

Package: tidylog (via r-universe)

September 27, 2026

Type Package

Title Logging for 'dplyr' and 'tidyr' Functions

Version 1.1.0.9000

Description Provides feedback about 'dplyr' and 'tidyr' operations.

License MIT + file LICENSE

Imports dplyr, tidyr, glue, cli, rlang (>= 0.4.3)

Suggests testthat, units, nycflights13, covr, forcats, knitr,
rmarkdown, bench, purrr

Encoding UTF-8

URL <https://github.com/elbersb/tidylog/>

BugReports <https://github.com/elbersb/tidylog/issues>

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev

Repository <https://elbersb.r-universe.dev>

Date/Publication 2026-06-23 15:02:56 UTC

RemoteUrl <https://github.com/elbersb/tidylog>

RemoteRef HEAD

RemoteSha 65a398acbd42e444b86fd67e7bb9625e2b3103f9

Contents

add_count	3
add_tally	4
anti_join	5
count	6
display_changed_rows	7
distinct	8

distinct_all	9
distinct_at	10
distinct_if	11
drop_na	12
fill	12
filter	13
filter_all	14
filter_at	15
filter_if	16
filter_out	17
full_join	18
gather	20
group_by	21
group_by_all	22
group_by_at	23
group_by_if	24
inner_join	25
left_join	27
mutate	30
mutate_all	31
mutate_at	32
mutate_if	33
pivot_longer	34
pivot_wider	36
relocate	38
rename	39
rename_all	39
rename_at	40
rename_if	41
rename_with	42
replace_na	42
right_join	43
sample_frac	46
sample_n	47
select	48
select_all	48
select_at	49
select_if	50
semi_join	51
separate_wider_delim	52
separate_wider_position	54
separate_wider_regex	55
slice	57
slice_head	58
slice_max	60
slice_min	61
slice_sample	62
slice_tail	63

spread	65
summarise	66
summarise_all	67
summarise_at	68
summarise_if	69
summarize	70
summarize_all	71
summarize_at	72
summarize_if	73
tally	74
tidylog	75
top_frac	75
top_n	76
transmute	77
transmute_all	78
transmute_at	79
transmute_if	80
uncount	81
ungroup	81

Index	83
--------------	-----------

add_count	<i>Wrapper around <code>dplyr::add_count</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::add_count()` that prints information about the operation.

Usage

```
add_count(x, ...)
```

Arguments

x	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>).
...	Arguments passed on to <code>dplyr::add_count</code>
wt	<code><data-masking></code> Frequency weights. Can be NULL or a variable: • If NULL (the default), counts the number of rows in each group. • If a variable, computes <code>sum(wt)</code> for each group.
sort	If TRUE, will show the largest groups at the top.
name	The name of the new column in the output. If omitted, it will default to <code>n</code> . If there's already a column called <code>n</code> , it will use <code>nn</code> . If there's a column called <code>n</code> and <code>nn</code> , it'll use <code>nnn</code> , and so on, adding <code>ns</code> until it gets a new name.

.drop Handling of factor levels that don't appear in the data, passed on to `group_by()`.

For `count()`: if `FALSE` will include counts for empty groups (i.e. for levels of factors that don't exist in the data).

[Defunct] For `add_count()`: defunct since it can't actually affect the output.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::add_count()`

See Also

`dplyr::add_count()`

<code>add_tally</code>	<i>Wrapper around <code>dplyr::add_tally</code> that prints information about the operation</i>
------------------------	---

Description

Wrapper around `dplyr::add_tally()` that prints information about the operation.

Usage

```
add_tally(x, ...)
```

Arguments

- | | |
|------------------|---|
| <code>x</code> | A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). |
| <code>...</code> | Arguments passed on to <code>dplyr::add_tally</code> |
- `wt` *<data-masking>* Frequency weights. Can be `NULL` or a variable:
- If `NULL` (the default), counts the number of rows in each group.
 - If a variable, computes `sum(wt)` for each group.
- `sort` If `TRUE`, will show the largest groups at the top.
- `name` The name of the new column in the output.
 If omitted, it will default to `n`. If there's already a column called `n`, it will use `nn`. If there's a column called `n` and `nn`, it'll use `nnn`, and so on, adding `ns` until it gets a new name.

.drop Handling of factor levels that don't appear in the data, passed on to `group_by()`.

For `count()`: if `FALSE` will include counts for empty groups (i.e. for levels of factors that don't exist in the data).

[Defunct] For `add_count()`: defunct since it can't actually affect the output.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::add_tally()`

See Also

`dplyr::add_tally()`

anti_join	<i>Wrapper around <code>dplyr::anti_join</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::anti_join()` that prints information about the operation.

Usage

```
anti_join(x, y, by = NULL, ...)
```

Arguments

- | | |
|------|---|
| x, y | A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details. |
| by | <p>A join specification created with <code>join_by()</code>, or a character vector of variables to join by.</p> <p>If <code>NULL</code>, the default, <code>*_join()</code> will perform a natural join, using all variables in common across <code>x</code> and <code>y</code>. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly.</p> <p>To join on different variables between <code>x</code> and <code>y</code>, use a <code>join_by()</code> specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>.</p> <p>To join by multiple variables, use a <code>join_by()</code> specification with multiple expressions. For example, <code>join_by(a == b, c == d)</code> will match <code>x\$a</code> to <code>y\$b</code> and <code>x\$c</code> to <code>y\$d</code>. If the column names are the same between <code>x</code> and <code>y</code>, you can shorten this by listing only the variable names, like <code>join_by(a, c)</code>.</p> |

`join_by()` can also be used to perform inequality, rolling, and overlap joins. See the documentation at [?join_by](#) for details on these types of joins.

For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, `by = c("a", "b")` joins `x$a` to `y$a` and `x$b` to `y$b`. If variable names differ between `x` and `y`, use a named character vector like `by = c("x_a" = "y_a", "x_b" = "y_b")`.

To perform a cross-join, generating all combinations of `x` and `y`, see `cross_join()`.

... Arguments passed on to `dplyr::anti_join`

`x, y` A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.

`copy` If `x` and `y` are not from the same data source, and `copy` is `TRUE`, then `y` will be copied into the same `src` as `x`. This allows you to join tables across `srcs`, but it is a potentially expensive operation so you must opt into it.

`na_matches` Should two NA or two NaN values match?

- "na", the 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. This is similar to joins for database sources and to `base::merge(incomparables = NA)`.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::anti_join()`

See Also

`dplyr::anti_join()`

count	<i>Wrapper around <code>dplyr::count</code> that prints information about the operation</i>
-------	---

Description

Wrapper around `dplyr::count()` that prints information about the operation.

Usage

`count(x, ...)`

Arguments

- `x` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`).
- `...` Arguments passed on to `dplyr::count`
- `wt` `<data-masking>` Frequency weights. Can be `NULL` or a variable:
- If `NULL` (the default), counts the number of rows in each group.
 - If a variable, computes `sum(wt)` for each group.
- `sort` If `TRUE`, will show the largest groups at the top.
- `name` The name of the new column in the output.
If omitted, it will default to `n`. If there's already a column called `n`, it will use `nn`. If there's a column called `n` and `nn`, it'll use `nnn`, and so on, adding `ns` until it gets a new name.
- `.drop` Handling of factor levels that don't appear in the data, passed on to `group_by()`.
For `count()`: if `FALSE` will include counts for empty groups (i.e. for levels of factors that don't exist in the data).
[Defunct] For `add_count()`: defunct since it can't actually affect the output.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::count()`

See Also

`dplyr::count()`

`display_changed_rows` *Displays messages related to changing row number.*

Description

Unlike `log_...()` functions, this assumes the data manipulation is performed elsewhere and only displays messages.

Usage

```
display_changed_rows(.olddata, .newdata, .funname)
```

Arguments

<code>.olddata</code>	Data frame before transformation.
<code>.newdata</code>	Data frame after transformation.
<code>.funname</code>	String: name of function that should be used in messages.

<code>distinct</code>	<i>Wrapper around <code>dplyr::distinct</code> that prints information about the operation</i>
-----------------------	--

Description

Wrapper around `dplyr::distinct()` that prints information about the operation.

Usage

```
distinct(.data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	Arguments passed on to <code>dplyr::distinct</code> <code>.keep_all</code> If TRUE, keep all variables in <code>.data</code> . If a combination of <code>...</code> is not distinct, this keeps the first row of values.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::distinct()`

See Also

`dplyr::distinct()`

distinct_all	<i>Wrapper around <code>dplyr::distinct_all</code> that prints information about the operation</i>
--------------	--

Description

Wrapper around `dplyr::distinct_all()` that prints information about the operation.

Usage

```
distinct_all(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::distinct_all</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.keep_all</code>	If TRUE, keep all variables in <code>.data</code> . If a combination of <code>...</code> is not distinct, this keeps the first row of values.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::distinct_all()`

See Also

`dplyr::distinct_all()`

distinct_at	<i>Wrapper around <code>dplyr::distinct_at</code> that prints information about the operation</i>
-------------	---

Description

Wrapper around `dplyr::distinct_at()` that prints information about the operation.

Usage

```
distinct_at(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to <code>dplyr::distinct_at</code>
<code>.funs</code>	A function <code>fun</code> , a quosure style <code>lambda ~ fun(.)</code> or a list of either form.
<code>.keep_all</code>	If <code>TRUE</code> , keep all variables in <code>.data</code> . If a combination of <code>...</code> is not distinct, this keeps the first row of values.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or <code>NULL</code> .
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns <code>TRUE</code> are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::distinct_at()`

See Also

`dplyr::distinct_at()`

distinct_if	<i>Wrapper around <code>dplyr::distinct_if</code> that prints information about the operation</i>
-------------	---

Description

Wrapper around `dplyr::distinct_if()` that prints information about the operation.

Usage

```
distinct_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to <code>dplyr::distinct_if</code>
<code>.funs</code>	A function <code>fun</code> , a quosure style <code>lambda ~ fun(.)</code> or a list of either form.
<code>.keep_all</code>	If <code>TRUE</code> , keep all variables in <code>.data</code> . If a combination of <code>...</code> is not distinct, this keeps the first row of values.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or <code>NULL</code> .
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns <code>TRUE</code> are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::distinct_if()`

See Also

`dplyr::distinct_if()`

drop_na	<i>Wrapper around <code>tidyr::drop_na</code> that prints information about the operation</i>
---------	---

Description

Wrapper around `tidyr::drop_na()` that prints information about the operation.

Usage

```
drop_na(data, ...)
```

Arguments

data	A data frame.
...	<code><tidy-select></code> Columns to inspect for missing values. If empty, all columns are used.

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::drop_na()`

See Also

`tidyr::drop_na()`

fill	<i>Wrapper around <code>tidyr::fill</code> that prints information about the operation</i>
------	--

Description

Wrapper around `tidyr::fill()` that prints information about the operation.

Usage

```
fill(data, ...)
```

Arguments

- `data` A data frame.
- `...` Arguments passed on to `tidyr::fill`
- `.by` **[Experimental]** `<tidy-select>` Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see `?dplyr_by`.
- `.direction` Direction in which to fill missing values. Currently either "down" (the default), "up", "downup" (i.e. first down and then up) or "updown" (first up and then down).

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::fill()`

See Also

`tidyr::fill()`

<code>filter</code>	<i>Wrapper around <code>dplyr::filter</code> that prints information about the operation</i>
---------------------	--

Description

Wrapper around `dplyr::filter()` that prints information about the operation.

Usage

```
filter(.data, ...)
```

Arguments

- `.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.
- `...` Arguments passed on to `dplyr::filter`
- `.by` `<tidy-select>` Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see `?dplyr_by`.
- `.preserve` Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::filter()`

See Also

`dplyr::filter()`

filter_all	<i>Wrapper around <code>dplyr::filter_all</code> that prints information about the operation</i>
------------	--

Description

Wrapper around `dplyr::filter_all()` that prints information about the operation.

Usage

```
filter_all(.tbl, ...)
```

Arguments

- `.tbl` A tbl object.
- `...` Arguments passed on to `dplyr::filter_all`
- `.vars_predicate` A quoted predicate expression as returned by `all_vars()` or `any_vars()`.
Can also be a function or purrr-like formula. In this case, the intersection of the results is taken by default and there's currently no way to request the union.
- `.preserve` when FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise it is kept as is.
- `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns TRUE are selected. This argument is passed to `rlang::as_function()` and thus supports quosure-style lambda functions and strings representing function names.
- `.vars` A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::filter_all\(\)](#)

See Also

[dplyr::filter_all\(\)](#)

filter_at	<i>Wrapper around dplyr::filter_at that prints information about the operation</i>
-----------	--

Description

Wrapper around [dplyr::filter_at\(\)](#) that prints information about the operation.

Usage

```
filter_at(.tbl, ...)
```

Arguments

- `.tbl` A `tbl` object.
- `...` Arguments passed on to [dplyr::filter_at](#)
- `.vars_predicate` A quoted predicate expression as returned by [all_vars\(\)](#) or [any_vars\(\)](#).
Can also be a function or purrr-like formula. In this case, the intersection of the results is taken by default and there's currently no way to request the union.
- `.preserve` when FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise it is kept as is.
- `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns TRUE are selected. This argument is passed to [rlang::as_function\(\)](#) and thus supports quosure-style lambda functions and strings representing function names.
- `.vars` A list of columns generated by [vars\(\)](#), a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::filter_at\(\)](#)

See Also

[dplyr::filter_at\(\)](#)

filter_if	<i>Wrapper around <code>dplyr::filter_if</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::filter_if()` that prints information about the operation.

Usage

```
filter_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::filter_if</code>
<code>.vars_predicate</code>	A quoted predicate expression as returned by <code>all_vars()</code> or <code>any_vars()</code> . Can also be a function or purrr-like formula. In this case, the intersection of the results is taken by default and there's currently no way to request the union.
<code>.preserve</code>	when FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise it is kept as is.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::filter_if()`

See Also

`dplyr::filter_if()`

filter_out	<i>Wrapper around <code>dplyr::filter_out</code> that prints information about the operation</i>
------------	--

Description

Wrapper around `dplyr::filter_out()` that prints information about the operation.

Usage

```
filter_out(.data, ...)
```

Arguments

- | | |
|------------------------|--|
| <code>.data</code> | A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details. |
| <code>...</code> | Arguments passed on to <code>dplyr::filter_out</code> |
| <code>.by</code> | <code><tidy-select></code> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to <code>group_by()</code> . For details and examples, see <code>?dplyr_by</code> . |
| <code>.preserve</code> | Relevant when the <code>.data</code> input is grouped. If <code>.preserve = FALSE</code> (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is. |

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::filter_out()`

See Also

`dplyr::filter_out()`

full_join	<i>Wrapper around <code>dplyr::full_join</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::full_join()` that prints information about the operation.

Usage

```
full_join(x, y, by = NULL, ...)
```

Arguments

x, y	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
by	A join specification created with <code>join_by()</code>, or a character vector of variables to join by. If <code>NULL</code>, the default, <code>*_join()</code> will perform a natural join, using all variables in common across <code>x</code> and <code>y</code>. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly. To join on different variables between <code>x</code> and <code>y</code>, use a <code>join_by()</code> specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>. To join by multiple variables, use a <code>join_by()</code> specification with multiple expressions. For example, <code>join_by(a == b, c == d)</code> will match <code>x\$a</code> to <code>y\$b</code> and <code>x\$c</code> to <code>y\$d</code>. If the column names are the same between <code>x</code> and <code>y</code>, you can shorten this by listing only the variable names, like <code>join_by(a, c)</code>. <code>join_by()</code> can also be used to perform inequality, rolling, and overlap joins. See the documentation at <code>?join_by</code> for details on these types of joins. For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, <code>by = c("a", "b")</code> joins <code>x\$a</code> to <code>y\$a</code> and <code>x\$b</code> to <code>y\$b</code>. If variable names differ between <code>x</code> and <code>y</code>, use a named character vector like <code>by = c("x_a" = "y_a", "x_b" = "y_b")</code>. To perform a cross-join, generating all combinations of <code>x</code> and <code>y</code>, see <code>cross_join()</code>.
...	Arguments passed on to <code>dplyr::full_join</code>
x, y	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
copy	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same src as <code>x</code> . This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
suffix	If there are non-joined duplicate variables in <code>x</code> and <code>y</code> , these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
keep	Should the join keys from both <code>x</code> and <code>y</code> be preserved in the output?

- If NULL, the default, joins on equality retain only the keys from x, while joins on inequality retain the keys from both inputs.
- If TRUE, all keys from both inputs are retained.
- If FALSE, only keys from x are retained. For right and full joins, the data in key columns corresponding to rows that only exist in y are merged into the key columns from x. Can't be used when joining on inequality conditions.

na_matches Should two NA or two NaN values match?

- "na", the 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. This is similar to joins for database sources and to `base::merge(incomparables = NA)`.

multiple Handling of rows in x with multiple matches in y. For each row of x:

- "all", the default, returns every match detected in y. This is the same behavior as SQL.
- "any" returns one match detected in y, with no guarantees on which match will be returned. It is often faster than "first" and "last" if you just need to detect if there is at least one match.
- "first" returns the first match detected in y.
- "last" returns the last match detected in y.

unmatched How should unmatched keys that would result in dropped rows be handled?

- "drop" drops unmatched keys from the result.
- "error" throws an error if unmatched keys are detected.

unmatched is intended to protect you from accidentally dropping rows during a join. It only checks for unmatched keys in the input that could potentially drop rows.

- For left joins, it checks y.
- For right joins, it checks x.
- For inner joins, it checks both x and y. In this case, unmatched is also allowed to be a character vector of length 2 to specify the behavior for x and y independently.

relationship Handling of the expected relationship between the keys of x and y. If the expectations chosen from the list below are invalidated, an error is thrown.

- NULL, the default, doesn't expect there to be any relationship between x and y. However, for equality joins it will check for a many-to-many relationship (which is typically unexpected) and will warn if one occurs, encouraging you to either take a closer look at your inputs or make this relationship explicit by specifying "many-to-many". See the *Many-to-many relationships* section for more details.
- "one-to-one" expects:
 - Each row in x matches at most 1 row in y.
 - Each row in y matches at most 1 row in x.

- "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" doesn't perform any relationship checks, but is provided to allow you to be explicit about this relationship if you know it exists.

relationship doesn't handle cases where there are zero matches. For that, see `unmatched`.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::full_join()`

See Also

`dplyr::full_join()`

gather	<i>Wrapper around <code>tidyr::gather</code> that prints information about the operation</i>
--------	--

Description

Wrapper around `tidyr::gather()` that prints information about the operation.

Usage

```
gather(data, ...)
```

Arguments

data	A data frame.
...	Arguments passed on to <code>tidyr::gather</code>

key, value Names of new key and value columns, as strings or symbols.
 This argument is passed by expression and supports [quasiquotation](#) (you can unquote strings and symbols). The name is captured from the expression with `rlang::ensym()` (note that this kind of interface where symbols do not represent actual objects is now discouraged in the tidyverse; we support it here for backward compatibility).

na.rm If TRUE, will remove rows from output where the value column is NA.

`convert` If TRUE will automatically run `type.convert()` on the key column. This is useful if the column types are actually numeric, integer, or logical.

`factor_key` If FALSE, the default, the key values will be stored as a character vector. If TRUE, will be stored as a factor, which preserves the original ordering of the columns.

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::gather()`

See Also

`tidyr::gather()`

group_by	<i>Wrapper around <code>dplyr::group_by</code> that prints information about the operation</i>
----------	--

Description

Wrapper around `dplyr::group_by()` that prints information about the operation.

Usage

```
group_by(.data, ...)
```

Arguments

`.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.

`...` Arguments passed on to `dplyr::group_by`

`.add` When FALSE, the default, `group_by()` will override existing groups. To add to the existing groups, use `.add = TRUE`.

`.drop` Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when `.data` has been previously grouped with `.drop = FALSE`. See `group_by_drop_default()` for details.

x A `tbl()`

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::group_by\(\)](#)

See Also

[dplyr::group_by\(\)](#)

group_by_all	<i>Wrapper around dplyr::group_by_all that prints information about the operation</i>
--------------	---

Description

Wrapper around [dplyr::group_by_all\(\)](#) that prints information about the operation.

Usage

```
group_by_all(.tbl, ...)
```

Arguments

.tbl	A tbl object.
...	Arguments passed on to dplyr::group_by_all
.funs	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
.add	See group_by()
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when .data has been previously grouped with .drop = FALSE. See group_by_drop_default() for details.
.vars	A list of columns generated by vars() , a character vector of column names, a numeric vector of column positions, or NULL.
.predicate	A predicate function to be applied to the columns or a logical vector. The variables for which .predicate is or returns TRUE are selected. This argument is passed to rlang::as_function() and thus supports quosure-style lambda functions and strings representing function names.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::group_by_all\(\)](#)

See Also

[dplyr::group_by_all\(\)](#)

group_by_at	<i>Wrapper around <code>dplyr::group_by_at</code> that prints information about the operation</i>
-------------	---

Description

Wrapper around `dplyr::group_by_at()` that prints information about the operation.

Usage

```
group_by_at(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to <code>dplyr::group_by_at</code>
<code>.funs</code>	A function <code>fun</code> , a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.add</code>	See <code>group_by()</code>
<code>.drop</code>	Drop groups formed by factor levels that don't appear in the data? The default is <code>TRUE</code> except when <code>.data</code> has been previously grouped with <code>.drop = FALSE</code> . See <code>group_by_drop_default()</code> for details.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or <code>NULL</code> .
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns <code>TRUE</code> are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::group_by_at()`

See Also

`dplyr::group_by_at()`

group_by_if	<i>Wrapper around <code>dplyr::group_by_if</code> that prints information about the operation</i>
-------------	---

Description

Wrapper around `dplyr::group_by_if()` that prints information about the operation.

Usage

```
group_by_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to <code>dplyr::group_by_if</code>
<code>.funs</code>	A function <code>fun</code> , a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.add</code>	See <code>group_by()</code>
<code>.drop</code>	Drop groups formed by factor levels that don't appear in the data? The default is <code>TRUE</code> except when <code>.data</code> has been previously grouped with <code>.drop = FALSE</code> . See <code>group_by_drop_default()</code> for details.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or <code>NULL</code> .
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns <code>TRUE</code> are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::group_by_if()`

See Also

`dplyr::group_by_if()`

inner_join	<i>Wrapper around <code>dplyr::inner_join</code> that prints information about the operation</i>
------------	--

Description

Wrapper around `dplyr::inner_join()` that prints information about the operation.

Usage

```
inner_join(x, y, by = NULL, ...)
```

Arguments

<code>x, y</code>	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>by</code>	A join specification created with <code>join_by()</code>, or a character vector of variables to join by. If <code>NULL</code>, the default, <code>*_join()</code> will perform a natural join, using all variables in common across <code>x</code> and <code>y</code>. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly. To join on different variables between <code>x</code> and <code>y</code>, use a <code>join_by()</code> specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>. To join by multiple variables, use a <code>join_by()</code> specification with multiple expressions. For example, <code>join_by(a == b, c == d)</code> will match <code>x\$a</code> to <code>y\$b</code> and <code>x\$c</code> to <code>y\$d</code>. If the column names are the same between <code>x</code> and <code>y</code>, you can shorten this by listing only the variable names, like <code>join_by(a, c)</code>. <code>join_by()</code> can also be used to perform inequality, rolling, and overlap joins. See the documentation at <code>?join_by</code> for details on these types of joins. For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, <code>by = c("a", "b")</code> joins <code>x\$a</code> to <code>y\$a</code> and <code>x\$b</code> to <code>y\$b</code>. If variable names differ between <code>x</code> and <code>y</code>, use a named character vector like <code>by = c("x_a" = "y_a", "x_b" = "y_b")</code>. To perform a cross-join, generating all combinations of <code>x</code> and <code>y</code>, see <code>cross_join()</code>.
<code>...</code>	Arguments passed on to <code>dplyr::inner_join</code>
<code>x, y</code>	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>copy</code>	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same src as <code>x</code> . This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
<code>suffix</code>	If there are non-joined duplicate variables in <code>x</code> and <code>y</code> , these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
<code>keep</code>	Should the join keys from both <code>x</code> and <code>y</code> be preserved in the output?

- If NULL, the default, joins on equality retain only the keys from x, while joins on inequality retain the keys from both inputs.
- If TRUE, all keys from both inputs are retained.
- If FALSE, only keys from x are retained. For right and full joins, the data in key columns corresponding to rows that only exist in y are merged into the key columns from x. Can't be used when joining on inequality conditions.

`na_matches` Should two NA or two NaN values match?

- "na", the 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. This is similar to joins for database sources and to `base::merge(incomparables = NA)`.

`multiple` Handling of rows in x with multiple matches in y. For each row of x:

- "all", the default, returns every match detected in y. This is the same behavior as SQL.
- "any" returns one match detected in y, with no guarantees on which match will be returned. It is often faster than "first" and "last" if you just need to detect if there is at least one match.
- "first" returns the first match detected in y.
- "last" returns the last match detected in y.

`unmatched` How should unmatched keys that would result in dropped rows be handled?

- "drop" drops unmatched keys from the result.
- "error" throws an error if unmatched keys are detected.

`unmatched` is intended to protect you from accidentally dropping rows during a join. It only checks for unmatched keys in the input that could potentially drop rows.

- For left joins, it checks y.
- For right joins, it checks x.
- For inner joins, it checks both x and y. In this case, `unmatched` is also allowed to be a character vector of length 2 to specify the behavior for x and y independently.

`relationship` Handling of the expected relationship between the keys of x and y. If the expectations chosen from the list below are invalidated, an error is thrown.

- NULL, the default, doesn't expect there to be any relationship between x and y. However, for equality joins it will check for a many-to-many relationship (which is typically unexpected) and will warn if one occurs, encouraging you to either take a closer look at your inputs or make this relationship explicit by specifying "many-to-many". See the *Many-to-many relationships* section for more details.
- "one-to-one" expects:
 - Each row in x matches at most 1 row in y.
 - Each row in y matches at most 1 row in x.

- "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" doesn't perform any relationship checks, but is provided to allow you to be explicit about this relationship if you know it exists.

relationship doesn't handle cases where there are zero matches. For that, see `unmatched`.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::inner_join()`

See Also

`dplyr::inner_join()`

left_join	<i>Wrapper around <code>dplyr::left_join</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::left_join()` that prints information about the operation.

Usage

```
left_join(x, y, by = NULL, ...)
```

Arguments

- | | |
|------|---|
| x, y | A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details. |
| by | <p>A join specification created with <code>join_by()</code>, or a character vector of variables to join by.</p> <p>If <code>NULL</code>, the default, <code>*_join()</code> will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly.</p> <p>To join on different variables between x and y, use a <code>join_by()</code> specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>.</p> <p>To join by multiple variables, use a <code>join_by()</code> specification with multiple expressions. For example, <code>join_by(a == b, c == d)</code> will match <code>x\$a</code> to <code>y\$b</code> and</p> |

`x$c` to `y$d`. If the column names are the same between `x` and `y`, you can shorten this by listing only the variable names, like `join_by(a, c)`.

`join_by()` can also be used to perform inequality, rolling, and overlap joins. See the documentation at [?join_by](#) for details on these types of joins.

For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, `by = c("a", "b")` joins `x$a` to `y$a` and `x$b` to `y$b`. If variable names differ between `x` and `y`, use a named character vector like `by = c("x_a" = "y_a", "x_b" = "y_b")`.

To perform a cross-join, generating all combinations of `x` and `y`, see `cross_join()`.

...

Arguments passed on to `dplyr::left_join`

`x, y` A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.

`copy` If `x` and `y` are not from the same data source, and `copy` is `TRUE`, then `y` will be copied into the same `src` as `x`. This allows you to join tables across `srcs`, but it is a potentially expensive operation so you must opt into it.

`suffix` If there are non-joined duplicate variables in `x` and `y`, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.

`keep` Should the join keys from both `x` and `y` be preserved in the output?

- If `NULL`, the default, joins on equality retain only the keys from `x`, while joins on inequality retain the keys from both inputs.
- If `TRUE`, all keys from both inputs are retained.
- If `FALSE`, only keys from `x` are retained. For right and full joins, the data in key columns corresponding to rows that only exist in `y` are merged into the key columns from `x`. Can't be used when joining on inequality conditions.

`na_matches` Should two NA or two NaN values match?

- `"na"`, the 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. This is similar to joins for database sources and to `base::merge(incomparables = NA)`.

`multiple` Handling of rows in `x` with multiple matches in `y`. For each row of `x`:

- `"all"`, the default, returns every match detected in `y`. This is the same behavior as `SQL`.
- `"any"` returns one match detected in `y`, with no guarantees on which match will be returned. It is often faster than `"first"` and `"last"` if you just need to detect if there is at least one match.
- `"first"` returns the first match detected in `y`.
- `"last"` returns the last match detected in `y`.

`unmatched` How should unmatched keys that would result in dropped rows be handled?

- `"drop"` drops unmatched keys from the result.
- `"error"` throws an error if unmatched keys are detected.

unmatched is intended to protect you from accidentally dropping rows during a join. It only checks for unmatched keys in the input that could potentially drop rows.

- For left joins, it checks y.
- For right joins, it checks x.
- For inner joins, it checks both x and y. In this case, unmatched is also allowed to be a character vector of length 2 to specify the behavior for x and y independently.

relationship Handling of the expected relationship between the keys of x and y. If the expectations chosen from the list below are invalidated, an error is thrown.

- NULL, the default, doesn't expect there to be any relationship between x and y. However, for equality joins it will check for a many-to-many relationship (which is typically unexpected) and will warn if one occurs, encouraging you to either take a closer look at your inputs or make this relationship explicit by specifying "many-to-many". See the *Many-to-many relationships* section for more details.
- "one-to-one" expects:
 - Each row in x matches at most 1 row in y.
 - Each row in y matches at most 1 row in x.
- "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" doesn't perform any relationship checks, but is provided to allow you to be explicit about this relationship if you know it exists.

relationship doesn't handle cases where there are zero matches. For that, see unmatched.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::left_join()`

See Also

`dplyr::left_join()`

mutate	<i>Wrapper around <code>dplyr::mutate</code> that prints information about the operation</i>
--------	--

Description

Wrapper around `dplyr::mutate()` that prints information about the operation.

Usage

```
mutate(.data, ...)
```

Arguments

- | | |
|--|---|
| <code>.data</code> | A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details. |
| <code>...</code> | Arguments passed on to <code>dplyr::mutate</code> |
| <code>.by</code> | <code><tidy-select></code> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to <code>group_by()</code> . For details and examples, see <code>?dplyr_by</code> . |
| <code>.keep</code> | Control which columns from <code>.data</code> are retained in the output. Grouping columns and columns created by <code>...</code> are always kept. <ul style="list-style-type: none"> • "all" retains all columns from <code>.data</code>. This is the default. • "used" retains only the columns used in <code>...</code> to create new columns. This is useful for checking your work, as it displays inputs and outputs side-by-side. • "unused" retains only the columns <i>not</i> used in <code>...</code> to create new columns. This is useful if you generate new columns, but no longer need the columns used to generate them. • "none" doesn't retain any extra columns from <code>.data</code>. Only the grouping variables and columns created by <code>...</code> are kept. |
| <code>.before</code> , <code>.after</code> | <code><tidy-select></code> Optionally, control where new columns should appear (the default is to add to the right hand side). See <code>relocate()</code> for more details. |

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::mutate()`

See Also

`dplyr::mutate()`

mutate_all	<i>Wrapper around <code>dplyr::mutate_all</code> that prints information about the operation</i>
------------	--

Description

Wrapper around `dplyr::mutate_all()` that prints information about the operation.

Usage

```
mutate_all(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::mutate_all</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::mutate_all()`

See Also

`dplyr::mutate_all()`

mutate_at	<i>Wrapper around <code>dplyr::mutate_at</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::mutate_at()` that prints information about the operation.

Usage

```
mutate_at(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::mutate_at</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::mutate_at()`

See Also

`dplyr::mutate_at()`

mutate_if	<i>Wrapper around <code>dplyr::mutate_if</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::mutate_if()` that prints information about the operation.

Usage

```
mutate_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::mutate_if</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::mutate_if()`

See Also

`dplyr::mutate_if()`

pivot_longer	<i>Wrapper around tidyr::pivot_longer that prints information about the operation</i>
--------------	---

Description

Wrapper around `tidyr::pivot_longer()` that prints information about the operation.

Usage

```
pivot_longer(data, ...)
```

Arguments

data	A data frame to pivot.
...	Arguments passed on to <code>tidyr::pivot_longer</code>

cols `<tidy-select>` Columns to pivot into longer format.

cols_vary When pivoting cols into longer format, how should the output rows be arranged relative to their original row number?

- "fastest", the default, keeps individual rows from cols close together in the output. This often produces intuitively ordered output when you have at least one key column from data that is not involved in the pivoting process.
- "slowest" keeps individual columns from cols close together in the output. This often produces intuitively ordered output when you utilize all of the columns from data in the pivoting process.

names_to A character vector specifying the new column or columns to create from the information stored in the column names of data specified by cols.

- If length 0, or if NULL is supplied, no columns will be created.
- If length 1, a single column will be created which will contain the column names specified by cols.
- If length >1, multiple columns will be created. In this case, one of names_sep or names_pattern must be supplied to specify how the column names should be split. There are also two additional character values you can take advantage of:
 - NA will discard the corresponding component of the column name.
 - ".value" indicates that the corresponding component of the column name defines the name of the output column containing the cell values, overriding values_to entirely.

names_prefix A regular expression used to remove matching text from the start of each variable name.

names_sep, names_pattern If names_to contains multiple values, these arguments control how the column name is broken up.

`names_sep` takes the same specification as `separate()`, and can either be a numeric vector (specifying positions to break on), or a single string (specifying a regular expression to split on).

`names_pattern` takes the same specification as `extract()`, a regular expression containing matching groups (`()`).

If these arguments do not give you enough control, use `pivot_longer_spec()` to create a spec object and process manually as needed.

`names_ptypes, values_ptypes` Optionally, a list of column name-prototype pairs. Alternatively, a single empty prototype can be supplied, which will be applied to all columns. A prototype (or ptype for short) is a zero-length vector (like `integer()` or `numeric()`) that defines the type, class, and attributes of a vector. Use these arguments if you want to confirm that the created columns are the types that you expect. Note that if you want to change (instead of confirm) the types of specific columns, you should use `names_transform` or `values_transform` instead.

`names_transform, values_transform` Optionally, a list of column name-function pairs. Alternatively, a single function can be supplied, which will be applied to all columns. Use these arguments if you need to change the types of specific columns. For example, `names_transform = list(week = as.integer)` would convert a character variable called `week` to an integer.

If not specified, the type of the columns generated from `names_to` will be character, and the type of the variables generated from `values_to` will be the common type of the input columns used to generate them.

`names_repair` What happens if the output has invalid column names? The default, "check_unique" is to error if the columns are duplicated. Use "minimal" to allow duplicates in the output, or "unique" to de-duplicated by adding numeric suffixes. See `vctrs::vec_as_names()` for more options.

`values_to` A string specifying the name of the column to create from the data stored in cell values. If `names_to` is a character containing the special `.value` sentinel, this value will be ignored, and the name of the value column will be derived from part of the existing column names.

`values_drop_na` If TRUE, will drop rows that contain only NAs in the `values_to` column. This effectively converts explicit missing values to implicit missing values, and should generally be used only when missing values in data were created by its structure.

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::pivot_longer()`

See Also

`tidyr::pivot_longer()`

pivot_wider	<i>Wrapper around tidyr::pivot_wider that prints information about the operation</i>
-------------	--

Description

Wrapper around `tidyr::pivot_wider()` that prints information about the operation.

Usage

```
pivot_wider(data, ...)
```

Arguments

data	A data frame to pivot.
...	Arguments passed on to <code>tidyr::pivot_wider</code>

id_cols `<tidy-select>` A set of columns that uniquely identify each observation. Typically used when you have redundant variables, i.e. variables whose values are perfectly correlated with existing variables. Defaults to all columns in data except for the columns specified through `names_from` and `values_from`. If a tidyselect expression is supplied, it will be evaluated on data after removing the columns specified through `names_from` and `values_from`.

id_expand Should the values in the `id_cols` columns be expanded by `expand()` before pivoting? This results in more rows, the output will contain a complete expansion of all possible values in `id_cols`. Implicit factor levels that aren't represented in the data will become explicit. Additionally, the row values corresponding to the expanded `id_cols` will be sorted.

names_from, values_from `<tidy-select>` A pair of arguments describing which column (or columns) to get the name of the output column (`names_from`), and which column (or columns) to get the cell values from (`values_from`). If `values_from` contains multiple values, the value will be added to the front of the output column.

names_prefix String added to the start of every variable name. This is particularly useful if `names_from` is a numeric vector and you want to create syntactic variable names.

names_sep If `names_from` or `values_from` contains multiple variables, this will be used to join their values together into a single string to use as a column name.

names_glue Instead of `names_sep` and `names_prefix`, you can supply a glue specification that uses the `names_from` columns (and special `.value`) to create custom column names.

names_sort Should the column names be sorted? If `FALSE`, the default, column names are ordered by first appearance.

names_vary When `names_from` identifies a column (or columns) with multiple unique values, and multiple `values_from` columns are provided, in what order should the resulting column names be combined?

- "fastest" varies `names_from` values fastest, resulting in a column naming scheme of the form: `value1_name1`, `value1_name2`, `value2_name1`, `value2_name2`. This is the default.
- "slowest" varies `names_from` values slowest, resulting in a column naming scheme of the form: `value1_name1`, `value2_name1`, `value1_name2`, `value2_name2`.

names_expand Should the values in the `names_from` columns be expanded by `expand()` before pivoting? This results in more columns, the output will contain column names corresponding to a complete expansion of all possible values in `names_from`. Implicit factor levels that aren't represented in the data will become explicit. Additionally, the column names will be sorted, identical to what `names_sort` would produce.

names_repair What happens if the output has invalid column names? The default, "check_unique" is to error if the columns are duplicated. Use "minimal" to allow duplicates in the output, or "unique" to de-duplicated by adding numeric suffixes. See `vctrs::vec_as_names()` for more options.

values_fill Optionally, a (scalar) value that specifies what each value should be filled in with when missing.
This can be a named list if you want to apply different fill values to different value columns.

values_fn Optionally, a function applied to the value in each cell in the output. You will typically use this when the combination of `id_cols` and `names_from` columns does not uniquely identify an observation.
This can be a named list if you want to apply different aggregations to different `values_from` columns.

unused_fn Optionally, a function applied to summarize the values from the unused columns (i.e. columns not identified by `id_cols`, `names_from`, or `values_from`).

The default drops all unused columns from the result.

This can be a named list if you want to apply different aggregations to different unused columns.

`id_cols` must be supplied for `unused_fn` to be useful, since otherwise all unspecified columns will be considered `id_cols`.

This is similar to grouping by the `id_cols` then summarizing the unused columns using `unused_fn`.

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::pivot_wider()`

See Also

[tidyr::pivot_wider\(\)](#)

relocate	<i>Wrapper around dplyr::relocate that prints information about the operation</i>
----------	---

Description

Wrapper around [dplyr::relocate\(\)](#) that prints information about the operation.

Usage

```
relocate(.data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	Arguments passed on to dplyr::relocate <code>.before</code> , <code>.after</code> <tidy-select> Destination of columns selected by <code>...</code> . Supplying neither will move columns to the left-hand side; specifying both is an error.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See [dplyr::relocate\(\)](#)

See Also

[dplyr::relocate\(\)](#)

rename	<i>Wrapper around <code>dplyr::rename</code> that prints information about the operation</i>
--------	--

Description

Wrapper around `dplyr::rename()` that prints information about the operation.

Usage

```
rename(.data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	Arguments passed on to <code>dplyr::rename</code>
<code>.fn</code>	A function used to transform the selected <code>.cols</code> . Should return a character vector the same length as the input.
<code>.cols</code>	<code><tidy-select></code> Columns to rename; defaults to all columns.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::rename()`

See Also

`dplyr::rename()`

rename_all	<i>Wrapper around <code>dplyr::rename_all</code> that prints information about the operation</i>
------------	--

Description

Wrapper around `dplyr::rename_all()` that prints information about the operation.

Usage

```
rename_all(.tbl, ...)
```

Arguments

- `.tbl` A `tbl` object.
- `...` Arguments passed on to `dplyr::rename_all`
- `.funs` A function `fun`, a purrr style lambda `~ fun(.)` or a list of either form.
- `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns TRUE are selected. This argument is passed to `rlang::as_function()` and thus supports quosure-style lambda functions and strings representing function names.
- `.vars` A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::rename_all()`

See Also

`dplyr::rename_all()`

<code>rename_at</code>	<i>Wrapper around <code>dplyr::rename_at</code> that prints information about the operation</i>
------------------------	---

Description

Wrapper around `dplyr::rename_at()` that prints information about the operation.

Usage

```
rename_at(.tbl, ...)
```

Arguments

- `.tbl` A `tbl` object.
- `...` Arguments passed on to `dplyr::rename_at`
- `.funs` A function `fun`, a purrr style lambda `~ fun(.)` or a list of either form.
- `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns TRUE are selected. This argument is passed to `rlang::as_function()` and thus supports quosure-style lambda functions and strings representing function names.
- `.vars` A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::rename_at\(\)](#)

See Also

[dplyr::rename_at\(\)](#)

rename_if	<i>Wrapper around dplyr::rename_if that prints information about the operation</i>
-----------	--

Description

Wrapper around [dplyr::rename_if\(\)](#) that prints information about the operation.

Usage

```
rename_if(.tbl, ...)
```

Arguments

- | | |
|-------------------------|---|
| <code>.tbl</code> | A tbl object. |
| <code>...</code> | Arguments passed on to dplyr::rename_if |
| <code>.funs</code> | A function fun, a purrr style lambda <code>~ fun(.)</code> or a list of either form. |
| <code>.predicate</code> | A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to rlang::as_function() and thus supports quosure-style lambda functions and strings representing function names. |
| <code>.vars</code> | A list of columns generated by vars() , a character vector of column names, a numeric vector of column positions, or NULL. |

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::rename_if\(\)](#)

See Also

[dplyr::rename_if\(\)](#)

rename_with	<i>Wrapper around <code>dplyr::rename_with</code> that prints information about the operation</i>
-------------	---

Description

Wrapper around `dplyr::rename_with()` that prints information about the operation.

Usage

```
rename_with(.data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	Arguments passed on to <code>dplyr::rename_with</code>
<code>.fn</code>	A function used to transform the selected <code>.cols</code> . Should return a character vector the same length as the input.
<code>.cols</code>	<code><tidy-select></code> Columns to rename; defaults to all columns.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::rename_with()`

See Also

`dplyr::rename_with()`

replace_na	<i>Wrapper around <code>tidyr::replace_na</code> that prints information about the operation</i>
------------	--

Description

Wrapper around `tidyr::replace_na()` that prints information about the operation.

Usage

```
replace_na(data, ...)
```

Arguments

`data` A data frame or vector.

`...` Arguments passed on to `tidyr::replace_na`

`replace` If `data` is a data frame, `replace` takes a named list of values, with one value for each column that has missing values to be replaced. Each value in `replace` will be cast to the type of the column in `data` that it being used as a replacement in.

If `data` is a vector, `replace` takes a single value. This single value replaces all of the missing values in the vector. `replace` will be cast to the type of `data`.

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::replace_na()`

See Also

`tidyr::replace_na()`

<code>right_join</code>	<i>Wrapper around <code>dplyr::right_join</code> that prints information about the operation</i>
-------------------------	--

Description

Wrapper around `dplyr::right_join()` that prints information about the operation.

Usage

```
right_join(x, y, by = NULL, ...)
```

Arguments

`x, y` A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.

`by` A join specification created with `join_by()`, or a character vector of variables to join by.

If `NULL`, the default, `*_join()` will perform a natural join, using all variables in common across `x` and `y`. A message lists the variables so that you can check they're correct; suppress the message by supplying `by` explicitly.

To join on different variables between `x` and `y`, use a `join_by()` specification. For example, `join_by(a == b)` will match `x$a` to `y$b`.

To join by multiple variables, use a `join_by()` specification with multiple expressions. For example, `join_by(a == b, c == d)` will match `x$a` to `y$b` and `x$c` to `y$d`. If the column names are the same between `x` and `y`, you can shorten this by listing only the variable names, like `join_by(a, c)`.

`join_by()` can also be used to perform inequality, rolling, and overlap joins. See the documentation at [?join_by](#) for details on these types of joins.

For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, `by = c("a", "b")` joins `x$a` to `y$a` and `x$b` to `y$b`. If variable names differ between `x` and `y`, use a named character vector like `by = c("x_a" = "y_a", "x_b" = "y_b")`.

To perform a cross-join, generating all combinations of `x` and `y`, see `cross_join()`.

...

Arguments passed on to `dplyr::right_join`

x, y A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.

copy If `x` and `y` are not from the same data source, and `copy` is `TRUE`, then `y` will be copied into the same `src` as `x`. This allows you to join tables across `srcs`, but it is a potentially expensive operation so you must opt into it.

suffix If there are non-joined duplicate variables in `x` and `y`, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.

keep Should the join keys from both `x` and `y` be preserved in the output?

- If `NULL`, the default, joins on equality retain only the keys from `x`, while joins on inequality retain the keys from both inputs.
- If `TRUE`, all keys from both inputs are retained.
- If `FALSE`, only keys from `x` are retained. For right and full joins, the data in key columns corresponding to rows that only exist in `y` are merged into the key columns from `x`. Can't be used when joining on inequality conditions.

na_matches Should two NA or two NaN values match?

- `"na"`, the 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. This is similar to joins for database sources and to `base::merge(incomparables = NA)`.

multiple Handling of rows in `x` with multiple matches in `y`. For each row of `x`:

- `"all"`, the default, returns every match detected in `y`. This is the same behavior as `SQL`.
- `"any"` returns one match detected in `y`, with no guarantees on which match will be returned. It is often faster than `"first"` and `"last"` if you just need to detect if there is at least one match.
- `"first"` returns the first match detected in `y`.
- `"last"` returns the last match detected in `y`.

unmatched How should unmatched keys that would result in dropped rows be handled?

- `"drop"` drops unmatched keys from the result.

- "error" throws an error if unmatched keys are detected.

unmatched is intended to protect you from accidentally dropping rows during a join. It only checks for unmatched keys in the input that could potentially drop rows.

- For left joins, it checks y.
- For right joins, it checks x.
- For inner joins, it checks both x and y. In this case, unmatched is also allowed to be a character vector of length 2 to specify the behavior for x and y independently.

relationship Handling of the expected relationship between the keys of x and y. If the expectations chosen from the list below are invalidated, an error is thrown.

- NULL, the default, doesn't expect there to be any relationship between x and y. However, for equality joins it will check for a many-to-many relationship (which is typically unexpected) and will warn if one occurs, encouraging you to either take a closer look at your inputs or make this relationship explicit by specifying "many-to-many". See the *Many-to-many relationships* section for more details.
- "one-to-one" expects:
 - Each row in x matches at most 1 row in y.
 - Each row in y matches at most 1 row in x.
- "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" doesn't perform any relationship checks, but is provided to allow you to be explicit about this relationship if you know it exists.

relationship doesn't handle cases where there are zero matches. For that, see unmatched.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::right_join()`

See Also

`dplyr::right_join()`

sample_frac	<i>Wrapper around <code>dplyr::sample_frac</code> that prints information about the operation</i>
-------------	---

Description

Wrapper around `dplyr::sample_frac()` that prints information about the operation.

Usage

```
sample_frac(tbl, ...)
```

Arguments

tbl	A data.frame.
...	Arguments passed on to <code>dplyr::sample_frac</code>
size	<code><tidy-select></code> For <code>sample_n()</code> , the number of rows to select. For <code>sample_frac()</code> , the fraction of rows to select. If <code>tbl</code> is grouped, <code>size</code> applies to each group.
replace	Sample with or without replacement?
weight	<code><tidy-select></code> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
.env	DEPRECATED.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::sample_frac()`

See Also

`dplyr::sample_frac()`

sample_n	<i>Wrapper around <code>dplyr::sample_n</code> that prints information about the operation</i>
----------	--

Description

Wrapper around `dplyr::sample_n()` that prints information about the operation.

Usage

```
sample_n(tbl, ...)
```

Arguments

tbl	A data.frame.
...	Arguments passed on to <code>dplyr::sample_n</code>
size	<code><tidy-select></code> For <code>sample_n()</code> , the number of rows to select. For <code>sample_frac()</code> , the fraction of rows to select. If <code>tbl</code> is grouped, size applies to each group.
replace	Sample with or without replacement?
weight	<code><tidy-select></code> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
.env	DEPRECATED.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::sample_n()`

See Also

`dplyr::sample_n()`

select	<i>Wrapper around <code>dplyr::select</code> that prints information about the operation</i>
--------	--

Description

Wrapper around `dplyr::select()` that prints information about the operation.

Usage

```
select(.data, ...)
```

Arguments

.data	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
...	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like <code>x:y</code> can be used to select a range of variables.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::select()`

See Also

`dplyr::select()`

select_all	<i>Wrapper around <code>dplyr::select_all</code> that prints information about the operation</i>
------------	--

Description

Wrapper around `dplyr::select_all()` that prints information about the operation.

Usage

```
select_all(.tbl, ...)
```

Arguments

- `.tbl` A `tbl` object.
- `...` Arguments passed on to `dplyr::select_all`
- `.funs` A function `fun`, a purrr style lambda `~ fun(.)` or a list of either form.
- `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns TRUE are selected. This argument is passed to `rlang::as_function()` and thus supports quosure-style lambda functions and strings representing function names.
- `.vars` A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::select_all()`

See Also

`dplyr::select_all()`

select_at	<i>Wrapper around <code>dplyr::select_at</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::select_at()` that prints information about the operation.

Usage

```
select_at(.tbl, ...)
```

Arguments

- `.tbl` A `tbl` object.
- `...` Arguments passed on to `dplyr::select_at`
- `.funs` A function `fun`, a purrr style lambda `~ fun(.)` or a list of either form.
- `.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns TRUE are selected. This argument is passed to `rlang::as_function()` and thus supports quosure-style lambda functions and strings representing function names.
- `.vars` A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::select_at\(\)](#)

See Also

[dplyr::select_at\(\)](#)

select_if	<i>Wrapper around dplyr::select_if that prints information about the operation</i>
-----------	--

Description

Wrapper around [dplyr::select_if\(\)](#) that prints information about the operation.

Usage

```
select_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to dplyr::select_if
<code>.funs</code>	A function <code>fun</code> , a purrr style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to rlang::as_function() and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by vars() , a character vector of column names, a numeric vector of column positions, or NULL.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::select_if\(\)](#)

See Also

[dplyr::select_if\(\)](#)

semi_join	<i>Wrapper around <code>dplyr::semi_join</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::semi_join()` that prints information about the operation.

Usage

```
semi_join(x, y, by = NULL, ...)
```

Arguments

x, y	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
by	A join specification created with <code>join_by()</code>, or a character vector of variables to join by. If <code>NULL</code>, the default, <code>*_join()</code> will perform a natural join, using all variables in common across <code>x</code> and <code>y</code>. A message lists the variables so that you can check they're correct; suppress the message by supplying <code>by</code> explicitly. To join on different variables between <code>x</code> and <code>y</code>, use a <code>join_by()</code> specification. For example, <code>join_by(a == b)</code> will match <code>x\$a</code> to <code>y\$b</code>. To join by multiple variables, use a <code>join_by()</code> specification with multiple expressions. For example, <code>join_by(a == b, c == d)</code> will match <code>x\$a</code> to <code>y\$b</code> and <code>x\$c</code> to <code>y\$d</code>. If the column names are the same between <code>x</code> and <code>y</code>, you can shorten this by listing only the variable names, like <code>join_by(a, c)</code>. <code>join_by()</code> can also be used to perform inequality, rolling, and overlap joins. See the documentation at <code>?join_by</code> for details on these types of joins. For simple equality joins, you can alternatively specify a character vector of variable names to join by. For example, <code>by = c("a", "b")</code> joins <code>x\$a</code> to <code>y\$a</code> and <code>x\$b</code> to <code>y\$b</code>. If variable names differ between <code>x</code> and <code>y</code>, use a named character vector like <code>by = c("x_a" = "y_a", "x_b" = "y_b")</code>. To perform a cross-join, generating all combinations of <code>x</code> and <code>y</code>, see <code>cross_join()</code>.
...	Arguments passed on to <code>dplyr::semi_join</code>
x, y	A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
copy	If <code>x</code> and <code>y</code> are not from the same data source, and <code>copy</code> is <code>TRUE</code> , then <code>y</code> will be copied into the same <code>src</code> as <code>x</code> . This allows you to join tables across <code>srcs</code> , but it is a potentially expensive operation so you must opt into it.
na_matches	Should two NA or two NaN values match? "na", the default, treats two NA or two NaN values as equal, like <code>%in%</code>, <code>match()</code>, and <code>merge()</code>. "never" treats two NA or two NaN values as different, and will never match them together or to any other values. This is similar to joins for database sources and to <code>base::merge(incomparables = NA)</code>.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::semi_join()`

See Also

`dplyr::semi_join()`

separate_wider_delim	<i>Wrapper around <code>tidyr::separate_wider_delim</code> that prints information about the operation</i>
----------------------	--

Description

Wrapper around `tidyr::separate_wider_delim()` that prints information about the operation.

Usage

```
separate_wider_delim(data, ...)
```

Arguments

data	A data frame.
...	Arguments passed on to <code>tidyr::separate_wider_delim</code>
	<code>cols</code> < <code>tidy-select</code> > Columns to separate.
	<code>delim</code> For <code>separate_wider_delim()</code> , a string giving the delimiter between values. By default, it is interpreted as a fixed string; use <code>stringr::regex()</code> and friends to split in other ways.
	<code>names</code> For <code>separate_wider_delim()</code> , a character vector of output column names. Use NA if there are components that you don't want to appear in the output; the number of non-NA elements determines the number of new columns in the result.
	<code>names_sep</code> If supplied, output names will be composed of the input column name followed by the separator followed by the new column name. Required when <code>cols</code> selects multiple columns. For <code>separate_wider_delim()</code> you can specify instead of <code>names</code> , in which case the names will be generated from the source column name, <code>names_sep</code> , and a numeric suffix.
	<code>names_repair</code> Used to check that output data frame has valid names. Must be one of the following options: • "minimal": no name repair or checks, beyond basic existence, • "unique": make sure names are unique and not empty,

- "check_unique": (the default), no name repair, but check they are unique,
- "universal": make the names unique and syntactic
- a function: apply custom name repair.
- [tidyr_legacy](#): use the name repair from tidyr 0.8.
- a formula: a purrr-style anonymous function (see [rlang::as_function\(\)](#))

See [vctrs::vec_as_names\(\)](#) for more details on these terms and the strategies used to enforce them.

too_few What should happen if a value separates into too few pieces?

- "error", the default, will throw an error.
- "debug" adds additional columns to the output to help you locate and resolve the underlying problem. This option is intended to help you debug the issue and address and should not generally remain in your final code.
- "align_start" aligns starts of short matches, adding NA on the end to pad to the correct length.
- "align_end" ([separate_wider_delim\(\)](#) only) aligns the ends of short matches, adding NA at the start to pad to the correct length.

too_many What should happen if a value separates into too many pieces?

- "error", the default, will throw an error.
- "debug" will add additional columns to the output to help you locate and resolve the underlying problem.
- "drop" will silently drop any extra pieces.
- "merge" ([separate_wider_delim\(\)](#) only) will merge together any additional pieces.

cols_remove Should the input cols be removed from the output? Always FALSE if **too_few** or **too_many** are set to "debug".

widths A named numeric vector where the names become column names, and the values specify the column width. Unnamed components will match, but not be included in the output.

patterns A named character vector where the names become column names and the values are regular expressions that match the contents of the vector. Unnamed components will match, but not be included in the output.

Details

Documentation generated from tidyr version 1.3.2.

Value

See [tidyr::separate_wider_delim\(\)](#)

See Also

[tidyr::separate_wider_delim\(\)](#)

```
separate_wider_position
```

Wrapper around `tidyr::separate_wider_position` that prints information about the operation

Description

Wrapper around `tidyr::separate_wider_position()` that prints information about the operation.

Usage

```
separate_wider_position(data, ...)
```

Arguments

`data` A data frame.

`...` Arguments passed on to `tidyr::separate_wider_position`

`cols` `<tidy-select>` Columns to separate.

`delim` For `separate_wider_delim()`, a string giving the delimiter between values. By default, it is interpreted as a fixed string; use `stringr::regex()` and friends to split in other ways.

`names` For `separate_wider_delim()`, a character vector of output column names. Use NA if there are components that you don't want to appear in the output; the number of non-NA elements determines the number of new columns in the result.

`names_sep` If supplied, output names will be composed of the input column name followed by the separator followed by the new column name. Required when `cols` selects multiple columns.

For `separate_wider_delim()` you can specify instead of `names`, in which case the names will be generated from the source column name, `names_sep`, and a numeric suffix.

`names_repair` Used to check that output data frame has valid names. Must be one of the following options:

- "minimal": no name repair or checks, beyond basic existence,
- "unique": make sure names are unique and not empty,
- "check_unique": (the default), no name repair, but check they are unique,
- "universal": make the names unique and syntactic
- a function: apply custom name repair.
- `tidyr_legacy`: use the name repair from tidyr 0.8.
- a formula: a purrr-style anonymous function (see `rlang::as_function()`)

See `vctrs::vec_as_names()` for more details on these terms and the strategies used to enforce them.

- too_few** What should happen if a value separates into too few pieces?
- "error", the default, will throw an error.
 - "debug" adds additional columns to the output to help you locate and resolve the underlying problem. This option is intended to help you debug the issue and address and should not generally remain in your final code.
 - "align_start" aligns starts of short matches, adding NA on the end to pad to the correct length.
 - "align_end" (separate_wider_delim() only) aligns the ends of short matches, adding NA at the start to pad to the correct length.
- too_many** What should happen if a value separates into too many pieces?
- "error", the default, will throw an error.
 - "debug" will add additional columns to the output to help you locate and resolve the underlying problem.
 - "drop" will silently drop any extra pieces.
 - "merge" (separate_wider_delim() only) will merge together any additional pieces.
- cols_remove** Should the input cols be removed from the output? Always FALSE if too_few or too_many are set to "debug".
- widths** A named numeric vector where the names become column names, and the values specify the column width. Unnamed components will match, but not be included in the output.
- patterns** A named character vector where the names become column names and the values are regular expressions that match the contents of the vector. Unnamed components will match, but not be included in the output.

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::separate_wider_position()`

See Also

`tidyr::separate_wider_position()`

separate_wider_regex	<i>Wrapper around tidyr::separate_wider_regex that prints information about the operation</i>
----------------------	---

Description

Wrapper around `tidyr::separate_wider_regex()` that prints information about the operation.

Usage

```
separate_wider_regex(data, ...)
```

Arguments

- data** A data frame.
- ...** Arguments passed on to [tidyr::separate_wider_regex](#)
- cols** [<tidy-select>](#) Columns to separate.
- delim** For `separate_wider_delim()`, a string giving the delimiter between values. By default, it is interpreted as a fixed string; use [stringr::regex\(\)](#) and friends to split in other ways.
- names** For `separate_wider_delim()`, a character vector of output column names. Use NA if there are components that you don't want to appear in the output; the number of non-NA elements determines the number of new columns in the result.
- names_sep** If supplied, output names will be composed of the input column name followed by the separator followed by the new column name. Required when `cols` selects multiple columns. For `separate_wider_delim()` you can specify instead of `names`, in which case the names will be generated from the source column name, `names_sep`, and a numeric suffix.
- names_repair** Used to check that output data frame has valid names. Must be one of the following options:
- "minimal": no name repair or checks, beyond basic existence,
 - "unique": make sure names are unique and not empty,
 - "check_unique": (the default), no name repair, but check they are unique,
 - "universal": make the names unique and syntactic
 - a function: apply custom name repair.
 - [tidyr_legacy](#): use the name repair from tidyr 0.8.
 - a formula: a purrr-style anonymous function (see [rlang::as_function\(\)](#))
- See [vctrs::vec_as_names\(\)](#) for more details on these terms and the strategies used to enforce them.
- too_few** What should happen if a value separates into too few pieces?
- "error", the default, will throw an error.
 - "debug" adds additional columns to the output to help you locate and resolve the underlying problem. This option is intended to help you debug the issue and address and should not generally remain in your final code.
 - "align_start" aligns starts of short matches, adding NA on the end to pad to the correct length.
 - "align_end" (`separate_wider_delim()` only) aligns the ends of short matches, adding NA at the start to pad to the correct length.
- too_many** What should happen if a value separates into too many pieces?
- "error", the default, will throw an error.

- "debug" will add additional columns to the output to help you locate and resolve the underlying problem.
- "drop" will silently drop any extra pieces.
- "merge" (`separate_wider_delim()` only) will merge together any additional pieces.

`cols_remove` Should the input cols be removed from the output? Always FALSE if `too_few` or `too_many` are set to "debug".

`widths` A named numeric vector where the names become column names, and the values specify the column width. Unnamed components will match, but not be included in the output.

`patterns` A named character vector where the names become column names and the values are regular expressions that match the contents of the vector. Unnamed components will match, but not be included in the output.

Details

Documentation generated from tidyr version 1.3.2.

Value

See [tidyr::separate_wider_regex\(\)](#)

See Also

[tidyr::separate_wider_regex\(\)](#)

<code>slice</code>	<i>Wrapper around <code>dplyr::slice</code> that prints information about the operation</i>
--------------------	---

Description

Wrapper around [dplyr::slice\(\)](#) that prints information about the operation.

Usage

```
slice(.data, ...)
```

Arguments

- | | |
|------------------------------------|--|
| <code>.data</code> | A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details. |
| <code>...</code> | Arguments passed on to dplyr::slice |
| <code>.by</code> , <code>by</code> | <code><tidy-select></code> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by() . For details and examples, see ?dplyr_by . |

`.preserve` Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

`n,prop` Provide either `n`, the number of rows, or `prop`, the proportion of rows to select. If neither are supplied, `n = 1` will be used. If `n` is greater than the number of rows in the group (or `prop > 1`), the result will be silently truncated to the group size. `prop` will be rounded towards zero to generate an integer number of rows.

A negative value of `n` or `prop` will be subtracted from the group size. For example, `n = -2` with a group of 5 rows will select $5 - 2 = 3$ rows; `prop = -0.25` with 8 rows will select $8 * (1 - 0.25) = 6$ rows.

`order_by` [<data-masking>](#) Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.

`with_ties` Should ties be kept together? The default, `TRUE`, may return more rows than you request. Use `FALSE` to ignore ties, and return the first `n` rows.

`na_rm` Should missing values in `order_by` be removed from the result? If `FALSE`, NA values are sorted to the end (like in [arrange\(\)](#)), so they will only be included if there are insufficient non-missing values to reach `n/prop`.

`weight_by` [<data-masking>](#) Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1. See the Details section for more technical details regarding these weights.

`replace` Should sampling be performed with (`TRUE`) or without (`FALSE`, the default) replacement.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::slice\(\)](#)

See Also

[dplyr::slice\(\)](#)

slice_head

Wrapper around `dplyr::slice_head` that prints information about the operation

Description

Wrapper around [dplyr::slice_head\(\)](#) that prints information about the operation.

Usage

```
slice_head(.data, ...)
```

Arguments

- `.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.
- `...` Arguments passed on to `dplyr::slice_head`
- `.by, by <tidy-select>` Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see `?dplyr_by`.
- `.preserve` Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
- `n, prop` Provide either `n`, the number of rows, or `prop`, the proportion of rows to select. If neither are supplied, `n = 1` will be used. If `n` is greater than the number of rows in the group (or `prop > 1`), the result will be silently truncated to the group size. `prop` will be rounded towards zero to generate an integer number of rows.
A negative value of `n` or `prop` will be subtracted from the group size. For example, `n = -2` with a group of 5 rows will select $5 - 2 = 3$ rows; `prop = -0.25` with 8 rows will select $8 * (1 - 0.25) = 6$ rows.
- `order_by <data-masking>` Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.
- `with_ties` Should ties be kept together? The default, `TRUE`, may return more rows than you request. Use `FALSE` to ignore ties, and return the first `n` rows.
- `na_rm` Should missing values in `order_by` be removed from the result? If `FALSE`, NA values are sorted to the end (like in `arrange()`), so they will only be included if there are insufficient non-missing values to reach `n/prop`.
- `weight_by <data-masking>` Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1. See the *Details* section for more technical details regarding these weights.
- `replace` Should sampling be performed with (`TRUE`) or without (`FALSE`, the default) replacement.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::slice_head()`

See Also

`dplyr::slice_head()`

slice_max	<i>Wrapper around <code>dplyr::slice_max</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::slice_max()` that prints information about the operation.

Usage

```
slice_max(.data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	Arguments passed on to <code>dplyr::slice_max</code>
<code>.by, by</code>	<code><tidy-select></code> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to <code>group_by()</code> . For details and examples, see <code>?dplyr_by</code> .
<code>.preserve</code>	Relevant when the <code>.data</code> input is grouped. If <code>.preserve = FALSE</code> (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
<code>n, prop</code>	Provide either <code>n</code> , the number of rows, or <code>prop</code> , the proportion of rows to select. If neither are supplied, <code>n = 1</code> will be used. If <code>n</code> is greater than the number of rows in the group (or <code>prop > 1</code>), the result will be silently truncated to the group size. <code>prop</code> will be rounded towards zero to generate an integer number of rows. A negative value of <code>n</code> or <code>prop</code> will be subtracted from the group size. For example, <code>n = -2</code> with a group of 5 rows will select $5 - 2 = 3$ rows; <code>prop = -0.25</code> with 8 rows will select $8 * (1 - 0.25) = 6$ rows.
<code>order_by</code>	<code><data-masking></code> Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.
<code>with_ties</code>	Should ties be kept together? The default, <code>TRUE</code> , may return more rows than you request. Use <code>FALSE</code> to ignore ties, and return the first <code>n</code> rows.
<code>na_rm</code>	Should missing values in <code>order_by</code> be removed from the result? If <code>FALSE</code> , NA values are sorted to the end (like in <code>arrange()</code>), so they will only be included if there are insufficient non-missing values to reach <code>n/prop</code> .
<code>weight_by</code>	<code><data-masking></code> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1. See the Details section for more technical details regarding these weights.
<code>replace</code>	Should sampling be performed with (<code>TRUE</code>) or without (<code>FALSE</code> , the default) replacement.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::slice_max()`

See Also

`dplyr::slice_max()`

slice_min	<i>Wrapper around <code>dplyr::slice_min</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::slice_min()` that prints information about the operation.

Usage

```
slice_min(.data, ...)
```

Arguments

- `.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.
- `...` Arguments passed on to `dplyr::slice_min`
- `.by, by <tidy-select>` Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see `?dplyr_by`.
- `.preserve` Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
- `n, prop` Provide either `n`, the number of rows, or `prop`, the proportion of rows to select. If neither are supplied, `n = 1` will be used. If `n` is greater than the number of rows in the group (or `prop > 1`), the result will be silently truncated to the group size. `prop` will be rounded towards zero to generate an integer number of rows.
A negative value of `n` or `prop` will be subtracted from the group size. For example, `n = -2` with a group of 5 rows will select $5 - 2 = 3$ rows; `prop = -0.25` with 8 rows will select $8 * (1 - 0.25) = 6$ rows.
- `order_by <data-masking>` Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.
- `with_ties` Should ties be kept together? The default, `TRUE`, may return more rows than you request. Use `FALSE` to ignore ties, and return the first `n` rows.

`na_rm` Should missing values in `order_by` be removed from the result? If `FALSE`, NA values are sorted to the end (like in `arrange()`), so they will only be included if there are insufficient non-missing values to reach `n/prop`.

`weight_by` [<data-masking>](#) Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1. See the [Details](#) section for more technical details regarding these weights.

`replace` Should sampling be performed with (`TRUE`) or without (`FALSE`, the default) replacement.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::slice_min\(\)](#)

See Also

[dplyr::slice_min\(\)](#)

<code>slice_sample</code>	<i>Wrapper around <code>dplyr::slice_sample</code> that prints information about the operation</i>
---------------------------	--

Description

Wrapper around [dplyr::slice_sample\(\)](#) that prints information about the operation.

Usage

```
slice_sample(.data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	Arguments passed on to dplyr::slice_sample
<code>.by, by</code>	<tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by() . For details and examples, see ?dplyr_by .
<code>.preserve</code>	Relevant when the <code>.data</code> input is grouped. If <code>.preserve = FALSE</code> (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

`n, prop` Provide either `n`, the number of rows, or `prop`, the proportion of rows to select. If neither are supplied, `n = 1` will be used. If `n` is greater than the number of rows in the group (or `prop > 1`), the result will be silently truncated to the group size. `prop` will be rounded towards zero to generate an integer number of rows.

A negative value of `n` or `prop` will be subtracted from the group size. For example, `n = -2` with a group of 5 rows will select $5 - 2 = 3$ rows; `prop = -0.25` with 8 rows will select $8 * (1 - 0.25) = 6$ rows.

`order_by` [<data-masking>](#) Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.

`with_ties` Should ties be kept together? The default, `TRUE`, may return more rows than you request. Use `FALSE` to ignore ties, and return the first `n` rows.

`na_rm` Should missing values in `order_by` be removed from the result? If `FALSE`, NA values are sorted to the end (like in [arrange\(\)](#)), so they will only be included if there are insufficient non-missing values to reach `n/prop`.

`weight_by` [<data-masking>](#) Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1. See the Details section for more technical details regarding these weights.

`replace` Should sampling be performed with (`TRUE`) or without (`FALSE`, the default) replacement.

Details

Documentation generated from dplyr version 1.2.1.

Value

See [dplyr::slice_sample\(\)](#)

See Also

[dplyr::slice_sample\(\)](#)

slice_tail

Wrapper around [dplyr::slice_tail](#) that prints information about the operation

Description

Wrapper around [dplyr::slice_tail\(\)](#) that prints information about the operation.

Usage

```
slice_tail(.data, ...)
```

Arguments

- `.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.
- `...` Arguments passed on to `dplyr::slice_tail`
- `.by, by <tidy-select>` Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see `?dplyr_by`.
- `.preserve` Relevant when the `.data` input is grouped. If `.preserve = FALSE` (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
- `n, prop` Provide either `n`, the number of rows, or `prop`, the proportion of rows to select. If neither are supplied, `n = 1` will be used. If `n` is greater than the number of rows in the group (or `prop > 1`), the result will be silently truncated to the group size. `prop` will be rounded towards zero to generate an integer number of rows.
A negative value of `n` or `prop` will be subtracted from the group size. For example, `n = -2` with a group of 5 rows will select $5 - 2 = 3$ rows; `prop = -0.25` with 8 rows will select $8 * (1 - 0.25) = 6$ rows.
- `order_by <data-masking>` Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.
- `with_ties` Should ties be kept together? The default, `TRUE`, may return more rows than you request. Use `FALSE` to ignore ties, and return the first `n` rows.
- `na_rm` Should missing values in `order_by` be removed from the result? If `FALSE`, NA values are sorted to the end (like in `arrange()`), so they will only be included if there are insufficient non-missing values to reach `n/prop`.
- `weight_by <data-masking>` Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1. See the *Details* section for more technical details regarding these weights.
- `replace` Should sampling be performed with (`TRUE`) or without (`FALSE`, the default) replacement.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::slice_tail()`

See Also

`dplyr::slice_tail()`

spread	<i>Wrapper around <code>tidyr::spread</code> that prints information about the operation</i>
--------	--

Description

Wrapper around `tidyr::spread()` that prints information about the operation.

Usage

```
spread(data, ...)
```

Arguments

data	A data frame.
...	Arguments passed on to <code>tidyr::spread</code>
key, value	< tidy-select > Columns to use for key and value.
fill	If set, missing values will be replaced with this value. Note that there are two types of missingness in the input: explicit missing values (i.e. NA), and implicit missings, rows that simply aren't present. Both types of missing value will be replaced by fill.
convert	If TRUE, <code>type.convert()</code> with <code>asis = TRUE</code> will be run on each of the new columns. This is useful if the value column was a mix of variables that was coerced to a string. If the class of the value column was factor or date, note that will not be true of the new columns that are produced, which are coerced to character before type conversion.
drop	If FALSE, will keep factor levels that don't appear in the data, filling in missing combinations with fill.
sep	If NULL, the column names will be taken from the values of key variable. If non-NULL, the column names will be given by "<key_name><sep><key_value>".

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::spread()`

See Also

`tidyr::spread()`

summarise	<i>Wrapper around <code>dplyr::summarise</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::summarise()` that prints information about the operation.

Usage

```
summarise(.data, ...)
```

Arguments

- | | |
|----------------------|--|
| <code>.data</code> | A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details. |
| <code>...</code> | Arguments passed on to <code>dplyr::summarise</code> |
| <code>.by</code> | <code><tidy-select></code> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to <code>group_by()</code> . For details and examples, see <code>?dplyr_by</code> . |
| <code>.groups</code> | <p>[Experimental] Grouping structure of the result.</p> <ul style="list-style-type: none"> "drop_last": drops the last level of grouping. This was the only supported option before version 1.0.0. "drop": All levels of grouping are dropped. "keep": Same grouping structure as <code>.data</code>. "rowwise": Each row is its own group. |

When `.groups` is not specified, it is set to "drop_last" for a grouped data frame, and "keep" for a rowwise data frame. In addition, a message informs you of how the result will be grouped unless the result is ungrouped, the option `"dplyr.summarise.inform"` is set to FALSE, or when `summarise()` is called from a function in a package.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::summarise()`

See Also

`dplyr::summarise()`

summarise_all	<i>Wrapper around <code>dplyr::summarise_all</code> that prints information about the operation</i>
---------------	---

Description

Wrapper around `dplyr::summarise_all()` that prints information about the operation.

Usage

```
summarise_all(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::summarise_all</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::summarise_all()`

See Also

`dplyr::summarise_all()`

summarise_at	<i>Wrapper around <code>dplyr::summarise_at</code> that prints information about the operation</i>
--------------	--

Description

Wrapper around `dplyr::summarise_at()` that prints information about the operation.

Usage

```
summarise_at(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::summarise_at</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::summarise_at()`

See Also

`dplyr::summarise_at()`

summarise_if	<i>Wrapper around <code>dplyr::summarise_if</code> that prints information about the operation</i>
--------------	--

Description

Wrapper around `dplyr::summarise_if()` that prints information about the operation.

Usage

```
summarise_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::summarise_if</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::summarise_if()`

See Also

`dplyr::summarise_if()`

summarize	<i>Wrapper around <code>dplyr::summarize</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::summarize()` that prints information about the operation.

Usage

```
summarize(.data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details.
<code>...</code>	Arguments passed on to <code>dplyr::summarize</code>
<code>.by</code>	<code><tidy-select></code> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to <code>group_by()</code> . For details and examples, see <code>?dplyr_by</code> .
<code>.groups</code>	[Experimental] Grouping structure of the result. "drop_last": drops the last level of grouping. This was the only supported option before version 1.0.0. "drop": All levels of grouping are dropped. "keep": Same grouping structure as <code>.data</code>. "rowwise": Each row is its own group. When <code>.groups</code> is not specified, it is set to "drop_last" for a grouped data frame, and "keep" for a rowwise data frame. In addition, a message informs you of how the result will be grouped unless the result is ungrouped, the option <code>"dplyr.summarise.inform"</code> is set to FALSE, or when <code>summarise()</code> is called from a function in a package.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::summarize()`

See Also

`dplyr::summarize()`

summarize_all	<i>Wrapper around <code>dplyr::summarize_all</code> that prints information about the operation</i>
---------------	---

Description

Wrapper around `dplyr::summarize_all()` that prints information about the operation.

Usage

```
summarize_all(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to <code>dplyr::summarize_all</code>
<code>.funs</code>	A function <code>fun</code> , a quosure style <code>lambda ~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns <code>TRUE</code> are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or <code>NULL</code> .
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit <code>dplyr</code> 's terminology and is deprecated.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::summarize_all()`

See Also

`dplyr::summarize_all()`

summarize_at	<i>Wrapper around <code>dplyr::summarize_at</code> that prints information about the operation</i>
--------------	--

Description

Wrapper around `dplyr::summarize_at()` that prints information about the operation.

Usage

```
summarize_at(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to <code>dplyr::summarize_at</code>
<code>.funs</code>	A function <code>fun</code> , a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit <code>dplyr</code> 's terminology and is deprecated.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::summarize_at()`

See Also

`dplyr::summarize_at()`

summarize_if	<i>Wrapper around <code>dplyr::summarize_if</code> that prints information about the operation</i>
--------------	--

Description

Wrapper around `dplyr::summarize_if()` that prints information about the operation.

Usage

```
summarize_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A <code>tbl</code> object.
<code>...</code>	Arguments passed on to <code>dplyr::summarize_if</code>
<code>.funs</code>	A function <code>fun</code> , a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit <code>dplyr</code> 's terminology and is deprecated.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::summarize_if()`

See Also

`dplyr::summarize_if()`

tally	<i>Wrapper around <code>dplyr::tally</code> that prints information about the operation</i>
-------	---

Description

Wrapper around `dplyr::tally()` that prints information about the operation.

Usage

```
tally(x, ...)
```

Arguments

- x** A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`).
- ...** Arguments passed on to `dplyr::tally`
- wt** `<data-masking>` Frequency weights. Can be `NULL` or a variable:
- If `NULL` (the default), counts the number of rows in each group.
 - If a variable, computes `sum(wt)` for each group.
- sort** If `TRUE`, will show the largest groups at the top.
- name** The name of the new column in the output.
If omitted, it will default to `n`. If there's already a column called `n`, it will use `nn`. If there's a column called `n` and `nn`, it'll use `nnn`, and so on, adding `ns` until it gets a new name.
- .drop** Handling of factor levels that don't appear in the data, passed on to `group_by()`.
For `count()`: if `FALSE` will include counts for empty groups (i.e. for levels of factors that don't exist in the data).
[Defunct] For `add_count()`: defunct since it can't actually affect the output.

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::tally()`

See Also

`dplyr::tally()`

tidylog	<i>outputs some information about the data frame/tbl</i>
---------	--

Description

outputs some information about the data frame/tbl

Usage

```
tidylog(.data)
```

Arguments

`.data` a tbl/data frame

Value

same as `.data`

Examples

```
tidylog(mtcars)
#> tidylog: data.frame with 32 rows and 11 columns
```

top_frac	<i>Wrapper around <code>dplyr::top_frac</code> that prints information about the operation</i>
----------	--

Description

Wrapper around `dplyr::top_frac()` that prints information about the operation.

Usage

```
top_frac(x, ...)
```

Arguments

`x` A data frame.

`...` Arguments passed on to `dplyr::top_frac`

`n` Number of rows to return for `top_n()`, fraction of rows to return for `top_frac()`. If `n` is positive, selects the top rows. If negative, selects the bottom rows. If `x` is grouped, this is the number (or fraction) of rows per group. Will include more rows if there are ties.

`wt` (Optional). The variable to use for ordering. If not specified, defaults to the last variable in the tbl.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::top_frac()`

See Also

`dplyr::top_frac()`

top_n	<i>Wrapper around <code>dplyr::top_n</code> that prints information about the operation</i>
-------	---

Description

Wrapper around `dplyr::top_n()` that prints information about the operation.

Usage

`top_n(x, ...)`

Arguments

- x A data frame.
- ... Arguments passed on to `dplyr::top_n`
- n Number of rows to return for `top_n()`, fraction of rows to return for `top_frac()`. If n is positive, selects the top rows. If negative, selects the bottom rows. If x is grouped, this is the number (or fraction) of rows per group. Will include more rows if there are ties.
- wt (Optional). The variable to use for ordering. If not specified, defaults to the last variable in the tbl.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::top_n()`

See Also

`dplyr::top_n()`

transmute	<i>Wrapper around <code>dplyr::transmute</code> that prints information about the operation</i>
-----------	---

Description

Wrapper around `dplyr::transmute()` that prints information about the operation.

Usage

```
transmute(.data, ...)
```

Arguments

- | | |
|--------------------|--|
| <code>.data</code> | A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code>). See <i>Methods</i> , below, for more details. |
| <code>...</code> | <code><data-masking></code> Name-value pairs. The name gives the name of the column in the output.
The value can be: <ul style="list-style-type: none">• A vector of length 1, which will be recycled to the correct length.• A vector the same length as the current group (or the whole data frame if ungrouped).• <code>NULL</code>, to remove the column.• A data frame or tibble, to create multiple columns in the output. |

Details

Documentation generated from `dplyr` version 1.2.1.

Value

See `dplyr::transmute()`

See Also

`dplyr::transmute()`

transmute_all	<i>Wrapper around <code>dplyr::transmute_all</code> that prints information about the operation</i>
---------------	---

Description

Wrapper around `dplyr::transmute_all()` that prints information about the operation.

Usage

```
transmute_all(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::transmute_all</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::transmute_all()`

See Also

`dplyr::transmute_all()`

transmute_at	<i>Wrapper around <code>dplyr::transmute_at</code> that prints information about the operation</i>
--------------	--

Description

Wrapper around `dplyr::transmute_at()` that prints information about the operation.

Usage

```
transmute_at(.tbl, ...)
```

Arguments

- | | |
|-------------------------|--|
| <code>.tbl</code> | A tbl object. |
| <code>...</code> | Arguments passed on to <code>dplyr::transmute_at</code> |
| <code>.funs</code> | A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form. |
| <code>.predicate</code> | A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names. |
| <code>.vars</code> | A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL. |
| <code>.cols</code> | This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated. |

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::transmute_at()`

See Also

`dplyr::transmute_at()`

transmute_if	<i>Wrapper around <code>dplyr::transmute_if</code> that prints information about the operation</i>
--------------	--

Description

Wrapper around `dplyr::transmute_if()` that prints information about the operation.

Usage

```
transmute_if(.tbl, ...)
```

Arguments

<code>.tbl</code>	A tbl object.
<code>...</code>	Arguments passed on to <code>dplyr::transmute_if</code>
<code>.funs</code>	A function fun, a quosure style lambda <code>~ fun(.)</code> or a list of either form.
<code>.predicate</code>	A predicate function to be applied to the columns or a logical vector. The variables for which <code>.predicate</code> is or returns TRUE are selected. This argument is passed to <code>rlang::as_function()</code> and thus supports quosure-style lambda functions and strings representing function names.
<code>.vars</code>	A list of columns generated by <code>vars()</code> , a character vector of column names, a numeric vector of column positions, or NULL.
<code>.cols</code>	This argument has been renamed to <code>.vars</code> to fit dplyr's terminology and is deprecated.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::transmute_if()`

See Also

`dplyr::transmute_if()`

uncount	<i>Wrapper around <code>tidyr::uncount</code> that prints information about the operation</i>
---------	---

Description

Wrapper around `tidyr::uncount()` that prints information about the operation.

Usage

```
uncount(data, ...)
```

Arguments

<code>data</code>	A data frame, tibble, or grouped tibble.
<code>...</code>	Arguments passed on to <code>tidyr::uncount</code>
<code>weights</code>	A vector of weights. Evaluated in the context of data; supports quasiquotation.
<code>.remove</code>	If TRUE, and <code>weights</code> is the name of a column in data, then this column is removed.
<code>.id</code>	Supply a string to create a new variable which gives a unique identifier for each created row.

Details

Documentation generated from tidyr version 1.3.2.

Value

See `tidyr::uncount()`

See Also

`tidyr::uncount()`

ungroup	<i>Wrapper around <code>dplyr::ungroup</code> that prints information about the operation</i>
---------	---

Description

Wrapper around `dplyr::ungroup()` that prints information about the operation.

Usage

```
ungroup(x, ...)
```

Arguments

- `x` A `tbl()`
- `...` Arguments passed on to `dplyr::ungroup`
- `.data` A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from `dbplyr` or `dtplyr`). See *Methods*, below, for more details.
- `.add` When FALSE, the default, `group_by()` will override existing groups. To add to the existing groups, use `.add = TRUE`.
- `.drop` Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when `.data` has been previously grouped with `.drop = FALSE`. See `group_by_drop_default()` for details.

Details

Documentation generated from dplyr version 1.2.1.

Value

See `dplyr::ungroup()`

See Also

`dplyr::ungroup()`

Index

?dplyr_by, [13](#), [17](#), [30](#), [57](#), [59–62](#), [64](#), [66](#), [70](#)
?join_by, [6](#), [18](#), [25](#), [28](#), [44](#), [51](#)

add_count, [3](#)
add_tally, [4](#)
all_vars(), [14–16](#)
anti_join, [5](#)
any_vars(), [14–16](#)
arrange(), [58–60](#), [62–64](#)

count, [6](#)
cross_join(), [6](#), [18](#), [25](#), [28](#), [44](#), [51](#)

display_changed_rows, [7](#)
distinct, [8](#)
distinct_all, [9](#)
distinct_at, [10](#)
distinct_if, [11](#)
dplyr::add_count, [3](#)
dplyr::add_count(), [3](#), [4](#)
dplyr::add_tally, [4](#)
dplyr::add_tally(), [4](#), [5](#)
dplyr::anti_join, [6](#)
dplyr::anti_join(), [5](#), [6](#)
dplyr::count, [7](#)
dplyr::count(), [6](#), [7](#)
dplyr::distinct, [8](#)
dplyr::distinct(), [8](#)
dplyr::distinct_all, [9](#)
dplyr::distinct_all(), [9](#)
dplyr::distinct_at, [10](#)
dplyr::distinct_at(), [10](#)
dplyr::distinct_if, [11](#)
dplyr::distinct_if(), [11](#)
dplyr::filter, [13](#)
dplyr::filter(), [13](#), [14](#)
dplyr::filter_all, [14](#)
dplyr::filter_all(), [14](#), [15](#)
dplyr::filter_at, [15](#)
dplyr::filter_at(), [15](#)

dplyr::filter_if, [16](#)
dplyr::filter_if(), [16](#)
dplyr::filter_out, [17](#)
dplyr::filter_out(), [17](#)
dplyr::full_join, [18](#)
dplyr::full_join(), [18](#), [20](#)
dplyr::group_by, [21](#)
dplyr::group_by(), [21](#), [22](#)
dplyr::group_by_all, [22](#)
dplyr::group_by_all(), [22](#)
dplyr::group_by_at, [23](#)
dplyr::group_by_at(), [23](#)
dplyr::group_by_if, [24](#)
dplyr::group_by_if(), [24](#)
dplyr::inner_join, [25](#)
dplyr::inner_join(), [25](#), [27](#)
dplyr::left_join, [28](#)
dplyr::left_join(), [27](#), [29](#)
dplyr::mutate, [30](#)
dplyr::mutate(), [30](#)
dplyr::mutate_all, [31](#)
dplyr::mutate_all(), [31](#)
dplyr::mutate_at, [32](#)
dplyr::mutate_at(), [32](#)
dplyr::mutate_if, [33](#)
dplyr::mutate_if(), [33](#)
dplyr::relocate, [38](#)
dplyr::relocate(), [38](#)
dplyr::rename, [39](#)
dplyr::rename(), [39](#)
dplyr::rename_all, [40](#)
dplyr::rename_all(), [39](#), [40](#)
dplyr::rename_at, [40](#)
dplyr::rename_at(), [40](#), [41](#)
dplyr::rename_if, [41](#)
dplyr::rename_if(), [41](#)
dplyr::rename_with, [42](#)
dplyr::rename_with(), [42](#)
dplyr::right_join, [44](#)

dplyr::right_join(), 43, 45
 dplyr::sample_frac, 46
 dplyr::sample_frac(), 46
 dplyr::sample_n, 47
 dplyr::sample_n(), 47
 dplyr::select(), 48
 dplyr::select_all, 49
 dplyr::select_all(), 48, 49
 dplyr::select_at, 49
 dplyr::select_at(), 49, 50
 dplyr::select_if, 50
 dplyr::select_if(), 50
 dplyr::semi_join, 51
 dplyr::semi_join(), 51, 52
 dplyr::slice, 57
 dplyr::slice(), 57, 58
 dplyr::slice_head, 59
 dplyr::slice_head(), 58, 59
 dplyr::slice_max, 60
 dplyr::slice_max(), 60, 61
 dplyr::slice_min, 61
 dplyr::slice_min(), 61, 62
 dplyr::slice_sample, 62
 dplyr::slice_sample(), 62, 63
 dplyr::slice_tail, 64
 dplyr::slice_tail(), 63, 64
 dplyr::summarise, 66
 dplyr::summarise(), 66
 dplyr::summarise_all, 67
 dplyr::summarise_all(), 67
 dplyr::summarise_at, 68
 dplyr::summarise_at(), 68
 dplyr::summarise_if, 69
 dplyr::summarise_if(), 69
 dplyr::summarize, 70
 dplyr::summarize(), 70
 dplyr::summarize_all, 71
 dplyr::summarize_all(), 71
 dplyr::summarize_at, 72
 dplyr::summarize_at(), 72
 dplyr::summarize_if, 73
 dplyr::summarize_if(), 73
 dplyr::tally, 74
 dplyr::tally(), 74
 dplyr::top_frac, 75
 dplyr::top_frac(), 75, 76
 dplyr::top_n, 76
 dplyr::top_n(), 76
 dplyr::transmute(), 77
 dplyr::transmute_all, 78
 dplyr::transmute_all(), 78
 dplyr::transmute_at, 79
 dplyr::transmute_at(), 79
 dplyr::transmute_if, 80
 dplyr::transmute_if(), 80
 dplyr::ungroup, 82
 dplyr::ungroup(), 81, 82
 drop_na, 12
 expand(), 36, 37
 extract(), 35
 fill, 12
 filter, 13
 filter_all, 14
 filter_at, 15
 filter_if, 16
 filter_out, 17
 full_join, 18
 gather, 20
 group_by, 21
 group_by(), 4, 5, 7, 13, 17, 22–24, 30, 57, 59–62, 64, 66, 70, 74
 group_by_all, 22
 group_by_at, 23
 group_by_drop_default(), 21–24, 82
 group_by_if, 24
 inner_join, 25
 join_by(), 5, 6, 18, 25, 27, 28, 43, 44, 51
 left_join, 27
 match(), 6, 19, 26, 28, 44, 51
 merge(), 6, 19, 26, 28, 44, 51
 mutate, 30
 mutate_all, 31
 mutate_at, 32
 mutate_if, 33
 pivot_longer, 34
 pivot_wider, 36
 quasiquotation, 20
 relocate, 38

relocate(), 30
 rename, 39
 rename_all, 39
 rename_at, 40
 rename_if, 41
 rename_with, 42
 replace_na, 42
 right_join, 43
 rlang::as_function(), 9–11, 14–16, 22–24, 31–33, 40, 41, 49, 50, 53, 54, 56, 67–69, 71–73, 78–80
 rlang::ensym(), 20

 sample_frac, 46
 sample_n, 47
 select, 48
 select_all, 48
 select_at, 49
 select_if, 50
 semi_join, 51
 separate(), 35
 separate_wider_delim, 52
 separate_wider_position, 54
 separate_wider_regex, 55
 slice, 57
 slice_head, 58
 slice_max, 60
 slice_min, 61
 slice_sample, 62
 slice_tail, 63
 spread, 65
 stringr::regex(), 52, 54, 56
 summarise, 66
 summarise_all, 67
 summarise_at, 68
 summarise_if, 69
 summarize, 70
 summarize_all, 71
 summarize_at, 72
 summarize_if, 73

 tally, 74
 tbl(), 21, 82
 tidylog, 75
 tidyr::drop_na(), 12
 tidyr::fill, 13
 tidyr::fill(), 12, 13
 tidyr::gather, 20
 tidyr::gather(), 20, 21

 tidyr::pivot_longer, 34
 tidyr::pivot_longer(), 34, 35
 tidyr::pivot_wider, 36
 tidyr::pivot_wider(), 36–38
 tidyr::replace_na, 43
 tidyr::replace_na(), 42, 43
 tidyr::separate_wider_delim, 52
 tidyr::separate_wider_delim(), 52, 53
 tidyr::separate_wider_position, 54
 tidyr::separate_wider_position(), 54, 55
 tidyr::separate_wider_regex, 56
 tidyr::separate_wider_regex(), 55, 57
 tidyr::spread, 65
 tidyr::spread(), 65
 tidyr::uncount, 81
 tidyr::uncount(), 81
 tidyr_legacy, 53, 54, 56
 top_frac, 75
 top_n, 76
 transmute, 77
 transmute_all, 78
 transmute_at, 79
 transmute_if, 80
 type.convert(), 21, 65

 uncount, 81
 ungroup, 81

 vars(), 9–11, 14–16, 22–24, 31–33, 40, 41, 49, 50, 67–69, 71–73, 78–80
 vctrs::vec_as_names(), 35, 37, 53, 54, 56