row-wise condition holds for every row*

Description

Assert that a row-wise condition holds for every row

Usage

```
assertrows(data, condition)
```

Arguments

data	A data.frame.
condition	A logical expression evaluated in the context of data.

Value

data, invisibly, if the assertion passes.

asserttype *Assert that a column has an expected class*

Description

Assert that a column has an expected class

Usage

```
asserttype(data, column, class)
```

Arguments

data	A data.frame.
column	Name of a single column.
class	Class name to check against.

Value

data, invisibly, if the assertion passes.

assertunique *Assert that selected columns are unique*

Description

Assert that selected columns are unique

Usage

```
assertunique(data, cols)
```

Arguments

data	A data.frame.
cols	Character vector of column names.

Value

data, invisibly, if the assertion passes.

assertvalues *Assert that a column only contains allowed values*

Description

Assert that a column only contains allowed values

Usage

```
assertvalues(data, column, allowed)
```

Arguments

data	A data.frame.
column	Name of a single column.
allowed	Vector of permitted values.

Value

data, invisibly, if the assertion passes.

assert_cols *Validation and schema helpers*

Description

Low-level helpers for validating columns and keys or comparing schemas.

Usage

```
assert_cols(data, cols)

assert_key(data, by)

common_names(x, y)

duplicated_keys(data, by)
```

Arguments

data, x, y	A data.frame.
cols, by	Character vectors naming columns.

Value

assert_cols() and assert_key() return their input invisibly. common_names() returns a character vector. duplicated_keys() returns a data frame of duplicated key combinations.

Examples

```
common_names(mtcars, iris)
assert_cols(mtcars, c("mpg", "hp"))
duplicated_keys(data.frame(id = c(1, 1, 2)), "id")
```

as_basetable *Coerce an object to a basetable*

Description

as_basetable() turns a data frame, a list of equal-length columns, a matrix, or anything else as.data.frame() accepts into a basetable: an ordinary data frame carrying basetable's lightweight class, the same object every basetable verb returns. Use it when a package or script wants to hand its own data to basetable explicitly, rather than relying on a verb to stamp the class.

Usage

```
as_basetable(x, ...)

## S3 method for class 'basetable'
as_basetable(x, ...)

## S3 method for class 'data.frame'
as_basetable(x, ...)

## Default S3 method:
as_basetable(x, ...)
```

Arguments

x	An object to coerce.
...	Passed to `as.data.frame()` by the default method; ignored otherwise.

Details

The data are left unchanged; only the class and the `row.names` attribute are normalised. An object that is already a `basetable` is returned as-is, and `as.data.frame()` strips the class again.

Value

A `basetable`: a data frame with the `basetable` class.

See Also

`is_basetable()`

Examples

```
bt <- as_basetable(data.frame(g = c("a", "b"), x = 1:2))
class(bt)
identical(as_basetable(bt), bt)
as_basetable(list(g = c("a", "b"), x = 1:2))
```

`betweendates` *Test whether a date falls within a range*

Description

Test whether a date falls within a range

Usage

```
betweendates(x, start, end)
```

Arguments

x	An atomic vector.
start	Start bound (inclusive).
end	End bound (inclusive).

Value

A logical vector.

---

blanktona *Recode blank strings to* NA

Description

Recode blank strings to NA

Usage

```
blanktona(x)
```

Arguments

x	An atomic vector.

Value

x with blanks replaced by NA.

---

btread *Read a delimited text file*

Description

A from-scratch delimited-file reader written in C++: the file is memory mapped, scanned once for row boundaries (RFC 4180 quoting), type-guessed from a bounded sample, and materialised into typed vectors. Numeric columns are filled by a thread pool; character columns are built on the R thread.

Usage

```
btread(
  file,
  delim = NULL,
  header = TRUE,
  col_names = NULL,
  col_types = NULL,
  col_select = NULL,
  na = c("NA", ""),
  quote = "\"",
  comment = "",
  trim_ws = FALSE,
  skip = 0,
  n_max = Inf,
  guess_max = 10000,
  lazy = FALSE,
  n_threads = bt_default_threads(),
  as = c("data.frame", "basetable")
)
```

Arguments

file	Path to a delimited text file. .gz inputs are decompressed to a temporary file first.
delim	Field delimiter. NULL (default) sniffs the first line for one of , \t ; | and a space.
header	Logical; does the first row hold column names?
col_names	Optional character vector of names, used only when header = FALSE. Defaults to V1, V2, ...
col_types	NULL to guess, or a character vector of "logical", "integer", "double", "character", "skip", "guess". Length 1 is recycled; otherwise it must have one entry per column.
col_select	Optional integer positions or character names of the columns to keep. Other columns are not parsed.
na	Character vector of strings to read as NA.
quote	Quoting character.
comment	Lines beginning with this one-character string are skipped ("" disables).
trim_ws	Logical; strip leading/trailing spaces and tabs from unquoted fields.
skip	Number of raw lines to discard before reading.
n_max	Maximum number of data rows to read (Inf for all).
guess_max	Number of rows sampled for type guessing.
lazy	Logical; return numeric columns as lazily-parsed ALTREP vectors.
n_threads	Number of worker threads for the eager numeric fill.
as	Return "data.frame" (default) or "basetable".

Details

With `lazy = TRUE` the integer and double columns are returned as ALTREP vectors that parse their column only the first time it is touched, which makes "read a few columns from a wide file" close to free.

Value

A data.frame.

Examples

```
p <- tempfile(fileext = ".csv")
write.csv(head(iris), p, row.names = FALSE)
btread(p)
```

`btwrite` *Write a data frame to a delimited text file*

Description

A threaded C++ writer: each column is resolved to a plain C array on the R thread, then disjoint row ranges are formatted into private buffers and flushed in order.

Usage

```
btwrite(
  x,
  file,
  delim = ",",
  na = "NA",
  col_names = TRUE,
  quote = "\"",
  digits = 15,
  append = FALSE,
  n_threads = bt_default_threads()
)
```

Arguments

`x`	A data.frame.
`file`	Output path.
`delim`	Field delimiter.
`na`	String to write for `NA`.
`col_names`	Logical; write a header row?
`quote`	Quoting character; a field is quoted only if it contains the delimiter, the quote character, or a newline.

digits	Significant digits for double columns (passed to %g).
append	Logical; append to file instead of overwriting (no header is written when appending).
n_threads	Number of worker threads.

Value

file, invisibly.

Examples

```
p <- tempfile(fileext = ".csv")
btwrite(head(iris), p)
btread(p)
```

---

cardinality *Distinct-value count (or proportion) per column*

Description

Distinct-value count (or proportion) per column

Usage

```
cardinality(data, cols, prop = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
prop	Return the proportion of distinct values instead of the count.

Value

A named numeric vector.

casewhen *Multi-branch selection by condition*

Description

For each position, take the label of the first condition that is TRUE there. Conditions are a named list of equal-length logical vectors, in the same shape as collapsevalues()'s groups: the name of an element is the value used where that element is the first TRUE. Positions matching no condition get default. NA in a condition counts as not matching.

Usage

```
casewhen(conditions, default = NA)
```

Arguments

conditions	A named list of equal-length logical vectors. Each name is the value returned where its vector is the first TRUE.
default	Value for positions where no condition holds. Default NA.

Value

A vector as long as the conditions, holding matched labels and default elsewhere.

Examples

```
x <- c(-3, 0, 4, 25)
casewhen(list(neg = x < 0, low = x < 10), default = "high")
```

ceilingdate *Round a date up to a unit*

Description

Round a date up to a unit

Usage

```
ceilingdate(x, unit = c("day", "week", "month", "year"))
```

Arguments

x	An atomic vector.
unit	Time unit to round to.

Value

A Date vector.

center *Center a vector at its mean*

Description

Center a vector at its mean

Usage

```
center(x)
```

Arguments

x	An atomic vector.

Value

A centered numeric vector.

changedcols *Compare the columns of two tables*

Description

Compare the columns of two tables

Usage

```
changedcols(old, new)
```

Arguments

old	Baseline data.frame ("before" state).
new	Updated data.frame ("after" state), or replacement values when used for value substitution.

Value

See `compareschema()`.

changedrows *Rows whose key appears in both tables*

Description

Rows whose key appears in both tables

Usage

```
changedrows(old, new, by = NULL)
```

Arguments

old	Baseline data.frame ("before" state).
new	Updated data.frame ("after" state), or replacement values when used for value substitution.
by	Character vector of column names identifying groups or join keys.

Value

The matched rows from both tables, suffixed `.old`/`.new`.

classes *Column classes*

Description

Column classes

Usage

```
classes(data)
```

Arguments

data	A data.frame.

Value

A basetable with one row per column giving its class.

cleannames *Clean column names to a unique, syntactic form*

Description

Clean column names to a unique, syntactic form

Usage

```
cleannames(data)
```

Arguments

data	A data.frame.

Value

data with cleaned column names.

collapselevels *Collapse factor levels into named groups*

Description

Collapse factor levels into named groups

Usage

```
collapselevels(x, groups)
```

Arguments

x	An atomic vector.
groups	Named list mapping a replacement label to the values it should collapse.

Value

A character vector with levels collapsed.

collapsetext *Collapse a vector into a single string*

Description

Collapse a vector into a single string

Usage

```
collapsetext(x, sep = "")
```

Arguments

x	An atomic vector.
sep	Separator string.

Value

A single character string.

collapsevalues *Collapse values into named groups*

Description

Collapse values into named groups

Usage

```
collapsevalues(x, groups)
```

Arguments

x	An atomic vector.
groups	Named list mapping a replacement label to the values it should collapse.

Value

x with values replaced by their group label.

colnames *Column names*

Description

Column names

Usage

```
colnames(data)
```

Arguments

data	A data.frame.

Value

A character vector of column names.

commonnames *Column names shared by two tables*

Description

Column names shared by two tables

Usage

```
commonnames(x, y)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.

Value

See common_names().

compact *Drop NULL elements from a list*

Description

Remove every NULL element of .x, keeping the rest (including other falsy-but-not-NULL values like NA, 0, or ""). The common companion to map() when some calls legitimately return nothing – e.g. an empty API response – and the NULLs must be dropped before foldr(), rbindfill(), or similar.

Usage

```
compact(.x)
```

Arguments

.x	A list.

Value

.x with its NULL elements removed.

Examples

```
compact(list(1, NULL, 2, NULL, 3))
compact(list(a = 1, b = NULL, c = NA))
```

compare *Compare tables*

Description

Compare dimensions, names, types, missingness, and key overlap for two tables.

Usage

```
compare(x, y, by = NULL)
```

Arguments

x, y	Data frames or data tables to compare.
by	Optional key columns for overlap checks.

Value

A list of comparison tables.

compareschema *Compare the schemas of two tables*

Description

Compare the schemas of two tables

Usage

```
compareschema(x, y)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.

Value

A basetable describing shared and differing columns/types.

completegrid *Complete a table grid*

Description

Create all combinations of selected columns and merge them back to the input.

Usage

```
completegrid(data, cols, fill = list())
```

Arguments

data	A data frame or data table.
cols	Character vector of columns to complete across.
fill	Named list of values used to fill missing cells.

Value

A basetable containing the completed grid.

constants *Columns with a single distinct non-missing value*

Description

Columns with a single distinct non-missing value

Usage

```
constants(data)
```

Arguments

data	A data.frame.

Value

A character vector of constant column names.

containstext *Test for a pattern match anywhere in the string*

Description

Test for a pattern match anywhere in the string

Usage

```
containstext(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when fixed = TRUE).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A logical vector.

convertcols *Convert selected columns with a function*

Description

Convert selected columns with a function

Usage

```
convertcols(data, cols, fun, ...)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
fun	Function applied to each element, column, or group.
...	Additional arguments (unused, or passed through depending on the function).

Value

data with the selected columns converted.

count *Count rows by group*

Description

Count rows in a table by one or more grouping columns.

Usage

```
count(data, by, sort = TRUE, name = "n", ...)
```

Arguments

data	A data frame, or a single path to a delimited text file for the fused one-pass file mode.
by	Character vector of grouping columns.
sort	Whether to sort by descending counts. Ties (groups with the same count) are broken by ascending group key, so the row order is fully determined by by and data – never by the grouping engine's internal enumeration order, which is otherwise unspecified.
name	Name of the count column.
...	In file mode, reader options such as where, delim.

Value

A basetable with one row per group.

`countmatch` *Count pattern matches per string*

Description

Count pattern matches per string

Usage

```
countmatch(x, pattern, fixed = FALSE)
```

Arguments

`x`	An atomic vector.
`pattern`	A regular expression (or fixed string when `fixed = TRUE`).
`fixed`	Use fixed (non-regex) matching instead of regular expressions.

Value

An integer vector.

`crossmerge` *Cartesian join two tables*

Description

Return every combination of rows from `x` and `y`.

Usage

```
crossmerge(x, y)
```

Arguments

`x, y`	Data frames or data tables to combine.

Value

A basetable containing the Cartesian product of the rows.

cumavg *Cumulative mean*

Description

Cumulative mean

Usage

```
cumavg(x)
```

Arguments

x	An atomic vector.

Value

A numeric vector.

cumcount *Cumulative row counter*

Description

Cumulative row counter

Usage

```
cumcount(x)
```

Arguments

x	An atomic vector.

Value

An integer sequence along x.

cumedist *Cumulative empirical distribution*

Description

Cumulative empirical distribution

Usage

```
cumedist(x)
```

Arguments

x	An atomic vector.

Value

A numeric vector.

datediff *Difference between two dates*

Description

Difference between two dates

Usage

```
datediff(x, y, units = "days")
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
units	Unit used to express the difference.

Value

A numeric vector of differences.

dateseq *Sequence of dates*

Description

Sequence of dates

Usage

```
dateseq(from, to, by = "day", length.out = NULL)
```

Arguments

from	Start date.
to	End date, or target range for rescaling.
by	Character vector of column names identifying groups or join keys.
length.out	Desired sequence length.

Value

A Date vector.

day *Extract the day of month*

Description

Extract the day of month

Usage

```
day(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

denserank *Dense rank of a vector*

Description

Dense rank of a vector

Usage

```
denserank(x)
```

Arguments

x	An atomic vector.

Value

An integer vector of dense ranks.

describe *Column descriptions*

Description

Summarize a table with basic counts, missingness, distinct values, and descriptive statistics.

Usage

```
describe(data, cols = NULL, top_n = 3)
```

Arguments

data	A data frame or data table.
cols	Optional character vector of columns to describe.
top_n	Number of most frequent values to show for non-numeric columns.

Value

A basetable with one row per column.

difference *Lagged difference*

Description

Lagged difference

Usage

```
difference(x, lag = 1L)
```

Arguments

x	An atomic vector.
lag	Lag used for the difference.

Value

A numeric vector of differences.

diffrows *Set difference of rows*

Description

Set difference of rows

Usage

```
diffrows(x, y, by = NULL)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.

Value

Rows of x absent from y.

dims *Table dimensions*

Description

Return the number of rows and columns in a table.

Usage

```
dims(data)
```

Arguments

data	A data frame or data table.

Value

A one-row data frame with the row and column counts.

drop *Drop columns*

Description

Drop columns by name.

Usage

```
drop(data, cols)
```

Arguments

data	A data frame or data table.
cols	Character vector of column names to remove.

Value

A basetable with the selected columns removed.

duplicatekeys *Duplicated key combinations*

Description

Duplicated key combinations

Usage

```
duplicatekeys(data, by)
```

Arguments

data	A data.frame.
by	Character vector of column names identifying groups or join keys.

Value

See duplicated_keys().

duplicatenames *Duplicated column names*

Description

Duplicated column names

Usage

```
duplicatenames(data)
```

Arguments

data	A data.frame.

Value

A character vector of the duplicated names.

duplicaterows *Rows involved in a duplicate*

Description

Rows involved in a duplicate

Usage

```
duplicaterows(data)
```

Arguments

data	A data.frame.

Value

A data.frame of the duplicated rows.

emptycols *Columns that are entirely blank or missing*

Description

Columns that are entirely blank or missing

Usage

```
emptycols(data)
```

Arguments

data	A data.frame.

Value

A character vector of column names.

emptyrows *Rows that are entirely blank or missing*

Description

Rows that are entirely blank or missing

Usage

```
emptyrows(data)
```

Arguments

data	A data.frame.

Value

A data.frame of the fully blank rows.

endswith *Test whether strings end with a pattern*

Description

Test whether strings end with a pattern

Usage

```
endswith(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when `fixed = TRUE`).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A logical vector.

equaldata *Compare two tables for equality*

Description

Compare two tables for equality

Usage

```
equaldata(
  x,
  y,
  ignoreorder = FALSE,
  ignorerownames = TRUE,
  tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
ignoreorder	Ignore row order when comparing.
ignorerownames	Ignore row names when comparing.
tolerance	Numeric join/comparison tolerance.

Value

A single logical value.

equalrows *Compare two tables' rows for equality*

Description

Compare two tables' rows for equality

Usage

```
equalrows(x, y, by = NULL)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.

Value

A single logical value.

`expandlevels` *Add levels to a factor*

Description

Add levels to a factor

Usage

```
expandlevels(x, levels)
```

Arguments

`x`	An atomic vector.
`levels`	Character vector of factor levels.

Value

A factor including the additional levels.

`expandrows` *Repeat rows in a table*

Description

Repeat each row of a table according to a count vector.

Usage

```
expandrows(data, times)
```

Arguments

`data`	A data frame or data table.
`times`	A scalar or row-wise count vector giving repetitions.

Value

A basetable with repeated rows.

extract *Extract the first pattern match*

Description

Extract the first pattern match

Usage

```
extract(x, pattern, ...)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when fixed = TRUE).
...	Additional arguments (unused, or passed through depending on the function).

Value

A character vector of matches.

extractall *Extract all pattern matches*

Description

Extract all pattern matches

Usage

```
extractall(x, pattern, ...)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when fixed = TRUE).
...	Additional arguments (unused, or passed through depending on the function).

Value

A list of character vectors of matches.

extractbetween *Extract text between two markers*

Description

Extract text between two markers

Usage

```
extractbetween(x, left, right)
```

Arguments

x	An atomic vector.
left	Left marker.
right	Right marker.

Value

A character vector.

extractint *Extract an integer value from text*

Description

Extract an integer value from text

Usage

```
extractint(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

extractnum *Extract a numeric value from text*

Description

Extract a numeric value from text

Usage

```
extractnum(x)
```

Arguments

x	An atomic vector.

Value

A numeric vector.

fillboth *Fill missing values both forward and backward within groups*

Description

Fill missing values both forward and backward within groups

Usage

```
fillboth(data, cols, by = NULL)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
by	Character vector of column names identifying groups or join keys.

Value

data with cols filled in both directions.

filldown *Fill missing values downward*

Description

Fill missing values in selected columns using the last observed value.

Usage

```
filldown(data, cols, by = NULL)
```

Arguments

data	A data frame or data table.
cols	Character vector of columns to fill.
by	Optional grouping columns.

Value

A basetable with missing values filled downward.

fillup *Fill missing values upward*

Description

Fill missing values in selected columns using the next observed value.

Usage

```
fillup(data, cols, by = NULL)
```

Arguments

data	A data frame or data table.
cols	Character vector of columns to fill.
by	Optional grouping columns.

Value

A basetable with missing values filled upward.

firstby *First row within each group*

Description

First row within each group

Usage

```
firstby(data, by, order = NULL)
```

Arguments

data	A data.frame.
by	Character vector of column names identifying groups or join keys.
order	Optional column(s) used to break ties before picking first/last rows.

Value

One row per group.

firstcols *Move columns to the front*

Description

Move columns to the front

Usage

```
firstcols(data, cols)
```

Arguments

data	A data.frame.
cols	Character vector of column names.

Value

data with cols moved first.

firstrows *First* n *rows*

Description

First n rows

Usage

```
firstrows(data, n = 1L)
```

Arguments

data	A data.frame.
n	Integer count.

Value

The first n rows of data.

floordate *Round a date down to a unit*

Description

Round a date down to a unit

Usage

```
floordate(x, unit = c("day", "week", "month", "year"))
```

Arguments

x	An atomic vector.
unit	Time unit to round to.

Value

A Date vector.

foldr *Fold a vector or list from the right*

Description

Reduce `.x` with the two-argument function `.f`, associating from the right, a thin wrapper around `base::Reduce()` with `right = TRUE`. With `.accumulate = TRUE` the intermediate results are returned as well.

Usage

```
foldr(.x, .f, .init = NULL, .accumulate = FALSE, .simplify = TRUE)
```

Arguments

`.x`	A vector or list.
`.f`	A two-argument reducing function.
`.init`	Optional initial value.
`.accumulate`	Return intermediate accumulated values.
`.simplify`	Simplify accumulated results when possible.

Value

The folded value, or accumulated values when `.accumulate = TRUE`.

Examples

```
foldr(1:4, `+`)
foldr(letters[1:4], paste0, .accumulate = TRUE)
```

freq *Frequency table*

Description

Count the occurrences of values in a column, optionally within groups.

Usage

```
freq(data, column, by = NULL, prop = FALSE, sort = TRUE, ...)
```

Arguments

`data`	A data frame, or a single path to a delimited text file. In file mode the second argument is taken as the grouping column(s).
`column`	Name of the column to tabulate.
`by`	Optional grouping column or columns.
`prop`	Whether to include proportions.
`sort`	Whether to sort rows by descending frequency.
`...`	In file mode, reader options such as `where`, `delim`.

Value

A basetable containing frequencies, and proportions when requested.

`headtail` *Head and tail rows*

Description

Return the first and last ‘n‘ rows of a table.

Usage

```
headtail(data, n = 3)
```

Arguments

`data`	A data frame or data table.
`n`	Number of rows to show from the start and end.

Value

A basetable containing the selected rows.

hour *Extract the hour*

Description

Extract the hour

Usage

```
hour(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

intersectrows *Set intersection of rows*

Description

Set intersection of rows

Usage

```
intersectrows(x, y, by = NULL)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.

Value

Rows of x also present in y.

invalidrows *Rows that fail a condition*

Description

Rows that fail a condition

Usage

```
invalidrows(data, condition)
```

Arguments

data	A data.frame.
condition	A logical expression evaluated in the context of data.

Value

The rows of data for which condition is not TRUE.

invalidvalues *Rows whose value is not in the allowed set*

Description

Rows whose value is not in the allowed set

Usage

```
invalidvalues(data, column, allowed)
```

Arguments

data	A data.frame.
column	Name of a single column.
allowed	Vector of permitted values.

Value

The rows of data with disallowed values.

isalpha *Test for alphabetic-only strings*

Description

Test for alphabetic-only strings

Usage

```
isalpha(x)
```

Arguments

x	An atomic vector.

Value

A logical vector.

isalphanumeric *Test for alphanumeric-only strings*

Description

Test for alphanumeric-only strings

Usage

```
isalphanumeric(x)
```

Arguments

x	An atomic vector.

Value

A logical vector.

isblank *Test for blank (missing or empty) values*

Description

Test for blank (missing or empty) values

Usage

```
isblank(x)
```

Arguments

x	An atomic vector.

Value

A logical vector.

isemail *Test whether text looks like an email address*

Description

Test whether text looks like an email address

Usage

```
isemail(x)
```

Arguments

x	An atomic vector.

Value

A logical vector.

isintegertext *Test whether text looks like an integer*

Description

Test whether text looks like an integer

Usage

```
isintegertext(x)
```

Arguments

x	An atomic vector.

Value

A logical vector.

isnumerictext *Test whether text looks numeric*

Description

Test whether text looks numeric

Usage

```
isnumerictext(x)
```

Arguments

x	An atomic vector.

Value

A logical vector.

isurl *Test whether text looks like a URL*

Description

Test whether text looks like a URL

Usage

```
isurl(x)
```

Arguments

x	An atomic vector.

Value

A logical vector.

is_basetable *Test whether an object is a basetable*

Description

Test whether an object is a basetable

Usage

```
is_basetable(x)
```

Arguments

x	An object.

Value

A single logical.

See Also

as_basetable()

Examples

```
is_basetable(as_basetable(mtcars))
is_basetable(mtcars)
```

joinrelationship *Cardinality of a join key relationship*

Description

Cardinality of a join key relationship

Usage

```
joinrelationship(x, y, by)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.

Value

One of `"one-to-one"`, `"one-to-many"`, `"many-to-one"`, `"many-to-many"`.

jointext *Concatenate values element-wise*

Description

Concatenate values element-wise

Usage

```
jointext(...)
```

Arguments

...	Additional arguments (unused, or passed through depending on the function).

Value

A character vector.

keepmissing *Keep rows with missing values*

Description

Return rows where selected columns are missing or blank.

Usage

```
keepmissing(data, cols = NULL, mode = c("any", "all"))
```

Arguments

data	A data frame or data table.
cols	Optional character vector of columns to inspect.
mode	Whether any or all selected columns must be missing.

Value

A basetable containing the matching rows.

lagvalue *Lag a vector*

Description

Lag a vector

Usage

```
lagvalue(x, n = 1L, default = NA)
```

Arguments

x	An atomic vector.
n	Integer count.
default	Value used to fill positions with no lagged observation.

Value

x shifted forward by n positions.

lastby *Last row within each group*

Description

Last row within each group

Usage

```
lastby(data, by, order = NULL)
```

Arguments

data	A data.frame.
by	Character vector of column names identifying groups or join keys.
order	Optional column(s) used to break ties before picking first/last rows.

Value

One row per group.

lastcols *Move columns to the back*

Description

Move columns to the back

Usage

```
lastcols(data, cols)
```

Arguments

data	A data.frame.
cols	Character vector of column names.

Value

data with cols moved last.

lastrows *Last* n *rows*

Description

Last n rows

Usage

```
lastrows(data, n = 1L)
```

Arguments

data	A data.frame.
n	Integer count.

Value

The last n rows of data.

leadvalue *Lead a vector*

Description

Lead a vector

Usage

```
leadvalue(x, n = 1L, default = NA)
```

Arguments

x	An atomic vector.
n	Integer count.
default	Value used to fill positions with no leading observation.

Value

x shifted backward by n positions.

left *First* n *characters*

Description

First n characters

Usage

```
left(x, n)
```

Arguments

x	An atomic vector.
n	Integer count.

Value

A character vector.

locate *Position of the first pattern match*

Description

Position of the first pattern match

Usage

```
locate(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when `fixed = TRUE`).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

An integer vector of match positions (see `regexpr()`).

locateall *Positions of all pattern matches*

Description

Positions of all pattern matches

Usage

```
locateall(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when fixed = TRUE).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A list of match positions (see gregexpr()).

lower *Convert to lower case*

Description

Convert to lower case

Usage

```
lower(x)
```

Arguments

x	An atomic vector.

Value

A lower-case character vector.

lump — *Lump infrequent values into "Other"*

Description

Lump infrequent values into "Other"

Usage

```
lump(x, n = 5, other = "Other")
```

Arguments

x	An atomic vector.
n	Integer count.
other	Label used for values outside the top n.

Value

A character vector.

map — *Map over vectors and lists*

Description

A small base-R-style mapping helper with no external dependency: apply .f to each element of .x and return the results in a list, like base::lapply() with a friendlier argument name.

Usage

```
map(.x, .f, ...)
```

Arguments

.x	A vector or list.
.f	A function or function name.
...	Additional arguments passed to .f.

Value

map() returns a list.

Examples

```
map(1:3, function(x) x + 1)
```

matchedkeys *Rows of* x *whose key is present in* y

Description

Rows of x whose key is present in y

Usage

```
matchedkeys(x, y, by)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.

Value

Rows of x whose key is present in y.

matchestext *Test for a full-string pattern match*

Description

Test for a full-string pattern match

Usage

```
matchestext(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when fixed = TRUE).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A logical vector.

merge *Merge two tables*

Description

Merge two tables using shared keys.

Usage

```
merge(x, y, by = NULL, all = FALSE, all.x = all, all.y = all,
  sort = FALSE, suffixes = c(".x", ".y"))
```

Arguments

x, y	Data frames or data tables to merge.
by	Optional character vector of join columns.
all, all.x, all.y	Controls for keeping unmatched rows.
sort	Whether to sort the result.
suffixes	Suffixes used for overlapping column names.

Value

A basetable containing the merged rows.

middle *Substring between two positions*

Description

Substring between two positions

Usage

```
middle(x, start, end)
```

Arguments

x	An atomic vector.
start	Start bound (inclusive).
end	End bound (inclusive).

Value

A character vector.

minute *Extract the minute*

Description

Extract the minute

Usage

```
minute(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

missingindicator *Add missingness indicators*

Description

Append logical indicators showing whether selected columns are missing or blank.

Usage

```
missingindicator(data, cols = NULL, prefix = "missing_")
```

Arguments

data	A data frame or data table.
cols	Optional character vector of columns to inspect.
prefix	Prefix used for the added indicator columns.

Value

A basetable with logical missingness indicator columns appended.

missingness *Summarize missing values by column or row*

Description

Summarize missing values by column or row

Usage

```
missingness(data, margin = c("column", "row"))
```

Arguments

data	A data.frame.
margin	Summarize missingness "column"-wise (the default) or "row"-wise.

Value

A basetable with one row per column (or per row) describing the count and proportion of missing values.

missingrows *Return rows with missing values*

Description

Return rows where selected columns are missing or blank.

Usage

```
missingrows(data, cols = NULL, mode = c("any", "all"))
```

Arguments

data	A data frame or data table.
cols	Optional character vector of columns to inspect.
mode	Whether any or all selected columns must be missing.

Value

A basetable containing the matching rows.

month *Extract the month*

Description

Extract the month

Usage

```
month(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

move *Move columns before or after another column*

Description

Move columns before or after another column

Usage

```
move(data, cols, before = NULL, after = NULL)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
before	Column identifying where to insert cols before.
after	Column identifying where to insert cols after.

Value

data with columns reordered.

naif *Recode matching values to* NA

Description

Recode matching values to NA

Usage

```
naif(x, values)
```

Arguments

x	An atomic vector.
values	Vector or list of replacement values.

Value

x with matches replaced by NA.

nato *Replace* NA *with a value*

Description

Replace NA with a value

Usage

```
nato(x, value)
```

Arguments

x	An atomic vector.
value	Value to test, assign, or match against.

Value

x with missing values replaced.

natoblank *Replace* NA *with an empty string*

Description

Replace NA with an empty string

Usage

```
natoblank(x)
```

Arguments

x	An atomic vector.

Value

x with missing values replaced by "".

ncols *Number of columns*

Description

Return the number of columns in a table.

Usage

```
ncols(data)
```

Arguments

data	A data frame or data table.

Value

An integer scalar.

nearestmerge *Nearest-key join*

Description

Nearest-key join

Usage

```
nearestmerge(x, y, by, tolerance = Inf)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.
tolerance	Numeric join/comparison tolerance.

Value

See rollingmerge().

nearesttext *Nearest string match*

Description

Nearest string match

Usage

```
nearesttext(x, choices)
```

Arguments

x	An atomic vector.
choices	Candidate strings to match against.

Value

A character vector of nearest matches.

nonequimerge *Merge two tables using non-equi (inequality) conditions*

Description

Unlike a plain equi-merge, nonequimerge() supports inequality conditions between columns of x and y (for example, matching each x row to every y row whose date range contains x's date), in addition to any exact-match columns in by.

Usage

```
nonequimerge(x, y, by, ...)
```

Arguments

x, y	Data frames or data tables to join.
by	Character vector of join conditions. A plain column name (present in both x and y) is matched exactly; an entry containing a comparison operator, written as "xcol<op>ycol" (e.g. "date>=start_date", "date<=end_date"), is matched as an inequality between x's xcol and y's ycol. This uses "col>=other" style comparison strings.
...	Currently ignored; reserved for future join (e.g. mult, roll).

Value

A basetable with one row per matching x/y pair (an inner join: rows of x with no matching y row are dropped).

Examples

```
x <- data.frame(id = 1, date = as.Date("2024-01-15"))
y <- data.frame(
  id = 1,
  start_date = as.Date(c("2024-01-01", "2024-02-01")),
  end_date = as.Date(c("2024-01-31", "2024-02-28")),
  period = c("Jan", "Feb")
)
nonequimerge(x, y, by = c("id", "date>=start_date", "date<=end_date"))
```

normalizeencoding *Normalize text encoding to UTF-8*

Description

Normalize text encoding to UTF-8

Usage

```
normalizeencoding(x)
```

Arguments

x	An atomic vector.

Value

A character vector.

normalizeunicode *Normalize Unicode text*

Description

Normalize Unicode text

Usage

```
normalizeunicode(x)
```

Arguments

x	An atomic vector.

Value

A character vector (currently returned unchanged).

nrows *Number of rows*

Description

Return the number of rows in a table.

Usage

```
nrows(data)
```

Arguments

data	A data frame or data table.

Value

An integer scalar.

omitmissing *Drop rows with missing values*

Description

Drop rows where selected columns are missing or blank.

Usage

```
omitmissing(data, cols = NULL, mode = c("any", "all"))
```

Arguments

data	A data frame or data table.
cols	Optional character vector of columns to inspect.
mode	Whether any or all selected columns must be missing before a row is dropped.

Value

A basetable without the omitted rows.

orderrows *Order rows by one or more columns*

Description

Order rows by one or more columns

Usage

```
orderrows(data, by, decreasing = FALSE, na.last = TRUE)
```

Arguments

data	A data.frame.
by	Character vector of column names identifying groups or join keys.
decreasing	Sort in decreasing order.
na.last	Placement of missing values when sorting.

Value

data sorted by by.

outofrange *Rows whose value falls outside a range*

Description

Rows whose value falls outside a range

Usage

```
outofrange(data, column, lower, upper)
```

Arguments

data	A data.frame.
column	Name of a single column.
lower	Lower bound.
upper	Upper bound.

Value

The rows of data outside [lower, upper].

overlapmerge *Merge tables on overlapping intervals*

Description

Join tables by overlapping interval ranges.

Usage

```
overlapmerge(x, y, startx, endx, starty, endy, by = NULL)
```

Arguments

x, y	Data frames or data tables to join.
startx, endx	Interval bounds in x.
starty, endy	Interval bounds in y.
by	Optional character vector of exact-match grouping columns.

Value

A basetable containing the overlap matches.

padcenter *Pad text on both sides*

Description

Pad text on both sides

Usage

```
padcenter(x, width, pad = " ")
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
pad	Padding character.

Value

A character vector padded to width.

padleft *Pad text on the left*

Description

Pad text on the left

Usage

```
padleft(x, width, pad = " ")
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
pad	Padding character.

Value

A character vector padded to `width`.

padright *Pad text on the right*

Description

Pad text on the right

Usage

```
padright(x, width, pad = " ")
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
pad	Padding character.

Value

A character vector padded to `width`.

parsecurrency *Parse a currency string as a number*

Description

Parse a currency string as a number

Usage

```
parsecurrency(x, strict = FALSE)
```

Arguments

x	An atomic vector.
strict	Raise an error if any non-missing value fails to parse.

Value

A numeric vector.

parsedate *Parse text as dates, trying multiple formats*

Description

Parse text as dates, trying multiple formats

Usage

```
parsedate(x, formats, tz = "UTC", strict = FALSE)
```

Arguments

x	An atomic vector.
formats	Candidate format strings tried in order.
tz	Time zone used when parsing.
strict	Raise an error if any non-missing value fails to parse.

Value

A Date vector.

parsedatetime *Parse text as date-times, trying multiple formats*

Description

Parse text as date-times, trying multiple formats

Usage

```
parsedatetime(x, formats, tz = "UTC", strict = FALSE)
```

Arguments

x	An atomic vector.
formats	Candidate format strings tried in order.
tz	Time zone used when parsing.
strict	Raise an error if any non-missing value fails to parse.

Value

A POSIXct vector.

parsefailures *Rows where parsing failed*

Description

Rows where parsing failed

Usage

```
parsefailures(x, fun, ...)
```

Arguments

x	An atomic vector.
fun	Function applied to each element, column, or group.
...	Additional arguments (unused, or passed through depending on the function).

Value

A basetable of the failed indices and values.

parseint *Parse text as integers*

Description

Parse text as integers

Usage

```
parseint(x, na = character(), strict = FALSE)
```

Arguments

x	An atomic vector.
na	Values treated as missing before parsing.
strict	Raise an error if any non-missing value fails to parse.

Value

An integer vector.

parselogical *Parse text as logicals*

Description

Parse text as logicals

Usage

```
parselogical(x, na = character(), strict = FALSE)
```

Arguments

x	An atomic vector.
na	Values treated as missing before parsing.
strict	Raise an error if any non-missing value fails to parse.

Value

A logical vector.

parsenum *Parse text as numbers*

Description

Parse text as numbers

Usage

```
parsenum(x, decimal = ".", grouping = ",", na = character(), strict = FALSE)
```

Arguments

x	An atomic vector.
decimal	Decimal mark used in the input strings.
grouping	Thousands-grouping mark used in the input strings.
na	Values treated as missing before parsing.
strict	Raise an error if any non-missing value fails to parse.

Value

A numeric vector.

parsepercent *Parse a percentage string as a number*

Description

Parse a percentage string as a number

Usage

```
parsepercent(x, strict = FALSE)
```

Arguments

x	An atomic vector.
strict	Raise an error if any non-missing value fails to parse.

Value

A numeric vector.

percentchange *Period-over-period percent change*

Description

Period-over-period percent change

Usage

```
percentchange(x)
```

Arguments

x	An atomic vector.

Value

A numeric vector of percent changes.

percentrank *Percentile rank of a vector*

Description

Percentile rank of a vector

Usage

```
percentrank(x)
```

Arguments

x	An atomic vector.

Value

A numeric vector of percentile ranks in `[0, 1]`.

pick *Select columns*

Description

Select columns by name.

Usage

```
pick(data, cols)
```

Arguments

data	A data frame or data table.
cols	Character vector of column names.

Value

A basetable containing the selected columns.

preview *Compact table preview*

Description

Print a concise preview of a table with its row and column counts and a short summary of each variable.

Usage

```
preview(data, width = getOption("width"))
```

Arguments

data	A data frame or data table.
width	Maximum line width for printed output.

Value

The input data, invisibly.

profile *Compact variable profile*

Description

Return a compact profile table for the variables in a data set.

Usage

```
profile(data, cols = NULL, top_n = 3)
```

Arguments

data	A data frame or data table.
cols	Optional character vector of columns to profile.
top_n	Number of most frequent values to show for non-numeric columns.

Value

A basetable with one row per column.

propcount *Grouped counts with proportions*

Description

Grouped counts with proportions

Usage

```
propcount(data, by, margin = NULL)
```

Arguments

data	A data.frame.
by	Character vector of column names identifying groups or join keys.
margin	Column(s) defining the totals used to compute proportions.

Value

A basetable of counts (and proportions when `margin` is supplied).

quantilegroup *Bin a vector into quantile groups*

Description

Bin a vector into quantile groups

Usage

```
quantilegroup(x, n = 5)
```

Arguments

x	An atomic vector.
n	Integer count.

Value

A factor of quantile bins.

quarter *Extract the calendar quarter*

Description

Extract the calendar quarter

Usage

```
quarter(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

rangemerge *Merge two tables by a range condition*

Description

For each row of x, keep the row(s) of y whose value column falls between x's lower and upper bounds (inclusive), within the exact-match by key.

Usage

```
rangemerge(x, y, by, lower, upper, value)
```

Arguments

x, y	Data frames or data tables to join.
by	Character vector of exact-match join columns, present in both x and y.
lower, upper	Single columns in x describing the inclusive range bounds to test against.
value	Single column in y whose value is tested against each matching x row's [lower, upper] range.

Value

A basetable with one row per x row for each y row whose value falls within [lower, upper] (matching on by); an x row with no matching y row appears once with NA for y's columns.

Examples

```
x <- data.frame(id = c(1, 1, 2), lower = c(0, 0, 0), upper = c(10, 10, 20))
y <- data.frame(id = c(1, 1, 2), val = c(5, 15, 5), label = c("a", "b", "c"))
rangemerge(x, y, by = "id", lower = "lower", upper = "upper", value = "val")
```

rbindfill *Row-bind tables, filling missing columns*

Description

Combine data frames by row, filling columns that are missing from some inputs with NA.

Usage

```
rbindfill(..., id = NULL, fill = TRUE, typeconflict = c("error", "coerce"))
```

Arguments

`...`	Data frames to combine, or a single list of them.
`id`	Optional name for a source identifier column recording which input each row came from.
`fill`	Fill missing columns with `NA` instead of erroring when inputs have different columns.
`typeconflict`	How to handle a column whose type differs across inputs. `"error"` (the default) stops with a message naming the column and the conflicting types before any coercion happens. `"coerce"` skips the check and allows ordinary vector coercion while binding.

Value

A combined basetable.

`recode` *Recode values by lookup (alias)*

Description

Recode values by lookup (alias)

Usage

```
recode(x, old, new)
```

Arguments

`x`	An atomic vector.
`old`	Baseline data.frame ("before" state).
`new`	Updated data.frame ("after" state), or replacement values when used for value substitution.

Value

See `replacevalues()`.

removeaccents *Strip accents from text*

Description

Fold accented Latin letters to their unaccented ASCII form (`"café" -> "cafe"`), following ICU's `Latin-ASCII` mapping: covers the Latin-1 Supplement and Latin Extended-A blocks, plus a few common extras (ß -> `"ss"`, Æ -> `"AE"`, Romanian ș/ț). Characters with no mapping, including non-Latin scripts, pass through unchanged. No Unicode library dependency.

Usage

```
removeaccents(x)
```

Arguments

x	An atomic vector.

Value

A character vector the same length as `x`.

See Also

`transliterate()` for Greek and Cyrillic as well.

Examples

```
removeaccents(c("café", "naïve", "Zürich", "Kraków"))
```

removeall *Remove text matching a pattern (alias)*

Description

Remove text matching a pattern (alias)

Usage

```
removeall(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when `fixed = TRUE`).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A character vector.

`removedrows` *Rows present in* `old` *but not in* `new`

Description

Rows present in `old` but not in `new`

Usage

```
removedrows(old, new, by = NULL)
```

Arguments

`old`	Baseline data.frame ("before" state).
`new`	Updated data.frame ("after" state), or replacement values when used for value substitution.
`by`	Character vector of column names identifying groups or join keys.

Value

The removed rows.

`removeduplicates` *Remove duplicate rows, optionally by key*

Description

Remove duplicate rows, optionally by key

Usage

```
removeduplicates(data, by = NULL, keep = c("first", "last", "none"))
```

Arguments

`data`	A data.frame.
`by`	Character vector of column names identifying groups or join keys.
`keep`	Which duplicate to keep.

Value

`data` with duplicates removed.

removetext *Remove text matching a pattern*

Description

Remove text matching a pattern

Usage

```
removetext(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when fixed = TRUE).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A character vector.

renamecols *Rename columns*

Description

Rename columns using new = old pairs. Old column names may be supplied as bare names or character strings.

Usage

```
renamecols(data, ...)
```

Arguments

data	A data.frame.
...	Named rename expressions, as new = old.

Value

data with the named columns renamed.

renamewith *Rename columns with a function*

Description

Rename columns with a function

Usage

```
renamewith(data, cols, fun)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
fun	Function applied to each element, column, or group.

Value

data with the selected columns renamed.

reorderlevels *Reorder factor levels*

Description

Reorder factor levels

Usage

```
reorderlevels(x, by)
```

Arguments

x	An atomic vector.
by	Character vector of column names identifying groups or join keys.

Value

A factor with reordered levels.

repairnames *Repair column names*

Description

Repair column names

Usage

```
repairnames(data, method = c("unique", "universal", "minimal"))
```

Arguments

data	A data.frame.
method	Cleaning strategy to apply to names.

Value

data with repaired column names.

replaceall *Recode values by lookup (alias)*

Description

Recode values by lookup (alias)

Usage

```
replaceall(x, old, new)
```

Arguments

x	An atomic vector.
old	Existing value(s) to replace.
new	Replacement value(s) matching old by position.

Value

See replacevalues().

replacecols *Replace selected columns with new values*

Description

Replace selected columns with new values

Usage

```
replacecols(data, cols, values)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
values	Vector or list of replacement values.

Value

data with the selected columns replaced.

replacetext *Replace a pattern with replacement text*

Description

Replace a pattern with replacement text

Usage

```
replacetext(x, pattern, replacement, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when fixed = TRUE).
replacement	Replacement text.
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A character vector.

replacevalues *Recode values by lookup*

Description

Recode values by lookup

Usage

```
replacevalues(x, old, new)
```

Arguments

x	An atomic vector.
old	Baseline data.frame ("before" state).
new	Updated data.frame ("after" state), or replacement values when used for value substitution.

Value

x with matched values replaced.

replacewhere *Replace values in selected columns where a condition holds*

Description

Replace values in selected columns where a condition holds

Usage

```
replacewhere(data, condition, cols, value)
```

Arguments

data	A data.frame.
condition	A logical expression evaluated in the context of data.
cols	Character vector of column names.
value	Value to test, assign, or match against.

Value

data with matching values replaced.

rescale *Rescale a vector to a new range*

Description

Rescale a vector to a new range

Usage

```
rescale(x, to = c(0, 1))
```

Arguments

x	An atomic vector.
to	End date, or target range for rescaling.

Value

A rescaled numeric vector.

reverse *Reverse row order*

Description

Reverse row order

Usage

```
reverse(data)
```

Arguments

data	A data.frame.

Value

data with rows in reverse order.

right *Last* n *characters*

Description

Last n characters

Usage

```
right(x, n)
```

Arguments

x	An atomic vector.
n	Integer count.

Value

A character vector.

rollapply *Rolling window apply*

Description

Rolling window apply

Usage

```
rollapply(
  x,
  width,
  FUN,
  ...,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
FUN	Function applied to each rolling window.
...	Additional arguments (unused, or passed through depending on the function).
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.

Value

A vector with one result per window.

---

rollingmerge *Rolling join two tables*

Description

Join tables using a rolling match on the last key column.

Usage

```
rollingmerge(x, y, by, direction = c("backward", "forward", "nearest"),
  tolerance = Inf)
```

Arguments

x, y	Data frames or data tables to join.
by	Character vector of join columns. The last column is treated as the rolling key.
direction	Rolling direction: backward, forward, or nearest.
tolerance	Maximum distance allowed for rolling matches.

Value

A basetable with rows from x joined to nearest matches in y.

rollmax *Rolling maximum*

Description

Rolling maximum

Usage

```
rollmax(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of rolling maximums.

rollmean *Rolling mean*

Description

Rolling mean

Usage

```
rollmean(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of rolling means.

---

rollmedian *Rolling median*

Description

Rolling median

Usage

```
rollmedian(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of rolling medians.

---

rollmin *Rolling minimum*

Description

Rolling minimum

Usage

```
rollmin(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of rolling minimums.

`rollprod` *Rolling product*

Description

Rolling product

Usage

```
rollprod(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

`x`	An atomic vector.
`width`	Target width, in characters or rolling-window size depending on the function.
`align`	Alignment of the rolling window relative to each position.
`fill`	Value used for positions where no window/result is available.
`partial`	Allow partial windows at the boundaries.
`na.rm`	Drop missing values before computing the result.

Value

A numeric vector of rolling products.

`rollsd` *Rolling standard deviation*

Description

Rolling standard deviation

Usage

```
rollsd(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of rolling standard deviations.

---

rollsum *Rolling sum*

Description

Rolling sum

Usage

```
rollsum(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of rolling sums.

---

rollvar *Rolling variance*

Description

Rolling variance

Usage

```
rollvar(
  x,
  width,
  align = c("right", "left", "center"),
  fill = NA,
  partial = FALSE,
  na.rm = FALSE
)
```

Arguments

x	An atomic vector.
width	Target width, in characters or rolling-window size depending on the function.
align	Alignment of the rolling window relative to each position.
fill	Value used for positions where no window/result is available.
partial	Allow partial windows at the boundaries.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of rolling variances.

roundddate *Round a date to the nearest unit*

Description

Round a date to the nearest unit

Usage

```
roundddate(x, unit = c("day", "week", "month", "year"))
```

Arguments

x	An atomic vector.
unit	Time unit to round to.

Value

A Date vector.

rowall *Row-wise* all() *across columns*

Description

Row-wise all() across columns

Usage

```
rowall(data, cols = NULL, na.rm = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
na.rm	Drop missing values before computing the result.

Value

A logical vector.

rowany *Row-wise* any() *across columns*

Description

Row-wise any() across columns

Usage

```
rowany(data, cols = NULL, na.rm = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
na.rm	Drop missing values before computing the result.

Value

A logical vector.

rowapply *Apply a function row-wise*

Description

Apply a function row-wise

Usage

```
rowapply(data, cols = NULL, fun, ...)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
fun	Function applied to each element, column, or group.
...	Additional arguments (unused, or passed through depending on the function).

Value

The result of fun applied to each row.

rowcount *Count matches of a value across columns, per row*

Description

Count matches of a value across columns, per row

Usage

```
rowcount(data, cols = NULL, value = TRUE, na.rm = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
value	Value to test, assign, or match against.
na.rm	Drop missing values before computing the result.

Value

An integer vector.

rowfirst *First non-missing value across columns, per row*

Description

First non-missing value across columns, per row

Usage

```
rowfirst(data, cols = NULL, na.rm = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
na.rm	Drop missing values before computing the result.

Value

A vector of first values.

rowlast *Last non-missing value across columns, per row*

Description

Last non-missing value across columns, per row

Usage

```
rowlast(data, cols = NULL, na.rm = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
na.rm	Drop missing values before computing the result.

Value

A vector of last values.

rowmax *Row-wise maximum across columns*

Description

Row-wise maximum across columns

Usage

```
rowmax(data, cols = NULL, na.rm = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of row maximums.

rowmin *Row-wise minimum across columns*

Description

Row-wise minimum across columns

Usage

```
rowmin(data, cols = NULL, na.rm = FALSE)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
na.rm	Drop missing values before computing the result.

Value

A numeric vector of row minimums.

rownames *Row names*

Description

Row names

Usage

```
rownames(data)
```

Arguments

data	A data.frame.

Value

A character vector of row names.

rownumber *Row position*

Description

Row position

Usage

```
rownumber(x)
```

Arguments

x	An atomic vector.

Value

An integer sequence along x.

sameschema *Test whether two tables share the same column names*

Description

Test whether two tables share the same column names

Usage

```
sameschema(x, y)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.

Value

A single logical value.

samplefrac *Sample a fraction of rows without replacement*

Description

Sample a fraction of rows without replacement

Usage

```
samplefrac(data, frac, by = NULL)
```

Arguments

data	A data.frame.
frac	Fraction of rows to sample.
by	Optional character vector of grouping columns; sample frac of each group (rounded up). Sampled rows are returned in their original order.

Value

A random sample of rows.

samplerows *Sample* n *rows without replacement*

Description

Sample n rows without replacement

Usage

```
samplerows(data, n, by = NULL)
```

Arguments

data	A data.frame.
n	Integer count.
by	Optional character vector of grouping columns; sample n rows within each group (capped at the group size). Sampled rows are returned in their original order.

Value

A random sample of n rows.

second *Extract the second*

Description

Extract the second

Usage

```
second(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

semimerge *Semi-join two tables*

Description

Keep rows in x whose keys exist in y.

Usage

```
semimerge(x, y, by)
```

Arguments

x, y	Data frames or data tables to compare.
by	Character vector of join columns.

Value

A basetable containing rows from x with matches in y.

sentencecase *Convert to sentence case*

Description

Convert to sentence case

Usage

```
sentencecase(x)
```

Arguments

x	An atomic vector.

Value

A sentence-case character vector.

separate *Split one column into several*

Description

Split one column into several

Usage

```
separate(
  data,
  column,
  into,
  sep,
  remove = TRUE,
  extra = c("warn", "drop", "merge"),
  fill = c("warn", "left", "right")
)
```

Arguments

data	A data.frame.
column	Name of a single column.
into	Names of the columns produced by splitting.
sep	Separator string.
remove	Drop the source column(s) after the operation.
extra	How to handle extra pieces beyond into.
fill	Value used for positions where no window/result is available.

Value

`data` with `column` split into `into`.

`setthreads` *Set basetable thread count for the session*

Description

Controls the default thread count used by basetable's native reader and writer when a verb does not receive `n_threads` explicitly.

Usage

```
setthreads(
  threads = NULL,
  restore_after_fork = NULL,
  percent = NULL,
  throttle = NULL
)
```

Arguments

`threads`	Integer thread count, or `NULL` to reread environment settings.
`restore_after_fork`	Ignored; kept for API compatibility.
`percent`	Percentage of detected logical CPUs to use.
`throttle`	Ignored; kept for API compatibility.

Value

The previous thread count.

`similartext` *Nearest string match (alias)*

Description

Nearest string match (alias)

Usage

```
similartext(x, choices)
```

Arguments

x	An atomic vector.
choices	Candidate strings to match against.

Value

See nearesttext().

---

split *Split a table by groups*

Description

Split a table into pieces by one or more grouping columns.

Usage

```
split(data, by, drop = FALSE, keep.by = TRUE)
```

Arguments

data	A data frame or data table.
by	Character vector of grouping columns.
drop	Whether to drop empty groups.
keep.by	Whether to keep grouping columns in each piece.

Value

A named list of basetables.

---

splitfirst *First piece after splitting on a separator*

Description

First piece after splitting on a separator

Usage

```
splitfirst(x, sep, fixed = FALSE)
```

Arguments

x	An atomic vector.
sep	Separator string.
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A character vector.

---

splitlast *Last piece after splitting on a separator*

Description

Last piece after splitting on a separator

Usage

```
splitlast(x, sep, fixed = FALSE)
```

Arguments

x	An atomic vector.
sep	Separator string.
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A character vector.

---

splittext *Split text on a separator*

Description

Split text on a separator

Usage

```
splittext(x, sep, fixed = FALSE)
```

Arguments

x	An atomic vector.
sep	Separator string.
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A list of character vectors.

---

squish *Trim and collapse internal whitespace*

Description

Trim and collapse internal whitespace

Usage

```
squish(x)
```

Arguments

x	An atomic vector.

Value

A cleaned character vector.

---

stack *Stack columns*

Description

Convert selected columns into long format with an indicator column.

Usage

```
stack(data, select = NULL, drop = FALSE)
```

Arguments

data	A data frame or data table.
select	Optional character vector of columns to stack.
drop	Whether to drop rows with missing values.

Value

A basetable with a stacked value column.

standardize *Standardize a vector to mean 0, SD 1*

Description

Standardize a vector to mean 0, SD 1

Usage

```
standardize(x)
```

Arguments

x	An atomic vector.

Value

A standardized numeric vector.

startswith *Test whether strings start with a pattern*

Description

Test whether strings start with a pattern

Usage

```
startswith(x, pattern, fixed = FALSE)
```

Arguments

x	An atomic vector.
pattern	A regular expression (or fixed string when `fixed = TRUE`).
fixed	Use fixed (non-regex) matching instead of regular expressions.

Value

A logical vector.

subset *Core data manipulation operations*

Description

Base-faithful table manipulation helpers built around explicit arguments, returning a basetable. See aggregate, count, drop, merge, move, orderrows, pick, renamecols, split, stack, summaries, uniquerows, unstack, and within for the remaining core verbs, each documented on its own page.

Usage

```
subset(data, subset = NULL, select = NULL, drop = FALSE, by = NULL)

transform(data, ..., .keep = TRUE, by = NULL)

reshape(data, ..., direction)
```

Arguments

data	A data.frame.
subset	A base-style expression evaluated against columns.
select, by	Character vectors naming columns. For subset, by names optional grouping columns: the subset expression is then evaluated within every group, so aggregate references in it (mean(x), max(x), ...) are per group; kept rows are returned in their original order. For transform, by names optional grouping columns and each expression in ... is evaluated within every group (e.g. cumsum(x) or x - mean(x) computed per group), so the result still has one row per input row. Use summaries for one row per group instead.
drop, .keep	Control flags.
...	Additional expressions or arguments passed through to helpers.
direction	Additional operation-specific controls.

Value

Most functions return a basetable.

Examples

```
subset(mtcars, cyl == 6, select = c("mpg", "hp"))
subset(mtcars, mpg > mean(mpg), by = "cyl")
transform(mtcars, power = hp / wt)
transform(mtcars, dev_mpg = mpg - mean(mpg), by = "cyl")
```

summaries *Named grouped summaries*

Description

Compute multiple named summary expressions, optionally by group.

Usage

```
summaries(data, by = NULL, ...)
```

Arguments

data	A data frame or data table.
by	Optional grouping columns.
...	Named summary expressions.

Value

A basetable containing the requested summaries.

summarytab *Table 1-style descriptive summary*

Description

Build a publication-style summary table of one or more variables, optionally stratified by a grouping column, in the manner of a clinical "Table 1".

Usage

```
summarytab(
  data,
  vars = NULL,
  by = NULL,
  overall = TRUE,
  p_value = FALSE,
  digits = 1
)
```

Arguments

`data`	A data.frame.
`vars`	Character vector of variables to summarize. Defaults to every column other than `by`.
`by`	Optional single column name used to stratify the summary.
`overall`	Include an "Overall" column alongside the by-group columns.
`p_value`	Include a column of between-group p-values. Requires `by`.
`digits`	Number of decimal places used when formatting numbers.

Value

A basetable with one row per variable (or variable level), and one column per stratum plus `Overall`/`p_value` as requested.

`textdist` *Pairwise string distances*

Description

Pairwise string distances

Usage

```
textdist(x, y)
```

Arguments

`x`	An atomic vector.
`y`	An atomic vector, data.frame, or basetable, depending on the function.

Value

A numeric distance matrix (see `utils::adist()`).

textlen *Character length*

Description

Character length

Usage

```
textlen(x)
```

Arguments

x	An atomic vector.

Value

An integer vector of character counts.

titlecase *Convert to title case*

Description

Convert to title case

Usage

```
titlecase(x)
```

Arguments

x	An atomic vector.

Value

A title-case character vector.

tolong *Reshape columns into key/value rows*

Description

Reshape columns into key/value rows

Usage

```
tolong(
  data,
  cols,
  names = "variable",
  values = "value",
  idcols = NULL,
  na.rm = FALSE
)
```

Arguments

data	A data.frame.
cols	Character vector of column names.
names	Name for the resulting key/value column, depending on the function.
values	Vector or list of replacement values.
idcols	Columns to keep as row identifiers.
na.rm	Drop missing values before computing the result.

Value

A long-format basetable.

towide *Reshape rows into columns*

Description

Reshape rows into columns

Usage

```
towide(data, names, values, idcols = NULL, fun = NULL, fill = NA)
```

Arguments

data	A data.frame.
names	Name for the resulting key/value column, depending on the function.
values	Vector or list of replacement values.
idcols	Columns to keep as row identifiers.
fun	Function applied to each element, column, or group.
fill	Value used for positions where no window/result is available.

Value

A wide-format basetable.

---

transliterate *Transliterate text to ASCII*

Description

Like removeaccents(), but also romanises the Greek and Cyrillic blocks (the Greek and Cyrillic spellings of "Athena" and "Moskva" fold to those ASCII forms), following ICU's Any-Latin; Latin-ASCII mapping. Other non-Latin scripts (Han, Kana, Arabic, Hebrew, Devanagari, Thai, ...) are left unchanged rather than guessed at. No Unicode library dependency.

Usage

```
transliterate(x)
```

Arguments

x	An atomic vector.

Value

A character vector the same length as x.

See Also

removeaccents() for Latin only.

Examples

```
# "Zurich", Greek "Athena", Cyrillic "Moskva" (spelt via code points so
# this help page stays ASCII)
greek <- intToUtf8(c(0x391, 0x3b8, 0x3ae, 0x3bd, 0x3b1))
cyrillic <- intToUtf8(c(0x41c, 0x43e, 0x441, 0x43a, 0x432, 0x430))
transliterate(c("Zurich", greek, cyrillic))
```

transpose *Transpose a table*

Description

Transpose a table

Usage

```
transpose(data)
```

Arguments

data	A data.frame.

Value

A transposed basetable.

traverse *Traverse a list of arguments*

Description

Call `.f` once per position across several equal-length vectors or lists, passing the i-th element of each as a positional argument. The parallel-map companion to `map()`.

Usage

```
traverse(.l, .f, ...)
```

Arguments

.l	A list of vectors or lists with a common length.
.f	A function or function name.
...	Additional arguments passed to `.f`.

Value

`traverse()` returns a list.

Examples

```
traverse(list(a = 1:2, b = 10:11), function(a, b) a + b)
```

trim *Trim leading and trailing whitespace*

Description

Trim leading and trailing whitespace

Usage

```
trim(x)
```

Arguments

x	An atomic vector.

Value

A trimmed character vector.

truncate *Truncate text with an ellipsis*

Description

Truncate text with an ellipsis

Usage

```
truncate(x, n, ellipsis = "...")
```

Arguments

x	An atomic vector.
n	Integer count.
ellipsis	String appended to truncated text.

Value

A character vector.

types *Column types*

Description

Return the column name, class, and storage type for each variable in a table.

Usage

```
types(data)
```

Arguments

data	A data frame or data table.

Value

A data frame with one row per column.

unionrows *Set union of rows*

Description

Set union of rows

Usage

```
unionrows(x, y, by = NULL)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.

Value

The union of x and y.

uniquerows *Unique rows, in memory or straight off a delimited file*

Description

Return unique rows, optionally considering only selected columns. With a single file path as `data`, it is a fused one-pass scan that returns the distinct combinations of `cols` without ever materialising the file as R vectors (see `aggregate()`'s file mode).

Usage

```
uniquerows(data, cols = NULL, .keep_all = FALSE, ...)
```

Arguments

`data`	A data.frame, or a single path to a delimited text file.
`cols`	Optional character vector of columns used to determine uniqueness. Required when `data` is a file path.
`.keep_all`	Keep all columns when `cols` is supplied.
`...`	For the file form, passed to the file reader (`where`, `delim`, `n_threads`, ...).

Value

A basetable of the unique rows.

uniques *Count of distinct values per column*

Description

Count of distinct values per column

Usage

```
uniques(data)
```

Arguments

`data`	A data.frame.

Value

A named integer vector of distinct-value counts.

unite — *Combine several columns into one*

Description

Combine several columns into one

Usage

```
unite(data, column, cols, sep = "_", remove = TRUE, na.rm = FALSE)
```

Arguments

data	A data.frame.
column	Name of a single column.
cols	Character vector of column names.
sep	Separator string.
remove	Drop the source column(s) after the operation.
na.rm	Drop missing values before computing the result.

Value

data with cols combined into column.

unmatchedkeys — *Distinct keys of* x *absent from* y

Description

Distinct keys of x absent from y

Usage

```
unmatchedkeys(x, y, by)
```

Arguments

x	An atomic vector.
y	An atomic vector, data.frame, or basetable, depending on the function.
by	Character vector of column names identifying groups or join keys.

Value

Rows of x whose key is not present in y.

unstack *Unstack a table*

Description

Reshape a stacked table back into wide form.

Usage

```
unstack(data, form, ...)
```

Arguments

data	A data frame or data table.
form	A formula or equivalent description of the reshape.
...	Additional arguments passed to the underlying method.

Value

A basetable reshaped into wide format.

updatemerge *Update rows from another table*

Description

Update matching rows in x from y.

Usage

```
updatemerge(x, y, by, cols = NULL)
```

Arguments

x, y	Data frames or data tables.
by	Character vector of key columns.
cols	Optional columns to update.

Value

A basetable with selected columns updated from y.

upper *Convert to upper case*

Description

Convert to upper case

Usage

```
upper(x)
```

Arguments

x	An atomic vector.

Value

An upper-case character vector.

week *Extract the week of year*

Description

Extract the week of year

Usage

```
week(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

weekday *Extract the weekday name*

Description

Extract the weekday name

Usage

```
weekday(x)
```

Arguments

x	An atomic vector.

Value

A character vector.

winsorize *Winsorize a vector at given quantiles*

Description

Winsorize a vector at given quantiles

Usage

```
winsorize(x, probs = c(0.01, 0.99))
```

Arguments

x	An atomic vector.
probs	Quantile probabilities.

Value

A winsorized numeric vector.

within *Modify a table within an environment*

Description

Evaluate expressions inside a table and return the modified result.

Usage

```
within(data, expr)
```

Arguments

data	A data frame or data table.
expr	An expression evaluated in the data's environment.

Value

A basetable containing the modified table.

year *Extract the year*

Description

Extract the year

Usage

```
year(x)
```

Arguments

x	An atomic vector.

Value

An integer vector.

`yearday` *Extract the day of year*

Description

Extract the day of year

Usage

```
yearday(x)
```

Arguments

`x`	An atomic vector.

Value

An integer vector.

Index