Package: svTidy (via r-universe)

June 10, 2026

Type Package

Version 0.2.1

Title 'SciViews::R' - Tidy Functions

Description SciViews equivalent functions of 'dplyr' and 'tidyr', but faster and using a standard evaluation of arguments or formulas.

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 4.2.0)

Imports cli (>= 3.6.4), collapse (>= 2.0.12), data.table (>= 1.15.4), data.trame (>= 0.9.0), dplyr (>= 1.1.4), rlang (>= 1.1.1), svBase (>= 1.7.4), svMisc (>= 1.6.0), tibble (>= 3.2.1), tidyr (>= 1.3.0), tidyselect(>= 1.2.1), utils (>= 4.2.0)

Suggests babynames(>= 1.0.1), bench (>= 1.1.4), dtplyr (>= 1.3.1), covr (>= 3.5.0), knitr (>= 1.42), rmarkdown (>= 2.21), spelling (>= 2.2.1), testthat (>= 3.0.0)

Remotes SciViews/data.trame, SciViews/svBase, SciViews/svMisc

License MIT + file LICENSE

URL https://github.com/SciViews/svTidy, https://www.sciviews.org/svTidy/, https://sciviews.r-universe.dev/svTidy

BugReports https://github.com/SciViews/svTidy/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Encoding UTF-8

Language en-US

ByteCompile yes

Config/testthat/edition 3

Config/pak/sysreqs cmake make libicu-dev libuv1-dev

Repository https://sciviews.r-universe.dev

Date/Publication 2026-06-10 09:35:17 UTC

RemoteUrl https://github.com/SciViews/svTidy

RemoteRef HEAD

RemoteSha c5fd05648865670b60f1ff10e50e870a2deda54e

Contents

arrange_ 2
bind_rows_ 3
filtering 4
grouping 6
joining 10
library_dplyr 16
list_sciviews_functions 17
mutating 17
pivoting 20
selecting 24
summarising 28
tidying 33

Index **38**

`arrange_` *Arranging Functions*

Description

Functions for arranging (sorting) rows. These are SciViews::R versions with standard evaluation and formula-based non-standard evaluation (ending with underscore _).

Usage

```
arrange_(
  .data = (.),
  ...,
  .by_group = FALSE,
  .locale = "C",
  .decreasing = FALSE
)
```

Arguments

.data	A data frame (data.frame, data.table or tibble's tbl_df). If not provided, the data-dot mechanism injects . as .data= automatically.
...	Either standard (quoted) column names of data. of formulas like ~col_name (formula-masking).
.by_group	Logical. If TRUE rows are first arranged by the grouping variables if any (applies only to grouped data frames). FALSE by default.
.locale	The locale to sort character vectors in. If NULL(default), use the "dplyr.legacy_locale" option (same one as dplyr::arrange()), and if not specified, it uses a "C" locale.
.decreasing	Sort in decreasing order (no, FALSE, by default)?

Details

For the way missing data are handled, see dplyr::arrange().

Value

A similar object as .data with all columns, all attributes and groups preserved, but row rearranged according to the specified order.

See Also

dplyr::arrange()

Examples

```
library(svTidy)
data(mtcars, package = 'datasets')
mtcars <- data.trame::as.data.trame(mtcars)
# Standard evaluation (provide quoted names of the columns to sort)
# You cannot use desc(col) here, but must specify what you want in the
# .decreasing argument
arrange_(mtcars, 'cyl', 'disp', .decreasing = c(FALSE, TRUE))
# With formula masking, you can use desc()
arrange_(mtcars, ~cyl, ~desc(disp))
```

---

bind_rows_ *Binding Functions*

Description

Functions for binding data frames by rows or columns. These are SciViews::R versions with standard evaluation and formula-based non-standard evaluation (ending with underscore _).

#' **Functions:**

- bind_rows_() - Stack two or more data frames one on top of the other
- bind_cols_() - Stack two or more data frames side by side

Usage

```
bind_rows_(..., .id = NULL, .use_names = TRUE, .fill = TRUE)

bind_cols_(
  ...,
  .name_repair = c("unique", "universal", "check_unique", "minimal")
)
```

Arguments

...	Data frames to bind.
.id	The name of the column for the origin id, either names if all other arguments are named, or numbers.
.use_names	If TRUE (default), bind by matching names, if FALSE, bind by position. If NULL, warns if all items do not have the same name in the same order, and then proceeds as if FALSE (but will be as if TRUE in the future).
.fill	If TRUE (default), fill missing columns with NA or NULL for missing list columns, if FALSE, do not fill.
.name_repair	How should the name be "repaired" to avoid duplicate

Value

A data frame of the same type as the first one provided in ....

See Also

dplyr::bind_rows(), dplyr::bind_cols()

---

filtering *Filter Rows and Slice Data Frames*

Description

Functions for subsetting rows based on conditions or by position.

These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- filter_() - Keep rows that match conditions
- filter_out_() - Remove rows that match conditions (inverse of filter_())
- distinct_() - Keep only unique/distinct rows based on columns
- slice_() - Select rows by position (index)
- slice_head_() - Select first n rows or proportion
- slice_tail_() - Select last n rows or proportion

Usage

```
filter_(.data = (.), ..., .by = NULL, .preserve = FALSE)

filter_out_(.data = (.), ..., .by = NULL, .preserve = FALSE)

distinct_(.data = (.), ..., .keep_all = FALSE, .method = "auto")

slice_(.data = (.), ..., .by = NULL, .preserve = NULL)

slice_head_(.data = (.), ..., n = 1L, prop, by = NULL, sort = TRUE)

slice_tail_(.data = (.), ..., n = 1L, prop, by = NULL, sort = TRUE)
```

Arguments

.data	A data frame (data.frame, data.table, or tibble)
...	For filter_() and filter_out_(): conditions as formulas (e.g., ~mpg > 20). For distinct_(): columns to use for uniqueness. For slice_(): row positions.
.by	A list of names of the columns to use for grouping the data.
.preserve	Logical. When TRUE, preserve the grouping structure in the result. When FALSE (default), recalculate grouping based on the filtered data.
.keep_all	Logical. For distinct_(), if TRUE, keep all columns in the result. If FALSE (default), keep only the distinct columns.
.method	The algorithm to use for grouping: "radix", "hash", or "auto" (by default). "auto" chose "radix" when sort = TRUE and "hash" otherwise.
n	Number of rows to keep
prop	Proportion of rows to keep, between 0 and 1. Provide either n, or prop but not both simultaneously. If none is provided, n = 1 is used. @param by A list of names of the columns to use for joining the two data frames.
by	A list of names of the columns to use for grouping the data.
sort	If TRUE largest group will be shown on top.

Value

A data frame with filtered/selected rows, maintaining the same class as the input (data.frame, data.table, or tibble).

Note

From {dplyr}, the slice_min(), slice_max() and slice_sample() functions are not added yet.

See Also

dplyr::filter(), dplyr::distinct(), dplyr::slice(), dplyr::slice_head(), dplyr::slice_tail()

Examples

```
library(svTidy)
data(mtcars)

# Filter rows with condition
mtcars |> filter_(~mpg > 20)

# Remove rows matching condition (inverse of filter_())
mtcars |> filter_out_(~mpg > 20)

# Multiple conditions (AND logic)
mtcars |> filter_(~mpg > 20, ~cyl == 4)

# Get distinct values for columns
mtcars |> distinct_(~cyl, ~gear)

# Distinct with all columns kept
mtcars |> distinct_(~cyl, .keep_all = TRUE)

# Slice specific rows
mtcars |> slice_(1, 5, 10)

# Select first 5 rows
mtcars |> slice_head_(n = 5)

# Select last 10% of rows
mtcars |> slice_tail_(prop = 0.1)

# Grouped filtering
mtcars |>
  group_by_(~cyl) |>
  filter_(~mpg > mean(~mpg))
```

---

grouping *Grouping Functions and Group Metadata*

Description

Functions for grouping data frames and accessing group metadata.

These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- `group_by_()` - Group data by one or more variables
- `ungroup_()` - Remove grouping variables
- `group_vars_()` - Get names or info about grouping variables

- `group_rows_()` - Get row indices for each group
- `group_data_()` - Get a tibble with grouping data and row indices
- `group_indices_()` - Get group index for each row
- `group_keys_()` - Get unique values of grouping variables
- `groups_()` - Get grouping variables as symbols
- `group_size_()` - Get number of rows in each group
- `n_groups_()` - Get total number of groups
- `as.grouped_df()` / `as_grouped_df()` - Convert to grouped_df object
- `is.grouped_df()` - Test if object is a grouped_df

Usage

```
group_by_(
  .data = (.),
  ...,
  .add = FALSE,
  .drop = NULL,
  .sort = get_collapse("sort"),
  .decreasing = FALSE,
  .na.last = TRUE,
  .return.groups = TRUE,
  .return.order = .sort,
  .method = "auto"
)

ungroup_(.data = (.), ..., .na.last = TRUE, .method = "auto")

group_vars_(.data = (.), return = "names")

group_rows_(.data = (.))

group_data_(.data = (.))

group_indices_(.data = (.), ...)

group_keys_(.data = (.), ...)

groups_(.data = (.))

group_size_(.data = (.))

n_groups_(.data = (.))

as.grouped_df(x, ...)

as_grouped_df(x, ...)
```

```
## Default S3 method:
as.grouped_df(x, ...)

## S3 method for class 'grouped_df'
as.grouped_df(x, ...)

## S3 method for class 'GRP_df'
as.grouped_df(x, ...)

## S3 method for class 'grouped_df'
print(x, ...)

is.grouped_df(x, collapse = FALSE)
```

Arguments

.data	A data frame (data.frame, data.table, or tibble)
...	For group_by_(): grouping variables as formulas (e.g., ~cyl) or character names. For ungroup_(): optional variables to remove from grouping (if omitted, removes all grouping).
.add	Logical. If TRUE, add new grouping variables to existing ones. If FALSE (default), replace existing grouping.
.drop	Logical. Should unused factor levels be dropped? Default is TRUE, unless the data was previously grouped with .drop = FALSE.
.sort	Logical. Should groups be sorted? Default uses the collapse package setting.
.decreasing	Logical. Should sorting be in decreasing order? Default is FALSE.
.na.last	Logical. Should NA values be sorted last? Default is TRUE.
.return.groups	Logical. Should group information be stored? Default is TRUE.
.return.order	Logical. Should group order be stored? Default follows .sort.
.method	Character. Method for grouping: "auto" (default), "hash", or "radix".
return	What to return: "data" or 1, "unique" or 2 for unique rows of grouping columns, "names" or 3 (default) for names of grouping columns, "indices" or 4 for integer indices of grouping columns, "named_indices" or 5 for named indices, "logicial" or 6 for logical selection vector of grouping columns, or "named_logical" or 7 for named logical.
x	An object to convert to **grouped_df**, or to check as such
collapse	Logical. If TRUE, check if it is a special version of **grouped_df** build by group_by_() or collapse::fgroup_by(), that is, a **GRP_df** object only partly compatible with an object obtained with dplyr::group_by().

Value

- group_by_() returns a grouped data frame (GRP_df or grouped_df class)

- ungroup_() returns the data frame without grouping (or with partial grouping if specific variables removed)
- group_vars_() returns names, data, or indices depending on return arg
- group_rows_() returns a list of integer vectors with row indices per group
- group_data_() returns a tibble with grouping columns and a .rows column
- group_indices_() returns an integer vector with group ID for each row
- group_keys_() returns a data frame with unique grouping variable values
- groups_() returns a list of symbols (grouping variable names)
- group_size_() returns an integer vector with row counts per group
- n_groups_() returns a single integer (total number of groups)
- as.grouped_df() returns a grouped_df object

See Also

dplyr::group_by(), dplyr::ungroup(), dplyr::group_vars(), dplyr::group_rows(), dplyr::group_data(), dplyr::group_indices(), dplyr::group_keys(), dplyr::groups(), dplyr::group_size(), dplyr::n_groups(), collapse::fgroup_by()

Examples

```
library(svTidy)
data(mtcars)

# Group by single variable
mtcars |> group_by_(~cyl)

# Group by multiple variables using formulas
mtcars_grouped <- mtcars |> group_by_(~cyl, ~gear)

# Group using character names
mtcars |> group_by_('cyl', 'gear')

# Add grouping variables to existing groups
mtcars |>
  group_by_(~cyl) |>
  group_by_(~gear, .add = TRUE)

# Get grouping variable names
mtcars_grouped |> group_vars_()

# Get number of groups
mtcars_grouped |> n_groups_()

# Get size of each group
mtcars_grouped |> group_size_()

# Get group indices for each row
mtcars_grouped |> group_indices_()
```

```
# Get unique grouping keys
mtcars_grouped |> group_keys_()

# Get row indices for each group
mtcars_grouped |> group_rows_()

# Get complete group data
mtcars_grouped |> group_data_()

# Ungroup completely
mtcars_grouped |> ungroup_()

# Ungroup specific variables
mtcars |>
  group_by_(~cyl, ~gear, ~am) |>
  ungroup_(~gear)

# Use with other operations
mtcars |>
  group_by_(~cyl) |>
  summarise_(~mean(mpg), ~mean(hp))
```

`joining` *Joining Functions*

Description

Functions for joining two data frames based on matching values in key columns.

These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- `left_join_()` - Keep all rows from x, add matching columns from y
- `right_join_()` - Keep all rows from y, add matching columns from x
- `inner_join_()` - Keep only rows with matches in both x and y
- `full_join_()` - Keep all rows from both x and y
- `semi_join_()` - Keep rows in x that have a match in y (no columns from y)
- `anti_join_()` - Keep rows in x that do NOT have a match in y
- `join_()` - Generic join function with `how` parameter

Usage

```
join_(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  unmatched = "drop",
  relationship = NULL,
  sort = FALSE,
  verbose = 0,
  column = NULL,
  attr = NULL,
  how = "full"
)

left_join_(
  x = (.),
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  unmatched = "drop",
  relationship = NULL,
  sort = FALSE,
  verbose = 0,
  column = NULL,
  attr = NULL
)

right_join_(
  x = (.),
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
```

```
  unmatched = "drop",
  relationship = NULL,
  sort = FALSE,
  verbose = 0,
  column = NULL,
  attr = NULL
)

inner_join_(
  x = (.),
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  unmatched = "drop",
  relationship = NULL,
  sort = FALSE,
  verbose = 0,
  column = NULL,
  attr = NULL
)

full_join_(
  x = (.),
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  relationship = NULL,
  sort = FALSE,
  verbose = 0,
  column = NULL,
  attr = NULL
)

semi_join_(
  x = (.),
  y,
  by = NULL,
  copy = FALSE,
```

```
  ...,
  na_matches = c("na", "never"),
  sort = FALSE,
  verbose = 0,
  column = NULL,
  attr = NULL
)

anti_join_(
  x = (.),
  y,
  by = NULL,
  copy = FALSE,
  ...,
  na_matches = c("na", "never"),
  sort = FALSE,
  verbose = 0,
  column = NULL,
  attr = NULL
)
```

Arguments

x	A data frame (data.frame, data.table, or tibble)
y	A data frame to join with x
by	A character vector of column names to join by. If NULL, uses all columns with common names across x and y. Can also be a named character vector to join by different column names (e.g., c("a" = "b") joins x$a to y$b), or a dplyr::join_by() object for complex joins.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same source as x. This allows you to join tables across data sources, but is potentially expensive.
suffix	Character vector of length 2 specifying suffixes to append to duplicate column names. Default is c(".x", ".y").
...	Additional arguments (currently unused, for compatibility)
keep	Should the join keys from both x and y be preserved in the output? If NULL (default), keeps join keys from x only for equality joins, but keeps keys from both for inequality joins. If TRUE, all keys from both inputs are retained. If FALSE, only keys from x are retained. Note: when keep = TRUE, calculation is delegated to dplyr join methods.
na_matches	Should two NA or two NaN values match? "na" (default) treats two NA or two NaN values as equal, like %in%, match(), and merge(). "never" treats two NA or two NaN values as different, and will never match them together or to any other values. When "never", calculation is delegated to dplyr join methods.
multiple	Handling of rows in x with multiple matches in y. Options: "all" (default) returns every match detected in y (SQL behavior), "first" returns the first match

detected in y, "last" returns the last match detected in y, "any" returns one match (faster but non-deterministic). For "any" and "last" (and "first" for right joins), calculation is delegated to dplyr join methods.

unmatched — How should unmatched keys that would result in dropped rows be handled? "drop" (default) drops unmatched keys from the result. "error" throws an error if unmatched keys are detected. Can also be a named list like list(x = 1, y = 0.5, fail = "warning") specifying the proportions that must match and the action ("message", "warning", or "error"). Not available for full_join_(), semi_join_(), and anti_join_().

relationship — Expected relationship between the keys of x and y: NULL (default) has no expectations but warns for many-to-many, "one-to-one" expects each row in x matches at most 1 row in y and vice versa, "one-to-many" expects each row in y matches at most 1 row in x, "many-to-one" expects each row in x matches at most 1 row in y, "many-to-many" has no restrictions (explicit about relationship).

sort — Logical. If TRUE, the result is sorted by the grouping variables. Default is FALSE.

verbose — Integer controlling information printed about the join: 0 means no output (default), 1 prints join information, and 2 additionally prints the classes of the by columns. Note: This parameter is ignored when using dplyr join methods.

column — Name for an extra column to generate in the output indicating which dataset a record came from. TRUE creates a column named ".join", or provide a custom name as a character string. NULL (default) does not add this column. Note: This parameter is ignored when using dplyr join methods.

attr — Name for an attribute providing information about the join performed (including output of collapse::fmatch()) attached to the result. TRUE creates an attribute named "join.match", or provide a custom name. NULL (default) does not add this attribute. Note: This parameter is ignored when using dplyr join methods.

how — Character string specifying the join type for join_(): "full" (default), "inner", "left", "right", "semi", or "anti".

Value

A data frame of the same type as x``. The order of the rows and columns of x‘ is preserved as much as possible.

- left_join_() returns all rows from x, and all columns from x and y
- right_join_() returns all rows from y, and all columns from x and y
- inner_join_() returns only rows with matches in both x and y
- full_join_() returns all rows from both x and y
- semi_join_() returns rows from x (no columns added from y)
- anti_join_() returns rows from x with no match in y (no columns from y)

See Also

dplyr::left_join(), dplyr::right_join(), dplyr::inner_join(), dplyr::full_join(), dplyr::semi_join(), dplyr::anti_join(), collapse::join(), dplyr::join_by()

Examples

```
library(svTidy)

# Create example datasets
band_members <- data.frame(
  name = c("Mick", "John", "Paul"),
  band = c("Stones", "Beatles", "Beatles")
)
band_instruments <- data.frame(
  name = c("John", "Paul", "Keith"),
  plays = c("guitar", "bass", "guitar")
)

# Left join - keep all rows from x
band_members |> left_join_(band_instruments, by = "name")

# Right join - keep all rows from y
band_members |> right_join_(band_instruments, by = "name")

# Inner join - keep only matching rows
band_members |> inner_join_(band_instruments, by = "name")

# Full join - keep all rows from both
band_members |> full_join_(band_instruments, by = "name")

# Semi join - filter x to rows matching y (no columns from y)
band_members |> semi_join_(band_instruments, by = "name")

# Anti join - filter x to rows NOT matching y
band_members |> anti_join_(band_instruments, by = "name")

# Join by different column names
band_instruments2 <- data.frame(
  artist = c("John", "Paul", "Keith"),
  plays = c("guitar", "bass", "guitar")
)
band_members |> left_join_(band_instruments2, by = c("name" = "artist"))

# Add suffix to duplicate columns
df1 <- data.frame(id = 1:3, value = c("a", "b", "c"))
df2 <- data.frame(id = 2:4, value = c("B", "C", "D"))
df1 |> full_join_(df2, by = "id", suffix = c("_x", "_y"))

# Control handling of multiple matches
df_x <- data.frame(key = c(1, 1, 2), x = c("a", "b", "c"))
df_y <- data.frame(key = c(1, 1, 2), y = c("A", "B", "C"))
df_x |> left_join_(df_y, by = "key", multiple = "all")
df_x |> left_join_(df_y, by = "key", multiple = "first")

# Validate relationships
df_one <- data.frame(id = 1:3, val = c("a", "b", "c"))
df_many <- data.frame(id = c(1, 1, 2), val = c("A", "B", "C"))
```

```
## Not run:
# This will error - expects one-to-one but is one-to-many
df_one |> left_join_(df_many, by = "id", relationship = "one-to-one")

## End(Not run)
# This works - explicitly one-to-many
df_one |> left_join_(df_many, by = "id", relationship = "one-to-many")

# Add indicator column showing source
band_members |>
  full_join_(band_instruments, by = "name", column = "source")

# Use generic join_() with how parameter
band_members |> join_(band_instruments, by = "name", how = "inner")
band_members |> join_(band_instruments, by = "name", how = "left")

# Handle unmatched keys
## Not run:
# Error if any keys don't match
band_members |>
  inner_join_(band_instruments, by = "name", unmatched = "error")

## End(Not run)
```

`library_dplyr` *Load {dplyr} or {tidyr} without functions ending with an underscore*

Description

This is the same as `library(dplyr)` or `library(tidyr)`, but excluding all functions that end with an underscore and that may conflict with {svTidy} corresponding ones. Note that these functions are deprecated in recent versions of {dplyr} and {tidyr}, and they are even defunct in version 1.2.0 or greater of {dplyr}.

Usage

```
library_dplyr(..., exclude)

library_tidyr(..., exclude)
```

Arguments

`...`	Further arguments passed to `base::library()`
`exclude`	A list of functions to exclude. Leave this argument missing to exclude all underscore functions from the package by default.

Value

The list of attached packages invisibly, or `TRUE`/`FALSE` to indicate success if `logical.return = TRUE` is indicated.

See Also

base::library()

Examples

```
library_dplyr()
library_tidyr()
search()
# However, the functions with underscore are not directly accessible
# (unless you make library(svTidy) of course)
get0('mutate_')
```

---

list_sciviews_functions

List SciViews Functions

Description

Give a comprehensive list of the SciViews functions (ending with _ and similar to their dplyr or tidyr equivalents). There are two major differences from the original dplyr and tidyr functions: (1) the data-dot mechanism replaces . as .data= in case it is not provided, and (2) they use a formula-masking mechanism instead of data-masking or tidy-select. In this case, you can either use standard evaluation, specifying df$var like for many base R function, of a formula, like ~var.

Usage

```
list_sciviews_functions()
```

---

mutating

Mutating Functions

Description

Functions for creating new columns or modifying existing columns in a data frame. These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- mutate_() - Add new columns or modify existing ones
- transmute_() - Create new columns and drop all others

Usage

```
mutate_(
  .data = (.),
  ...,
  .by = NULL,
  .keep = "all",
  .before = NULL,
  .after = NULL,
  .cols = NULL
)

transmute_(.data, ...)
```

Arguments

.data	A data frame (data.frame, data.table, or tibble)
...	Name-value pairs specifying new columns or modifications. Names are column names; values are expressions to compute. Use formulas for non-standard evaluation (e.g., new_col = ~mean(old_col)), or provide character names and expressions for standard evaluation. New columns can refer to columns just created. Within formulas, you can provide column names use like this: 'new_col_name' ~ median(old_col). You can also use the name of an object that contains the column name. For instance, col_names <- c("new_col1", "new_col2"), then in mutate_(), use col_name[1] ~ median(old_col).
.by	Optional grouping variables for per-group computations. Provide as formulas (e.g., ~group_col) or character names. Groups are temporary and not preserved in the output. Cannot be used with grouped data frames.
.keep	Control which columns to keep in the output: • "all" (default) - Keep all existing columns plus new ones • "used" - Keep columns used to make new columns, and new columns • "unused" - Keep columns not used to make new columns, and new columns. This is useful if you do not need the columns that are used to create the new ones. • "none" - Keep only new columns (same as transmute_())
.before	Optional column name or position to place new columns before. Not yet implemented - use dplyr::mutate() instead.
.after	Optional column name or position to place new columns after. Not yet implemented - use dplyr::mutate() instead.
.cols	Optional character vector of column names to operate on. If provided, only these columns are modified or created.

Value

A data frame of the same type as .data with modified or new columns.

- mutate_() returns all columns (by default), including new/modified ones
- transmute_() returns only the newly created columns

See Also

dplyr::mutate(), dplyr::transmute(), collapse::fmutate()

Examples

```
library(svTidy)
data(mtcars)

# Create new columns using formulas
mtcars |> mutate_(hp_per_cyl = ~hp/cyl)

# Multiple new columns
mtcars |>
  mutate_(
    hp_per_cyl = ~hp/cyl,
    wt_kg = ~wt * 453.592
  )

# Modify existing column
mtcars |> mutate_(mpg = ~mpg * 1.5)

# Reference newly created columns
mtcars |>
  mutate_(
    hp_per_cyl = ~hp/cyl,
    hp_per_cyl_scaled = ~hp_per_cyl * 100
  )

# Use column name in a variable
col_name <- "power_ratio"
mtcars |> mutate_(col_name ~ hp/wt)

# Group-wise computations with .by
mtcars |>
  mutate_(
    mpg_centered = ~mpg - mean(mpg),
    .by = 'cyl'
  )

# Multiple grouping variables
mtcars |>
  mutate_(
    hp_rank = ~rank(hp),
    .by = c('cyl', 'gear')
  )

# Control which columns to keep
mtcars |>
  mutate_(
    hp_per_cyl = ~hp/cyl,
    .keep = "used"
  )
```

```
mtcars |>
  mutate_(
    efficiency = ~mpg/hp,
    .keep = "unused"
  )

# transmute_() keeps only new columns
mtcars |>
  transmute_(
    car = ~rownames(mtcars),
    hp_per_cyl = ~hp/cyl,
    efficiency = ~mpg/wt
  )

# Conditional mutations
mtcars |>
  mutate_(
    performance = ~ifelse(hp > 150, "high", "normal")
  )

# Use with grouped data
mtcars |>
  group_by_(~cyl) |>
  mutate_(mpg_ratio = ~mpg/mean(mpg))

# Complex transformations
mtcars |>
  mutate_(
    log_hp = ~log(hp),
    sqrt_wt = ~sqrt(wt),
    hp_wt_interaction = ~hp * wt
  )
```

`pivoting` *Pivoting Functions*

Description

Functions for pivoting data between long and wide formats.

These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- `pivot_longer_()` - Convert data from wide to long format
- `pivot_wider_()` - Convert data from long to wide format

Usage

```
pivot_longer_(
  .data = (.),
  cols,
  ...,
  cols_vary = "fastest",
  names_to = "name",
  names_prefix = NULL,
  values_to = "value",
  values_drop_na = FALSE,
  factor = FALSE
)

pivot_wider_(
  .data = (.),
  ...,
  id_cols = NULL,
  id_expand = FALSE,
  names_from = "",
  names_prefix = "",
  names_vary = "fastest",
  values_from = "",
  values_fill = NULL,
  values_fn = "last",
  drop = TRUE,
  sort = FALSE
)
```

Arguments

`.data`	A data frame (data.frame, data.table, or tibble)
`cols`	For `pivot_longer_()`: columns to pivot into longer format. Use tidy-select syntax with formulas (e.g., `~col1:col5`, `~starts_with("x")`), or provide column positions or names as a vector.
`...`	Additional arguments (currently unused, for compatibility)
`cols_vary`	Character. Either `"fastest"` (default) or `"slowest"`. If `"fastest"`, keeps individual rows from `cols` close together in the output. If `"slowest"`, keeps individual columns from `cols` close together.
`names_to`	Character string specifying the name of the column to create from the column names being pivoted. Default is `"name"`.
`names_prefix`	Character. A regular expression used to remove matching text from the start of each variable name before creating the names column.
`values_to`	Character string specifying the name of the column to create from the cell values. Default is `"value"`.
`values_drop_na`	Logical. If `TRUE`, rows containing only `NA` values in the `values_to` column are dropped from the result. Default is `FALSE`.

factor	Logical. If TRUE, convert the names and values columns to factors. If FALSE (default), leave them as character strings.
id_cols	For pivot_wider_(): columns that uniquely identify each observation. Use tidy-select syntax or NULL (default) to use all columns except names_from and values_from.
id_expand	Logical. If TRUE, expand the id_cols to include all possible combinations. Default is FALSE.
names_from	For pivot_wider_(): column(s) containing the names for the new columns. Provide as unquoted names or character vector. Default is name.
names_vary	Character. How column names are constructed when multiple names_from or values_from columns are provided: "fastest" (default), "slowest", "transpose", or "slowtranspose".
values_from	For pivot_wider_(): column(s) containing the values for the new columns. Provide as unquoted names or character vector. Default is value.
values_fill	Optional scalar value to use for missing combinations. Default is NULL (use NA).
values_fn	Function to apply when there are multiple values for a cell. Can be a string naming an internal function ("first", "last" (default), "count", "sum", "mean", "min", "max"), or a formula calling an external function with first argument .x (e.g., ~fmedian(.x, na.rm = TRUE)).
drop	Logical. Should unused factor levels be dropped? Default is TRUE.
sort	Logical. If TRUE, sort the result so the largest groups are shown on top. Default is FALSE.

Value

A data frame of the same type as .data in the pivoted format. pivot_longer_() returns a data frame with more rows and fewer columns. pivot_wider_() returns a data frame with fewer rows and more columns.

See Also

tidyr::pivot_longer(), tidyr::pivot_wider(), collapse::pivot()

Examples

```
library(svTidy)

# Create sample wide data
wide_data <- data.frame(
  id = 1:3,
  year = c(2020, 2021, 2022),
  q1 = c(100, 110, 120),
  q2 = c(105, 115, 125),
  q3 = c(110, 120, 130),
  q4 = c(115, 125, 135)
)
```

```
# Pivot from wide to long format
wide_data |>
  pivot_longer_(~q1:q4, names_to = "quarter", values_to = "sales")

# Use tidy-select helpers
wide_data |>
  pivot_longer_(~starts_with("q"), names_to = "quarter", values_to = "sales")

# Remove prefix from column names
wide_data |>
  pivot_longer_(
    ~q1:q4,
    names_to = "quarter",
    values_to = "sales",
    names_prefix = "q"
  )

# Control row ordering with cols_vary
wide_data |>
  pivot_longer_(~q1:q4, cols_vary = "slowest")

# Drop NA values
wide_na <- wide_data
wide_na$q3[2] <- NA
wide_na |>
  pivot_longer_(~q1:q4, values_drop_na = TRUE)

# Convert to factors
wide_data |>
  pivot_longer_(~q1:q4, factor = TRUE)

# Create sample long data
long_data <- data.frame(
  id = rep(1:3, each = 4),
  year = rep(c(2020, 2021, 2022), each = 4),
  quarter = rep(c("q1", "q2", "q3", "q4"), 3),
  sales = c(100, 105, 110, 115, 110, 115, 120, 125, 120, 125, 130, 135)
)

# Pivot from long to wide format
long_data |>
  pivot_wider_(names_from = "quarter", values_from = "sales")

# Specify id columns explicitly
long_data |>
  pivot_wider_(
    id_cols = ~c(id, year),
    names_from = "quarter",
    values_from = "sales"
  )

# Add prefix to new column names
long_data |>
```

```
  pivot_wider_(
    names_from = "quarter",
    values_from = "sales",
    names_prefix = "sales_"
  )

# Fill missing values
long_data |>
  pivot_wider_(
    names_from = "quarter",
    values_from = "sales",
    values_fill = 0
  )

# Handle multiple values with aggregation
long_dup <- rbind(long_data, long_data[1:3, ])
long_dup |>
  pivot_wider_(
    names_from = "quarter",
    values_from = "sales",
    values_fn = "mean"
  )

# Use custom aggregation function
long_dup |>
  pivot_wider_(
    names_from = "quarter",
    values_from = "sales",
    values_fn = ~sum(.x)
  )

# Multiple names_from and values_from
long_multi <- data.frame(
  id = rep(1:2, each = 4),
  metric = rep(c("sales", "profit"), 4),
  quarter = rep(c("q1", "q2"), each = 2, times = 2),
  value = c(100, 20, 105, 22, 110, 24, 115, 26)
)
long_multi |>
  pivot_wider_(
    names_from = c("quarter", "metric"),
    values_from = "value"
  )

# Round-trip: wide -> long -> wide
wide_data |>
  pivot_longer_(~q1:q4, names_to = "quarter", values_to = "sales") |>
  pivot_wider_(names_from = "quarter", values_from = "sales")
```

---

selecting *Selecting Functions*

Description

Functions for selecting, renaming, and extracting columns from a data frame.

These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- `select_()` - Select columns by name, position, or using tidy-select helpers
- `pull_()` - Extract a single column as a vector
- `rename_()` - Rename columns using new_name = old_name pairs
- `rename_with_()` - Rename columns using a function
- `all_of()` - Helper for selecting all specified columns (errors if missing)

Usage

```
select_(.data = (.), ...)

pull_(.data = (.), var = -1, name = NULL, ...)

rename_(.data = (.), ...)

rename_with_(.data = (.), .fn, .cols = ~everything(), ...)

all_of(x)
```

Arguments

`.data`	A data frame (data.frame, data.table, or tibble)
`...`	For `select_()` and `rename_()`: column names, positions, or tidy-select expressions. Use formulas for non-standard evaluation (e.g., `~starts_with("x")`). For `rename_()`, provide pairs as `new_name = old_name` or `new_name = ~old_name`. For `rename_with_()`: additional arguments passed to `.fn`.
`var`	For `pull_()`: the column to extract. Can be a column name (as character), a formula with column name on RHS (e.g., `~mpg`), a positive integer for position from left, or a negative integer for position from right. Default is `-1` (last column).
`name`	For `pull_()`: optional column to use as names for the resulting vector. Specified the same way as `var`.
`.fn`	For `rename_with_()`: a function to apply to column names, or a formula expression using `.x` as the column names (e.g., `~toupper(.x)`).
`.cols`	For `rename_with_()`: columns to rename. Use tidy-select syntax with formulas. Default is `~everything()` (all columns).
`x`	For `all_of()`: a character vector of column names. All must exist or an error is raised.

Value

- `select_()` returns a data frame with only the selected columns
- `pull_()` returns a vector (named or unnamed depending on `name` parameter)
- `rename_()` returns the data frame with renamed columns
- `rename_with_()` returns the data frame with renamed columns
- `all_of()` returns the input vector (used inside select/rename functions)

See Also

`dplyr::select()`, `dplyr::pull()`, `dplyr::rename()`, `dplyr::rename_with()`, `dplyr::all_of()`, `tidyselect::starts_with()`, `tidyselect::ends_with()`, `tidyselect::contains()`, `tidyselect::matches()`, `tidyselect::everything()`, `collapse::fselect()`

Examples

```
library(svTidy)
data(mtcars)

# Select specific columns by name
mtcars |> select_(~mpg, ~cyl, ~hp)

# Select columns by position
mtcars |> select_(1, 3, 5)

# Select range of columns
mtcars |> select_(~mpg:hp)

# Use tidy-select helpers
mtcars |> select_(~starts_with("d"))
mtcars |> select_(~ends_with("p"))
mtcars |> select_(~contains("a"))

# Exclude columns with minus
mtcars |> select_(~-c(mpg, cyl))

# Select all numeric columns
mtcars |> select_(~where(is.numeric))

# Combine multiple selection methods
mtcars |> select_(~mpg, ~starts_with("d"), ~hp)

# Use all_of() for programmatic selection
cols <- c("mpg", "cyl", "hp")
mtcars |> select_(~all_of(cols))

# Pull a column as a vector (by name)
mtcars |> pull_(~mpg)

# Pull by position (last column)
mtcars |> pull_(-1)
```

```
# Pull first column
mtcars |> pull_(1)

# Pull with names from another column
mtcars |> pull_(~mpg, name = ~cyl)

# Rename columns with new_name = old_name
mtcars |> rename_(miles_per_gallon = ~mpg, cylinders = ~cyl)

# Rename using column positions
mtcars |> rename_(miles_per_gallon = 1, cylinders = 2)

# Rename multiple columns
mtcars |>
  rename_(
    miles_per_gallon = ~mpg,
    cylinders = ~cyl,
    horsepower = ~hp
  )

# Rename all columns with a function
mtcars |> rename_with_(toupper)

# Rename using a formula with .x
mtcars |> rename_with_(~paste0("var_", .x))

# Rename with string manipulation
mtcars |> rename_with_(~tolower(.x))
mtcars |> rename_with_(~gsub("_", ".", .x))

# Rename only selected columns
mtcars |> rename_with_(toupper, .cols = ~starts_with("d"))

# Rename specific columns by name
mtcars |> rename_with_(toupper, .cols = c("mpg", "cyl", "hp"))

# Chain operations
mtcars |>
  select_(~mpg, ~cyl, ~hp, ~wt) |>
  rename_(efficiency = ~mpg, weight = ~wt) |>
  arrange_(~cyl)

# Use in data pipeline
mtcars |>
  select_(~where(is.numeric)) |>
  rename_with_(tolower) |>
  filter_(~cyl > 4) |>
  pull_(~mpg)
```

summarising *Summarising and Counting Functions*

Description

Functions for summarising data and counting observations in data frames.

These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- `summarise_()` / `summarize_()` - Compute summary statistics for groups
- `reframe_()` - Similar to summarise but always returns ungrouped data
- `count_()` - Count observations by group
- `tally_()` - Count total observations (wrapper around count_)
- `add_count_()` - Add count column to data frame
- `add_tally_()` - Add total count column to data frame

Usage

```
summarise_(
  .data = (.),
  ...,
  .by = NULL,
  .groups = "drop_last",
  .keep.group_vars = TRUE,
  .cols = NULL
)

summarize_(
  .data = (.),
  ...,
  .by = NULL,
  .groups = "drop_last",
  .keep.group_vars = TRUE,
  .cols = NULL
)

reframe_(
  .data,
  ...,
  .by = NULL,
  .groups = "drop",
  .keep.group_vars = TRUE,
  .cols = NULL
```

```
)

count_(
  .data = (.),
  ...,
  wt = NULL,
  name = "n",
  sort = FALSE,
  decreasing = TRUE,
  .drop = TRUE,
  add = FALSE
)

tally_(.data = (.), wt = NULL, name = "n", sort = FALSE, decreasing = TRUE)

add_count_(
  .data = (.),
  ...,
  wt = NULL,
  name = "n",
  sort = FALSE,
  decreasing = TRUE,
  .drop = TRUE
)

add_tally_(.data = (.), wt = NULL, name = "n", sort = FALSE, decreasing = TRUE)
```

Arguments

.data	A data frame (data.frame, data.table, or tibble)
...	For summarise_() and reframe_(): name-value pairs of summary functions. Names are column names in the output; values are expressions to compute. Use formulas for non-standard evaluation (e.g., ~mean_mpg = mean(~mpg)). For count_() and add_count_(): grouping variables specified as formulas (e.g., ~cyl, ~gear) or character names. Can include named expressions to create new grouping variables before counting.
.by	Optional temporary grouping variables for per-group computations. Provide as formulas (e.g., ~group_col) or character names. Groups are temporary and not preserved in the output. Cannot be used with grouped data frames.
.groups	Control grouping of the result. Options: • "drop_last" (default) - Drop the last grouping level • "drop" - Remove all grouping • "keep" - Keep all grouping levels • "rowwise" - Not implemented For reframe_(), only "drop" is allowed.
.keep.group_vars	Logical. If TRUE (default), keep grouping variables in the result.

.cols	Optional character vector of column names to operate on. Currently only NULL (default) is implemented.
wt	For count_(), tally_(), add_count_(), and add_tally_(): frequency weights. Can be NULL (default, counts rows), a numeric vector, a column name as character, or a formula (e.g., ~weight_col).
name	Character string specifying the name of the count column created in the output. Default is "n".
sort	Logical. If TRUE, sort the result by the count column in decreasing order (or as specified by decreasing). Default is FALSE.
decreasing	Logical. If TRUE (default), sort counts in decreasing order when sort = TRUE.
.drop	Logical. If TRUE (default), drop unused factor levels. Note: .drop = FALSE is not yet implemented in count_().
add	Logical. If TRUE, add the count column to the original data frame instead of returning a summary. Default is FALSE.

Value

- summarise_() returns a data frame with one row per group (or one row if ungrouped), containing the summary statistics. Grouping depends on .groups.
- reframe_() returns an ungrouped data frame (can have any number of rows per group).
- count_() returns a data frame with one row per unique combination of grouping variables, plus a count column.
- tally_() returns a data frame with one row per group showing the count.
- add_count_() returns the original data with an additional count column.
- add_tally_() returns the original data with an additional count column.

Note

The summarise_() function does not support n() as does dplyr::summarise(). You can use svBase::fn() instead, but then you must give a variable name as argument. The svBase::fn() alternative can also be used in dplyr::summarise() for homogeneous syntax between the two.

See Also

dplyr::summarise(), dplyr::reframe(), dplyr::count(), dplyr::tally(), dplyr::add_count(), dplyr::add_tally(), collapse::fsummarise(), collapse::fcount(), svBase::fn()

Examples

```
library(svTidy)
data(mtcars)

# Basic summarise - single summary statistic
mtcars |> summarise_(mean_mpg = ~mean(mpg))

# Multiple summary statistics
mtcars |>
```

```
  summarise_(
    mean_mpg = ~mean(mpg),
    sd_mpg   = ~sd(mpg),
    max_hp   = ~max(hp)
  )

# Summarise by groups
mtcars |>
  group_by_(~cyl) |>
  summarise_(
    mean_mpg = ~mean(mpg),
    mean_hp  = ~mean(hp)
  )

# Use .by for temporary grouping
mtcars |>
  summarise_(
    mean_mpg = ~mean(mpg),
    count    = ~length(mpg),
    .by = 'cyl'
  )

# Multiple grouping variables with .by
mtcars |>
  summarise_(
    mean_mpg = ~mean(mpg),
    .by = c('cyl', 'gear')
  )

# Control grouping of result
mtcars |>
  group_by_(~cyl, ~gear) |>
  summarise_(mean_mpg = ~mean(mpg), .groups = "drop")

mtcars |>
  group_by_(~cyl, ~gear) |>
  summarise_(mean_mpg = ~mean(mpg), .groups = "keep")

# Using standard evaluation (ungrouped data only)
mtcars |> summarise_(mean_mpg = mean(mtcars$mpg))

# reframe_() for summaries returning multiple rows per group
mtcars |>
  group_by_(~cyl) |>
  reframe_(quantile_mpg = ~quantile(mpg, c(0.25, 0.5, 0.75)))

# Count observations by group
mtcars |> count_(~cyl)

# Count by multiple variables
mtcars |> count_(~cyl, ~gear)

# Count with sorting
```

```
mtcars |> count_(~cyl, sort = TRUE)

# Count in increasing order
mtcars |> count_(~cyl, sort = TRUE, decreasing = FALSE)

# Count with weights
mtcars |> count_(~cyl, wt = ~mpg)

# Count with computed grouping variable
mtcars |> count_(high_mpg = ~mpg > 20)

# Combine grouping and computation
mtcars |> count_(~cyl, high_hp = ~hp > 150)

# tally_() - count rows (optionally by existing groups)
mtcars |> tally_()

mtcars |>
  group_by_(~cyl) |>
  tally_()

# tally with weights
mtcars |>
  group_by_(~cyl) |>
  tally_(wt = ~hp)

# add_count_() - add count column without collapsing
mtcars |> add_count_(~cyl)

# add_count with custom column name
mtcars |> add_count_(~cyl, name = "n_cyl")

# add_count by multiple variables
mtcars |> add_count_(~cyl, ~gear)

# add_tally_() - add total count to each row
mtcars |> add_tally_()

mtcars |>
  group_by_(~cyl) |>
  add_tally_()

# Chain operations
mtcars |>
  count_(~cyl, ~gear, sort = TRUE) |>
  mutate_(pct = ~n/sum(n) * 100)

# Use with filtering
mtcars |>
  add_count_(~cyl) |>
  filter_(~n > 10)
```

`tidying` *Tidying Functions*

Description

Functions for tidying data by separating, uniting, filling, and handling missing values.

These are SciViews::R versions of tidyverse functions with standard evaluation and formula-based non-standard evaluation (ending with underscore _). They work with data.frame, data.table, and tibbles.

Functions:

- `separate_()` - Separate one column into multiple columns by splitting on a separator
- `unite_()` - Unite multiple columns into one by pasting strings together
- `fill_()` - Fill missing values using previous or next non-missing value
- `drop_na_()` - Drop rows containing missing values
- `replace_na_()` - Replace missing values with a specified value
- `uncount_()` - Duplicate rows according to a weighting variable

Usage

```
separate_(
  .data = (.),
  col,
  into,
  sep = "[^[:alnum:]]+",
  remove = TRUE,
  convert = FALSE,
  extra = "warn",
  fill = "warn",
  fixed = FALSE,
  ...
)

unite_(.data = (.), col, ..., sep = "_", remove = TRUE, na.rm = FALSE)

fill_(.data = (.), ..., .direction = "down")

drop_na_(.data = (.), ..., .na.attr = FALSE, .prop = 0)

replace_na_(.data = (.), replace, ..., v = NULL)

uncount_(.data = (.), weights, ..., .remove = TRUE, .id = NULL)
```

Arguments

.data	A data frame (data.frame, data.table, or tibble)
col	For separate_(): the column to separate. Can be a column name as character, or a formula (e.g., ~col_name). For unite_(): the name of the new united column (character string or formula).
into	For separate_(): names of new variables to create as a character vector. Use NA to omit a variable in the output.
sep	For separate_() and unite_(): separator between columns. For separate_(), can be a character vector, a numeric vector of positions to split at, or a regular expression pattern. Default is "[^[:alnum:]]+" for separate_() and "_" for unite_().
remove	Logical. If TRUE (default), remove input columns from output.
convert	For separate_(): logical. If TRUE, attempts to convert new columns to appropriate types. Default is FALSE.
extra	For separate_() when sep is a character: what to do when there are too many pieces. Options: "warn" (default, warn and drop), "drop" (drop without warning), or "merge" (merge extra pieces with last).
fill	For separate_() when sep is a character: what to do when there are not enough pieces. Options: "warn" (default, warn and fill right with NA), "right" (fill right without warning), or "left" (fill left).
fixed	For separate_(): logical. If TRUE, sep is a fixed string. If FALSE (default), sep is a (perl) regular expression.
...	For separate_() and unite_(): additional arguments (currently unused). For fill_() and drop_na_(): columns to fill or check for NAs. Use formulas (e.g., ~col1, ~col2) or column names. If not provided, uses all columns.
na.rm	If TRUE, NAs are eliminated before uniting the values.
.direction	Direction in which to fill missing data: "down" (by default), "up", or "downup" (first down, then up), "updown" (the opposite).
.na.attr	logical. TRUE adds an attribute containing the removed cases. For compatibility reasons this is exactly the same format as na.omit(), i.e. the attribute is called "na.action" and of class **omit**
.prop	numeric. The proportion missing values in each case for the case to be considered as missing required to keep a
replace	If .data is a vector, a unique value to replace NAs, otherwise, a list of values, one per column of the data frame.
v	a vector where to replace NAs.
weights	A vector of weight to use to "uncount" .data.
.remove	If TRUE (default), and weights is the name of a column, that column is removed from .data.
.id	The name of the column for the origin id, either names if all other arguments are named, or numbers.

Value

A data frame of the same type as `.data` with the transformation applied.

- `separate_()` returns a data frame with the specified column split into multiple columns
- `unite_()` returns a data frame with specified columns combined into one
- `fill_()` returns a data frame with missing values filled
- `drop_na_()` returns a data frame with rows containing NAs removed
- `replace_na_()` returns a data frame or vector with NAs replaced by specified values
- `uncount_()` returns a data frame with rows duplicated according to weights

See Also

`tidyr::separate()`, `tidyr::unite()`, `tidyr::fill()`, `tidyr::drop_na()`, `tidyr::replace_na()`, `tidyr::uncount()`, `collapse::na_omit()`, `collapse::replace_na()`

Examples

```
library(svTidy)

# separate_() - split one column into multiple
df <- data.frame(x = c("a_b_c", "d_e_f", "g_h_i"))
df |> separate_(~x, into = c("A", "B", "C"), sep = "_")

# Use character name instead of formula
df |> separate_("x", into = c("A", "B", "C"), sep = "_")

# Drop a column with NA in into
df |> separate_(~x, into = c("A", NA, "C"), sep = "_")

# Keep original column
df |> separate_(~x, into = c("A", "B", "C"), sep = "_", remove = FALSE)

# Separate by numeric positions is not implemented yet
#df2 <- data.frame(date = c("20201231", "20210115", "20220601"))
#df2 |> separate_(~date, into = c("year", "month", "day"), sep = c(4, 6))

# Handle too many pieces
df3 <- data.frame(x = c("a_b_c", "d_e_f_g", "h_i"))
df3 |> separate_(~x, into = c("A", "B"), extra = "drop")
df3 |> separate_(~x, into = c("A", "B"), extra = "merge")

# Handle too few pieces
df3 |> separate_(~x, into = c("A", "B", "C"), fill = "right")

# unite_() - combine multiple columns into one
df4 <- data.frame(year = 2020:2022, month = 1:3, day = 10:12)
df4 |> unite_(~date, ~year, ~month, ~day, sep = "-")

# Keep original columns
df4 |> unite_(~date, ~year, ~month, ~day, sep = "-", remove = FALSE)
```

```
# Handle NAs in unite
df5 <- data.frame(x = c("a", "b", NA), y = c("d", NA, "f"))
df5 |> unite_(~z, ~x, ~y)
df5 |> unite_(~z, ~x, ~y, na.rm = TRUE)

# fill_() - fill missing values
df6 <- data.frame(
  group = c(1, 1, 1, 2, 2, 2),
  value = c(10, NA, NA, 20, NA, 30)
)
df6 |> fill_(~value)

# Fill upward
df6 |> fill_(~value, .direction = "up")

# Fill down then up
df6 |> fill_(~value, .direction = "downup")

# Fill specific columns
df7 <- data.frame(x = c(1, NA, 3), y = c(NA, 2, NA), z = c(1, 2, 3))
df7 |> fill_(~x, ~y, .direction = "down")

# Fill with grouped data
df6 |>
  group_by_(~group) |>
  fill_(~value)

# drop_na_() - remove rows with missing values
df8 <- data.frame(x = c(1, 2, NA), y = c("a", NA, "c"), z = 1:3)
df8 |> drop_na_()

# Drop NAs from specific columns only
df8 |> drop_na_(~x)
df8 |> drop_na_(~x, ~y)

# Use proportion threshold
df9 <- data.frame(x = c(1, NA, NA), y = c(NA, 2, NA), z = c(NA, NA, 3))
df9 |> drop_na_(.prop = 0.5)  # Drop rows with >= 50% NAs

# Keep track of removed rows
result <- df8 |> drop_na_(.na.attr = TRUE)
attr(result, "na.action")

# replace_na_() - replace NAs with a value
df10 <- data.frame(x = c(1, 2, NA), y = c(NA, "b", "c"))
df10 |> replace_na_(list(x = 0, y = "missing"))

# Replace in a single vector
vec <- c(1, 2, NA, 4, NA)
replace_na_(v = vec, replace = 0)

# Replace all NAs with same value (not standard tidyr)
```

```
df10 |> replace_na_(list(everywhere = 999))

# uncount_() - duplicate rows according to weights
df11 <- data.frame(x = c("a", "b", "c"), n = c(1, 2, 3))
df11 |> uncount_(~n)

# Keep the weight column
df11 |> uncount_(~n, .remove = FALSE)

# Add ID column to track original rows
df11 |> uncount_(~n, .id = "id")

# Use numeric weights vector
df12 <- data.frame(x = c("a", "b", "c"))
df12 |> uncount_(weights = c(2, 1, 3))
```

Index

add_count_ *(summarising)*, 28
add_tally_ *(summarising)*, 28
all_of *(selecting)*, 25
anti_join_ *(joining)*, 10
arrange_, 2
as.grouped_df *(grouping)*, 6
as_grouped_df *(grouping)*, 6

base::library(), *16, 17*
bind_cols_ *(bind_rows_)*, 3
bind_rows_, 3

collapse::fcount(), *30*
collapse::fgroup_by(), *8, 9*
collapse::fmatch(), *14*
collapse::fmutate(), *19*
collapse::fselect(), *26*
collapse::fsummarise(), *30*
collapse::join(), *14*
collapse::na_omit(), *35*
collapse::pivot(), *22*
collapse::replace_na(), *35*
count_ *(summarising)*, 28

distinct_ *(filtering)*, 4
dplyr::add_count(), *30*
dplyr::add_tally(), *30*
dplyr::all_of(), *26*
dplyr::anti_join(), *14*
dplyr::arrange(), *3*
dplyr::bind_cols(), *4*
dplyr::bind_rows(), *4*
dplyr::count(), *30*
dplyr::distinct(), *5*
dplyr::filter(), *5*
dplyr::full_join(), *14*
dplyr::group_by(), *8, 9*
dplyr::group_data(), *9*
dplyr::group_indices(), *9*
dplyr::group_keys(), *9*
dplyr::group_rows(), *9*
dplyr::group_size(), *9*
dplyr::group_vars(), *9*
dplyr::groups(), *9*
dplyr::inner_join(), *14*
dplyr::join_by(), *14*
dplyr::left_join(), *14*
dplyr::mutate(), *19*
dplyr::n_groups(), *9*
dplyr::pull(), *26*
dplyr::reframe(), *30*
dplyr::rename(), *26*
dplyr::rename_with(), *26*
dplyr::right_join(), *14*
dplyr::select(), *26*
dplyr::semi_join(), *14*
dplyr::slice(), *5*
dplyr::slice_head(), *5*
dplyr::slice_tail(), *5*
dplyr::summarise(), *30*
dplyr::tally(), *30*
dplyr::transmute(), *19*
dplyr::ungroup(), *9*
drop_na_ *(tidying)*, 33

fill_ *(tidying)*, 33
filter_ *(filtering)*, 4
filter_out_ *(filtering)*, 4
filtering, 4
full_join_ *(joining)*, 10

group_by_ *(grouping)*, 6
group_data_ *(grouping)*, 6
group_indices_ *(grouping)*, 6
group_keys_ *(grouping)*, 6
group_rows_ *(grouping)*, 6
group_size_ *(grouping)*, 6
group_vars_ *(grouping)*, 6
grouping, 6
groups_ *(grouping)*, 6

inner_join_ *(joining)*, 10
is.grouped_df *(grouping)*, 6

join_ *(joining)*, 10
joining, 10

left_join_ *(joining)*, 10
library_dplyr, 16
library_tidyr *(library_dplyr)*, 16
list_sciviews_functions, 17

mutate_ *(mutating)*, 17
mutating, 17

n_groups_ *(grouping)*, 6
na.omit(), *34*

pivot_longer_ *(pivoting)*, 20
pivot_wider_ *(pivoting)*, 20
pivoting, 20
print.grouped_df *(grouping)*, 6
pull_ *(selecting)*, 25

reframe_ *(summarising)*, 28
rename_ *(selecting)*, 25
rename_with_ *(selecting)*, 25
replace_na_ *(tidying)*, 33
right_join_ *(joining)*, 10

select_ *(selecting)*, 25
selecting, 24
semi_join_ *(joining)*, 10
separate_ *(tidying)*, 33
slice_ *(filtering)*, 4
slice_head_ *(filtering)*, 4
slice_tail_ *(filtering)*, 4
summarise_ *(summarising)*, 28
summarising, 28
summarize_ *(summarising)*, 28
svBase::fn(), *30*

tally_ *(summarising)*, 28
tidying, 33
tidyr::drop_na(), *35*
tidyr::fill(), *35*
tidyr::pivot_longer(), *22*
tidyr::pivot_wider(), *22*
tidyr::replace_na(), *35*
tidyr::separate(), *35*
tidyr::uncount(), *35*
tidyr::unite(), *35*
tidyselect::contains(), *26*
tidyselect::ends_with(), *26*
tidyselect::everything(), *26*
tidyselect::matches(), *26*
tidyselect::starts_with(), *26*
transmute_ *(mutating)*, 17

uncount_ *(tidying)*, 33
ungroup_ *(grouping)*, 6
unite_ *(tidying)*, 33