



- Committed to this repository
- Member of RStudio, and 1 more

Go to file



garrettgman Fixes #93.

Latest commit b06d4ba on 7 Aug 2019  History

1 contributor

1.36 MB

Download



Apply functions with purrr : : CHEAT SHEET



Apply Functions

Map functions apply a function iteratively to each element of a list or vector.

`map(fun, ...)` → `fun(...)` → `map(x, f, ...)` Apply a function to each element of a list or vector. `map(x, is.logical)`

`map2(fun, ...)` → `fun(...)` → `map2(x, y, f, ...)` Apply a function to pairs of elements from two lists, vectors. `map2(x, y, sum)`

`pmap(fun, ...)` → `fun(...)` → `pmap(list(x, y, z), sum, na.rm = TRUE)` Apply a function to groups of elements from list of lists, vectors. `pmap(list(x, y, z), sum, na.rm = TRUE)`

`invoke_map(f, x = list(NULL), ..., env=NULL)` Run each function in a list. Also `invoke`. `l <- list(var, sd); invoke_map(l, x = 1:9)`

`lmap(x, f, ...)` Apply function to each list-element of a list or vector. `imap(x, f, ...)` Apply `f` to each element of a list or vector and its index.

OUTPUT

`map()`, `map2()`, `pmap()`, `imap` and `invoke_map` each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. `map2_chr`, `pmap_lgl`, etc.

Use `walk`, `walk2`, and `pwalk` to trigger side effects. Each return its input invisibly.

function	returns
<code>map</code>	list
<code>map_chr</code>	character vector
<code>map_dbl</code>	double (numeric) vector
<code>map_dfc</code>	data frame (column bind)
<code>map_dfr</code>	data frame (row bind)
<code>map_int</code>	integer vector
<code>map_lgl</code>	logical vector
<code>walk</code>	triggers side effects, returns the input invisibly

SHORTCUTS - within a purrr function:

"name" becomes `function(x) x[["name"]]`, e.g. `map(l, "a")` extracts `a` from each element of `l`

`~x.y` becomes `function(x, y) x.y`, e.g. `map2(l, p, ~x+y)` becomes `map2(l, p, function(l, p) l + p)`

`~.x` becomes `function(x) x`, e.g. `map(l, ~2 + x)` becomes `map(l, function(x) 2 + x)`

`~..1..2` etc becomes `function(..1, ..2, etc) ..1 ..2 etc`, e.g. `pmap(list(a, b, c), ~..3 + ..1 - ..2)` becomes `pmap(list(a, b, c), function(a, b, c) c + a - b)`

Work with Lists

FILTER LISTS

`pluck(x, ..., .default=NULL)` Select an element by name or index, `pluck(x, "b")`, or its attribute with `attr_getter`. `pluck(x, "b", attr_getter("n"))`

`keep(x, p, ...)` Select elements that pass a logical test. `keep(x, is.na)`

`discard(x, p, ...)` Select elements that do not pass a logical test. `discard(x, is.na)`

`compact(x, p = identity)` Drop empty elements. `compact(x)`

`head_while(x, p, ...)` Return head elements until one does not pass. Also `tail_while`. `head_while(x, is.character)`

RESHAPE LISTS

`flatten(x)` Remove a level of indexes from a list. Also `flatten_chr`, `flatten_dbl`, `flatten_dfc`, `flatten_dfr`, `flatten_int`, `flatten_lgl`. `flatten(x)`

`transpose(l, .names = NULL)` Transposes the index order in a multi-level list. `transpose(x)`

SUMMARISE LISTS

`every(x, p, ...)` Do all elements pass a test? `every(x, is.character)`

`some(x, p, ...)` Do some elements pass a test? `some(x, is.character)`

`has_element(x, y)` Does a list contain an element? `has_element(x, "foo")`

`detect(x, f, ..., .right=FALSE, .p)` Find first element to pass. `detect(x, is.character)`

`detect_index(x, f, ..., .right = FALSE, .p)` Find index of first element to pass. `detect_index(x, is.character)`

`vec_depth(x)` Return depth (number of levels of indexes). `vec_depth(x)`

JOIN (TO) LISTS

`append(x, values, after = length(x))` Add to end of list. `append(x, list(d = 1))`

`prepend(x, values, before = 1)` Add to start of list. `prepend(x, list(d = 1))`

`splice(...)` Combine objects into a list, storing S3 objects as sub-lists. `splice(x, y, "foo")`

TRANSFORM LISTS

`modify(x, f, ...)` Apply function to each element. Also `map`, `map_chr`, `map_dbl`, `map_dfc`, `map_dfr`, `map_int`, `map_lgl`. `modify(x, ~.+2)`

`modify_at(x, .at, f, ...)` Apply function to elements by name or index. Also `map_at`. `modify_at(x, "b", ~.+2)`

`modify_if(x, p, f, ...)` Apply function to elements that pass a test. Also `map_if`. `modify_if(x, is.numeric, ~.+2)`

`modify_depth(x, depth, f, ...)` Apply function to each element at a given level of a list. `modify_depth(x, 1, ~.+2)`

WORK WITH LISTS

`array_tree(array, margin = NULL)` Turn array into list. Also `array_branch`. `array_tree(x, margin = 3)`

`cross2(x, y, filter = NULL)` All combinations of `x` and `y`. Also `cross`, `cross3`, `cross_df`. `cross2(1:3, 4:6)`

`set_names(x, nm = x)` Set the names of a vector/list directly or with a function. `set_names(x, c("p", "q", "r"))` `set_names(x, tolower)`

Reduce Lists

`func + a b c d` → `func(a, b)` → `func(c, d)` → `func(a, b, c, d)`

`func + a b c d` → `func(a, b)` → `func(c, d)` → `func(a, b, c, d)`

`reduce(x, f, ..., .init, .dir = c("forward", "backward"))` Apply function recursively to each element of a list or vector. Also `reduce2`. `reduce(x, sum)`

`accumulate(x, f, ..., .init)` Reduce, but also return intermediate results. Also `accumulate2`. `accumulate(x, sum)`

Modify function behavior

`compose()` Compose multiple functions.

`lift()` Change the type of input a function takes. Also `lift_dbl`, `lift_dv`, `lift_ld`, `lift_lv`, `lift_vd`, `lift_vl`.

`rerun()` Rerun expression `n` times.

`negate()` Negate a predicate function (a pipe friendly !)

`partial()` Create a version of a function that has some args preset to values.

`safely()` Modify func to return list of results and errors.

`quietly()` Modify function to return list of results, output, messages, warnings.

`possibly()` Modify function to return default value whenever an error occurs (instead of error).



RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at [purrr.tidyverse.org](#) • purrr 0.2.3 • Updated: 2019-08

Nested Data

A **nested data frame** stores individual tables within the cells of a larger, organizing table.

nested data frame

"cell" contents

n_iris\$data[[1]]

n_iris\$data[[2]]

n_iris\$data[[3]]

Use a nested data frame to:

- preserve relationships between observations and subsets of data
- manipulate many sub-tables at once with the **purrr** functions `map()`, `map2()`, or `pmap()`.

Use a two step process to create a nested data frame:

- Group the data frame into groups with `dplyr::group_by()`
- Use `nest()` to create a nested data frame with one row per group

n_iris <- iris %>% group_by(Species) %>% nest()

tidyr::nest(data, ..., .key = data)
For grouped data, moves groups into cells as data frames.

Unnest a nested data frame with `unnest()`:

n_iris %>% unnest()

tidyr::unnest(data, ..., .drop = NA, .id=NULL, .sep=NULL)
Unnests a nested data frame.



List Column Workflow

Nested data frames use a **list column**, a list that is stored as a column vector of a data frame. A typical **workflow** for list columns:

1 Make a list column

Species S.L S.W P.L P.W
setosa 5.1 3.5 1.4 0.2
setosa 4.9 3.0 1.4 0.2
setosa 4.7 3.2 1.3 0.2
setosa 4.6 3.1 1.5 0.2
setosa 5.0 3.6 1.4 0.2
versl 7.0 3.2 4.7 1.4
versl 6.4 3.2 4.5 1.5
versl 6.9 3.1 4.9 1.5
versl 5.5 2.3 4.0 1.3
virgini 6.3 3.3 6.0 2.5
virgini 5.8 2.7 5.1 1.9
virgini 7.1 3.0 5.9 2.1
virgini 6.3 2.9 5.6 1.8

n_iris <- iris %>%
group_by(Species) %>%
nest()

2 Work with list columns

mod_fun <- function(df)
lm(Sepal.Length ~ ., data = df)
m_iris <- n_iris %>%
mutate(model = map(data, mod_fun))

3 Simplify the list column

b_fun <- function(mod)
coefficients(mod)[1:1]
m_iris %>% transmute(Species,
beta = map_dbl(model, b_fun))

1. MAKE A LIST COLUMN - You can create list columns with functions in the **tibble** and **dplyr** packages, as well as **tidyr**'s `nest()`

tibble::tribble(...)
Makes list column when needed
tribble(~max, ~seq,
3, 1:3,
4, 1:4,
5, 1:5)

tibble::tribble(...)
Saves list input as list columns
tribble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))

dplyr::mutate(data, ...) Also **transmute()**
Returns list col when result returns list.
mtcars %>% mutate(seq = map(cyl, seq))

tibble::enframe(x, name="name", value="value")
Converts multi-level list to tibble with list cols
enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')

dplyr::summarise(data, ...)
Returns list col when result is wrapped with **list()**
mtcars %>% group_by(cyl) %>%
summarise(q = list(quantile(mpg)))

2. WORK WITH LIST COLUMNS - Use the purrr functions `map()`, `map2()`, and `pmap()` to apply a function that returns a result element-wise to the cells of a list column. `walk()`, `walk2()`, and `pwalk()` work the same way, but return a side effect.

purrr::map(x, f, ...)
Apply .f element-wise to .x as .f(.x)
n_iris %>% mutate(n = map(data, dim))
purrr::map2(x, y, f, ...)
Apply .f element-wise to .x and .y as .f(.x, .y)
n_iris %>% mutate(n = map2(data, model, list))
purrr::pmap(.l, f, ...)
Apply .f element-wise to vectors saved in .l
m_iris %>%
mutate(n = pmap(list(data, model, data), list))

map(
data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>
)
fun, ...
fun(
data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>
)
result
result 1
result 2
result 3

map2(
data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>
)
model
<S3: lms>
<S3: lms>
<S3: lms>
)
fun, ...
fun(
data
<tibble [50x4]>
<tibble [50x4]>
<tibble [50x4]>
)
model
<S3: lms>
<S3: lms>
<S3: lms>
)
result
result 1
result 2
result 3

3. SIMPLIFY THE LIST COLUMN (into a regular column)

Use the purrr functions `map_lgl()`, `map_int()`, `map_dbl()`, `map_chr()`, as well as **tidyr**'s `unnest()` to reduce a list column into a regular column.

purrr::map_lgl(x, f, ...)
Apply .f element-wise to .x, return a logical vector
n_iris %>% transmute(n = map_lgl(data, is.matrix))
purrr::map_int(x, f, ...)
Apply .f element-wise to .x, return an integer vector
n_iris %>% transmute(n = map_int(data, nrow))

purrr::map_dbl(x, f, ...)
Apply .f element-wise to .x, return a double vector
n_iris %>% transmute(n = map_dbl(data, nrow))
purrr::map_chr(x, f, ...)
Apply .f element-wise to .x, return a character vector
n_iris %>% transmute(n = map_chr(data, nrow))

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at [purrr.tidyverse.org](#) • purrr 0.2.3 • Updated: 2019-08



We use **optional** third-party analytics cookies to understand how you use GitHub.com so we can build better products. [Learn more.](#)

Accept

Reject

bout