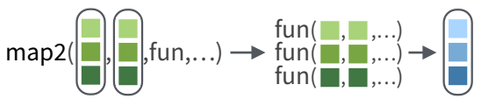
Aplicar funciones con purrr : : **GUÍA RÁPIDA**

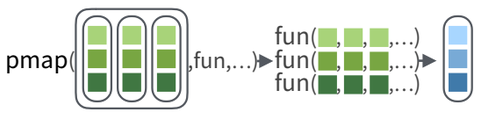
Applicar Funciones

Las funciones map aplican una función iterativamente a cada elemento de una lista o un vector.

map(.x, .f, …) Aplica una función a cada elemento de una lista o vector. *map(x, is.logical)*



map2(.x, ,y, .f, …) Aplica una función a pares de elementos de dos listas, vectores. *map2(x, y, sum)*



pmap(.l, .f, …) Aplica una función a grupos de elementos de listas de listas, vectores. *pmap(list(x, y, z), sum, na.rm = TRUE)*

invoke_map(.f, .x = list(NULL), …, .env=NULL) Ejecuta cada función en una lista. También **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Aplica una función a cada elemento de una lista o vector.
imap(.x, .f, ...) Aplica .f a cada elemento de una lista o vector y su índice.

SALIDA

map(), map2(), pmap(), imap y **invoke_map** devuelven una lista. Usar la versión con el sufijo para devolver el resultado de acuerdo a un tipo o un vector plano, e.g. **map2_chr**, **pmap_lgl**, etc.

Usar **walk**, **walk2**, y **pwalk** para producir efectos alternativos. Cada uno devuelve su entrada de forma invisible.

Función	devuelve
map	Lista
map_chr	vector caracter
map_dbl	vector double (numérico)
map_dfc	data frame (columna añadida)
map_dfr	data frame (fila añadida)
map_int	vector entero
map_lgl	vector lógico
walk	crea efectos adicionales devuelve la entrada de forma invisible

ATAJOS - en una función purrr:

"name" pasa a ser **function(x) x$name.** e.g. *map(l, "a")* extrae *$a* de cada elemento de *l*

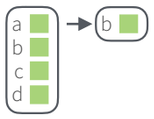
~ . pasa a ser **function(x) x**. e.g. *map(l, ~ 2 +.)* pasa a ser *map(l, function(x) 2 + x)*

~ .x .y pasa a ser **function(.x, .y) .x .y**. e.g. *map2(l, p, ~ .x +.y)* pasa a ser *map2(l, p, function(l, p) l + p)*

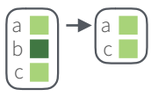
~ ..1 ..2 etc pasa a ser **function(..1, ..2,** etc**) ..1 ..2** etc e.g. *pmap(list(a, b, c), ~ ..3 + ..1 - ..2)* pasa a ser *pmap(list(a, b, c), function(a, b, c) c + a - b)*

Trabajar con Listas

FILTRAR LISTAS

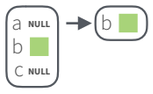


pluck(.x, ..., .default=NULL) Selecciona un elemento por nombre o índice, *pluck(x,"b")* ,o su atributo con **attr_getter**. *pluck(x,"b",attr_getter("n"))*

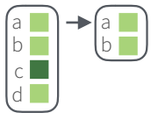


keep(.x, .p, …) Selecciona elementos que pasan una prueba lógica. *keep(x, is.na)*

discard(.x, .p, …) Selecciona elementos que no pasan una prueba lógica. *discard(x, is.na)*

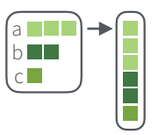


compact(.x, .p = identity) Elimina elementos vacíos. *compact(x)*

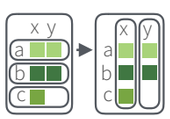


head_while(.x, .p, …) Devuelve elementos desde el principio hasta que uno no pasa. También **tail_while**. *head_while(x, is.character)*

REMODELAR LISTAS



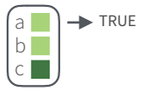
flatten(.x) Elimina un nivel de índices de una lista. También **flatten_chr**, **flatten_dbl**, **flatten_dfc**, **flatten_dfr**, **flatten_int**, **flatten_lgl**. *flatten(x)*



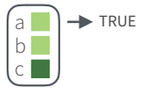
transpose(.l, .names = NULL) Transpone el orden del índice en una multi-lista. *transpose(x)*

RESUMIR LISTAS

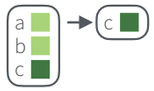
every(.x, .p, …) ¿Pasan todos los elementos una prueba? *every(x, is.character)*



some(.x, .p, …) ¿Pasan algunos elementos una prueba? *some(x, is.character)*

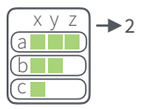


has_element(.x, .y) ¿Contiene la lista un elemento? *has_element(x, "foo")*



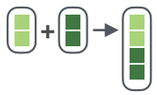
detect(.x, .f, ..., .right=FALSE, .p) Encuentra el primer elemento que pasa. *detect(x, is.character)*

detect_index(.x, .f, ..., .right = FALSE, .p) Encuentra el índice del primer elemento que pasa. *detect_index(x, is.character)*

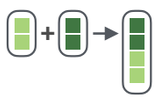


depth(x) Devuelve profundidad (número de niveles o índices). *depth(x)*

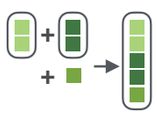
UNIR LISTAS



append(x, values, after = length(x)) Añade al final de una lista. *append(x, list(d = 1))*



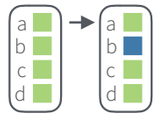
prepend(x, values, before = 1) Añade al principio de una lista. *prepend(x, list(d = 1))*



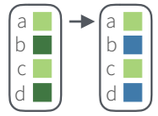
splice(…) Combina objetos en una lista, almacena objetos S3 como sub-listas. *splice(x, y, "foo")*

TRANSFORMAR LISTAS

modify(.x, .f, ...) Aplica una función a cada elemento. También **map**, **map_chr**, **map_dbl**, **map_dfc**, **map_dfr**, **map_int**, **map_lgl**. *modify(x, ~.+2)*



modify_at(.x, .at, .f, ...) Aplica una función a los elementos por nombre o índice. También **map_at**. *modify_at(x, "b", ~.+2)*

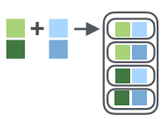


modify_if(.x, .p, .f, ...) Aplica una función a los elementos que pasan una prueba. También **map_if**. *modify_if(x, is.numeric,~.+2)*

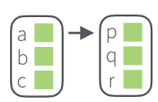
modify_depth(.x,.depth,.f,...) Aplica una función a cada elemento y un nivel dado de una lista. *modify_depth(x, 1, ~.+2)*

TRABAJAR CON LISTAS

array_tree(array, margin = NULL) Convierte una matriz en una lista. También **array_branch**. *array_tree(x, margin = 3)*

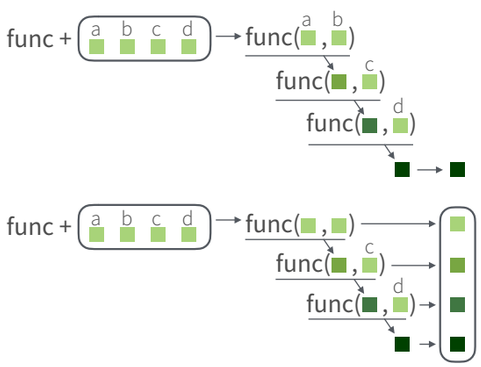


cross2(.x, .y, .filter = NULL) Todas las combinaciones de .x e .y. También **cross**, **cross3**, **cross_df**. *cross2(1:3, 4:6)*



set_names(x, nm = x) Fija el nombre de un vector/lista directamente o con una función. *set_names(x, c("p", "q", "r"))* *set_names(x, tolower)*

Reducir Listas



reduce(.x, .f, ..., .init) Aplica una función de forma recursiva a cada elemento de un a lista o vector. También **reduce_right**, **reduce2**, **reduce2_right**. *reduce(x, sum)*

accumulate(.x, .f, ..., .init) Reduce, pero también devuelve resultados intermedios. También **accumulate_right**. *accumulate(x, sum)*

Modifica el comportamiento

compose() Compone múltiples funciones.

lift() Cambia el tipo de la entrada que una recibe una función. También **lift_dl**, **lift_dv**, **lift_ld**, **lift_lv**, **lift_vd**, **lift_vl**.

rerun() Ejecuta una expresión n veces.

negate() Niega el predicado de una función (a pipe friendly !)

partial() Aplica parcialmente una función, completando algunos argumentos.

safely() Modifica la función para devolver lista de resultados y errores.

quietly() Modifica la función para devolver lista de resultados, salidas, mensajes y avisos.

possibly() Modifica la función para devolver el valor por defecto para cualquier tipo de error que ocurra (en vez del error).

Datos Anidados

Un **data frame anidado** almacena tablas individuales en las celdas de una tabla organizada más grande.

nested data frame

Species	data
setosa	<tibble [50 x 4]>
versicolor	<tibble [50 x 4]>
virginica	<tibble [50 x 4]>

n_iris

"cell" contents

Sepal.L	Sepal.W	Petal.L	Petal.W
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

n_iris$data[[1]]

Sepal.L	Sepal.W	Petal.L	Petal.W
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5

n_iris$data[[2]]

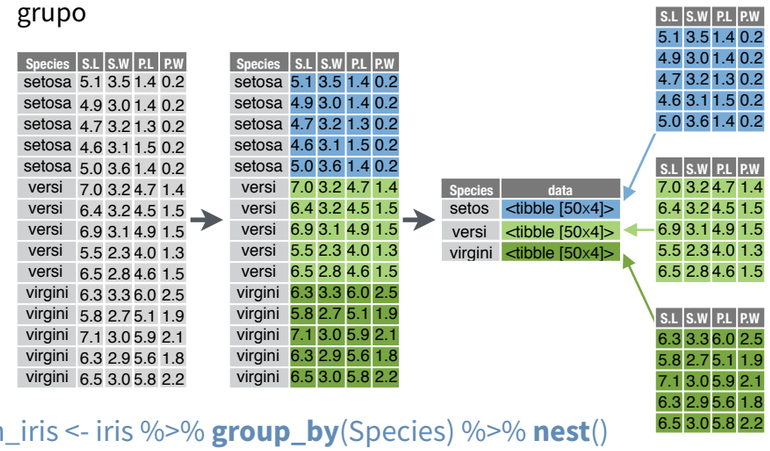
Sepal.L	Sepal.W	Petal.L	Petal.W
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
6.3	2.9	5.6	1.8
6.5	3.0	5.8	2.2

n_iris$data[[3]]

Usa un data frame anidado para:

- Preservar las relaciones entre las observaciones y los subconjuntos de datos
- manipular varias sub-tablas a la vez con las funciones **purrr** **map()**, **map2()**, o **pmap()**.

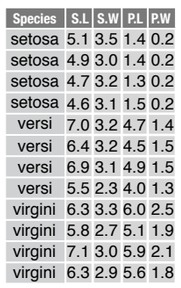
Usa un proceso a dos pasos para crear un data frame anidado:
1. Agrupa los data frames en grupos con **dplyr::group_by()**
2. Usa **nest()** para crear un data frame anidado con una fila por grupo



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

tidyr::**nest(**data, ..., .key = data**)**
Para datos agrupados, mueve grupos a celdas como data frames

Desanidar un data frame anidado con **unnest()**:



n_iris %>% **unnest**()

tidyr::**unnest(**data, ..., .drop = NA, .id=NULL, .sep=NULL**)**
Desanida un data frame anidado.

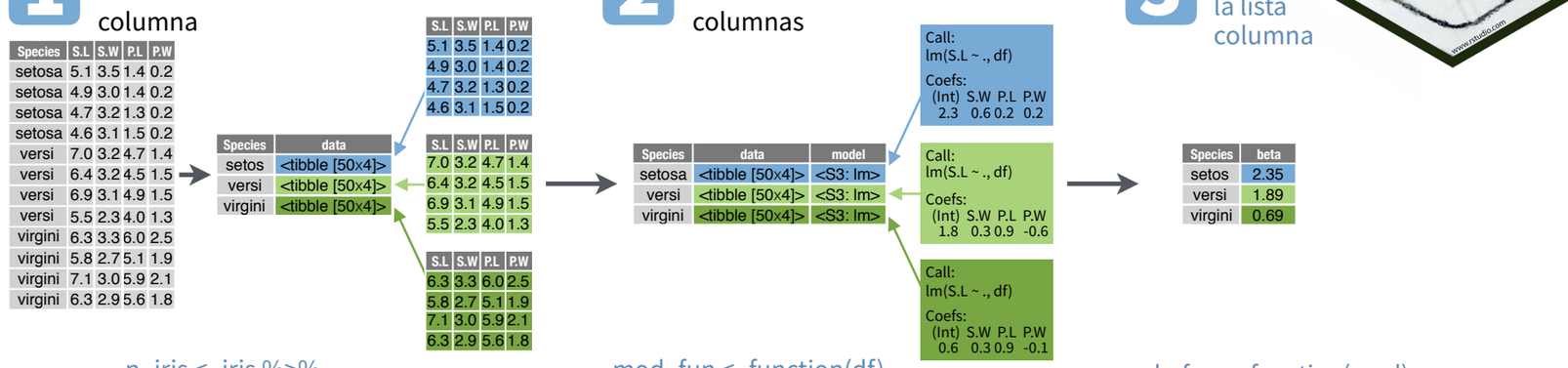
Flujo Lista Columna

Los data frames anidados usan una **lista columna**, una lista que es almacenada como un vector columna de un data frame. El **flujo de trabajo** para listas de columnas::

1 **Crear** una lista columna

2 **Trabajar con** listas columnas

3 **Simplicar** la lista columna



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. CREARE UNA LISTA COLUMNA - Se puede crear una lista columnas con funciones de los paquetes **tibble** y **dplyr**, también con nest() de **tidyr**

tibble::**tribble(**…**)**
Crea una lista columna cuando se necesita

```
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(**…**)**
Guarda una lista como lista columnas
tibble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))

tibble::**enframe(**x, name="name", value="value"**)**
Convierte una lista multilevel a un tibble con list columnas
enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')

dplyr::**mutate(**.data, …**)** También **transmute()**
Devuelve una lista col cuando el resultado devuelve una lista.
mtcars %>% **mutate**(seq = **map**(cyl, seq))

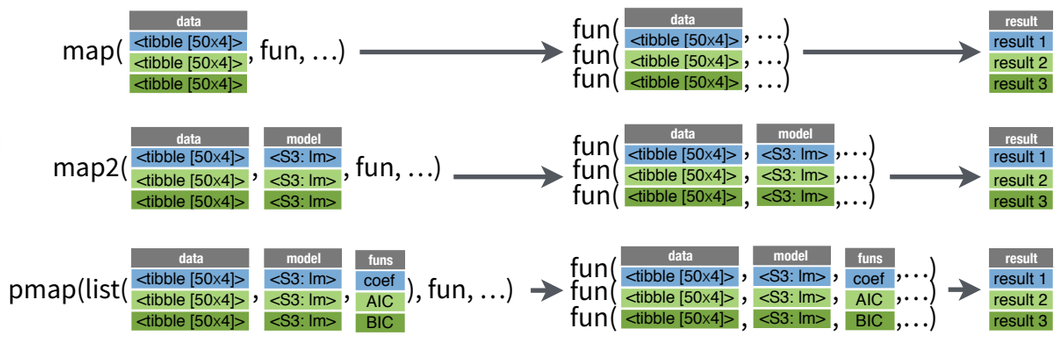
dplyr::**summarise(**.data, …**)**
Devuelve una lista col cuando el resultado se envuelve con **list()**
mtcars %>% group_by(cyl) %>%
summarise(q = **list**(quantile(mpg)))

2. TRABAJAR CON LISTA COLUMNAS - Usa las funciones de purrr **map()**, **map2()**, y **pmap()** para aplicar una función que devuelve un resultado elemento a elemento en las celdas de una lista columna. **walk()**, **walk2()**, y **pwalk()** funcionan de la misma forma, pero producen un efecto secundario

purrr::**map(**.x, .f, ...**)**
Aplica .f elemento a elemento a .x como .f(.x)
n_iris %>% **mutate**(n = **map**(data, dim))

purrr::**map2(**.x, .y, .f, ...**)**
Aplica .f elemento a elemento a .x e .y como .f(.x, .y)
m_iris %>% **mutate**(n = **map2**(data, model, list))

purrr::**pmap(**.l, .f, ...**)**
Aplica .f elemento a elemento a los vectores guardados en .l
m_iris %>%
mutate(n = **pmap**(list(data, model, data), list))



3. SIMPLIFICAR LA LISTA COLUMNA (en una columna regular)

Usa las funciones de purrr **map_lgl()**, **map_int()**, **map_dbl()**, **map_chr()**, también con **unnest()** de tidyr para reducir una lista columna a una columna regular.

purrr::**map_lgl(**.x, .f, ...**)**
Aplica .f elemento a elemento a .x, devuelve un vector lógico
n_iris %>% **transmute**(n = **map_lgl**(data, is.matrix))

purrr::**map_int(**.x, .f, ...**)**
Aplica .f elemento a elemento a .x, devuelve un vector entero
n_iris %>% **transmute**(n = **map_int**(data, nrow))

purrr::**map_dbl(**.x, .f, ...**)**
Aplica .f elemento a elemento a .x, devuelve un vector tipo doble
n_iris %>% **transmute**(n = **map_int**(data, nrow))

purrr::**map_chr(**.x, .f, ...**)**
Aplica .f elemento a elemento a .x, devuelve un vector tipo caracter
n_iris %>% **transmute**(n = **map_chr**(data, nrow))