

# Package ‘powerjoin’

November 3, 2022

**Title** Extensions of 'dplyr' and 'fuzzyjoin' Join Functions

**Version** 0.1.0

**Description** We extend 'dplyr' and 'fuzzyjoin' join functions with features to preprocess the data, apply various data checks, and deal with conflicting columns.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.2.1

**Imports** dplyr, glue, rlang, tidyselect, vctrs, purrr, tibble, tidyr, cli, methods

**URL** <https://github.com/moodymudskipper/powerjoin>

**BugReports** <https://github.com/moodymudskipper/powerjoin/issues>

**Suggests** testthat (>= 3.0.0)

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Antoine Fabri [aut, cre],  
Hadley Wickham [ctb] (aut/cre of dplyr,  
<<https://orcid.org/0000-0003-4757-117X>>),  
Romain François [ctb] (aut of dplyr,  
<<https://orcid.org/0000-0002-2444-4226>>),  
David Robinson [ctb] (aut of fuzzyjoin),  
RStudio [cph, fnd] (cph/fnd dplyr)

**Maintainer** Antoine Fabri <[antoine.fabri@gmail.com](mailto:antoine.fabri@gmail.com)>

**Repository** CRAN

**Date/Publication** 2022-11-03 15:20:07 UTC

## R topics documented:

check_specs . . . . .	2
coalesce_xy . . . . .	3

full_diagnostic . . . . .	4
paste_xy . . . . .	4
power_left_join . . . . .	5
preprocess_inputs . . . . .	9

<b>Index</b>	<b>11</b>
--------------	-----------

---

check_specs	<i>Build a checklist for power joins</i>
-------------	--

---

## Description

Build a checklist for power joins

## Usage

```
check_specs(
  implicit_keys = c("inform", "ignore", "warn", "abort"),
  column_conflict = c("ignore", "inform", "warn", "abort"),
  duplicate_keys_left = c("ignore", "inform", "warn", "abort"),
  duplicate_keys_right = c("ignore", "inform", "warn", "abort"),
  unmatched_keys_left = c("ignore", "inform", "warn", "abort"),
  unmatched_keys_right = c("ignore", "inform", "warn", "abort"),
  missing_key_combination_left = c("ignore", "inform", "warn", "abort"),
  missing_key_combination_right = c("ignore", "inform", "warn", "abort"),
  inconsistent_factor_levels = c("ignore", "inform", "warn", "abort"),
  inconsistent_type = c("ignore", "inform", "warn", "abort"),
  grouped_input = c("ignore", "inform", "warn", "abort"),
  na_keys = c("ignore", "inform", "warn", "abort")
)
```

## Arguments

`implicit_keys` What to do if keys are not given explicitly through the `by` argument

`column_conflict` What to do if the join creates a column conflict which is not handled by the `conflict` argument

`duplicate_keys_left` What to do if we find duplicate sets of keys in the left table

`duplicate_keys_right` What to do if we find duplicate sets of keys in the right table

`unmatched_keys_left` What to do if we find unmatched sets of keys in the left table

`unmatched_keys_right` What to do if we find unmatched sets of keys in the right table

`missing_key_combination_left` What to do if the left table doesn't contain all key combinations

missing_key_combination_right	What to do if the right table doesn't contain all key combinations
inconsistent_factor_levels	What to do if the key columns from both sides have inconsistent factor levels
inconsistent_type	What to do if we joined keys have a different type
grouped_input	What to do if one or both of the tables are grouped
na_keys	What to do if keys contain missing values

**Value**

A character vector of class "powerjoin\_check"

**Examples**

```
check_specs(
  implicit_keys = "ignore",
  grouped_input = "inform",
  column_conflict = "abort",
  na_keys = "warn")
```

---

 coalesce\_xy

*Coalesce helpers*


---

**Description**

These are wrappers around `dplyr::coalesce`, designed for convenient use in the `conflict` argument of `powerjoin`'s join functions. `coalesce_xy()` is just like `dplyr::coalesce` (except it takes only 2 arguments), `coalesce_yx()` looks first in `y` and then in `x` if `y` is missing.

**Usage**

```
coalesce_xy(x, y)
```

```
coalesce_yx(x, y)
```

**Arguments**

<code>x</code>	A vector
<code>y</code>	A vector

**Value**

A vector

**Examples**

```
coalesce_xy(c(NA, 2, 3), c(11, 12, NA))
coalesce_yx(c(NA, 2, 3), c(11, 12, NA))
```

---

full_diagnostic	<i>Inform on all potential issues</i>
-----------------	---------------------------------------

---

### Description

This is the output of `check_specs()` with all arguments set to "inform", it's useful for a complete join diagnostic.

### Usage

```
full_diagnostic
```

### Format

An object of class `powerjoin_check` of length 12.

---

paste_xy	<i>Paste helpers</i>
----------	----------------------

---

### Description

These are similar to `paste()` but by default ignore NA and empty strings (""). If they are found in a conflicting column we return the value from the other column without using the separator. If both columns have such values we return an empty string.

### Usage

```
paste_xy(x, y, sep = " ", na = NULL, ignore_empty = TRUE)
```

```
paste_yx(x, y, sep = " ", na = NULL, ignore_empty = TRUE)
```

### Arguments

x	A vector
y	A vector
sep	separator
na	How to treat NAs, they are ignored by default, if NA the result will be NA, just as with <code>stringr::str_c</code> , if "NA" NAs will be coerced to character just as with <code>paste()</code> . Any other string can be used
ignore_empty	Whether to ignore empty strings, to avoid trailing and leading separators

### Value

A character vector

**Examples**

```
paste_xy(letters[1:3], c("d", NA, ""))
paste_yx(letters[1:3], c("d", NA, ""))
paste_xy(letters[1:3], c("d", NA, ""), na = NA, ignore_empty = FALSE)
paste_xy(letters[1:3], c("d", NA, ""), na = "NA", ignore_empty = FALSE)
```

---

power_left_join	<i>Power joins</i>
-----------------	--------------------

---

**Description**

Power joins

**Usage**

```
power_left_join(  
  x,  
  y = NULL,  
  by = NULL,  
  copy = FALSE,  
  suffix = c(".x", ".y"),  
  keep = NULL,  
  na_matches = c("na", "never"),  
  check = check_specs(),  
  conflict = NULL,  
  fill = NULL  
)
```

```
power_right_join(  
  x,  
  y = NULL,  
  by = NULL,  
  copy = FALSE,  
  suffix = c(".x", ".y"),  
  keep = NULL,  
  na_matches = c("na", "never"),  
  check = check_specs(),  
  conflict = NULL,  
  fill = NULL  
)
```

```
power_inner_join(  
  x,  
  y = NULL,  
  by = NULL,  
  copy = FALSE,  
  suffix = c(".x", ".y"),
```

```

    keep = NULL,
    na_matches = c("na", "never"),
    check = check_specs(),
    conflict = NULL,
    fill = NULL
  )

power_full_join(
  x,
  y = NULL,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  na_matches = c("na", "never"),
  check = check_specs(),
  conflict = NULL,
  fill = 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 <i>Methods</i> , below, for more details.
by	As in dplyr, but extended so user can supply a formula or a list of character and formulas. Formulas are used for fuzzy joins and
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	A boolean for compatibility with dplyr, or a value among "left", "right", "both", "none" or "default". See details. The vales of the keep parameter work as follow : <ul style="list-style-type: none"> <li>• NULL (default) : merge keys and name them as the left table's keys, and keep columns used for fuzzy joins from both tables</li> <li>• left : keep only key columns for left table</li> <li>• right: keep only key columns for right table</li> <li>• both or TRUE: keep key columns from both tables, adding suffix if relevant</li> <li>• none : drop all key columns from the output</li> <li>• FALSE : merge keys and name them as the left table's keys, maps to none for fuzzy joins</li> </ul>
na_matches	Should NA and NaN values match one another? The default, "na", treats two NA or NaN values as equal, like %in%, <a href="#">match()</a> , <a href="#">merge()</a> . Use "never" to always treat two NA or NaN values as different, like joins for database sources, similarly to <code>merge(incomparables = FALSE)</code> .

check	A list created with <code>check_specs()</code>
conflict	A function, formula, the special value amongst "patch", or a named list of such items. If the LHS of the formula is <code>rw</code> the rhs will be applied rowwise. Note that the columns will be subsetted with <code>[]</code> so for list columns <code>.x</code> or <code>.y</code> will refer to length 1 lists and you might sometimes need <code>.x[[1]]</code> or <code>.y[[1]]</code> .
fill	Values used to replace missing values originating in unmatched keys, or a named list of such items.

**Value**

A data frame

**Examples**

```
# See README for a more verbose version
library(tibble)
male_penguins <- tribble(
  ~name, ~species, ~island, ~flipper_length_mm, ~body_mass_g,
  "Giordan", "Gentoo", "Biscoe", 222L, 5250L,
  "Lynden", "Adelie", "Torgersen", 190L, 3900L,
  "Reiner", "Adelie", "Dream", 185L, 3650L
)

female_penguins <- tribble(
  ~name, ~species, ~island, ~flipper_length_mm, ~body_mass_g,
  "Alonda", "Gentoo", "Biscoe", 211, 4500L,
  "Ola", "Adelie", "Dream", 190, 3600L,
  "Mishayla", "Gentoo", "Biscoe", 215, 4750L,
)

# apply different checks
power_inner_join(
  male_penguins[c("species", "island")],
  female_penguins[c("species", "island")],
  check = check_specs(implicit_keys = "ignore", duplicate_keys_right = "inform")
)

df1 <- tibble(id = 1:3, value = c(10, NA, 30))
df2 <- tibble(id = 2:4, value = c(22, 32, 42))

# handle conflicted columns when joining
power_left_join(df1, df2, by = "id", conflict = `+`)

# the most frequent use case is to coalesce
power_left_join(df1, df2, by = "id", conflict = coalesce_xy)
power_left_join(df1, df2, by = "id", conflict = coalesce_yx)

# the conflict function is applied colwise by default!
power_left_join(df1, df2, by = "id", conflict = ~ sum(.x, .y, na.rm = TRUE))

# apply conflict function rowwise
power_left_join(df1, df2, by = "id", conflict = rw ~ sum(.x, .y, na.rm = TRUE))
```

```

# subset columns without repeating keys
power_inner_join(
  male_penguins %>% select_keys_and(name),
  female_penguins %>% select_keys_and(female_name = name),
  by = c("species", "island")
)

# semi join
power_inner_join(
  male_penguins,
  female_penguins %>% select_keys_and(),
  by = c("species", "island")
)

# aggregate without repeating keys
power_left_join(
  male_penguins %>% summarize_by_keys(male_weight = mean(body_mass_g)),
  female_penguins %>% summarize_by_keys(female_weight = mean(body_mass_g)),
  by = c("species", "island")
)

# pack auxiliary columns without repeating keys
power_left_join(
  male_penguins %>% pack_along_keys(name = "m"),
  female_penguins %>% pack_along_keys(name = "f"),
  by = c("species", "island")
)

# fuzzy join
power_inner_join(
  male_penguins %>% select_keys_and(male_name = name),
  female_penguins %>% select_keys_and(female_name = name),
  by = c(~.x$flipper_length_mm < .y$flipper_length_mm, ~.x$body_mass_g > .y$body_mass_g)
)

# fuzzy + equi join
power_inner_join(
  male_penguins %>% select_keys_and(male_name = name),
  female_penguins %>% select_keys_and(female_name = name),
  by = c("island", ~.x$flipper_length_mm > .y$flipper_length_mm)
)

# define new column without repeating computation
power_inner_join(
  male_penguins %>% select_keys_and(male_name = name),
  female_penguins %>% select_keys_and(female_name = name),
  by = ~ (mass_ratio <- .y$body_mass_g / .x$body_mass_g) > 1.2
)
power_inner_join(
  male_penguins %>% select_keys_and(male_name = name),
  female_penguins %>% select_keys_and(female_name = name),
  by = ~ (mass_ratio <- .y$body_mass_g / .x$body_mass_g) > 1.2,

```



```

    keep = "none"
  )

# fill unmatched values
df1 <- tibble(id = 1:3)
df2 <- tibble(id = 1:2, value2 = c(2, NA), value3 = c(NA, 3))
power_left_join(df1, df2, by = "id", fill = 0)
power_left_join(df1, df2, by = "id", fill = list(value2 = 0))

# join recursively
df1 <- tibble(id = 1, a = "foo")
df2 <- tibble(id = 1, b = "bar")
df3 <- tibble(id = 1, c = "baz")
power_left_join(list(df1, df2, df3), by = "id")
power_left_join(df1, list(df2, df3), by = "id")

```

---

```
preprocess_inputs      Preprocess powerjoin inputs
```

---

## Description

These functions are named after the tidyverse functions `select`, `summarize`, `nest`, `pack`, `pivot_wider` and `pivot_longer` and are designed to avoid repetition of key columns when preprocessing the data for a join. They should only be used in the `x` and `y` arguments of `powerjoin` join functions. No further transformation should be applied on top of them.

## Usage

```

select_keys_and(.data, ...)

summarize_by_keys(.data, ...)

nest_by_keys(.data, ..., name = NULL)

pack_along_keys(.data, ..., name)

complete_keys(.data)

```

## Arguments

<code>.data</code>	A data frame to pivot.
<code>...</code>	Additional arguments passed on to methods.
<code>name</code>	Name of created column

## Details

Unlike their tidyverse counterparts these just add an attribute to the input and don't reshape it. The join function then preprocesses the inputs using these attributes and the keys.

**Value**

A data frame identical to the `.data` but with a `"powerjoin_preprocess"` attribute to be handled by the join functions

**Examples**

```
# in practice you'll mostly use those in join function calls directly
x <- select_keys_and(head(iris, 2), Sepal.Width)
# all it does is add an attribute that will be processed by the join function
attr(x, "powerjoin_preprocess")
# see `?power_left_join` or README for practical examples
```

# Index

## \* datasets

- full\_diagnostic, 4
  
- check\_specs, 2
- coalesce\_xy, 3
- coalesce\_yx (coalesce\_xy), 3
- complete\_keys (preprocess\_inputs), 9
  
- full\_diagnostic, 4
  
- match(), 6
- merge(), 6
  
- nest\_by\_keys (preprocess\_inputs), 9
  
- pack\_along\_keys (preprocess\_inputs), 9
- paste\_xy, 4
- paste\_yx (paste\_xy), 4
- power\_full\_join (power\_left\_join), 5
- power\_inner\_join (power\_left\_join), 5
- power\_left\_join, 5
- power\_right\_join (power\_left\_join), 5
- preprocess\_inputs, 9
  
- select\_keys\_and (preprocess\_inputs), 9
- summarize\_by\_keys (preprocess\_inputs), 9