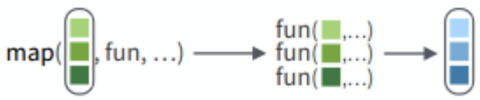
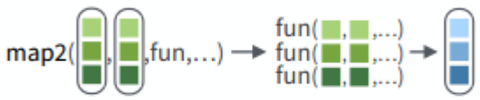
Nhóm hàm apply với purrr : : CHEAT SHEET

Các nhóm hàm apply

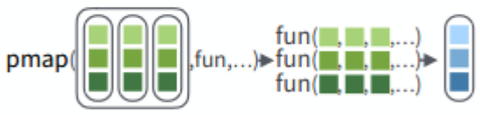
Các nhóm hàm **map** thực thi một hàm với từng giá trị của list hoặc vector



map(.x, .f, …) Thực hiện hàm f với mỗi giá trị của list hoặc vector. *map(x, is.logical)*



map2(.x, ,y, .f, …) Thực hiện hàm f với mỗi cặp giá trị x và y *map2(x, y, sum)*



pmap(.l, .f, …) Thực hiện hàm f với một nhóm các giá trị của list hoặc vector. *pmap(list(x, y, z), sum, na.rm = TRUE)*

invoke_map(.f, .x = list(NULL), …, .env=NULL) Thực hiện mỗi hàm trong một list. Xem thêm **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Thực hiện hàm .f với mỗi giá trị của list.
imap(.x, .f, ...) Thực hiện hàm .f với mỗi giá trị của x và index của x

KẾT QUẢ ĐẦU RA

map(), map2(), pmap(), **imap** & **invoke_map** trả ra kết quả là 1 list. Sử dụng các hàm có phần hậu tố để trả ra kết quả đúng định dạng mong muốn, VD. **map2_chr**, **pmap_lgl**,...

Sử dụng **walk**, **walk2**, & **pwalk** để thực hiện hiệu ứng biên (side effect). Kết quả mỗi hàm trả ra sẽ được ẩn đi.

Hàm	Kết quả
map	list
map_chr	Vector dạng character
map_dbl	Vector dạng số double
map_dfc	data frame (nối cột)
map_dfr	data frame (nối dòng)
map_int	Vector dạng số integer
map_lgl	Vector dạng số logic
walk	Thực hiện hiệu ứng biên, trả ra kết quả ẩn

SHORTCUTS – với các hàm purrr

"name" trở thành **function(x) x[["name"]]**, VD. *map(l, "a")* chiết xuất *a* từ mỗi giá trị của l

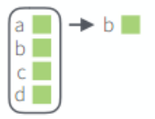
~ .x trở thành **function(x) x**, VD. *map(l, ~ 2 +.x)* trở thành *map(l, function(x) 2 + x)*

~ .x .y trở thành **function(.x, .y) .x .y**, *VD map2(l, p, ~ .x +.y)* trở thành *map2(l, p, function(l, p) l + p)*

~ ..1 ..2 trở thành **function(..1, ..2,** etc**) ..1 ..2** etc, e.g. *pmap(list(a, b, c), ~ ..3 + ..1 - ..2)* trở thành *pmap(list(a, b, c), function(a, b, c) c + a - b)*

Làm việc với list

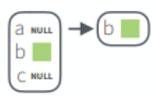
LỌC LIST



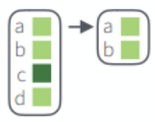
pluck(.x, ..., .default=NULL) Tìm giá trị theo tên hoặc index, *pluck(x,"b")*, hoặc thuộc tính với **attr_getter**. *pluck(x,"b",attr_getter("n"))*

keep(.x, .p, …) Tìm giá trị thỏa mãn điều kiện lọc. *keep(x, is.na)*

discard(.x, .p, …) Loại bỏ giá trị không thỏa mãn điều kiện lọc. *discard(x, is.na)*



compact(.x, .p = identity) Loại bỏ giá trị rỗng. *compact(x)*



head_while(.x, .p, …) Trả các kết quả theo thứ tự cho đến khi một giá trị không thỏa mãn điều kiện lọc. Xem thêm **tail_while**. *head_while(x, is.character)*

TỔNG HỢP LIST

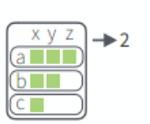
every(.x, .p, …) Kiểm tra tất cả các giá trị thỏa mãn điều kiện cho trước? *every(x, is.character)*

some(.x, .p, …) Liệu có tồn tại một vài giá trị thỏa mãn điều kiện? *some(x, is.character)*

has_element(.x, .y) list có chứa giá trị y? *has_element(x, "foo")*

detect(.x, .f, ..., .right=FALSE, .p) Tìm giá trị đầu tiên thỏa mãn điều kiện. *detect(x, is.character)*

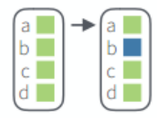
detect_index(.x, .f, ..., .right = FALSE, .p) Tìm index của giá trị đầu tiên thỏa mãn điều kiện. *detect_index(x, is.character)*



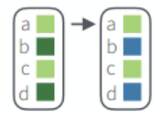
vec_depth(x) Trả ra bậc của x (số lượng giá trị các biến của index). *vec_depth(x)*

BIẾN ĐỔI LIST

modify(.x, .f, ...) Áp dụng hàm f với mỗi giá trị của x. Xem thêm **map**, **map_chr**, **map_dbl**, **map_dfc**, **map_dfr**, **map_int**, **map_lgl**. *modify(x, ~.+ 2)*



modify_at(.x, .at, .f, ...) Áp dụng hàm f với mỗi giá trị theo index hoặc tên. Xem thêm **map_at**. *modify_at(x, "b", ~.+ 2)*



modify_if(.x, .p, .f, ...) Áp dụng hàm f nếu x thỏa mãn điều kiện p. Xem thêm **map_if**. *modify_if(x, is.numeric,~.+2)*

modify_depth(.x,.depth,.f,...) Áp dụng hàm với mỗi giá trị ở level cho trước trong list. *modify_depth(x, 1, ~.+ 2)*

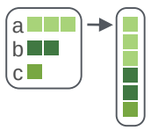
LÀM VIỆC VỚI LIST

array_tree(array, margin = NULL) Biến đổi dạng mảng thành list. Xem **array_branch**. *array_tree(x, margin = 3)*

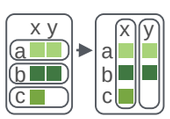
cross2(.x, .y, .filter = NULL) Tổ hợp của .x và .y. Xem **cross**, **cross3**, **cross_df**. *cross2(1:3, 4:6)*

set_names(x, nm = x) Đặt tên của vector/list *set_names(x, c("p", "q", "r"))* *set_names(x, tolower)*

TÁI CẤU TRÚC LIST

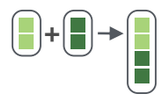


flatten(.x) Loại bỏ các bậc hoặc index của list. Xem **flatten_chr**, **flatten_dbl**, **flatten_dfc**, **flatten_dfr**, **flatten_int**, **flatten_lgl**. *flatten(x)*

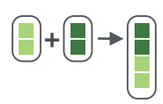


transpose(.l, .names = NULL) Xoay trục của thứ tự index với list có nhiều bậc. *transpose(x)*

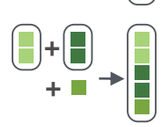
JOIN CÁC LIST



append(x, values, after = length(x)) Nối một list sau list khác. *append(x, list(d = 1))*

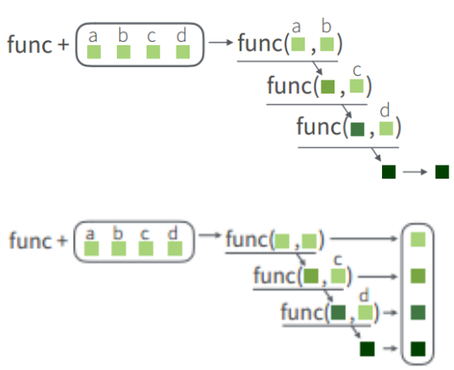


prepend(x, values, before = 1) Nối một list vào trước một list. *prepend(x, list(d = 1))*



splice(…) Nối các đối tượng vào một list, lưu dưới dạng S3. *splice(x, y, "foo")*

Thu gọn lists



reduce(.x, .f, ..., .init) Áp dụng hàm f liên tiếp với mỗi giá trị của x, trả ra kết quả cuối cùng. Xem **reduce_right**, **reduce2**, **reduce2_right**. *reduce(x, sum)*

accumulate(.x, .f, ..., .init) Tương tự như reduce, nhưng trả ra kết quả trung gian. Xem **accumulate_right**. *accumulate(x, sum)*

Thay đổi cách hoạt động của hàm

compose() Tích hợp nhiều hàm.

lift() Thay đổi loại dữ liệu đầu vào. Xem thêm **lift_dl**, **lift_dv**, **lift_ld**, **lift_lv**, **lift_vd**, **lift_vl**.

rerun() Chạy lại nhóm câu lệnh n lần

negate() Phủ định một hàm)

partial() Thay đổi hàm bằng cách điều chỉnh tham số mặc định

safely() Thay đổi hàm trả ra kết quả và lỗi.

quietly() Thay đổi hàm cho phép trả ra kết quả, thông báo, cảnh báo.

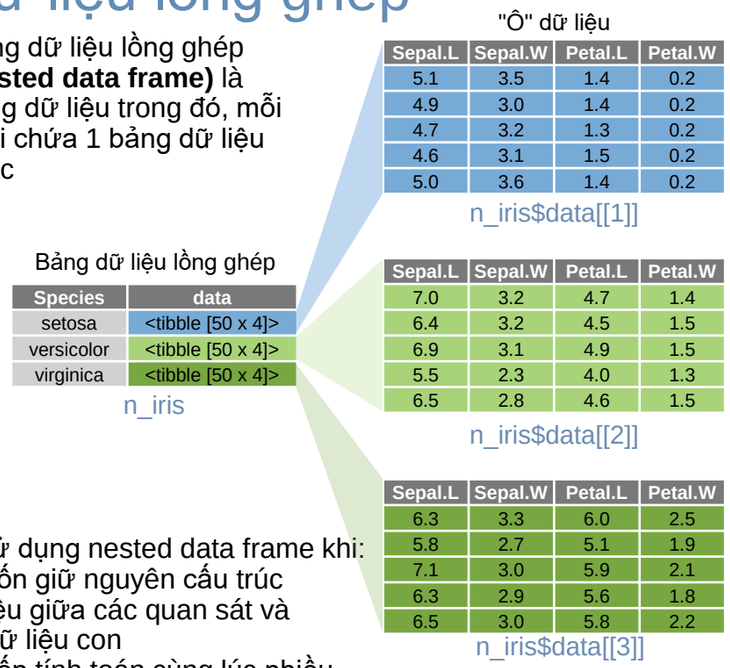
possibly() Thay đổi hàm cho phép trả giá trị mặc định khi xuất hiện lỗi.

Translator: ranalytics.vn

Xem thêm purrr.tidyverse.org • purrr 0.2.3 • Updated: 2017-09

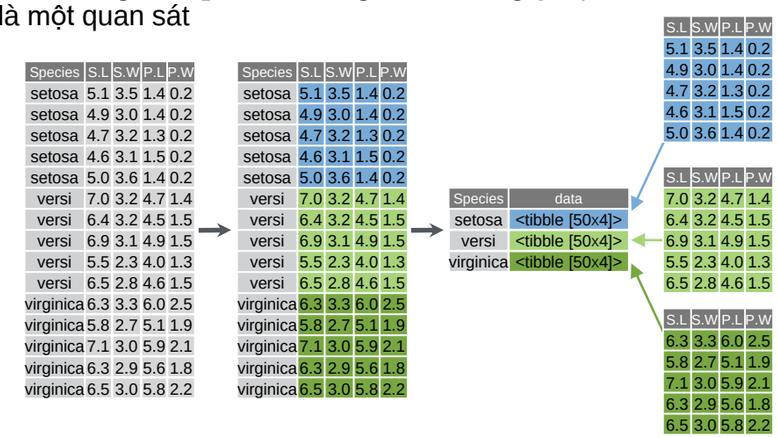
Dữ liệu lồng ghép

Bảng dữ liệu lồng ghép (**nested data frame)** là bảng dữ liệu trong đó, mỗi ô lại chứa 1 bảng dữ liệu khác



Ta sử dụng nested data frame khi:
- Muốn giữ nguyên cấu trúc dữ liệu giữa các quan sát và tập dữ liệu con
- Muốn tính toán cùng lúc nhiều bản dữ liệu con với **purrr** như **map()**, **map2()**, or **pmap()**.

Để tạo nested data frame, ta thực hiện hai bước sau
1. Nhóm data frame với **dplyr::group_by()**
2. Sử dụng **nest()** để tạo bảng dữ liệu lồng ghép, mỗi nhóm là một quan sát

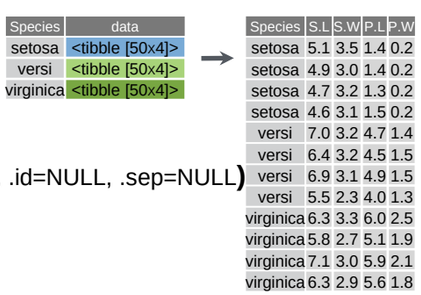


n_iris <- iris %>% **group_by**(Species) %>% **nest**()

tidyr::**nest(**data, ..., .key = data**)**

Với dữ liệu đã được nhóm, chỉ cần dùng hàm nest.

Phân rã bảng dữ liệu lồng ghép với **unnest()**:



n_iris %>%
unnest()

tidyr::**unnest(**data, ..., .drop = NA, .id=NULL, .sep=NULL**)**

Làm việc với biến chứa list

Nested data frames sử dụng **biến có chứa** list. Mô hình phân tích dữ liệu với biến chứa list diễn ra như sau:


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

```
mod_fun <- function(df) {

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

```
b_fun <- function(mod)

   coefficients(mod)[[1]]
m_iris %>% transmute(Species,
   beta = map_dbl(model,
   b_fun))
```

1. TẠO BIẾN CHỨA LIST – sử dụng gói **tibble** & **dplyr**, và hàm nest() của **tidyr**

tibble::**tribble(**…**)**
Tạo dữ liệu chứa list

```
tribble( ~max, ~seq,
          3,   1:3,
          4,   1:4,
          5,   1:5)
```

max	seq
3	<int [3]>
4	<int [4]>
5	<int [5]>

tibble::**tibble(**…**)**
Lưu biến đầu vào dạng list

```
tibble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))
```

tibble::**enframe(**x, name="name", value="value"**)**
Biến đổi định dạng list thành nested data frame

```
enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')
```

dplyr::**mutate(**.data, …**)** Xem thêm **transmute()**
Tạo biến chứa list

```
mtcars %>% mutate(seq = map(cyl, seq))
```

dplyr::**summarise(**.data, …**)**
Tổng hợp dữ liệu và trả ra kết quả biến dạng list với hàm list()

```
mtcars %>%  group_by(cyl) %>%
 summarise(q = list(quantile(mpg)))
```

2. LÀM VIỆC VỚI BIẾN CHỨA LIST – Sử dụng các hàm **map()**, **map2()**, và **pmap()** trong gói purrr. Các hàm **walk()**, **walk2()** & **pwalk()** trả ra kết quả tương tự nhưng có thêm hiệu ứng biên

purrr::**map(**.x, .f, ...**)**
Áp dụng .f vào mỗi giá trị của .x trở thành .f(.x)

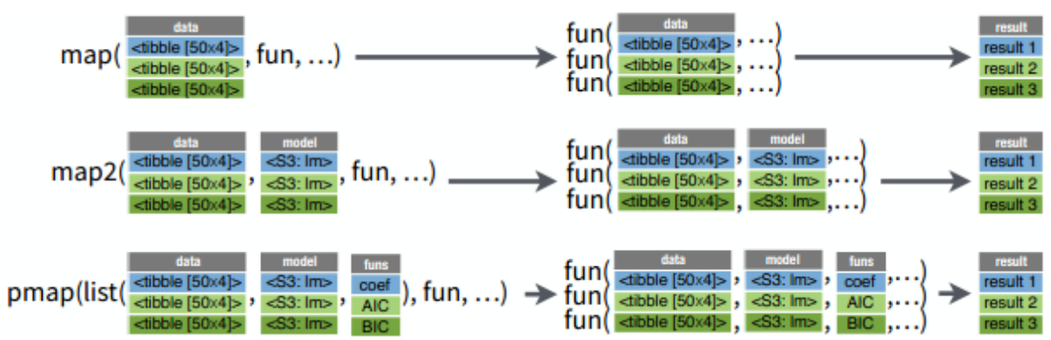
```
n_iris %>% mutate(n = map(data, dim))
```

purrr::**map2(**.x, .y, .f, ...**)**
Áp dụng .f vào mỗi cặp giá trị của .x và.y, trở thành .f(.x, .y)

```
m_iris %>% mutate(n = map2(data, model, list))
```

purrr::**pmap(**.l, .f, ...**)**
Áp dụng .f vào một vector giá trị lưu trong .l

```
m_iris %>%
  mutate(n = pmap(list(data, model, data), list))
```



3. ĐƠN GIẢN HÓA BIẾN CHỨA LIST (biến đổi thành định dạng cột thông thường)

Sử dụng các hàm **map_lgl()**, **map_int()**, **map_dbl()**, **map_chr()**, và **unnest()** để biến đổi biến chứa list trở thành biến định dạng cột thông thường

purrr::**map_lgl(**.x, .f, ...**)**
Áp dụng .f vào mỗi giá trị của .x, trả ra kết quả vector logic

```
n_iris %>% transmute(n = map_lgl(data, is.matrix))
```

purrr::**map_int(**.x, .f, ...**)**
Áp dụng .f vào mỗi giá trị của .x , trả ra kết quả vector integer

```
n_iris %>% transmute(n = map_int(data, nrow))
```

purrr::**map_dbl(**.x, .f, ...**)**
Áp dụng .f vào mỗi giá trị của .x , trả ra kết quả vector double

```
n_iris %>% transmute(n = map_dbl(data, nrow))
```

purrr::**map_chr(**.x, .f, ...**)**
Áp dụng .f vào mỗi giá trị của .x , trả ra kết quả vector định dạng character

```
n_iris %>% transmute(n = map_chr(data, nrow))
```


Translator: ranalytics.vn

Xem thêm purrr.tidyverse.org • purrr 0.2.3 • Updated: 2017-09