Package ‘tidyplus’

July 18, 2026

Title Additional 'tidyverse' Functions

Version 0.2.1

Description Provides functions such as str_crush(), add_missing_column(), coalesce_data() and drop_na_all() that complement 'tidyverse' functionality or functions that provide alternative behaviors such as if_else2() and str_detect2().

License MIT + file LICENSE

URL https://poissonconsulting.github.io/tidyplus/, https://github.com/poissonconsulting/tidyplus

BugReports https://github.com/poissonconsulting/tidyplus/issues

Depends R (>= 4.1.0)

Imports chk, dplyr, lifecycle, rlang, stringi, stringr, tibble, tidyr, tidyselect, vctrs

Suggests covr, readr, sf, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/Needs/website poissonconsulting/poissontemplate

Config/roxygen2/version 8.0.0.9000

NeedsCompilation no

Author Joe Thorley [aut] (ORCID: <https://orcid.org/0000-0002-7683-4592>), Ayla Pearson [aut, cre] (ORCID: <https://orcid.org/0000-0001-7388-1222>), Duncan Kennedy [ctb] (ORCID: <https://orcid.org/0009-0001-4880-5751>), Poisson Consulting [cph, fnd]

Maintainer Ayla Pearson <ayla@poissonconsulting.ca>

Repository CRAN

Date/Publication 2026-07-18 19:30:02 UTC

Contents

add_missing_column 2
coalesce_data 3
collapse_comments 4
drop_na_all 5
drop_uninformative_columns 6
duplicates 7
if_else2 7
only 9
replace_na_if 10
str_crush 11
str_detect2 11
str_replace_vec 12
str_to_snake_case 13
summarise2 14
unite_str 16

Index **17**

`add_missing_column` *Add missing columns to a data frame*

Description

This is a convenient way to add one more columns (if not already present) to an existing data frame. It is useful to ensure that all required columns are present in a data frame.

Usage

```
add_missing_column(
  .data,
  ...,
  .before = NULL,
  .after = NULL,
  .name_repair = c("check_unique", "unique", "universal", "minimal")
)
```

Arguments

`.data`	Data frame to append to.
`...`	<`dynamic-dots`> Name-value pairs, passed on to `tibble()`. All values must have the same size of `.data` or size 1.
`.before`, `.after`	One-based column index or column name where to add the new columns, default: after last column.
`.name_repair`	Treatment of problematic column names:

- `"minimal"`: No name repair or checks, beyond basic existence,

- "unique": Make sure names are unique and not empty,
- "check_unique": (default value), no name repair, but check they are unique,
- "universal": Make the names unique and syntactic
- "unique_quiet": Same as "unique", but "quiet"
- "universal_quiet": Same as "universal", but "quiet"
- a function: apply custom name repair (e.g., .name_repair = make.names for names in the style of base R).
- A purrr-style anonymous function, see rlang::as_function()

This argument is passed on as repair to vctrs::vec_as_names(). See there for more details on these terms and the strategies used to enforce them.

Details

It is wrapper on tibble::add_column() that doesn't error if the column is already present.

Value

The original data frame with missing columns added if not already present.

See Also

tibble::add_column()

Examples

```
data <- tibble::tibble(x = 1:3, y = 3:1)

tibble::add_column(data, z = -1:1, w = 0)
add_missing_column(data, z = -1:1, .before = "y")

# add_column errors if already present
try(tibble::add_column(data, x = 4:6))

# add_missing_column silently ignores
add_missing_column(data, x = 4:6)
```

---

coalesce_data *Coalesce Data*

Description

Coalesce values in multiple columns by finding the first non-missing value at each position. Coalesced columns are removed.

Usage

```
coalesce_data(x, coalesce = list(), quiet = FALSE)
```

Arguments

`x`	A data frame.
`coalesce`	A uniquely named list of character vectors where the names are the new column names and the values are the names of the columns to coalesce. If a single value is provided for a column it is treated as a regular expression.
`quiet`	A flag specifying whether to provide messages.

Details

Coalescence is performed in the order specified in the coalesce argument such that a column produced by coalescence can be further coalesced.

Value

The original data frame with one or more columns coalesced into a new column.

See Also

`dplyr::coalesce()`

Examples

```
data <- data.frame(x = c(1, NA, NA), y = c(NA, 3, NA), z = c(7, 8, 9), a = c(4, 5, 6))
coalesce_data(data, list(b = c("x", "y")), quiet = TRUE)
coalesce_data(data, list(z = c("y", "x"), d = c("z", "a")))
```

`collapse_comments` *Collapse Comments*

Description

Collapse comments coercing each element to a string (character scalar) and then collapsing into a single string using the '. ' separator.

Usage

```
collapse_comments(...)
```

Arguments

`...`	objects to be collapsed into a string.

Value

A string of the collapsed comments.

See Also

unite_str()

Examples

```
collapse_comments("Saw fish", character(0), "Nice. .", NA_character_)

data <- data.frame(
  visit = c(1, 1, 2, 2),
  fish = 1:4,
  comment = c("Sunny day.  ", "Skinny fish", "Lost boot", NA)
)

## Not run:
data |>
  dplyr::group_by(visit) |>
  dplyr::summarise(comment = collapse_comments(comment)) |>
  dplyr::ungroup()

## End(Not run)
```

---

drop_na_all *Drop rows containing all missing values*

Description

This is a convenient way to drop uninformative rows from a data frame.

Usage

```
drop_na_all(data, ...)
```

Arguments

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

Value

The original data frame with rows for which all values are missing dropped.

See Also

tidyr::drop_na and drop_uninformative_columns

Examples

```
data <- tibble::tibble(
  a = c(NA, NA, NA), b = c(1, 1, NA), c = c(2, NA, NA)
)

drop_na_all(data)
drop_na_all(data, a, c)
```

---

drop_uninformative_columns

Drop uninformative columns from a data frame

Description

This is a convenient way to drop columns which all have one value (missing or not) or if `na_distinct = FALSE` also drop columns which all have one value and/or missing values.

Usage

```
drop_uninformative_columns(data, na_distinct = TRUE)
```

Arguments

`data`	A data frame.
`na_distinct`	A flag specifying whether to treat missing values as distinct from other values.

Value

The original data frame with only informative columns.

Examples

```
data <- tibble::tibble(
  a = c(1, 1, 1), x = c(NA, NA, NA), b = c(1, 1, NA),
  z = c(1, 2, 2), e = c(1, 2, NA)
)

drop_uninformative_columns(data)
drop_uninformative_columns(data, na_distinct = FALSE)
```

duplicates — *Keep non-unique rows in a data frame*

Description

Keeps only non-unique rows within a data frame.

Usage

```
duplicates(.data, ..., .keep_all = TRUE)
```

Arguments

`.data`	A data.frame.
`...`	Optional variables to use when determining non-uniqueness. If omitted, will use all variables in the data frame.
`.keep_all`	A flag specifying whether to keep all variables in .data.

Value

The original data frame with only non-unique rows.

Examples

```
data <- tibble::tibble(x = c(1, 2, 1, 1), y = c(1, 1, 1, 5))

duplicates(data)
duplicates(data, x)
duplicates(data, y)
duplicates(data, x, y)
duplicates(data, y, .keep_all = FALSE)
```

if_else2 — *Vectorised if else.*

Description

Vectorised if else that if true returns first possibility otherwise returns second possibility (even if the condition is a missing value). When searching character vectors an alternative solution is to use `str_detect2()`.

Usage

```
if_else2(condition, true, false, error = FALSE)
```

Arguments

condition	A logical vector
true, false	Vectors to use for TRUE and FALSE values of condition. Both true and false will be recycled to the size of condition. true, false, and missing (if used) will be cast to their common type.
error	A logical value. If TRUE, provides an informative error message if no matches are found.

Value

Where condition is TRUE, the matching value from true, where it's FALSE or NA, the matching value from false.

See Also

ifelse() and dplyr::if_else().

Examples

```
# consider the following data frame
data <- tibble::tibble(
  x = c(TRUE, FALSE, NA),
  y = c("x is false", NA, "hello")
)

# with a single vector if_else2() behaves the same as the default call to if_else().
dplyr::mutate(data,
  y1 = dplyr::if_else(y != "x is false", "x is true", y),
  y2 = if_else2(y != "x is false", "x is true", y)
)

# however in the case of a second vector the use of
# if_else2() does not introduce missing values
dplyr::mutate(data,
  x1 = dplyr::if_else(stringr::str_detect(y, "x is false"), FALSE, x),
  x2 = if_else2(stringr::str_detect(y, "x is false"), FALSE, x)
)

# in the case of regular expression matching an alternative is to use
# str_detect2()
dplyr::mutate(data,
  x3 = dplyr::if_else(str_detect2(y, "x is false"), FALSE, x)
)
```

`only` *Extract the only distinct value from a vector*

Description

Extracts the only distinct value from an atomic vector or throws an informative error if no values or multiple distinct values.

Usage

```
only(x, na_rm = FALSE)
```

Arguments

`x`	An atomic vector.
`na_rm`	A flag indicating whether to exclude missing values.

Details

`only()` is useful when summarizing a vector by group while checking the assumption that it is constant within the group.

Value

The only distinct value from a vector otherwise throws an error.

See Also

`dplyr::first()`

Examples

```
only(c(1, 1))
only(c(NA, NA))
only(c(1, 1, NA), na_rm = TRUE)
try(only(character(0)))
try(only(c(1, NA)))
try(only(c(1, 2)))
```

replace_na_if *Conditional replacement of NAs with specified values*

Description

Unlike `tidyr::replace_na()`, it is only defined for vectors.

Usage

```
replace_na_if(x, condition, true)
```

Arguments

`x`	Vector with missing values to modify.
`condition`	A logical vector
`true`	The replacement values where condition is `TRUE`.

Details

`replace_na_if()` is a wrapper on `if_else2(is.na(x) & condition, true, x)`

Value

A modified version of x that replaces any missing values where condition is `TRUE` with `true`.

See Also

`tidyr::replace_na()` and `if_else2()`

Examples

```
data <- tibble::tibble(
  x = c(TRUE, FALSE, NA),
  y = c("x is false", NA, "x is false")
)

dplyr::mutate(data,
  x1 = tidyr::replace_na(x, FALSE),
  x3 = if_else2(is.na(x) & y == "x is false", FALSE, x),
  x4 = replace_na_if(x, y == "x is false", FALSE)
)
```

str_crush — *Remove whitespace from a string*

Description

`str_crush()`, which removes all whitespace from a string, is the logical extension to `stringr::str_trim()` and `stringr::str_squish()`.

Usage

```
str_crush(string)
```

Arguments

`string`	Input vector. Either a character vector, or something coercible to one.

Details

`str_crush()` is considered too specialized to be part of stringr.

Value

A character vector the same length as `string`.

See Also

`stringr::str_trim()` and `stringr::str_squish()`

Examples

```
str_crush("  String with trailing,  middle, and leading white space\t")
```

str_detect2 — *Detect the presence/absence of a match*

Description

Vectorised over `string` and `pattern`. Actually equivalent to `grepl(pattern, x)` as returns FALSE for `NA`s (unlike `stringr::str_detect()`). This behavior is useful when searching comments many of which are NA to indicate no comments present.

Usage

```
str_detect2(string, pattern, negate = FALSE)
```

Arguments

`string`	Input vector. Either a character vector, or something coercible to one.
`pattern`	Pattern to look for. The default interpretation is a regular expression, as described in `vignette("regular-expressions")`. Use `regex()` for finer control of the matching behaviour. Match a fixed string (i.e. by comparing only bytes), using `fixed()`. This is fast, but approximate. Generally, for matching human text, you'll want `coll()` which respects character matching rules for the specified locale. You can not match boundaries, including `""`, with this function.
`negate`	If `TRUE`, inverts the resulting boolean vector.

Value

A logical vector the same length as `string`/`pattern`.

See Also

`grepl()` and `stringr::str_detect()`

Examples

```
x <- c("b", NA, "ab")
pattern <- "^a"
grepl(pattern, x)
stringr::str_detect(x, pattern)
str_detect2(x, pattern)
```

---

str_replace_vec *String replace multiple strings*

Description

String replace multiple strings in a vector.

Usage

```
str_replace_vec(string, replace)
```

Arguments

`string`	Input vector. Either a character vector, or something coercible to one.
`replace`	A character vector where the names are the patterns to look for and the values are the replacement values (`c(pattern1 = replacement1)`)

Details

str_replace_vec() is a vectorized form of stringr::str_replace().

This is different from passing a named vector to stringr::str_replace_all, which performs multiple replacements but to all pattern matches in a string.

Value

A character vector the same length as string/pattern/replacement.

See Also

stringr::str_replace() and stringr::str_replace_all()

Examples

```
fruits <- c("two apples", "nine pears")
str_replace_vec(fruits, c("two" = "three", "nine" = "ten"))
```

---

str_to_snake_case *Converts strings to Snake Case*

Description

Converts strings to Snake Case

Usage

```
str_to_snake_case(x)
```

Arguments

x	input string or multiple strings to be converted to snake case

Value

string or strings converted to snake_case

Examples

```
str_to_snake_case("string of words")

str_to_snake_case("StringOfWords")

str_to_snake_case("s!t$ring of %char^&act*ers")

str_to_snake_case(c("multiples of strings", "strings in multiple", "many strings"))
```

summarise2 *Summarise Each Group Down to One Row*

Description

Wrapper on `dplyr::summarise` that sets the default for the .group variable to "keep". This means that all the groups set in `dplyr::group_by` are retained, not just the first group.

Usage

```
summarise2(.data, ..., .by = NULL, .groups = "keep")

summarize2(.data, ..., .by = NULL, .groups = "keep")
```

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.

`...` <`data-masking`> Name-value pairs of summary functions. The name will be the name of the variable in the result.

The value can be:

- A vector of length 1, e.g. `min(x)`, `n()`, or `sum(is.na(y))`.
- A data frame with 1 row, to add multiple columns from a single expression.

`.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.

`.groups` **[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 `.data`.
- `"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.

Value

An object *usually* of the same type as `.data`.

- The rows come from the underlying `group_keys()`.
- The columns are a combination of the grouping keys and the summary expressions that you provide.

- The grouping structure is controlled by the `.groups=` argument, the output may be another grouped_df, a tibble or a rowwise data frame.
- Data frame attributes are **not** preserved, because `summarise()` fundamentally creates a new data frame.

Useful functions

- Center: `mean()`, `median()`
- Spread: `sd()`, `IQR()`, `mad()`
- Range: `min()`, `max()`,
- Position: `first()`, `last()`, `nth()`,
- Count: `n()`, `n_distinct()`
- Logical: `any()`, `all()`

Backend variations

The data frame backend supports creating a variable and using it in the same summary. This means that previously created summary variables can be further transformed or combined within the summary, as in `mutate()`. However, it also means that summary variables with the same names as previous variables overwrite them, making those variables unavailable to later summary variables.

This behaviour may not be supported in other backends. To avoid unexpected results, consider using new names for your summary variables, especially when creating multiple summaries.

Methods

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages: no methods found.

See Also

`dplyr::summarise()` and `dplyr::summarize()`

Examples

```
df <- data.frame(
  group = c("A", "A", "B", "B"),
  id = c(1, 1, 2, 2),
  value = c(10, 4, 20, 6)
)
# summarise2 doesn't produce message about groups
df |>
  dplyr::group_by(group, id) |>
  summarise2(mean = mean(value))
# summarise doesn't retain all the groups set in `group_by`
df |>
  dplyr::group_by(group, id) |>
  dplyr::summarise(mean = mean(value))
```

unite_str — *Unite multiple character columns into one*

Description

Convenience function for combining character columns.

Usage

```
unite_str(data, col, ..., sep = ". ", remove = TRUE)
```

Arguments

data	A data frame.
col	The name of the new column, as a string or symbol. 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).
...	<`tidy-select`> Columns to unite
sep	Separator to use between values.
remove	If `TRUE`, remove input columns from output data frame.

Details

Blank values of "" are converted into missing values.

Value

The original data frame with the one or more columns combined as character vectors separated by a period.

See Also

`tidyr::unite()` and `collapse_comments()`

Examples

```
data <- tibble::tibble(x = c("good", "Saw fish.", "", NA), y = c("2021", NA, NA, NA))

# unite has poor handling of character vectors
tidyr::unite(data, "new", x, y, remove = FALSE)

unite_str(data, "new", x, y, remove = FALSE)
```

Index

?dplyr_by, *14*

add_missing_column, 2
all(), *15*
any(), *15*

coalesce_data, 3
coll(), *12*
collapse_comments, 4
collapse_comments(), *16*

dplyr::coalesce(), *4*
dplyr::first(), *9*
dplyr::if_else(), *8*
dplyr::summarise(), *15*
dplyr::summarize(), *15*
drop_na_all, 5
drop_uninformative_columns, *5*, 6
duplicates, 7

first(), *15*
fixed(), *12*

grepl(), *12*
group_by(), *14*
group_keys(), *14*
grouped_df, *15*

if_else2, 7
if_else2(), *10*
ifelse(), *8*
IQR(), *15*

last(), *15*

mad(), *15*
max(), *15*
mean(), *15*
median(), *15*
min(), *15*
mutate(), *15*

n(), *15*
n_distinct(), *15*
nth(), *15*

only, 9

quasiquotation, *16*

recycled, *8*
regex(), *12*
replace_na_if, 10
rlang::as_function(), *3*
rlang::ensym(), *16*
rowwise, *15*

sd(), *15*
str_crush, 11
str_detect2, 11
str_detect2(), *7*
str_replace_vec, 12
str_to_snake_case, 13
stringr::str_detect(), *11, 12*
stringr::str_replace(), *13*
stringr::str_replace_all, *13*
stringr::str_replace_all(), *13*
stringr::str_squish(), *11*
stringr::str_trim(), *11*
summarise2, 14
summarize2 *(*summarise2*)*, 14

tibble, *15*
tibble(), *2*
tibble::add_column(), *3*
tidyr::drop_na, *5*
tidyr::replace_na(), *10*
tidyr::unite(), *16*

unite_str, 16
unite_str(), *5*

vctrs::vec_as_names(), *3*