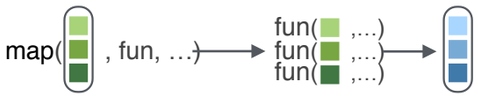
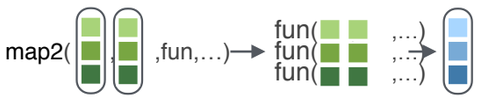
함수형 프로그래밍(purrr) : : 컨닝쪽지(CHEAT SHEET)

함수 적용

리스트 혹은 벡터 원소 각각에 대해 반복적으로 함수를 적용하는데 map 함수를 사용한다.



map(.x, .f, …) 리스트나 벡터 원소 각각에 함수를 적용시킴. map(x, is.logical)



map2(.x, ,y, .f, …) 리스트나 벡터가 두개인 경우 대응되는 원소 짝에 함수를 적용시킴. map2(x, y, sum)

pmap(.l, .f, …) 리스트의 리스트, 벡터에 원소 집단에 대해 함수를 적용시킴. pmap(list(x, y, z), sum, na.rm = TRUE)

invoke_map(.f, .x = list(NULL), …, .env=NULL) 내부 리스트에 함수 각각을 실행시킴. 다음도 가능. l <- list(var, sd); invoke_map(l, x = 1:9)

lmap(.x, .f, ...) 리스트나 벡터 리스트-원소 각각에 대해 함수를 적용.
imap(.x, .f, ...) 리스트나 벡터 원소와 대응되는 인덱스에 .f 함수를 적용.

출력결과

map(), **map2()**, **pmap()**, **imap**, **invoke_map** 모두 리스트를 반환한다. 특정 유형 벡터로 결과를 반환하려면 아래첨자를 사용. 예를 들어, **map2_chr**, **pmap_lgl**, 등.

부작용(side effects)을 일으키려면 **walk**, **walk2**, **pwalk** 함수를 사용. 보이지 않게 입력값을 반환한다.

함수명	반환값
map	리스트
map_ch	문자 벡터
map_dbl	더블 (숫자형) 벡터
map_dfc	데이터프레임 (열기준 합치기)
map_dfr	데이터프레임 (행기준 합치기)
map_int	정수형 벡터
map_lgl	논리형 벡터
walk	부작용(side effects)을 일으키고, 보이지 않게 입력을 반환

축약(SHORTCUTS) - purrr 함수 범위내부

"name" 은 다음과 같이 대치 **function(x) x$name.** 즉, map(l, "a") 경우 리스트 l 각각에서 원소 $a 를 추출

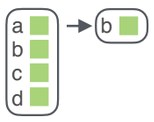
~ . 은 **function(x)** x 함수에 적용됨. 즉, map(l, ~ 2 +.) 경우, map(l, function(x) 2 + x) 와 같음.

~ .x .y 은 **function(.x, .y)** .x .y 함수에 적용됨. 즉, map2(l, p, ~ .x +.y) 경우, map2(l, p, function(l, p) l + p)와 같음.

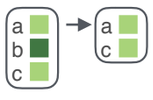
~ ..1 ..2 .. 은 **function(..1, ..2, ..)** 함수에 적용됨. 즉, pmap(list(a, b, c), ~ ..3 + ..1 - ..2) 경우 pmap(list(a, b, c), function(a, b, c) c + a - b) 와 같음.

리스트 작업

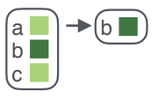
리스트 필터링



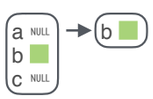
pluck(.x, ..., .default=NULL) 명칭 혹은 인덱스로 리스트 원소 추출 pluck(x,"b"), 혹은 속성 **attr_getter** 함수로 다음과 같이 pluck(x,"b",attr_getter("n")).



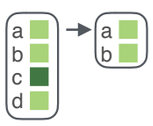
keep(.x, .p, …) 논리 테스트를 통과하는 원소만 추출. keep(x, is.na)



discard(.x, .p, …) 논리 테스트를 통과하지 않는 원소만 추출. discard(x, is.na)

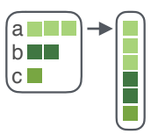


compact(.x, .p = identity) 빈 원소를 빼냄. compact(x)

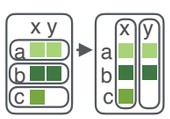


head_while(.x, .p, …) 테스트를 통과하지 않을 때까지 첫머리 원소를 반환. **tail_while**은 꼬리. head_while(x, is.character)

리스트 모양바꾸기



flatten(.x) 리스트에서 인덱스 혹은 범주(level)을 제거함. **flatten_chr**, **flatten_dbl**, **flatten_dfc**, **flatten_dfr**, **flatten_int**, **flatten_lgl**. flatten(x)



transpose(.l, .names = NULL) 다중범주(multi-level) 리스트에 인덱스 순서를 전치시킴. transpose(x)

리스트 요약하기

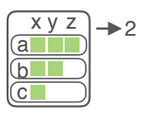
every(.x, .p, …) 모든 원소가 테스트를 통과했는가? every(x, is.character)

some(.x, .p, …) 일부 원소가 테스트를 통과했는가? some(x, is.character)

has_element(.x, .y) 리스트가 특정 원소를 포함하고 있는가? has_element(x, "foo")

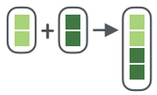
detect(.x, .f, ..., .right=FALSE, .p) 테스트를 통과하는 첫번째 원소를 찾아라. detect(x, is.character)

detect_index(.x, .f, ..., .right = FALSE, .p) 테스트를 통과하는 첫번째 원소 인덱스를 찾아라. detect_index(x, is.character)



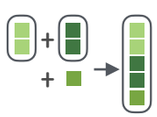
vec_depth(x) 리스트 깊이를 반환. (인덱스 수준 갯수). vec_depth(x)

리스트 병합하기



append(x, values, after = length(x)) 리스트 끝에 추가. append(x, list(d = 1))

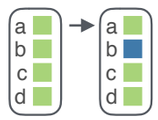
prepend(x, values, before = 1) 리스트 시작에 추가. prepend(x, list(d = 1))



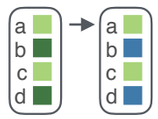
splice(…) 객체를 리스트와 결합해서 하위-리스트로 S3 객체 저장. splice(x, y, "foo")

리스트 변환하기

modify(.x, .f, ...) 리스트 원소 각각에 함수 적용. **map**, **map_chr**, **map_dbl**, **map_dfc**, **map_dfr**, **map_int**, **map_lgl** 동일. modify(x, ~.+ 2)



modify_at(.x, .at, .f, ...) 명칭 혹은 인덱스로 리스트 원소에 함수를 적용. **map_at**. modify_at(x, "b", ~.+ 2)

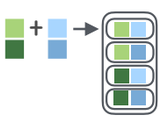


modify_if(.x, .p, .f, ...) 테스트를 통과하는 리스트 원소에 함수를 적용. **map_if**. modify_if(x, is.numeric,~.+2)

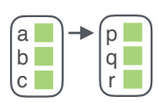
modify_depth(.x,.depth,.f,...) 주어진 리스트 범주(level)에 위치한 각 원소에 함수를 적용. modify_depth(x, 1, ~.+ 2)

리스트로 작업하기

array_tree(array, margin = NULL) 배열을 리스트로 변환. **array_branch** 동일. array_tree(x, margin = 3)

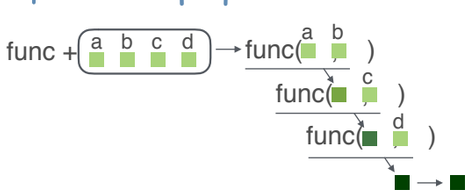


cross2(.x, .y, .filter = NULL) .x 와 .y의 모든 조합. **cross**, **cross3**, **cross_df** 동일. cross2(1:3, 4:6)

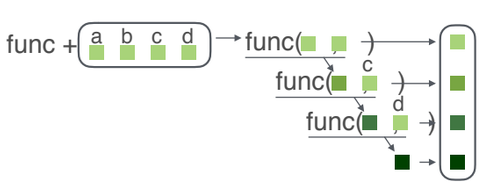


set_names(x, nm = x) 함수로 혹은 벡터/리스트 명칭으로 직접 이름을 설정. set_names(x, c("p", "q", "r")) set_names(x, tolower)

리스트 축약



reduce(.x, .f, ..., .init) 리스트 혹은 벡터 각 원소에 재귀적으로 함수를 적용. **reduce_right**, **reduce2**, **reduce2_right** 동일. reduce(x, sum)



accumulate(.x, .f, ..., .init) reduce와 동일하게 축약하지만, 중간 결과값을 반환. **accumulate_right** 동일. accumulate(x, sum)

함수 동작 변경

compose() 함수 다수를 조합.

lift() 함수가 입력값으로 받는 유형을 변경. **lift_dl**, **lift_dv**, **lift_ld**, **lift_lv**, **lift_vd**, **lift_vl** 동일.

rerun() 표현식을 n 번 재실행.

negate() 서술자 함수 (predicate function)를 부정 (파이프 친화적으로 !)

partial() 부분적으로 함수에 적용, 일부 인자만 채워 넣음.

safely() 함수(func)를 변경시켜서 결과와 오류 리스트를 반환.

quietly() 함수(func)를 변경시켜 결과, 출력, 메시지, 경고 리스트를 반환

possibly() 오류가 발생할 때마다, (오류 대신에) 함수(func)를 변경시켜 기본설정값을 반환.

중첩된 데이터

중첩된 데이터프레임(nested data frame)은 더 커다랗고 잘 조직된 표내부에 개별적인 표를 저장한다.

중첩된 데이터프레임

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

n_iris

"셀(cell)"에 담긴 내용

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

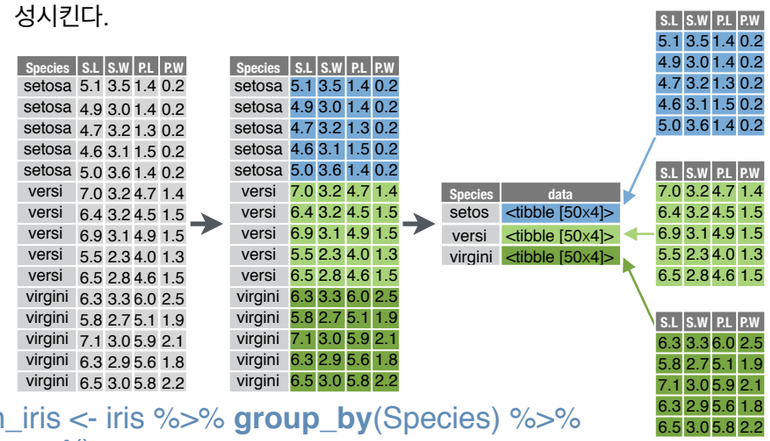
n_iris$data[[3]]

중첩된 데이터프레임 활용:

- 관측점과 데이터의 하위집합(subset) 사이 관계를 유지
- **purrr** 함수 **map()**, **map2()**, **pmap()**를 사용해서 한번에 많은 하위표(sub-table)를 조작

두 단계 과정을 거쳐 중첩된 데이터프레임을 생성시킨다:

1. **dplyr::group_by()** 명령어로 그룹으로 데이터프레임을 묶는다.
2. **nest()** 명령어를 사용해서 한행에 그룹을 갖는 중첩 데이터프레임을 생성시킨다.




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

tidyr::**nest(**data, ..., .key = data**)**
그룹으로 묶인 데이터에 대해서, 그룹을 데이터프레임 셀로 이동.

unnest() 명령어로 중첩된 데이터프레임의 중첩을 푼다:

```
n_iris %>%
  unnest()
```

tidyr::**unnest(**data, ..., .drop = NA, .id=NULL, .sep=NULL**)**
중첩된 데이터프레임의 중첩을 푼다.

Species	data
setos	<tibble [50x4]>
versi	<tibble [50x4]>
virgini	<tibble [50x4]>

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
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	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

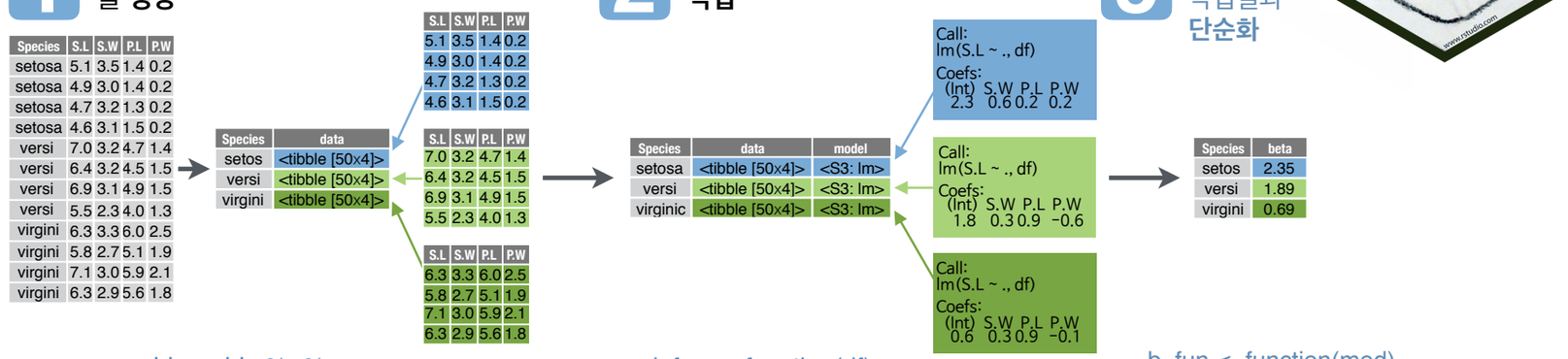
리스트열(List Column) 작업흐름

중첩된 데이터프레임은 **리스트 칼럼**(list column)을 사용한다. 리스트 칼럼은 리스트가 데이터프레임의 열벡터로 저장된다. 전형적인 **리스트 칼럼**에 대한 작업흐름은 다음과 같다:

1 리스트 칼럼을 **생성**

2 리스트 칼럼 **작업**

3 리스트 칼럼 작업결과 **단순화**




```
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. 리스트 칼럼 생성 - 리스트 칼럼을 생성하는데 **tibble**, **dplyr** 팩키지 함수 뿐만 아니라, **tidyr** 팩키지 nest() 함수도 사용한다.

tibble::**tribble(**…**)**
필요하면 다음과 같이 리스트 칼럼 생성

```
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(**…**)**
리스트 입력을 넣어 리스트 칼럼으로 저장

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

tibble::**enframe(**x, name="name", value="value"**)**
다단계(multi-level) 리스트를 리스트칼럼을 갖는 tibble로 변환

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

dplyr::**mutate(**.data, …**), transmute()** 동일
결과가 리스트를 반환하면, 리스트 칼럼을 자동 반환.

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

dplyr::**summarise(**.data, …**)**
결과가 **list()**로 감싸졌을 때 리스트 칼럼을 반환.

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

2. 리스트 칼럼 작업 - purrr **map()**, **map2()**, **pmap()** 함수를 사용해서 리스트 칼럼의 셀에 원소별로 함수를 적용해서 반환되는 결과를 저장한다. **walk()**, **walk2()**, **pwalk()**도 동일하게 동작하지만, 부작용(side effect)를 반환

purrr::**map(**.x, .f, ...**)**
.f(.x) 처럼 .f 함수를 .x 원소별로 적용시킨다.

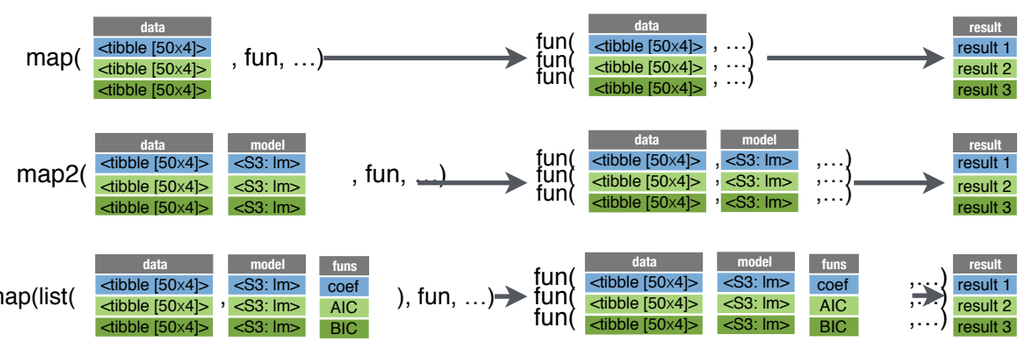
```
n_iris %>% mutate(n = map(data, dim))
```

purrr::**map2(**.x, .y, .f, ...**)**
.f(.x,.y) 처럼 .f 함수를 .x 와 .y 원소별로 적용시킨다.

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

purrr::**pmap(**.l, .f, ...**)**
.f 함수를 .x 원소별로 적용시켜 .l 에 저장시킨다.

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



3. (정규 칼럼 형태로) 리스트 칼럼 작업결과 단순화

purrr **map_lgl()**, **map_int()**, **map_dbl()**, **map_chr()**, 함수뿐만 아니라, tidyr **unnest()** 함수를 사용해서 리스트 칼럼을 정규 칼럼형태로 단순화

purrr::**map_lgl(**.x, .f, ...**)**
.f 함수를 .x 원소별로 적용해서, 논리벡터를 반환

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

purrr::**map_int(**.x, .f, ...**)**
.f 함수를 .x 원소별로 적용해서, 정수벡터를 반환

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

purrr::**map_dbl(**.x, .f, ...**)**
.f 함수를 .x 원소별로 적용해서, 숫자(double)벡터를 반환

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

purrr::**map_chr(**.x, .f, ...**)**
.f 함수를 .x 원소별로 적용해서, 문자벡터를 반환

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