Package 'purrrlyr'

September 12, 2025

Title Tools at the Intersection of 'purrr' and 'dplyr'

Version 0.0.10

Description Some functions at the intersection of 'dplyr' and
'purrr' that formerly lived in 'purrr'.

License GPL-3 | file LICENSE

URL https://github.com/hadley/purrrlyr

BugReports https://github.com/hadley/purrrlyr/issues

Imports dplyr (>= 0.8.0), magrittr (>= 1.5), purrr (>= 0.2.2), Rcpp

Suggests covr, testthat (>= 3.0.0)

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/build/compilation-database true

NeedsCompilation yes

Author Lionel Henry [aut, cre],
Hadley Wickham [ctb],
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Lionel Henry <lionel@posit.co>

Repository CRAN

Date/Publication 2025-09-12 05:30:11 UTC

Contents

---

by_row   *Apply a function to each row of a data frame*

Description

by_row() and invoke_rows() apply ..f to each row of .d. If ..f's output is not a data frame nor an atomic vector, a list-column is created. In all cases, by_row() and invoke_rows() create a data frame in tidy format.

Usage

```
by_row(
  .d,
  ..f,
  ...,
  .collate = c("list", "rows", "cols"),
  .to = ".out",
  .labels = TRUE
)

invoke_rows(
  .f,
  .d,
  ...,
  .collate = c("list", "rows", "cols"),
  .to = ".out",
  .labels = TRUE
)
```

Arguments

.d	A data frame.
...	Further arguments passed to ..f.
.collate	If "list", the results are returned as a list- column. Alternatively, if the results are data frames or atomic vectors, you can collate on "cols" or on "rows". Column collation require vector of equal length or data frames with same number of rows.
.to	Name of output column.
.labels	If TRUE, the returned data frame is prepended with the labels of the slices (the columns in .d used to define the slices). They are recycled to match the output size in each slice if necessary.
.f, ..f	A function to apply to each row. If ..f does not return a data frame or an atomic vector, a list-column is created under the name .out. If it returns a data frame, it should have the same number of rows within groups and the same number of columns between groups.

Details

By default, the whole row is appended to the result to serve as identifier (set `.labels` to `FALSE` to prevent this). In addition, if `..f` returns a multi-rows data frame or a non-scalar atomic vector, a `.row` column is appended to identify the row number in the original data frame.

`invoke_rows()` is intended to provide a version of `pmap()` for data frames. Its default collation method is `"cols"`, which makes it equivalent to `mdply()` from the plyr package. Note that `invoke_rows()` follows the signature pattern of the `invoke` family of functions and takes `.f` as its first argument.

The distinction between `by_row()` and `invoke_rows()` is that the former passes a data frame to `..f` while the latter maps the columns to its function call. This is essentially like using `invoke()` with each row. Another way to view this is that `invoke_rows()` is equivalent to using `by_row()` with a function lifted to accept dots (see `lift()`).

Value

A data frame.

See Also

[by_slice()](#)

Examples

```
# ..f should be able to work with a list or a data frame. As it
# happens, sum() handles data frame so the following works:
mtcars %>% by_row(sum)

# Other functions such as mean() may need to be adjusted with one
# of the lift_xy() helpers:
mtcars %>% by_row(purrr::lift_vl(mean))

# To run a function with invoke_rows(), make sure it is variadic (that
# it accepts dots) or that .f's signature is compatible with the
# column names
mtcars %>% invoke_rows(.f = sum)
mtcars %>% invoke_rows(.f = purrr::lift_vd(mean))

# invoke_rows() with cols collation is equivalent to plyr::mdply()
p <- expand.grid(mean = 1:5, sd = seq(0, 1, length = 10))
p %>% invoke_rows(.f = rnorm, n = 5, .collate = "cols")
## Not run:
p %>% plyr::mdply(rnorm, n = 5) %>% dplyr::tbl_df()

## End(Not run)

# To integrate the result as part of the data frame, use rows or
# cols collation:
mtcars[1:2] %>% by_row(function(x) 1:5)
mtcars[1:2] %>% by_row(function(x) 1:5, .collate = "rows")
mtcars[1:2] %>% by_row(function(x) 1:5, .collate = "cols")
```

---

by_slice   *Apply a function to slices of a data frame*

Description

by_slice() applies ..f on each group of a data frame. Groups should be set with slice_rows()
or dplyr::group_by().

Usage

```
by_slice(
  .d,
  ..f,
  ...,
  .collate = c("list", "rows", "cols"),
  .to = ".out",
  .labels = TRUE
)
```

Arguments

.d	A sliced data frame.
..f	A function to apply to each slice. If ..f does not return a data frame or an atomic vector, a list-column is created under the name .out. If it returns a data frame, it should have the same number of rows within groups and the same number of columns between groups.
...	Further arguments passed to ..f.
.collate	If "list", the results are returned as a list- column. Alternatively, if the results are data frames or atomic vectors, you can collate on "cols" or on "rows". Column collation require vector of equal length or data frames with same number of rows.
.to	Name of output column.
.labels	If TRUE, the returned data frame is prepended with the labels of the slices (the columns in .d used to define the slices). They are recycled to match the output size in each slice if necessary.

Details

by_slice() provides equivalent functionality to dplyr's dplyr::do() function. In combination
with map(), by_slice() is equivalent to dplyr::summarise_each() and dplyr::mutate_each().
The distinction between mutating and summarising operations is not as important as in dplyr be-
cause we do not act on the columns separately. The only constraint is that the mapped function must
return the same number of rows for each variable mapped on.

Value

A data frame.

See Also

by_row(), slice_rows(), dmap()

Examples

```
# Here we fit a regression model inside each slice defined by the
# unique values of the column "cyl". The fitted models are returned
# in a list-column.
mtcars %>%
  slice_rows("cyl") %>%
  by_slice(purrr::partial(lm, mpg ~ disp))

# by_slice() is especially useful in combination with map().

# To modify the contents of a data frame, use rows collation. Note
# that unlike dplyr, Mutating and summarising operations can be
# used indistinctly.

# Mutating operation:
df <- mtcars %>% slice_rows(c("cyl", "am"))
df %>% by_slice(dmap, ~ .x / sum(.x), .collate = "rows")

# Summarising operation:
df %>% by_slice(dmap, mean, .collate = "rows")

# Note that mapping columns within slices is best handled by dmap():
df %>% dmap(~ .x / sum(.x))
df %>% dmap(mean)

# If you don't need the slicing variables as identifiers, switch
# .labels to FALSE:
mtcars %>%
  slice_rows("cyl") %>%
  by_slice(purrr::partial(lm, mpg ~ disp), .labels = FALSE) %>%
  purrr::flatten() %>%
  purrr::map(coef)
```

---

dmap   *Map over the columns of a data frame*

Description

dmap() is just like purrr::map() but always returns a data frame. In addition, it handles grouped or sliced data frames.

Usage

```
dmap(.d, .f, ...)
```

```
dmap_at(.d, .at, .f, ...)

dmap_if(.d, .p, .f, ...)
```

Arguments

`.d`	A data frame.
`.f`	A function, specified in one of the following ways:

- A named function, e.g. `mean`.
- An anonymous function, e.g. `\(x) x + 1` or `function(x) x + 1`.
- A formula, e.g. `~ .x + 1`. You must use `.x` to refer to the first argument. No longer recommended.
- A string, integer, or list, e.g. `"idx"`, `1`, or `list("idx", 1)` which are short-hand for `\(x) pluck(x, "idx")`, `\(x) pluck(x, 1)`, and `\(x) pluck(x, "idx", 1)` respectively. Optionally supply `.default` to set a default value if the indexed element is `NULL` or does not exist.

[Experimental]

Wrap a function with [`in_parallel()`](#) to declare that it should be performed in parallel. See [`in_parallel()`](#) for more details. Use of `...` is not permitted in this context.

`...`	Additional arguments passed on to the mapped function.

We now generally recommend against using `...` to pass additional (constant) arguments to `.f`. Instead use a shorthand anonymous function:

```
# Instead of
x |> map(f, 1, 2, collapse = ",")
# do:
x |> map(\(x) f(x, 1, 2, collapse = ","))
```

This makes it easier to understand which arguments belong to which function and will tend to yield better error messages.

`.at`	A logical, integer, or character vector giving the elements to select. Alternatively, a function that takes a vector of names, and returns a logical, integer, or character vector of elements to select.

[Deprecated]: if the tidyselect package is installed, you can use `vars()` and tidyselect helpers to select elements.

`.p`	A single predicate function, a formula describing such a predicate function, or a logical vector of the same length as `.x`. Alternatively, if the elements of `.x` are themselves lists of objects, a string indicating the name of a logical element in the inner lists. Only those elements where `.p` evaluates to `TRUE` will be modified.

Details

`dmap_at()` and `dmap_if()` recycle length 1 vectors to the group sizes.

Examples

```
# dmap() always returns a data frame:
dmap(mtcars, summary)

# dmap() also supports sliced data frames:
sliced_df <- mtcars[1:5] %>% slice_rows("cyl")
sliced_df %>% dmap(mean)
sliced_df %>% dmap(~ .x / max(.x))

# This is equivalent to the combination of by_slice() and dmap()
# with 'rows' collation of results:
sliced_df %>% by_slice(dmap, mean, .collate = "rows")
```

---

slice_rows	*Slice a data frame into groups of rows*

Description

slice_rows() is equivalent to dplyr's [dplyr::group_by()](#) command but it takes a vector of column names or positions instead of capturing column names with special evaluation. unslice() removes the slicing attributes.

Usage

```
slice_rows(.d, .cols = NULL)

unslice(.d)
```

Arguments

.d	A data frame to slice or unslice.
.cols	A character vector of column names or a numeric vector of column positions. If NULL, the slicing attributes are removed.

Value

A sliced or unsliced data frame.

See Also

[by_slice()](#) and [dplyr::group_by()](#)

Index