Package 'joyn'

December 14, 2024

Type Package

Title Tool for Diagnosis of Tables Joins and Complementary Join Features

Version 0.2.4

Description Tool for diagnosing table joins. It combines the speed of `collapse` and `data.table`, the flexibility of `dplyr`, and the diagnosis and features of the `merge` command in `Stata`.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/randrescastaneda/joyn, https://randrescastaneda.github.io/joyn/

BugReports https://github.com/randrescastaneda/joyn/issues

Suggests badger, covr, knitr, rmarkdown, testthat (>= 3.0.0), withr, dplyr, tibble

Config/testthat/edition 3

Imports rlang, data.table, cli, utils, collapse (>= 2.0.15), lifecycle

Depends R (>= 2.10)

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author R.Andres Castaneda [aut, cre], Zander Prinsloo [aut], Rossana Tatulli [aut]

Maintainer R.Andres Castaneda <acastanedaa@worldbank.org>

Repository CRAN

Date/Publication 2024-12-13 23:20:02 UTC

Contents

`anti_join` *Anti join on two data frames*

Description

This is a `joyn` wrapper that works in a similar fashion to dplyr::anti_join

Usage

```
anti_join(
  x,
  y,
  by = intersect(names(x), names(y)),
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  relationship = "many-to-many",
  y_vars_to_keep = FALSE,
  reportvar = getOption("joyn.reportvar"),
  reporttype = c("factor", "character", "numeric"),
  roll = NULL,
  keep_common_vars = FALSE,
  sort = TRUE,
  verbose = getOption("joyn.verbose"),
  ...
)
```

Arguments

x	data frame: referred to as *left* in R terminology, or *master* in Stata terminology.
y	data frame: referred to as *right* in R terminology, or *using* in Stata terminology.
by	a character vector of variables to join by. If NULL, the default, joyn will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're correct (to suppress the message, simply explicitly list the variables that you want to join). To join by different variables on x and y use a vector of expressions. For example, `by = c("a = b", "z")` will use "a" in x, "b" in y, and "z" in both tables.
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.
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.

y_vars_to_keep character: Vector of variable names in y that will be kept after the merge. If TRUE (the default), it keeps all the brings all the variables in y into x. If FALSE or NULL, it does not bring any variable into x, but a report will be generated.

reportvar character: Name of reporting variable. Default is ".joyn". This is the same as variable "_merge" in Stata after performing a merge. If FALSE or NULL, the reporting variable will be excluded from the final table, though a summary of the join will be display after concluding.

reporttype character: One of *"character"* or *"numeric"*. Default is *"character"*. If *"numeric"*, the reporting variable will contain numeric codes of the source and the contents of each observation in the joined table. See below for more information.

roll double: *to be implemented*

keep_common_vars
logical: If TRUE, it will keep the original variable from y when both tables have common variable names. Thus, the prefix "y." will be added to the original name to distinguish from the resulting variable in the joined table.

sort logical: If TRUE, sort by key variables in by. Default is FALSE.

verbose logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

... Arguments passed on to joyn

match_type character: one of *"m:m"*, *"m:1"*, *"1:m"*, *"1:1"*. Default is *"1:1"* since this the most restrictive. However, following Stata's recommendation, it is better to be explicit and use any of the other three match types (See details in *match types sections*).

update_NAs logical: If TRUE, it will update NA values of all variables in x with actual values of variables in y that have the same name as the ones in x. If FALSE, NA values won't be updated, even if update_values is TRUE

update_values logical: If TRUE, it will update all values of variables in x with the actual of variables in y with the same name as the ones in x. **NAs from y won't be used to update actual values in x**. Yet, by default, NAs in x will be updated with values in y. To avoid this, make sure to set update_NAs = FALSE

allow.cartesian logical: Check documentation in official web site. Default is NULL, which implies that if the join is "1:1" it will be FALSE, but if the join has any "m" on it, it will be converted to TRUE. By specifying TRUE of FALSE you force the behavior of the join.

suffixes A character(2) specifying the suffixes to be used for making non-by column names unique. The suffix behaviour works in a similar fashion as the base::merge method does.

yvars **[Superseded]**: use now y_vars_to_keep

keep_y_in_x **[Superseded]**: use now keep_common_vars

msg_type character: type of messages to display by default

na.last logical. If TRUE, missing values in the data are placed last; if FALSE, they are placed first; if NA they are removed. na.last=NA is valid only for x[order(., na.last)] and its default is TRUE. setorder and setorderv only accept TRUE/FALSE with default FALSE.

Value

An data frame of the same class as x. The properties of the output are as close as possible to the ones returned by the dplyr alternative.

See Also

Other dplyr alternatives: full_join(), inner_join(), left_join(), right_join()

Examples

```
# Simple anti join
library(data.table)

x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = 11:15)
y1 = data.table(id = c(1,2, 4),
                y  = c(11L, 15L, 16))
anti_join(x1, y1, relationship = "many-to-one")
```

---

freq_table *Tabulate simple frequencies*

Description

tabulate one variable frequencies

Usage

```
freq_table(x, byvar, digits = 1, na.rm = FALSE, freq_var_name = "n")
```

Arguments

x	data frame
byvar	character: name of variable to tabulate. Use Standard evaluation.
digits	numeric: number of decimal places to display. Default is 1.
na.rm	logical: report NA values in frequencies. Default is FALSE.
freq_var_name	character: name for frequency variable. Default is "n"

Value

data.table with frequencies.

Examples

```
library(data.table)
x4 = data.table(id1 = c(1, 1, 2, 3, 3),
                id2 = c(1, 1, 2, 3, 4),
                t   = c(1L, 2L, 1L, 2L, NA_integer_),
                x   = c(16, 12, NA, NA, 15))
freq_table(x4, "id1")
```

---

full_join *Full join two data frames*

Description

This is a joyn wrapper that works in a similar fashion to dplyr::full_join

Usage

```
full_join(
  x,
  y,
  by = intersect(names(x), names(y)),
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  unmatched = "drop",
  relationship = "one-to-one",
  y_vars_to_keep = TRUE,
  update_values = FALSE,
  update_NAs = update_values,
  reportvar = getOption("joyn.reportvar"),
  reporttype = c("factor", "character", "numeric"),
  roll = NULL,
```

```
  keep_common_vars = FALSE,
  sort = TRUE,
  verbose = getOption("joyn.verbose"),
  ...
)
```

Arguments

x	data frame: referred to as *left* in R terminology, or *master* in Stata terminology.
y	data frame: referred to as *right* in R terminology, or *using* in Stata terminology.
by	a character vector of variables to join by. If NULL, the default, joyn will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're correct (to suppress the message, simply explicitly list the variables that you want to join). To join by different variables on x and y use a vector of expressions. For example, `by = c("a = b", "z")` will use "a" in x, "b" in y, and "z" in both tables.
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.

y_vars_to_keep character: Vector of variable names in y that will be kept after the merge. If TRUE (the default), it keeps all the brings all the variables in y into x. If FALSE or NULL, it does not bring any variable into x, but a report will be generated.

update_values logical: If TRUE, it will update all values of variables in x with the actual of variables in y with the same name as the ones in x. **NAs from y won't be used to update actual values in x**. Yet, by default, NAs in x will be updated with values in y. To avoid this, make sure to set update_NAs = FALSE

update_NAs logical: If TRUE, it will update NA values of all variables in x with actual values of variables in y that have the same name as the ones in x. If FALSE, NA values won't be updated, even if update_values is TRUE

reportvar character: Name of reporting variable. Default is ".joyn". This is the same as variable "_merge" in Stata after performing a merge. If FALSE or NULL, the reporting variable will be excluded from the final table, though a summary of the join will be display after concluding.

reporttype character: One of *"character"* or *"numeric"*. Default is *"character"*. If *"numeric"*, the reporting variable will contain numeric codes of the source and the

contents of each observation in the joined table. See below for more information.

`roll` double: *to be implemented*

`keep_common_vars`
logical: If TRUE, it will keep the original variable from y when both tables have common variable names. Thus, the prefix "y." will be added to the original name to distinguish from the resulting variable in the joined table.

`sort` logical: If TRUE, sort by key variables in by. Default is FALSE.

`verbose` logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

`...` Arguments passed on to `joyn`

`match_type` character: one of *"m:m"*, *"m:1"*, *"1:m"*, *"1:1"*. Default is *"1:1"* since this the most restrictive. However, following Stata's recommendation, it is better to be explicit and use any of the other three match types (See details in *match types sections*).

`allow.cartesian` logical: Check documentation in official web site. Default is `NULL`, which implies that if the join is "1:1" it will be `FALSE`, but if the join has any "m" on it, it will be converted to `TRUE`. By specifying `TRUE` of `FALSE` you force the behavior of the join.

`suffixes` A character(2) specifying the suffixes to be used for making non-by column names unique. The suffix behaviour works in a similar fashion as the base::merge method does.

`yvars` **[Superseded]**: use now `y_vars_to_keep`

`keep_y_in_x` **[Superseded]**: use now `keep_common_vars`

`msg_type` character: type of messages to display by default

`na.last` `logical`. If `TRUE`, missing values in the data are placed last; if `FALSE`, they are placed first; if `NA` they are removed. `na.last=NA` is valid only for `x[order(., na.last)]` and its default is `TRUE`. `setorder` and `setorderv` only accept `TRUE`/`FALSE` with default `FALSE`.

Value

An data frame of the same class as `x`. The properties of the output are as close as possible to the ones returned by the dplyr alternative.

See Also

Other dplyr alternatives: `anti_join()`, `inner_join()`, `left_join()`, `right_join()`

Examples

```
# Simple full join
library(data.table)

x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = 11:15)
```

```
y1 = data.table(id = c(1,2, 4),
                y  = c(11L, 15L, 16))
full_join(x1, y1, relationship = "many-to-one")
```

`get_joyn_options` *Get joyn options*

Description

This function aims to display and store info on joyn options

Usage

```
get_joyn_options(env = .joynenv, display = TRUE, option = NULL)
```

Arguments

`env`	environment, which is joyn environment by default
`display`	logical, if TRUE displays (i.e., print) info on joyn options and corresponding default and current values
`option`	character or NULL. If character, name of a specific joyn option. If NULL, all joyn options

Value

joyn options and values invisibly as a list

See Also

JOYn options functions `set_joyn_options()`

Examples

```
## Not run:

# display all joyn options, their default and current values
joyn:::get_joyn_options()

# store list of option = value pairs AND do not display info
joyn_options <- joyn:::get_joyn_options(display = FALSE)

# get info on one specific option and store it
joyn.verbose <- joyn:::get_joyn_options(option = "joyn.verbose")

# get info on two specific option
joyn:::get_joyn_options(option = c("joyn.verbose", "joyn.reportvar"))


## End(Not run)
```

inner_join *Inner join two data frames*

Description

This is a joyn wrapper that works in a similar fashion to dplyr::inner_join

Usage

```
inner_join(
  x,
  y,
  by = intersect(names(x), names(y)),
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  unmatched = "drop",
  relationship = "one-to-one",
  y_vars_to_keep = TRUE,
  update_values = FALSE,
  update_NAs = update_values,
  reportvar = getOption("joyn.reportvar"),
  reporttype = c("factor", "character", "numeric"),
  roll = NULL,
  keep_common_vars = FALSE,
  sort = TRUE,
  verbose = getOption("joyn.verbose"),
  ...
)
```

Arguments

x	data frame: referred to as *left* in R terminology, or *master* in Stata terminology.
y	data frame: referred to as *right* in R terminology, or *using* in Stata terminology.
by	a character vector of variables to join by. If NULL, the default, joyn will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're correct (to suppress the message, simply explicitly list the variables that you want to join). To join by different variables on x and y use a vector of expressions. For example, by = c("a = b", "z") will use "a" in x, "b" in y, and "z" in both tables.
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.

y_vars_to_keep character: Vector of variable names in y that will be kept after the merge. If TRUE (the default), it keeps all the brings all the variables in y into x. If FALSE or NULL, it does not bring any variable into x, but a report will be generated.

update_values logical: If TRUE, it will update all values of variables in x with the actual of variables in y with the same name as the ones in x. **NAs from y won't be used to update actual values in x**. Yet, by default, NAs in x will be updated with values in y. To avoid this, make sure to set update_NAs = FALSE

update_NAs logical: If TRUE, it will update NA values of all variables in x with actual values of variables in y that have the same name as the ones in x. If FALSE, NA values won't be updated, even if update_values is TRUE

reportvar character: Name of reporting variable. Default is ".joyn". This is the same as variable "_merge" in Stata after performing a merge. If FALSE or NULL, the reporting variable will be excluded from the final table, though a summary of the join will be display after concluding.

reporttype character: One of *"character"* or *"numeric"*. Default is *"character"*. If *"numeric"*, the reporting variable will contain numeric codes of the source and the contents of each observation in the joined table. See below for more information.

roll double: *to be implemented*

keep_common_vars
logical: If TRUE, it will keep the original variable from y when both tables have common variable names. Thus, the prefix "y." will be added to the original name to distinguish from the resulting variable in the joined table.

sort logical: If TRUE, sort by key variables in by. Default is FALSE.

verbose logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

... Arguments passed on to joyn

match_type character: one of *"m:m"*, *"m:1"*, *"1:m"*, *"1:1"*. Default is *"1:1"* since this the most restrictive. However, following Stata's recommendation, it is better to be explicit and use any of the other three match types (See details in *match types sections*).

allow.cartesian logical: Check documentation in official web site. Default is NULL, which implies that if the join is "1:1" it will be FALSE, but if the join has any "m" on it, it will be converted to TRUE. By specifying TRUE of FALSE you force the behavior of the join.

suffixes A character(2) specifying the suffixes to be used for making non-by column names unique. The suffix behaviour works in a similar fashion as the base::merge method does.

yvars **[Superseded]**: use now y_vars_to_keep

keep_y_in_x **[Superseded]**: use now keep_common_vars

msg_type character: type of messages to display by default

na.last logical. If TRUE, missing values in the data are placed last; if FALSE, they are placed first; if NA they are removed. na.last=NA is valid only for x[order(., na.last)] and its default is TRUE. setorder and setorderv only accept TRUE/FALSE with default FALSE.

Value

An data frame of the same class as x. The properties of the output are as close as possible to the ones returned by the dplyr alternative.

See Also

Other dplyr alternatives: anti_join(), full_join(), left_join(), right_join()

Examples

```
# Simple full join
library(data.table)

x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = 11:15)
y1 = data.table(id = c(1,2, 4),
                y  = c(11L, 15L, 16))
inner_join(x1, y1, relationship = "many-to-one")
```

---

is_balanced *Is data frame balanced by group?*

Description

Check if the data frame is balanced by group of columns, i.e., if it contains every combination of the elements in the specified variables

Usage

```
is_balanced(df, by, return = c("logic", "table"))
```

Arguments

df	data frame
by	character: variables used to check if df is balanced
return	character: either "logic" or "table". If "logic", returns TRUE or FALSE depending on whether data frame is balanced. If "table" returns the unbalanced observations - i.e. the combinations of elements in specified variables not found in input df

Value

logical, if return == "logic", else returns data frame of unbalanced observations

Examples

```
x1 = data.frame(id = c(1L, 1L, 2L, 3L, NA_integer_),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = 11:15)
is_balanced(df = x1,
            by = c("id", "t"),
        return = "table") # returns combination of elements in "id" and "t" not present in df
is_balanced(df = x1,
            by = c("id", "t"),
            return = "logic") # FALSE
```

---

is_id *Check if dt is uniquely identified by* by *variable*

Description

report if dt is uniquely identified by by var or, if report = TRUE, the duplicates in by variable

Usage

```
is_id(
  dt,
  by,
  verbose = getOption("joyn.verbose", default = FALSE),
  return_report = FALSE
)
```

Arguments

dt	either right of left table
by	variable to merge by
verbose	logical: if TRUE messages will be displayed
return_report	logical: if TRUE, returns data with summary of duplicates. If FALSE, returns logical value depending on whether dt is uniquely identified by by

Value

logical or data.frame, depending on the value of argument `return_report`

Examples

```
library(data.table)

# example with data frame not uniquely identified by `by` var

y <- data.table(id = c("c","b", "c", "a"),
                y  = c(11L, 15L, 18L, 20L))
is_id(y, by = "id")
is_id(y, by = "id", return_report = TRUE)

# example with data frame uniquely identified by `by` var

y1 <- data.table(id = c("1","3", "2", "9"),
                 y  = c(11L, 15L, 18L, 20L))
is_id(y1, by = "id")
```

---

joyn *Join two tables*

Description

This is the primary function in the joyn package. It executes a full join, performs a number of checks, and filters to allow the user-specified join.

Usage

```
joyn(
  x,
  y,
  by = intersect(names(x), names(y)),
  match_type = c("1:1", "1:m", "m:1", "m:m"),
  keep = c("full", "left", "master", "right", "using", "inner", "anti"),
  y_vars_to_keep = ifelse(keep == "anti", FALSE, TRUE),
  update_values = FALSE,
  update_NAs = update_values,
  reportvar = getOption("joyn.reportvar"),
  reporttype = c("factor", "character", "numeric"),
  roll = NULL,
  keep_common_vars = FALSE,
  sort = FALSE,
  verbose = getOption("joyn.verbose"),
  suffixes = getOption("joyn.suffixes"),
  allow.cartesian = deprecated(),
  yvars = deprecated(),
```

```
  keep_y_in_x = deprecated(),
  na.last = getOption("joyn.na.last"),
  msg_type = getOption("joyn.msg_type")
)
```

Arguments

x	data frame: referred to as *left* in R terminology, or *master* in Stata terminology.
y	data frame: referred to as *right* in R terminology, or *using* in Stata terminology.
by	a character vector of variables to join by. If NULL, the default, joyn will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're correct (to suppress the message, simply explicitly list the variables that you want to join). To join by different variables on x and y use a vector of expressions. For example, `by = c("a = b", "z")` will use "a" in x, "b" in y, and "z" in both tables.
match_type	character: one of *"m:m"*, *"m:1"*, *"1:m"*, *"1:1"*. Default is *"1:1"* since this the most restrictive. However, following Stata's recommendation, it is better to be explicit and use any of the other three match types (See details in *match types sections*).
keep	atomic character vector of length 1: One of *"full"*, *"left"*, *"master"*, *"right"*, *"using"*, *"inner"*. Default is *"full"*. Even though this is not the regular behavior of joins in R, the objective of joyn is to present a diagnosis of the join which requires a full join. That is why the default is a a full join. Yet, if *"left"* or *"master"*, it keeps the observations that matched in both tables and the ones that did not match in x. The ones in y will be discarded. If *"right"* or *"using"*, it keeps the observations that matched in both tables and the ones that did not match in y. The ones in x will be discarded. If *"inner"*, it only keeps the observations that matched both tables. Note that if, for example, a `keep = "left", the joyn()function still executes a full join under the hood and then filte`
y_vars_to_keep	character: Vector of variable names in y that will be kept after the merge. If TRUE (the default), it keeps all the brings all the variables in y into x. If FALSE or NULL, it does not bring any variable into x, but a report will be generated.
update_values	logical: If TRUE, it will update all values of variables in x with the actual of variables in y with the same name as the ones in x. **NAs from y won't be used to update actual values in x**. Yet, by default, NAs in x will be updated with values in y. To avoid this, make sure to set `update_NAs = FALSE`
update_NAs	logical: If TRUE, it will update NA values of all variables in x with actual values of variables in y that have the same name as the ones in x. If FALSE, NA values won't be updated, even if `update_values` is `TRUE`
reportvar	character: Name of reporting variable. Default is ".joyn". This is the same as variable "_merge" in Stata after performing a merge. If FALSE or NULL, the reporting variable will be excluded from the final table, though a summary of the join will be display after concluding.
reporttype	character: One of *"character"* or *"numeric"*. Default is *"character"*. If *"numeric"*, the reporting variable will contain numeric codes of the source and the contents of each observation in the joined table. See below for more information.

roll	double: *to be implemented*
keep_common_vars	logical: If TRUE, it will keep the original variable from y when both tables have common variable names. Thus, the prefix "y." will be added to the original name to distinguish from the resulting variable in the joined table.
sort	logical: If TRUE, sort by key variables in by. Default is FALSE.
verbose	logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.
suffixes	A character(2) specifying the suffixes to be used for making non-by column names unique. The suffix behaviour works in a similar fashion as the base::merge method does.
allow.cartesian	logical: Check documentation in official web site. Default is NULL, which implies that if the join is "1:1" it will be FALSE, but if the join has any "m" on it, it will be converted to TRUE. By specifying TRUE of FALSE you force the behavior of the join.
yvars	**[Superseded]**: use now y_vars_to_keep
keep_y_in_x	**[Superseded]**: use now keep_common_vars
na.last	logical. If TRUE, missing values in the data are placed last; if FALSE, they are placed first; if NA they are removed. na.last=NA is valid only for x[order(., na.last)] and its default is TRUE. setorder and setorderv only accept TRUE/FALSE with default FALSE.
msg_type	character: type of messages to display by default

Value

a data.table joining x and y.

match types

Using the same wording of the Stata manual

1:1: specifies a one-to-one match merge. The variables specified in by uniquely identify single observations in both table.

1:m and m:1: specify *one-to-many* and *many-to-one* match merges, respectively. This means that in of the tables the observations are uniquely identify by the variables in by, while in the other table many (two or more) of the observations are identify by the variables in by

m:m refers to *many-to-many merge*. variables in by does not uniquely identify the observations in either table. Matching is performed by combining observations with equal values in by; within matching values, the first observation in the master (i.e. left or x) table is matched with the first matching observation in the using (i.e. right or y) table; the second, with the second; and so on. If there is an unequal number of observations within a group, then the last observation of the shorter group is used repeatedly to match with subsequent observations of the longer group.

reporttype

If `reporttype = "numeric"`, then the numeric values have the following meaning:

1: row comes from x, i.e. "x" 2: row comes from y, i.e. "y" 3: row from both x and y, i.e. "x & y" 4: row has NA in x that has been updated with y, i.e. "NA updated" 5: row has valued in x that has been updated with y, i.e. "value updated" 6: row from x that has not been updated, i.e. "not updated"

NAs order

NAs are placed either at first or at last in the resulting data.frame depending on the value of `getOption("joyn.na.last")`. The Default is `FALSE` as it is the default value of data.table::setorderv.

Examples

```
# Simple join
library(data.table)
x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
t  = c(1L, 2L, 1L, 2L, NA_integer_),
x  = 11:15)

y1 = data.table(id = 1:2,
                y  = c(11L, 15L))

x2 = data.table(id = c(1, 1, 2, 3, NA),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = c(16, 12, NA, NA, 15))

y2 = data.table(id = c(1, 2, 5, 6, 3),
              yd = c(1, 2, 5, 6, 3),
              y  = c(11L, 15L, 20L, 13L, 10L),
              x  = c(16:20))
joyn(x1, y1, match_type = "m:1")

# Bad merge for not specifying by argument or match_type
joyn(x2, y2)

# good merge, ignoring variable x from y
joyn(x2, y2, by = "id", match_type = "m:1")

# update NAs in x variable form x
joyn(x2, y2, by = "id", update_NAs = TRUE, match_type = "m:1")

# Update values in x with variables from y
joyn(x2, y2, by = "id", update_values = TRUE, match_type = "m:1")
```

joyn_msg *display type of joyn message*

Description

display type of joyn message

Usage

```
joyn_msg(msg_type = getOption("joyn.msg_type"), msg = NULL)
```

Arguments

msg_type	character: one or more of the following: all, basic, info, note, warn, timing, or err
msg	character vector to be parsed to cli::cli_abort(). Default is NULL. It only works if "err" %in% msg_type. This is an internal argument.

Value

returns data frame with message invisibly. print message in console

See Also

Messages functions clear_joynenv(), joyn_msgs_exist(), joyn_report(), msg_type_dt(), store_msg(), style(), type_choices()

Examples

```
library(data.table)
x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
t  = c(1L, 2L, 1L, 2L, NA_integer_),
x  = 11:15)

y1 = data.table(id = 1:2,
                y  = c(11L, 15L))
df <- joyn(x1, y1, match_type = "m:1")
joyn_msg("basic")
joyn_msg("all")
```

joyn_report *Print JOYn report table*

Description

Print JOYn report table

Usage

```
joyn_report(verbose = getOption("joyn.verbose"))
```

Arguments

verbose	logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

Value

invisible table of frequencies

See Also

Messages functions clear_joynenv(), joyn_msg(), joyn_msgs_exist(), msg_type_dt(), store_msg(), style(), type_choices()

Examples

```
library(data.table)
x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
t  = c(1L, 2L, 1L, 2L, NA_integer_),
x  = 11:15)

y1 = data.table(id = 1:2,
                y  = c(11L, 15L))

d <- joyn(x1, y1, match_type = "m:1")
joyn_report(verbose = TRUE)
```

left_join *Left join two data frames*

Description

This is a joyn wrapper that works in a similar fashion to dplyr::left_join

Usage

```
left_join(
  x,
  y,
  by = intersect(names(x), names(y)),
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  unmatched = "drop",
  relationship = NULL,
  y_vars_to_keep = TRUE,
  update_values = FALSE,
  update_NAs = update_values,
  reportvar = getOption("joyn.reportvar"),
  reporttype = c("factor", "character", "numeric"),
  roll = NULL,
  keep_common_vars = FALSE,
  sort = TRUE,
  verbose = getOption("joyn.verbose"),
  ...
)
```

Arguments

x	data frame: referred to as *left* in R terminology, or *master* in Stata terminology.
y	data frame: referred to as *right* in R terminology, or *using* in Stata terminology.
by	a character vector of variables to join by. If NULL, the default, joyn will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're correct (to suppress the message, simply explicitly list the variables that you want to join). To join by different variables on x and y use a vector of expressions. For example, by = c("a = b", "z") will use "a" in x, "b" in y, and "z" in both tables.
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`.

`y_vars_to_keep` character: Vector of variable names in y that will be kept after the merge. If TRUE (the default), it keeps all the brings all the variables in y into x. If FALSE or NULL, it does not bring any variable into x, but a report will be generated.

`update_values` logical: If TRUE, it will update all values of variables in x with the actual of variables in y with the same name as the ones in x. **NAs from y won't be used to update actual values in x**. Yet, by default, NAs in x will be updated with values in y. To avoid this, make sure to set `update_NAs = FALSE`

`update_NAs` logical: If TRUE, it will update NA values of all variables in x with actual values of variables in y that have the same name as the ones in x. If FALSE, NA values won't be updated, even if `update_values` is `TRUE`

`reportvar` character: Name of reporting variable. Default is ".joyn". This is the same as variable "_merge" in Stata after performing a merge. If FALSE or NULL, the reporting variable will be excluded from the final table, though a summary of the join will be display after concluding.

`reporttype` character: One of *"character"* or *"numeric"*. Default is *"character"*. If *"numeric"*, the reporting variable will contain numeric codes of the source and the contents of each observation in the joined table. See below for more information.

`roll` double: *to be implemented*

`keep_common_vars`
logical: If TRUE, it will keep the original variable from y when both tables have common variable names. Thus, the prefix "y." will be added to the original name to distinguish from the resulting variable in the joined table.

`sort` logical: If TRUE, sort by key variables in by. Default is FALSE.

`verbose` logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

`...` Arguments passed on to `joyn`

`match_type` character: one of *"m:m"*, *"m:1"*, *"1:m"*, *"1:1"*. Default is *"1:1"* since this the most restrictive. However, following Stata's recommendation, it is better to be explicit and use any of the other three match types (See details in *match types sections*).

`allow.cartesian` logical: Check documentation in official web site. Default is `NULL`, which implies that if the join is "1:1" it will be `FALSE`, but if the join has any "m" on it, it will be converted to `TRUE`. By specifying `TRUE` of `FALSE` you force the behavior of the join.

`suffixes` A character(2) specifying the suffixes to be used for making non-by column names unique. The suffix behaviour works in a similar fashion as the base::merge method does.

`yvars` **[Superseded]**: use now `y_vars_to_keep`

`keep_y_in_x` **[Superseded]**: use now `keep_common_vars`

`msg_type` character: type of messages to display by default

`na.last` logical. If `TRUE`, missing values in the data are placed last; if `FALSE`, they are placed first; if `NA` they are removed. `na.last=NA` is valid only for `x[order(., na.last)]` and its default is `TRUE`. `setorder` and `setorderv` only accept `TRUE`/`FALSE` with default `FALSE`.

Value

An data frame of the same class as x. The properties of the output are as close as possible to the ones returned by the dplyr alternative.

See Also

Other dplyr alternatives: anti_join(), full_join(), inner_join(), right_join()

Examples

```
# Simple left join
library(data.table)

x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = 11:15)
y1 = data.table(id = c(1,2, 4),
                y  = c(11L, 15L, 16))
left_join(x1, y1, relationship = "many-to-one")
```

---

merge *Merge two data frames*

Description

This is a joyn wrapper that works in a similar fashion to base::merge and data.table::merge, which is why merge masks the other two.

Usage

```
merge(
  x,
  y,
  by = NULL,
  by.x = NULL,
  by.y = NULL,
  all = FALSE,
  all.x = all,
  all.y = all,
  sort = TRUE,
  suffixes = c(".x", ".y"),
  no.dups = TRUE,
  allow.cartesian = getOption("datatable.allow.cartesian"),
  match_type = c("m:m", "m:1", "1:m", "1:1"),
  keep_common_vars = TRUE,
  ...
)
```

Arguments

x, y	data tables. y is coerced to a data.table if it isn't one already.
by	A vector of shared column names in x and y to merge on. This defaults to the shared key columns between the two tables. If y has no key columns, this defaults to the key of x.
by.x, by.y	Vectors of column names in x and y to merge on.
all	logical; all = TRUE is shorthand to save setting both all.x = TRUE and all.y = TRUE.
all.x	logical; if TRUE, rows from x which have no matching row in y are included. These rows will have 'NA's in the columns that are usually filled with values from y. The default is FALSE so that only rows with data from both x and y are included in the output.
all.y	logical; analogous to all.x above.
sort	logical. If TRUE (default), the rows of the merged data.table are sorted by setting the key to the by / by.x columns. If FALSE, unlike base R's merge for which row order is unspecified, the row order in x is retained (including retaining the position of missing entries when all.x=TRUE), followed by y rows that don't match x (when all.y=TRUE) retaining the order those appear in y.
suffixes	A character(2) specifying the suffixes to be used for making non-by column names unique. The suffix behaviour works in a similar fashion as the merge.data.frame method does.
no.dups	logical indicating that suffixes are also appended to non-by.y column names in y when they have the same column name as any by.x.
allow.cartesian	See allow.cartesian in [.data.table.
match_type	character: one of *"m:m"*, *"m:1"*, *"1:m"*, *"1:1"*. Default is *"1:1"* since this the most restrictive. However, following Stata's recommendation, it is better to be explicit and use any of the other three match types (See details in *match types sections*).
keep_common_vars	logical: If TRUE, it will keep the original variable from y when both tables have common variable names. Thus, the prefix "y." will be added to the original name to distinguish from the resulting variable in the joined table.
...	Arguments passed on to joyn
	y_vars_to_keep character: Vector of variable names in y that will be kept after the merge. If TRUE (the default), it keeps all the brings all the variables in y into x. If FALSE or NULL, it does not bring any variable into x, but a report will be generated.
	reportvar character: Name of reporting variable. Default is ".joyn". This is the same as variable "_merge" in Stata after performing a merge. If FALSE or NULL, the reporting variable will be excluded from the final table, though a summary of the join will be display after concluding.
	update_NAs logical: If TRUE, it will update NA values of all variables in x with actual values of variables in y that have the same name as the ones in x. If FALSE, NA values won't be updated, even if update_values is TRUE

update_values logical: If TRUE, it will update all values of variables in x with the actual of variables in y with the same name as the ones in x. **NAs from y won't be used to update actual values in x**. Yet, by default, NAs in x will be updated with values in y. To avoid this, make sure to set update_NAs = FALSE

verbose logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

Value

data.table merging x and y

Examples

```
x1 = data.frame(id = c(1L, 1L, 2L, 3L, NA_integer_),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = 11:15)
y1 = data.frame(id = c(1,2, 4),
                y  = c(11L, 15L, 16))
joyn::merge(x1, y1, by = "id")
# example of using by.x and by.y
x2 = data.frame(id1 = c(1, 1, 2, 3, 3),
                id2 = c(1, 1, 2, 3, 4),
                t   = c(1L, 2L, 1L, 2L, NA_integer_),
                x   = c(16, 12, NA, NA, 15))
y2 = data.frame(id  = c(1, 2, 5, 6, 3),
                id2 = c(1, 1, 2, 3, 4),
                y   = c(11L, 15L, 20L, 13L, 10L),
                x   = c(16:20))
jn <- joyn::merge(x2,
            y2,
            match_type = "m:m",
            all.x = TRUE,
            by.x = "id1",
            by.y = "id2")
# example with all = TRUE
jn <- joyn::merge(x2,
            y2,
            match_type = "m:m",
            by.x = "id1",
            by.y = "id2",
            all = TRUE)
```

---

possible_ids *Find possible unique identifies of data frame*

Description

Identify possible combinations of variables that uniquely identifying dt

Usage

```
possible_ids(
  dt,
  vars = NULL,
  exclude = NULL,
  include = NULL,
  exclude_classes = NULL,
  include_classes = NULL,
  verbose = getOption("possible_ids.verbose", default = FALSE),
  min_combination_size = 1,
  max_combination_size = 5,
  max_processing_time = 60,
  max_numb_possible_ids = 100,
  get_all = FALSE
)
```

Arguments

dt	data frame
vars	character: A vector of variable names to consider for identifying unique combinations.
exclude	character: Names of variables to exclude from analysis
include	character: Name of variable to be included, that might belong to the group excluded in the exclude
exclude_classes	character: classes to exclude from analysis (e.g., "numeric", "integer", "date")
include_classes	character: classes to include in the analysis (e.g., "numeric", "integer", "date")
verbose	logical: If FALSE no message will be displayed. Default is TRUE
min_combination_size	numeric: Min number of combinations. Default is 1, so all combinations.
max_combination_size	numeric. Max number of combinations. Default is 5. If there is a combinations of identifiers larger than max_combination_size, they won't be found
max_processing_time	numeric: Max time to process in seconds. After that, it returns what it found.
max_numb_possible_ids	numeric: Max number of possible IDs to find. See details.
get_all	logical: get all possible combinations based on the parameters above.

Value

list with possible identifiers

Number of possible IDs

The number of possible IDs in a dataframe could be very large. This is why, `possible_ids()` makes use of heuristics to return something useful without wasting the time of the user. In addition, we provide multiple parameter so that the user can fine tune their search for possible IDs easily and quickly.

Say for instance that you have a dataframe with 10 variables. Testing every possible pair of variables will give you 90 possible unique identifiers for this dataframe. If you want to test all the possible IDs, you will have to test more 5000 combinations. If the dataframe has many rows, it may take a while.

Examples

```
library(data.table)
x4 = data.table(id1 = c(1, 1, 2, 3, 3),
                id2 = c(1, 1, 2, 3, 4),
                t   = c(1L, 2L, 1L, 2L, NA_integer_),
                x   = c(16, 12, NA, NA, 15))
possible_ids(x4)
```

---

rename_to_valid *Rename to syntactically valid names*

Description

Rename to syntactically valid names

Usage

```
rename_to_valid(name, verbose = getOption("joyn.verbose"))
```

Arguments

`name`	character: name to be coerced to syntactically valid name
`verbose`	logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

Value

valid character name

Examples

```
joyn:::rename_to_valid("x y")
```

right_join *Right join two data frames*

Description

This is a joyn wrapper that works in a similar fashion to dplyr::right_join

Usage

```
right_join(
  x,
  y,
  by = intersect(names(x), names(y)),
  copy = FALSE,
  suffix = c(".x", ".y"),
  keep = NULL,
  na_matches = c("na", "never"),
  multiple = "all",
  unmatched = "drop",
  relationship = "one-to-one",
  y_vars_to_keep = TRUE,
  update_values = FALSE,
  update_NAs = update_values,
  reportvar = getOption("joyn.reportvar"),
  reporttype = c("factor", "character", "numeric"),
  roll = NULL,
  keep_common_vars = FALSE,
  sort = TRUE,
  verbose = getOption("joyn.verbose"),
  ...
)
```

Arguments

x	data frame: referred to as *left* in R terminology, or *master* in Stata terminology.
y	data frame: referred to as *right* in R terminology, or *using* in Stata terminology.
by	a character vector of variables to join by. If NULL, the default, joyn will do a natural join, using all variables with common names across the two tables. A message lists the variables so that you can check they're correct (to suppress the message, simply explicitly list the variables that you want to join). To join by different variables on x and y use a vector of expressions. For example, by = c("a = b", "z") will use "a" in x, "b" in y, and "z" in both tables.
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.

y_vars_to_keep character: Vector of variable names in y that will be kept after the merge. If TRUE (the default), it keeps all the brings all the variables in y into x. If FALSE or NULL, it does not bring any variable into x, but a report will be generated.

update_values logical: If TRUE, it will update all values of variables in x with the actual of variables in y with the same name as the ones in x. **NAs from y won't be used to update actual values in x**. Yet, by default, NAs in x will be updated with values in y. To avoid this, make sure to set update_NAs = FALSE

update_NAs logical: If TRUE, it will update NA values of all variables in x with actual values of variables in y that have the same name as the ones in x. If FALSE, NA values won't be updated, even if update_values is TRUE

reportvar character: Name of reporting variable. Default is ".joyn". This is the same as variable "_merge" in Stata after performing a merge. If FALSE or NULL, the reporting variable will be excluded from the final table, though a summary of the join will be display after concluding.

reporttype character: One of *"character"* or *"numeric"*. Default is *"character"*. If *"numeric"*, the reporting variable will contain numeric codes of the source and the contents of each observation in the joined table. See below for more information.

roll double: *to be implemented*

keep_common_vars
logical: If TRUE, it will keep the original variable from y when both tables have common variable names. Thus, the prefix "y." will be added to the original name to distinguish from the resulting variable in the joined table.

sort logical: If TRUE, sort by key variables in by. Default is FALSE.

verbose logical: if FALSE, it won't display any message (programmer's option). Default is TRUE.

... Arguments passed on to joyn

match_type character: one of *"m:m"*, *"m:1"*, *"1:m"*, *"1:1"*. Default is *"1:1"* since this the most restrictive. However, following Stata's recommendation, it is better to be explicit and use any of the other three match types (See details in *match types sections*).

allow.cartesian logical: Check documentation in official web site. Default is NULL, which implies that if the join is "1:1" it will be FALSE, but if the join has any "m" on it, it will be converted to TRUE. By specifying TRUE of FALSE you force the behavior of the join.

suffixes A character(2) specifying the suffixes to be used for making non-by column names unique. The suffix behaviour works in a similar fashion as the base::merge method does.

yvars **[Superseded]**: use now y_vars_to_keep

keep_y_in_x **[Superseded]**: use now keep_common_vars

msg_type character: type of messages to display by default

na.last logical. If TRUE, missing values in the data are placed last; if FALSE, they are placed first; if NA they are removed. na.last=NA is valid only for x[order(., na.last)] and its default is TRUE. setorder and setorderv only accept TRUE/FALSE with default FALSE.

Value

An data frame of the same class as x. The properties of the output are as close as possible to the ones returned by the dplyr alternative.

See Also

Other dplyr alternatives: anti_join(), full_join(), inner_join(), left_join()

Examples

```
# Simple right join
library(data.table)

x1 = data.table(id = c(1L, 1L, 2L, 3L, NA_integer_),
                t  = c(1L, 2L, 1L, 2L, NA_integer_),
                x  = 11:15)
y1 = data.table(id = c(1,2, 4),
                y  = c(11L, 15L, 16))
right_join(x1, y1, relationship = "many-to-one")
```

---

set_joyn_options *Set joyn options*

Description

This function is used to change the value of one or more joyn options

Usage

```
set_joyn_options(..., env = .joynenv)
```

Arguments

...	pairs of option = value
env	environment, which is joyn environment by default

Value

joyn new options and values invisibly as a list

See Also

JOYn options functions `get_joyn_options()`

Examples

```
joyn:::set_joyn_options(joyn.verbose = FALSE, joyn.reportvar = "joyn_status")
joyn:::set_joyn_options() # return to default options
```

Index