Package ‘crossmap’

January 12, 2023

Title  Apply Functions to All Combinations of List Elements

Version 0.4.0

Description  Provides an extension to the 'purrr' family of mapping functions to apply a function to each combination of elements in a list of inputs. Also includes functions for automatically detecting output type in mapping functions, finding every combination of elements of lists or rows of data frames, and applying multiple models to multiple subsets of a dataset.

License  MIT + file LICENSE

URL  https://crossmap.rossellhayes.com,
     https://github.com/rossellhayes/crossmap

BugReports  https://github.com/rossellhayes/crossmap/issues

Imports  backports, cli, dplyr (>= 1.0.0), generics, lifecycle, purrr, rlang, stats, utils, vctrs

Suggests  broom, covr, crayon, estimatr, furrr, future, testthat, tibble, withr

Encoding  UTF-8

RoxygenNote  7.2.3

NeedsCompilation  no

Author  Alexander Rossell Hayes [aut, cre, cph]
        (<https://orcid.org/0000-0001-9412-0457>)

Maintainer  Alexander Rossell Hayes <alexander@rossellhayes.com>

Repository  CRAN

Date/Publication  2023-01-12 21:50:02 UTC

R topics documented:

 autonames ................................................................. 2
 cross_fit ................................................................. 3
 cross_fit_glm .......................................................... 5
autonames

Automatically generate names for vectors

Description

Automatically generate names for vectors

Usage

autonames(x, ..., trimws = TRUE)

Arguments

x A vector

... Additional arguments passed to format()

trimws Whether to trim whitespace surrounding automatically formatted names. Defaults to TRUE.

Value

Returns the names of a named vector and the elements of an unnamed vector formatted as characters.

Examples

autonames(c(a = "apple", b = "banana", c = "cantaloupe"))
autonames(c("apple", "banana", "cantaloupe"))

autonames(10^(1:4))
autonames(10^(1:4), big.mark = ",")
autonames(10^(1:4), scientific = TRUE)
cross_fit

Cross map a model across multiple formulas, subsets, and weights

Description
Applies a modeling function to every combination of a set of formulas and a set of data subsets.

Usage
cross_fit(
data,  
formulas,  
cols = NULL,  
weights = NULL,  
clusters = NULL,  
families = NULL,  
fn = lm,  
fn_args = list(),  
tidy = tidy_glance,  
tidy_args = list(),  
errors = c("stop", "warn")
)

Arguments
data A data frame
formulas A list of formulas to apply to each subset of the data. If named, these names will be used in the model column of the output. Otherwise, the formulas will be converted to strings in the model column.
cols Columns to subset the data. Can be any expression supported by <tidy-select>. If NULL, the data is not subset into columns. Defaults to NULL.
weights A list of columns passed to weights in fn. If one of the elements is NULL or NA, that model will not be weighted. Defaults to NULL.
clusters A list of columns passed to clusters if supported by fn. If one of the elements is NULL or NA, that model will not be clustered. Defaults to NULL.
families A list of glm model families passed to family if supported by fn. Defaults to gaussian("identity"), the equivalent of lm(). See family for examples.
fn The modeling function. Either an unquoted function name or a purrr-style lambda function with two arguments. To use multiple modeling functions, see cross_fit_glm(). Defaults to lm.
fn_args A list of additional arguments to fn.
tidy A logical or function to use to tidy model output into data.frame columns. If TRUE, uses the default tidying function: tidy_glance(). If FALSE, NA, or NULL, the untidied model output will be returned in a list column named fit. An
alternative function can be specified with an unquoted function name or a purrr-style lambda function with one argument (see usage with broom::tidy(conf.int = TRUE) in examples). Defaults to tidy_glance.

tidy_args A list of additional arguments to the tidy function
errors If "stop", the default, the function will stop and return an error if any subset produces an error. If "warn", the function will produce a warning for subsets that produce an error and return results for all subsets that do not.

Value
A tibble with a column for the model formula, columns for subsets, columns for the model family and type (if applicable), columns for the weights and clusters (if applicable), and columns of tidy model output or a list column of models (if tidy = FALSE)

See Also
cross_fit_glm() to map a model across multiple model types.
cross_fit_robust() to map robust linear models.
xmap() to apply any function to combinations of inputs.

Examples

cross_fit(mtcars, mpg ~ wt, cyl)
cross_fit(mtcars, list(mpg ~ wt, mpg ~ hp), cyl)
cross_fit(mtcars, list(wt = mpg ~ wt, hp = mpg ~ hp), cyl)

cross_fit(mtcars, list(mpg ~ wt, mpg ~ hp), c(cyl, vs))
cross_fit(mtcars, list(mpg ~ wt, mpg ~ hp), dplyr::starts_with("c"))

cross_fit(mtcars, list(hp = mpg ~ hp), cyl, weights = wt)
cross_fit(mtcars, list(hp = mpg ~ hp), cyl, weights = c(wt, NA))

cross_fit(
  mtcars, list(vs ~ cyl, vs ~ hp), am,
  fn = glm, fn_args = list(family = binomial(link = logit))
)
cross_fit(
  mtcars, list(vs ~ cyl, vs ~ hp), am,
  fn = ~ glm(.x, .y, family = binomial(link = logit))
)

cross_fit(mtcars, list(mpg ~ wt, mpg ~ hp), cyl, tidy = FALSE)
cross_fit(mtcars, list(mpg ~ wt, mpg ~ hp), cyl, tidy_args = c(conf.int = TRUE))

cross_fit(mtcars, list(mpg ~ wt, mpg ~ hp), cyl, tidy = broom::tidy)
cross_fit(
  mtcars, list(mpg ~ wt, mpg ~ hp), cyl,
  tidy = ~ broom::tidy(.), conf.int = TRUE)
Cross fit generalized linear models

Description

Cross fit generalized linear models

Usage

cross_fit_glm(
  data,
  formulas,
  cols = NULL,
  weights = NULL,
  families = gaussian(link = identity),
  fn_args = list(),
  tidy = tidy_glance,
  tidy_args = list(),
  errors = c("stop", "warn")
)

Arguments

data  A data frame
formulas  A list of formulas to apply to each subset of the data. If named, these names will be used in the model column of the output. Otherwise, the formulas will be converted to strings in the model column.
cols  Columns to subset the data. Can be any expression supported by <tidy-select>. If NULL, the data is not subset into columns. Defaults to NULL.
weights  A list of columns passed to weights in fn. If one of the elements is NULL or NA, that model will not be weighted. Defaults to NULL.
families  A list of glm model families. Defaults to gaussian("identity"), the equivalent of lm(). See family for examples.
fn_args  A list of additional arguments to glm().
tidy  A logical or function to use to tidy model output into data.frame columns. If TRUE, uses the default tidying function: tidy_glance(). If FALSE, NA, or NULL, the untidied model output will be returned in a list column named fit. An alternative function can be specified with an unquoted function name or a purrr-style lambda function with one argument (see usage with broom::tidy(conf.int = TRUE) in examples). Defaults to tidy_glance.
tidy_args  A list of additional arguments to the tidy function
errors  If "stop", the default, the function will stop and return an error if any subset produces an error. If "warn", the function will produce a warning for subsets that produce an error and return results for all subsets that do not.
Value
A tibble with a column for the model formula, columns for subsets, columns for the model family and type, columns for the weights (if applicable), and columns of tidy model output or a list column of models (if tidy = FALSE)

See Also
cross_fit() to use any modeling function.

Examples
cross_fit_glm(
  data = mtcars,
  formulas = list(am ~ gear, am ~ cyl),
  cols = vs,
  families = list(gaussian("identity"), binomial("logit"))
)

cross_fit_robust
Cross fit robust linear models

Description
Cross fit robust linear models

Usage
cross_fit_robust(
  data,
  formulas,
  cols = NULL,
  weights = NULL,
  clusters = NULL,
  fn_args = list(),
  tidy = tidy_glance,
  tidy_args = list(),
  errors = c("stop", "warn")
)

Arguments
data A data frame
formulas A list of formulas to apply to each subset of the data. If named, these names will be used in the model column of the output. Otherwise, the formulas will be converted to strings in the model column.
cols Columns to subset the data. Can be any expression supported by <tidy-select>. If NULL, the data is not subset into columns. Defaults to NULL.
weights A list of columns passed to weights in fn. If one of the elements is NULL or NA, that model will not be weighted. Defaults to NULL.

clusters A list of columns passed to clusters. If one of the elements is NULL or NA, that model will not be clustered. Defaults to NULL.

fn_args A list of additional arguments to estimatr::lm_robust().

tidy A logical or function to use to tidy model output into data.frame columns. If TRUE, uses the default tidying function: tidy_glance(). If FALSE, NA, or NULL, the untidied model output will be returned in a list column named fit. An alternative function can be specified with an unquoted function name or a purrr-style lambda function with one argument (see usage with broom::tidy(conf.int = TRUE) in examples). Defaults to tidy_glance.

tidy_args A list of additional arguments to the tidy function

errors If "stop", the default, the function will stop and return an error if any subset produces an error. If "warn", the function will produce a warning for subsets that produce an error and return results for all subsets that do not.

Value A tibble with a column for the model formula, columns for subsets, columns for the weights and clusters (if applicable), and columns of tidy model output or a list column of models (if tidy = FALSE)

See Also cross_fit() to use any modeling function.

Examples

cross_fit_robust(mtcars, mpg ~ wt, cyl, clusters = list(NULL, am))

cross_join

Description

Crossing join

Usage
cross_join(., copy = FALSE)
Arguments

... Data frames or a list of data frames – including data frame extensions (e.g. tibbles) and lazy data frames (e.g. from dbplyr or dplyr). NULL inputs are silently ignored.

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

Value

An object of the same type as the first input. The order of the rows and columns of the first input is preserved as much as possible. The output has the following properties:

- Rows from each input will be duplicated.
- Output columns include all columns from each input. If columns have the same name, suffixes are added to disambiguate.
- Groups are taken from the first input.

See Also
cross_list() to find combinations of elements of vectors and lists.

Examples

```r
fruits <- dplyr::tibble(
  fruit = c("apple", "banana", "cantaloupe"),
  color = c("red", "yellow", "orange")
)

desserts <- dplyr::tibble(
  dessert = c("cupcake", "muffin", "streudel"),
  makes = c(8, 6, 1)
)
cross_join(fruits, desserts)
cross_join(list(fruits, desserts))
cross_join(rep(list(fruits), 3))
```

---

cross_list List all combinations of values

Description

List all combinations of values
future_map_vec

Usage

cross_list(...)
cross_tbl(...)

Arguments

... Inputs or a list of inputs. NULL inputs are silently ignored.

Value

A list for cross_list() or tibble for cross_tbl(). Names will match the names of the inputs. Unnamed inputs will be left unnamed for cross_list() and automatically named for cross_tbl().

See Also

cross_join() to find combinations of data frame rows.
purrr::cross() for an implementation that results in a differently formatted list.
expand.grid() for an implementation that results in a data.frame.

Examples

fruits <- c("apple", "banana", "cantaloupe")
desserts <- c("cupcake", "muffin", "streudel")
cross_list(list(fruits, desserts))
cross_list(fruits, desserts)
cross_tbl(fruits, desserts)
cross_list(list(fruit = fruits, dessert = desserts))
cross_list(fruit = fruits, dessert = desserts)
cross_tbl(fruit = fruits, dessert = desserts)

future_map_vec Parallelized mapping functions that automatically determine type

Description

These functions work exactly the same as map_vec(), map2_vec(), pmap_vec(), imap_vec() and xmap_vec(), but allow you to map in parallel.
future_map_vec

 Usage

```r
future_map_vec(
  .x,
  .f,
  ..., 
  .class = NULL,
  .progress = FALSE,
  .options = furrr::furrr_options()
)
```

```r
future_map2_vec(
  .x,
  .y,
  .f,
  ..., 
  .class = NULL,
  .progress = FALSE,
  .options = furrr::furrr_options()
)
```

```r
future_pmap_vec(
  .l,
  .f,
  ..., 
  .class = NULL,
  .progress = FALSE,
  .options = furrr::furrr_options()
)
```

```r
future_imap_vec(
  .x,
  .f,
  ..., 
  .class = NULL,
  .progress = FALSE,
  .options = furrr::furrr_options()
)
```

```r
future_xmap_vec(
  .l,
  .f,
  ..., 
  .class = NULL,
  .progress = FALSE,
  .options = furrr::furrr_options()
)
```
Arguments

.x  A list or atomic vector.
.f  A function, formula, or vector (not necessarily atomic).

If a function, it is used as is.
If a formula, e.g. ~ .x + 2, it is converted to a function. There are three ways to refer to the arguments:

• For a single argument function, use .
• For a two argument function, use .x and .y
• For more arguments, use ..1, ..2, ..3 etc

This syntax allows you to create very compact anonymous functions.

If character vector, numeric vector, or list, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. If a component is not present, the value of .default will be returned.

... Additional arguments passed on to .f
.class  If .class is specified, all
.progress  A single logical. Should a progress bar be displayed? Only works with multisession, multicore, and multiprocess futures. Note that if a multicore/multisession future falls back to sequential, then a progress bar will not be displayed.

Warning: The .progress argument will be deprecated and removed in a future version of furrr in favor of using the more robust progressr package.

.options  The future specific options to use with the workers. This must be the result from a call to furrr_options().

.y  A vector the same length as .x. Vectors of length 1 will be recycled.
.l  A list of vectors, such as a data frame. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.

Value

Equivalent to map_vec(), map2_vec(), pmap_vec(), imap_vec() and xmap_vec()

See Also

The original functions: furrr::future_map(), furrr::future_map2(), furrr::future_pmap(), furrr::future_imap() and future_xmap()

Non-parallelized equivalents: map_vec(), map2_vec(), pmap_vec(), imap_vec() and xmap_vec()

Examples

fruits <- c("apple", "banana", "carrot", "durian", "eggplant")
desserts <- c("bread", "cake", "cupcake", "streudel", "muffin")
x <- sample(5)
y <- sample(5)
z <- sample(5)
names(z) <- fruits
```
future_map_vec(x, ^ 2)
future_map_vec(fruits, paste0, "s")

future_map2_vec(x, y, ~ .x + .y)
future_map2_vec(fruits, desserts, paste)

future_pmap_vec(list(x, y, z), sum)
future_pmap_vec(list(x, fruits, desserts), paste)

future imap_vec(x, ~ .x + .y)
future imap_vec(x, ~ paste0(.y, ": ", .x))
future imap_vec(z, paste)

future_xmap_vec(list(x, y), ~ .x * .y)
future_xmap_vec(list(fruits, desserts), paste)
```

---

**future_xmap**

*Map over each combination of list elements simultaneously via futures*

---

**Description**

These functions work exactly the same as `xmap()` functions, but allow you to run the map in parallel using `future::future()`

**Usage**

future_xmap(.l, .f, ..., .progress = FALSE, .options = furrr::furrr_options())

future_xmap_chr(
  .l,
  .f,
  ..., 
  .progress = FALSE,
  .options = furrr::furrr_options()
)

future_xmap_dbl(
  .l,
  .f,
  ..., 
  .progress = FALSE,
  .options = furrr::furrr_options()
)

future_xmap_dfc(
  .l,
  .f,
future_xmap

...,
.progress = FALSE,
.options = furrr::furrr_options()
)

future_xmap_dfr(
  .l,
  .f,
  ...,
  .id = NULL,
  .progress = FALSE,
  .options = furrr::furrr_options()
)

future_xmap_int(
  .l,
  .f,
  ...,
  .progress = FALSE,
  .options = furrr::furrr_options()
)

future_xmap_lgl(
  .l,
  .f,
  ...,
  .progress = FALSE,
  .options = furrr::furrr_options()
)

future_xmap_raw(
  .l,
  .f,
  ...,
  .progress = FALSE,
  .options = furrr::furrr_options()
)

future_xwalk(.l, .f, ..., .progress = FALSE, .options = furrr::furrr_options())

Arguments

.l

A list of vectors, such as a data frame. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.

.f

A function, formula, or vector (not necessarily atomic).

If a function, it is used as is.

If a formula, e.g. ~ .x + 2, it is converted to a function. There are three ways to refer to the arguments:
• For a single argument function, use .
• For a two argument function, use .x and .y
• For more arguments, use ..1, ..2, ..3 etc

This syntax allows you to create very compact anonymous functions.

If character vector, numeric vector, or list, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. If a component is not present, the value of .default will be returned.

... Additional arguments passed on to .f

.progress A single logical. Should a progress bar be displayed? Only works with multisession, multicore, and multiprocess futures. Note that if a multicore/multisession future falls back to sequential, then a progress bar will not be displayed.

Warning: The .progress argument will be deprecated and removed in a future version of furrr in favor of using the more robust progressr package.

.options The future specific options to use with the workers. This must be the result from a call to furrr_options().

.id Either a string or NULL. If a string, the output will contain a variable with that name, storing either the name (if .x is named) or the index (if .x is unnamed) of the input. If NULL, the default, no variable will be created.

Only applies to _dfr variant.

Value

An atomic vector, list, or data frame, depending on the suffix. Atomic vectors and lists will be named if the first element of .l is named.

If all input is length 0, the output will be length 0. If any input is length 1, it will be recycled to the length of the longest.

See Also

xmap() to run functions without parallel processing.
future_xmap_vec() to automatically determine output type.
future_xmap_mat() and future_xmap_arr() to return results in a matrix or array.
furr::future_map(), furrr::future_map2(), and furrr::future_pmap() for other parallelized mapping functions.

Examples

future_xmap(list(1:5, 1:5), ~ .y * .x)
future_xmap_dbl(list(1:5, 1:5), ~ .y * .x)
future_xmap_chr(list(1:5, 1:5), ~ paste(.y, "\n", .x, "\n", .y * .x))

apples_and_bananas <- list(
  x = c("apples", "bananas"),
  pattern = "a",
  replacement = c("oo", "ee"))
future_xmap_mat

future_xmap_chr(apples_and_bananas, gsub)

formulas <- list(mpg ~ wt, mpg ~ hp)
subsets  <- split(mtcars, mtcars$cyl)

future_xmap(list(subsets, formulas), ~ lm(.y, data = .x))

---

**future_xmap_mat**

Parallelized cross map returning a matrix or array

**Description**

Parallelized cross map returning a matrix or array

**Usage**

future_xmap_mat(
  .l,
  .f,
  ...
)

future_xmap_arr(
  .l,
  .f,
  ...
)

**Arguments**

.l  A list of vectors, such as a data frame. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.

.f  A function, formula, or vector (not necessarily atomic).

If a **function**, it is used as is.

If a **formula**, e.g. ~ .x + 2, it is converted to a function. There are three ways to refer to the arguments:

- For a single argument function, use .
- For a two argument function, use .x and .y
- For more arguments, use .1, .2, .3 etc
map_vec

This syntax allows you to create very compact anonymous functions.

If character vector, numeric vector, or list, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. If a component is not present, the value of .default will be returned.

... Additional arguments passed on to .f
.names A logical indicating whether to give names to the dimensions of the matrix or array. If inputs are named, the names are used. If inputs are unnamed, the elements of the input are used as names. Defaults to TRUE.
.progress A single logical. Should a progress bar be displayed? Only works with multisession, multicore, and multiprocess futures. Note that if a multicore/multisession future falls back to sequential, then a progress bar will not be displayed.

Warning: The .progress argument will be deprecated and removed in a future version of furrr in favor of using the more robust progressr package.
.options The future specific options to use with the workers. This must be the result from a call to furrr_options().

Value
A matrix (for future_xmap_mat()) or array (for future_xmap_arr()) with dimensions matching the lengths of each input in .l.

See Also
Unparallelized versions: xmap_mat() and xmap_arr()
future_xmap_vec() to return a vector.
future_xmap() for the underlying functions.

Examples
future_xmap_mat(list(1:3, 1:3), ~ ..1 * ..2)
fruits <- c(a = "apple", b = "banana", c = "cantaloupe")
future_xmap_mat(list(1:3, fruits), paste)
future_xmap_mat(list(1:3, fruits), paste, .names = FALSE)
future_xmap_arr(list(1:3, 1:3, 1:3), ~ ..1 * ..2 * ..3)

map_vec Mapping functions that automatically determine type

Description
These functions work exactly the same as typed variants of purrr::map(), purrr::map2(), purrr::pmap(), purrr::imap() and xmap() (e.g. purrr::map_chr()), but automatically determine the type.
Usage

map_vec(.x, .f, ..., .class = NULL)
map2_vec(.x, .y, .f, ..., .class = NULL)
pmap_vec(.l, .f, ..., .class = NULL)
imap_vec(.x, .f, ..., .class = NULL)
xmap_vec(.l, .f, ..., .class = NULL)

Arguments

.x A list or atomic vector.
.f A function, formula, or vector (not necessarily atomic).
   If a function, it is used as is.
   If a formula, e.g. ~ .x + 2, it is converted to a function. There are three ways to refer to the
   arguments:
   • For a single argument function, use .
   • For a two argument function, use .x and .y
   • For more arguments, use ..1, ..2, ..3 etc
   This syntax allows you to create very compact anonymous functions.

... Additional arguments passed on to .f
.class If .class is specified, all
.y A vector the same length as .x. Vectors of length 1 will be recycled.
.l A list of vectors, such as a data frame. The length of .l determines the number of arguments
   that .f will be called with. List names will be used if present.

Value

Equivalent to the typed variants of purrr::map(), purrr::map2(), purrr::pmap(), purrr::imap()
and xmap() with the type automatically determined.

If the output contains multiple types, the type is determined from the highest type of the components
in the hierarchy raw < logical < integer < double < complex < character < list (as in c()).
If the output contains elements that cannot be coerced to vectors (e.g. lists), the output will be a list.

See Also

The original functions: purrr::map(), purrr::map2(), purrr::pmap(), purrr::imap() and
xmap()
Parallelized equivalents: future_map_vec(), future_map2_vec(), future_pmap_vec(), future_imap_vec()
and future_xmap_vec()
Examples

```r
fruits <- c("apple", "banana", "cantaloupe", "durian", "eggplant")
desserts <- c("bread", "cake", "cupcake", "muffin", "streudel")
x <- sample(5)
y <- sample(5)
z <- sample(5)
names(z) <- fruits

map_vec(x, ~ . ^ 2)
map_vec(fruits, paste0, "s")

map2_vec(x, y, ~ .x + .y)
map2_vec(fruits, desserts, paste)

pmap_vec(list(x, y, z), sum)
pmap_vec(list(x, fruits, desserts), paste)

imap_vec(x, ~ .x + .y)
imap_vec(x, ~ paste0(.y, ": ", .x))
imap_vec(z, paste)

xmap_vec(list(x, y), ~ .x * .y)
xmap_vec(list(fruits, desserts), paste)
```

---

**tidy_glance**  
*Turn an object into a tidy tibble with glance information*

Description

Apply both `generics::tidy()` and `generics::glance()` to an object and return a single tibble with both sets of information.

Usage

```r
tidy_glance(x, ..., tidy_args = list(), glance_args = list())
```

Arguments

- `x`  
  An object to be converted into a tidy tibble.
- `...`  
  Additional arguments passed to `generics::tidy()` and `generics::glance()`. Arguments are passed to both methods, but should be ignored by the inapplicable method. For example, if called on a `lm` object, `conf.int` will affect `generics::tidy()` but not `generics::glance()`.
- `tidy_args`  
  A list of additional arguments passed only to `generics::tidy()`.
- `glance_args`  
  A list of additional arguments passed only to `generics::glance()`.
Value

A tibble with columns and rows from generics::tidy() and columns of repeated rows from generics::glance().

Column names that appear in both the tidy data and glance data will be disambiguated by appending "model." to the glance column names.

Examples

```r
mod <- lm(mpg ~ wt + qsec, data = mtcars)
tidy_glance(mod)
tidy_glance(mod, conf.int = TRUE)
tidy_glance(mod, tidy_args = list(conf.int = TRUE))
```

xmap

Map over each combination of list elements

Description

These functions are variants of purrr::pmap() that iterate over each combination of elements in a list.

Usage

xmap(.l, .f, ...)

xmap_chr(.l, .f, ...)

xmap_dbl(.l, .f, ...)

xmap_dfc(.l, .f, ...)

xmap_dfr(.l, .f, ..., .id = NULL)

xmap_int(.l, .f, ...)

xmap_lgl(.l, .f, ...)

xmap_raw(.l, .f, ...)

xwalk(.l, .f, ...)

Arguments

.l A list of vectors, such as a data frame. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.
A function, formula, or vector (not necessarily atomic).

If a **function**, it is used as is.

If a **formula**, e.g., $\sim .x + 2$, it is converted to a function. There are three ways to refer to the arguments:

- For a single argument function, use `.`
- For a two argument function, use `.x` and `.y`
- For more arguments, use `.1`, `.2`, `.3` etc

This syntax allows you to create very compact anonymous functions.

If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. If a component is not present, the value of `.default` will be returned.

... Additional arguments passed on to `.f`

**.id** Either a string or NULL. If a string, the output will contain a variable with that name, storing either the name (if `.x` is named) or the index (if `.x` is unnamed) of the input. If NULL, the default, no variable will be created.

Only applies to `_dfr` variant.

**Details**

Typed variants return a vector of the specified type. To automatically determine type, try `xmap_vec()`.

To return results as a matrix or array, try `xmap_mat()` and `xmap_arr()`.

Note that a data frame is a very important special case, in which case `xmap()` and `xwalk()` apply the function `.f` to each row. `xmap_dfr()` and `xmap_dfcc()` return data frames created by row-binding and column-binding respectively.

**Value**

An atomic vector, list, or data frame, depending on the suffix. Atomic vectors and lists will be named if the first element of `.l` is named.

If all input is length 0, the output will be length 0. If any input is length 1, it will be recycled to the length of the longest.

**See Also**

`xmap_vec()` to automatically determine output type.
`xmap_mat()` and `xmap_arr()` to return results in a matrix or array.
`future_xmap()` to run `xmap` functions with parallel processing.
`cross_fit()` to apply multiple models to multiple subsets of data.
`cross_list()` to find combinations of list elements.
`purrr::map()`, `purrr::map2()`, and `purrr::pmap()` for other mapping functions.
Examples

\[
\begin{align*}
\text{xmap} & \left( \text{list}(1:5, 1:5), \sim .y \ast .x \right) \\
\text{xmap dbl} & \left( \text{list}(1:5, 1:5), \sim .y \ast .x \right) \\
\text{xmap chr} & \left( \text{list}(1:5, 1:5), \sim \text{paste}(y, "\ast", .x, "\ast", .y \ast .x) \right)
\end{align*}
\]

\[
\text{apples_and_bananas} <- \text{list}(
  \begin{align*}
  & x = c("apples", "bananas") \\
  & \text{pattern} = "a", \\
  & \text{replacement} = c("oo", "ee")
\end{align*}
) \]

\[
\text{xmap chr} (\text{apples_and_bananas}, \text{gsub})
\]

\[
\text{formulas} <- \text{list} (\text{mpg} \sim \text{wt}, \text{mpg} \sim \text{hp}) \\
\text{subsets} <- \text{split} (\text{mtcars}, \text{mtcars$cyl})
\]

\[
\text{xmap} (\text{list} (\text{subsets, formulas}), \sim \text{lm}(y, \text{data} = .x)) \\
\text{xmap} (\text{list} (\text{data} = \text{subsets}, \text{formula} = \text{formulas}), \text{lm})
\]

---

**xmap_mat**

*Return a table applying a function to all combinations of list elements*

Description

Return a table applying a function to all combinations of list elements

Usage

\[
\text{xmap mat} (.l, .f, ..., .names = \text{TRUE})
\]

\[
\text{xmap arr} (.l, .f, ..., .names = \text{TRUE})
\]

Arguments

- `.l` A list of vectors, such as a data frame. The length of `.l` determines the number of arguments that `.f` will be called with. List names will be used if present.
- `.f` A function, formula, or vector (not necessarily atomic). If a **function**, it is used as is. If a **formula**, e.g. \( \sim .x + 2 \), it is converted to a function. There are three ways to refer to the arguments:
  - For a single argument function, use `.x`.
  - For a two argument function, use `.x` and `.y`
  - For more arguments, use `.x, .y` etc
This syntax allows you to create very compact anonymous functions.
- **Character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. If a component is not present, the value of `.default` will be returned.
xpluck

... Additional arguments passed on to .f
.names A logical indicating whether to give names to the dimensions of the matrix or array. If inputs are named, the names are used. If inputs are unnamed, the elements of the input are used as names. Defaults to TRUE.

Value
A matrix (for xmap_mat()) or array (for xmap_arr()) with dimensions equal to the lengths of each input in .l.

See Also
future_xmap_mat() and future_xmap_arr() to run functions in parallel.
xmap_vec() to return a vector.
xmap() for the underlying functions.

Examples
xmap_mat(list(1:3, 1:3), ~ ..1 * ..2)
fruits <- c(a = "apple", b = "banana", c = "cantaloupe")
xmap_mat(list(1:3, fruits), paste)
xmap_mat(list(1:3, fruits), paste, .names = FALSE)
xmap_arr(list(1:3, 1:3, 1:3), ~ ..1 * ..2 * ..3)

---
xpluck

Get one or more elements deep within a nested data structure

Description
xpluck() provides an alternative to purrr::pluck(). Unlike purrr::pluck(), xpluck() allows you to extract multiple indices at each nesting level.

Usage
xpluck(.x, ..., .default = NULL)

Arguments
.x A list or vector
... A list of accessors for indexing into the object. Can be positive integers, negative integers (to index from the right), strings (to index into names) or missing (to keep all elements at a given level).
Unlike purrr::pluck(), each accessor may be a vector to extract multiple elements.
.default Value to use if target is NULL or absent.
xpluck

Value

A list or vector.

Examples

```
obj1 <- list("a", list(1, elt = "foo"))
obj2 <- list("b", list(2, elt = "bar"))
x <- list(obj1, obj2)

xpluck(x, 1:2, 2)
xpluck(x, , 2)

xpluck(x, , 2, 1)
xpluck(x, , 2, 2)
xpluck(x, , 2, 1:2)
```
Index

autonames, 2
broom::tidy(conf.int = TRUE), 4, 5, 7
c(), 17
cross_fit(), 6, 7, 20
future::future(), 11, 14
future::future_map(), 11, 14
future::future_pmap(), 11, 14
future::future_options(), 11, 14, 16
future::future(), 12
future::future_imap_vec(), 9
future::future_map2_vec(), 17
future::future_map(), 17
future::future_map2_vec(), 17
future::future_map(), 9
future::future_map_vec(), 17
future::future_pmap_vec(), 17
future::future_xmap(), 12
future::future_xmap(), 11, 16, 20
future::future_xmap_arr (future_xmap_mat), 15
future::future_xmap_arr(), 14, 22
future::future_xmap_chr (future_xmap), 12
future::future_xmap_dbl (future_xmap), 12
future::future_xmap_dfc (future_xmap), 12
future::future_xmap_dfr (future_xmap), 12
future::future_xmap_int (future_xmap), 12
future::future_xmap_lgl (future_xmap), 12
future::future_xmap_mat, 15
future::future_xmap_mat(), 14, 22
future::future_xmap_raw (future_xmap), 12
future::future_xmap_vec (future_map_vec), 9
future::future_xmap_vec(), 14, 16, 17
future::future_xwalk (future_xmap), 12
gaussian(identity), 3, 5
generics::glance(), 18, 19
generics::tidy(), 18, 19
glm, 3, 5
glm(), 5
imap_vec (map_vec), 16
imap_vec(), 9, 11
integer, 17
list, 8, 9, 17, 22, 23
lm, 3, 18
lm(), 3, 5
logical, 17
map2_vec (map_vec), 16
map2_vec(), 9, 11
map_vec, 16
map_vec(), 9, 11

NA, 3, 5, 7
NULL, 3, 5–9, 22
INDEX

pmap_vec (map_vec), 16
pmap_vec(), 9, 11
purrr, 3–5, 7
purrr::cross(), 9
purrr::imap(), 16, 17
purrr::map(), 16, 17, 20
purrr::map2(), 16, 17, 20
purrr::map_chr(), 16
purrr::pluck(), 22
purrr::pmap(), 16, 17, 19, 20
raw, 17
tibble, 9, 18, 19
tibbles, 8
tidy_glance, 4, 5, 7, 18
tidy_glance(), 3, 5, 7
vector, 22, 23
xmap, 19
xmap(), 4, 12, 14, 16, 17, 22
xmap_arr (xmap_mat), 21
xmap_arr(), 16, 20
xmap_chr (xmap), 19
xmap_dbl (xmap), 19
xmap_dfc (xmap), 19
xmap_dfr (xmap), 19
xmap_int (xmap), 19
xmap_lgl (xmap), 19
xmap_mat, 21
xmap_mat(), 16, 20
xmap_raw (xmap), 19
xmap_vec (map_vec), 16
xmap_vec(), 9, 11, 20, 22
xpluck(22
xwalk (xmap), 19