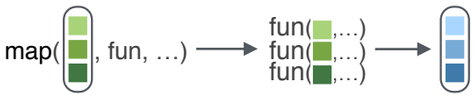
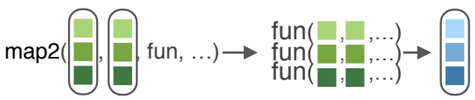
Применение функций с purrr : : ШПАРГАЛКА

Применение функций

Map-функции итеративно применяют функции к каждому элементу списка или вектора.



map(.x, .f, …) Применяет функцию к каждому элементу списка или вектора. *map(x, is.logical)*



map2(.x, ,y, .f, …) Применяет функцию к парам элементов из двух списков, векторов. *map2(x, y, sum)*

pmap(.l, .f, …) Применяет функцию к группам элементов из списка списков, векторов. *pmap(list(x, y, z), sum, na.rm = TRUE)*

invoke_map(.f, .x = list(NULL), …, .env=NULL) Вычисляет каждую функцию из списка. Также **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Применяет .f к списковым элементам списка, вектора.
imap(.x, .f, ...) Применяет .f к элементам списка, вектора и его индексу.

РЕЗУЛЬТАТ

map(), map2(), pmap(), **imap** и **invoke_map** возвращают список. Используйте версии с суффиксом для получения результата определенного типа, например **map2_chr**, **pmap_lgl**, и т.д.

Используйте **walk**, **walk2** и **pwalk** для совершения побочных действий. Все невидимо возвращают входные данные.

функция	результат
map	список
map_chr	символьный вектор
map_dbl	числовой вектор
map_dfc	data frame (соед. по столбцам)
map_dfr	data frame (соед. по строкам)
map_int	целочисленный вектор
map_lgl	логический вектор
walk	совершает побоч. действия, невидимо возвр. входные данные

СОКРАЩЕНИЯ - внутри функций purrr:

"name" становится **function(x) x[["name"]]**, напр. *map(l, "a")* извлекает *a* из каждого элемента *l*

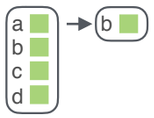
~ . становится **function(x) x**, напр. *map(l, ~ 2 +.)* становится *map(l, function(x) 2 + x)*

~ .x .y становится **function(.x, .y) .x .y**, напр. *map2(l, p, ~ .x +.y)* становится *map2(l, p, function(l, p) l + p)*

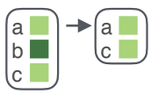
~ ..1 ..2 etc становится **function(..1, ..2,** etc**) ..1 ..2** etc напр. *pmap(list(a, b, c), ~ ..3 + ..1 - ..2)* становится *pmap(list(a, b, c), function(a, b, c) c + a - b)*

Работа со списками

ФИЛЬТРОВАНИЕ СПИСКОВ

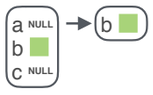


pluck(.x, ..., .default=NULL) Выбирает элемент по имени или индексу, *pluck(x,"b")*, или его атрибут (с **attr_getter**). *pluck(x,"b",attr_getter("n"))*

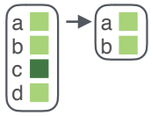


keep(.x, .p, …) Выбирает эл-ты, удовлетв. логич. условию. *keep(x, is.na)*

discard(.x, .p, …) Выбирает эл-ты, не удовлетв. логич. условию. *discard(x, is.na)*

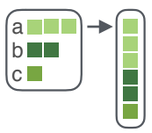


compact(.x, .p = identity) Удаляет пустые эл-ты. *compact(x)*

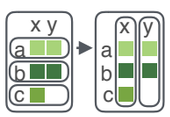


head_while(.x, .p, …) Возвращает начальные эл-ты до первого не подходящего. Также **tail_while**. *head_while(x, is.character)*

ФОРМАТИРОВАНИЕ СПИСКОВ



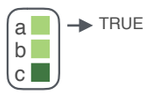
flatten(.x) Удаляет уровень индексов из списка. Также **flatten_chr**, **flatten_dbl**, **flatten_dfc**, **flatten_dfr**, **flatten_int**, **flatten_lgl**. *flatten(x)*



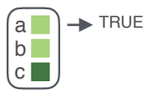
transpose(.l, .names = NULL) Транспонирует порядок индексов в многоуровневом списке. *transpose(x)*

СУММИРОВАНИЕ СПИСКОВ

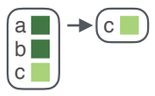
every(.x, .p, …) Все ли эл-ты удовлетв. условию? *every(x, is.character)*



some(.x, .p, …) Есть ли эл-ты, удовлетв. условию? *some(x, is.character)*

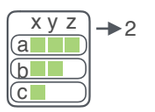


has_element(.x, .y) Содержит ли список эл-т? *has_element(x, "foo")*



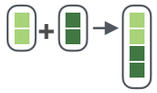
detect(.x, .f, ..., .right=FALSE, .p) Находит первый подходящий эл-т. *detect(x, is.character)*

detect_index(.x, .f, ..., .right = FALSE, .p) Находит индекс первого подходящего эл-та. *detect_index(x, is.character)*

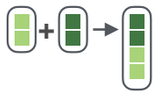


vec_depth(x) Возвращает глубину (кол-во уровней индексов). *vec_depth(x)*

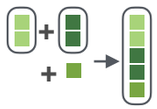
СОЕДИНЕНИЕ СПИСКОВ



append(x, values, after = length(x)) Добавляет эл-т в конец списка. *append(x, list(d = 1))*



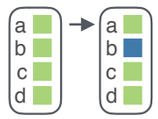
prepend(x, values, before = 1) Добав. эл-т в начало списка. *prepend(x, list(d = 1))*



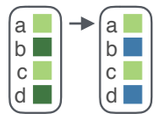
splice(…) Совмещает объекты в список, храня S3 объекты как подсписки. *splice(x, y, "foo")*

ПРЕОБРАЗОВАНИЕ СПИСКОВ

modify(.x, .f, ...) Применяет функцию к каждому эл-ту. Также **map**, **map_chr**, **map_dbl**, **map_dfc**, **map_dfr**, **map_int**, **map_lgl**. *modify(x, ~.+ 2)*



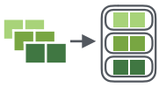
modify_at(.x, .at, .f, ...) Применяет функцию к эл-там по имени или индексу. Также **map_at**. *modify_at(x, "b", ~.+ 2)*



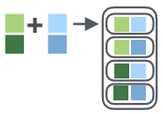
modify_if(.x, .p, .f, ...) Применяет функцию к эл-там, удовлетв. условию. Также **map_if**. *modify_if(x, is.numeric,~.+2)*

modify_depth(.x,.depth,.f,...) Применяет функцию к каждому эл-ту списка на опр. уровне. *modify_depth(x, 1, ~.+ 2)*

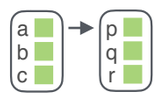
РАБОТА СО СПИСКАМИ



array_tree(array, margin = NULL) Преобразовывает массив в список. Также **array_branch**. *array_tree(x, margin = 3)*

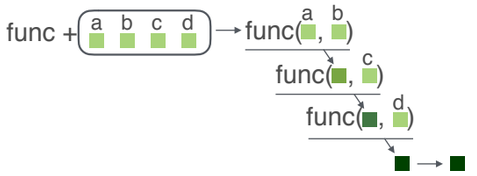


cross2(.x, .y, .filter = NULL) Все комбинации .x и .y. Также **cross**, **cross3**, **cross_df**. *cross2(1:3, 4:6)*

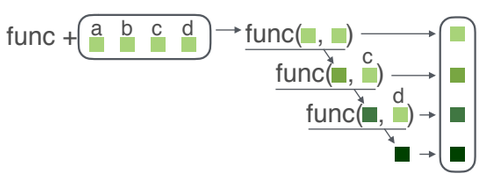


set_names(x, nm = x) Устанавливает имена вектора/списка напрямую или с помощью функции. *set_names(x, c("p", "q", "r"))* *set_names(x, tolower)*

Свертка списков



reduce(.x, .f, ..., .init) Применяет функцию рекурсивно к каждому эл-ту списка, вектора. Также **reduce_right**, **reduce2**, **reduce2_right**. *reduce(x, sum)*



accumulate(.x, .f, ..., .init) Как reduce(), но возвращает промежут. рез-ты. Также **accumulate_right**. *accumulate(x, sum)*

Изменение функций

compose() Создает композицию функций.

lift() Изменяет тип входных данных функции. Также **lift_dl**, **lift_dv**, **lift_ld**, **lift_lv**, **lift_vd**, **lift_vl**.

rerun() Независимо выполняет выражение n раз.

negate() Создает отрицание предикатной функции (можно использовать в конвейере !)

partial() Изменяет функцию, задавая опр. параметры.

safely() Изменяет функцию, чтобы возвращала список рез-тов и ошибок.

quietly() Изменяет функцию, чтобы возвращала список рез-тов, вывода, сообщений и предупреждений.

possibly() Изменяет функцию, чтобы возвращала значение по умолчанию при появлении ошибки (вместо ошибки).

Вложенные данные

Вложенный data frame хранит отдельные таблицы внутри ячеек большей упорядочивающей таблицы

содержимое "ячеек"

вложенный data frame

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

n_iris

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

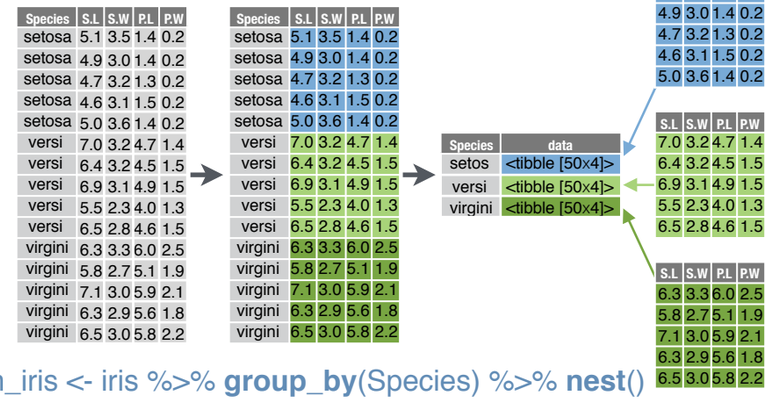
n_iris$data[[3]]

Используйте вложенный data frame для:

- сохранения отношений между наблюдениями и подмножествами данных
- манипулирования сразу несколькими подтаблицами с **purrr** функциями **map()**, **map2()** или **pmap()**.

Используйте двухэтапный процесс для создания вложенного data frame:

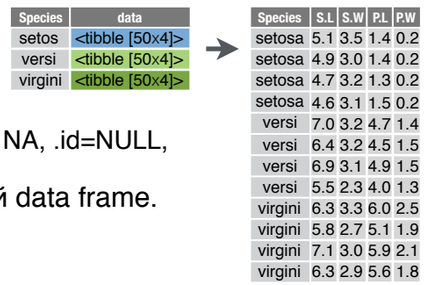
1. Сгруппируйте data frame в группы с помощью **dplyr::group_by()**
2. Используйте **nest()** для создания вложенного data frame с одной строкой на группу.



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

tidyr::**nest(**data, ..., .key = data**)**
Сворачивает группы в ячейки в виде data frame (для сгрупп. данных).

Разворачивайте вложенный data frame при помощи **unnest()**:



n_iris %>% **unnest**()

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

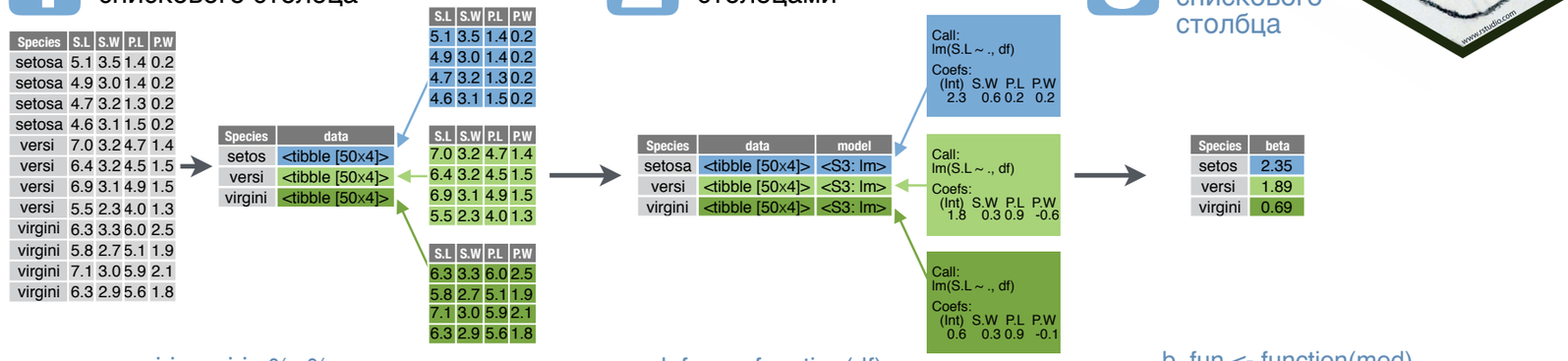
Процесс работы со списковыми столбцами

Вложенный data frame использует **списковый столбец**, т.е. список, который хранится как столбец в data frame. Типичный **рабочий процесс** со списковыми столбцами:

1 **Создание** спискового столбца

2 **Работа** со списковыми столбцами

3 **Упрощение** спискового столбца




```
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. СОЗДАНИЕ СПИСКОВОГО СТОЛБЦА - создавайте с помощью функций из пакетов **tibble** и **dplyr**, а также **tidyr**::nest()

tibble::**tribble(**…**)**
Создает списковый столбец при необходимости

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(**…**)**
Сохраняет список как списковый столбец.
tibble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))

tibble::**enframe(**x, name="name", value="value"**)**
Конвертирует многоуровневый список в tibble со списковыми столбцами.
enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')

dplyr::**mutate(**.data, …**)** Также **transmute()**
Возвращает списковый столбец, когда результат - список.
mtcars %>% **mutate**(seq = **map**(cyl, seq))

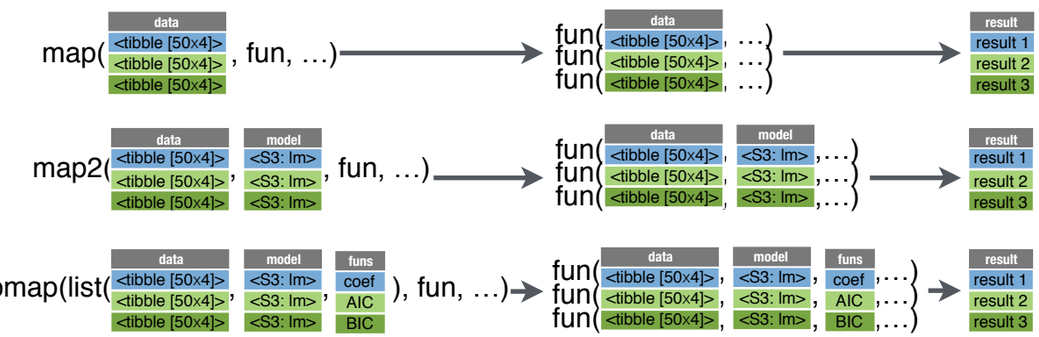
dplyr::**summarise(**.data, …**)**
Возвращает списковый столбец, когда результат создан с **list().**
mtcars %>% group_by(cyl) %>%
summarise(q = **list**(quantile(mpg)))

2. РАБОТА СО СПИСКОВЫМИ СТОЛБЦАМИ - Используйте функции purrr **map()**, **map2()** и **pmap()** для поэлементного применения функции с возвращением результата в списковый столбец. **walk()**, **walk2()** и **pwalk()** работают аналогично, но совершают побочные действия.

purrr::**map(**.x, .f, ...**)**
Применяет .f поэлементно к .x как .f(.x)
n_iris %>% **mutate**(n = **map**(data, dim))

purrr::**map2(**.x, .y, .f, ...**)**
Применяет .f поэлементно к .x и .y как .f(.x, .y)
m_iris %>% **mutate**(n = **map2**(data, model, list))

purrr::**pmap(**.l, .f, ...**)**
Применяет .f поэлементно к векторам из .l
m_iris %>%
mutate(n = **pmap**(list(data, model, data), list))



3. УПРОЩЕНИЕ СПИСКОВОГО СТОЛБЦА (в обычный столбец)

Используйте функции purrr **map_lgl()**, **map_int()**, **map_dbl()**, **map_chr()**, а также tidyr::**unnest()** для преобразования спискового столбца в обычный.

purrr::**map_lgl(**.x, .f, ...**)** Применяет .f поэл-но к .x, возвращает логич. вектор
n_iris %>% **transmute**(n = **map_lgl**(data, is.matrix))

purrr::**map_int(**.x, .f, ...**)** Применяет .f поэл-но к .x, возвращает целочисл. вектор
n_iris %>% **transmute**(n = **map_int**(data, nrow))

purrr::**map_dbl(**.x, .f, ...**)** Применяет .f поэл-но к .x, возвращает числ. вектор
n_iris %>% **transmute**(n = **map_dbl**(data, nrow))

purrr::**map_chr(**.x, .f, ...**)** Применяет .f поэл-но к .x, возвращает симв. вектор
n_iris %>% **transmute**(n = **map_chr**(data, nrow))