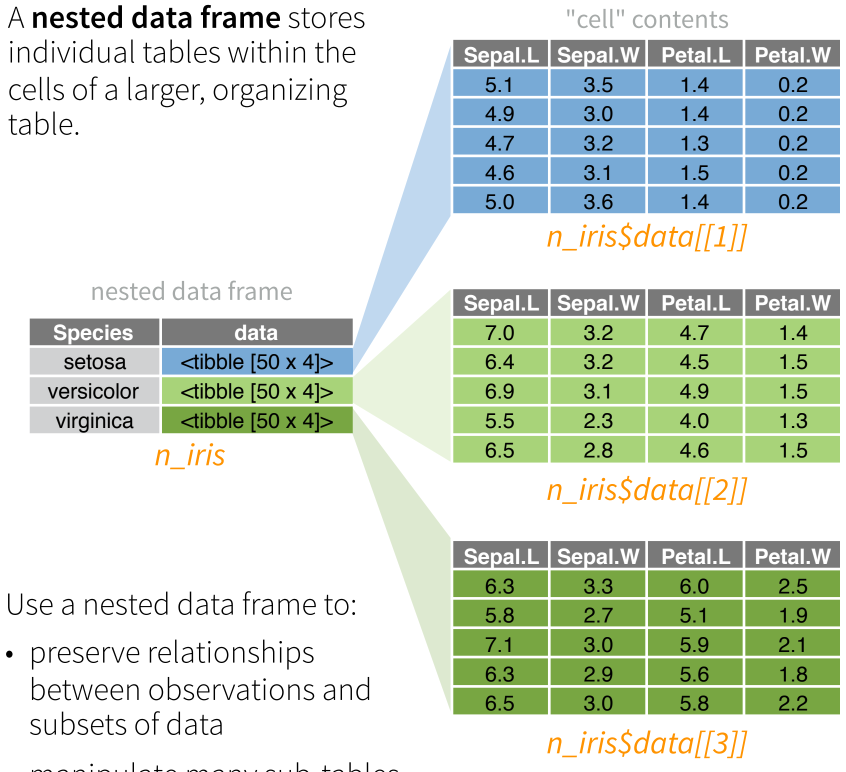
Nested Data

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



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]]

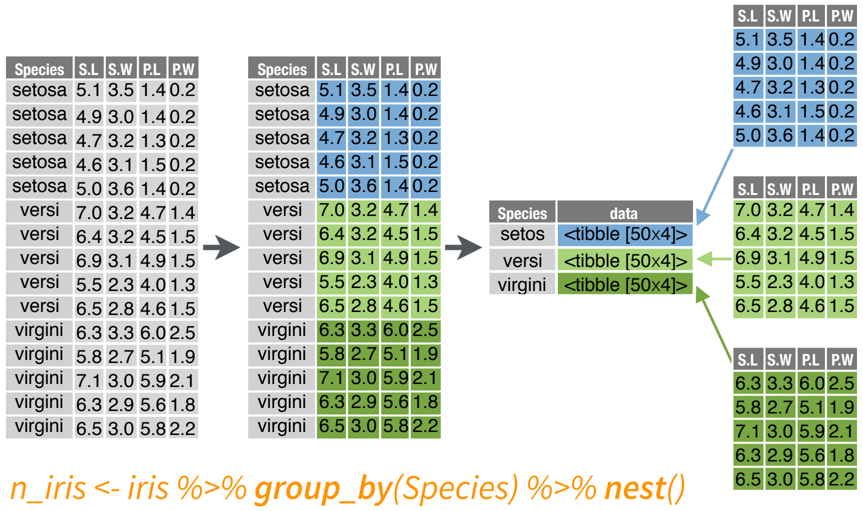
Sepal.L	Sepal.W	Petal.L	Petal.W
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
6.3	2.9	5.6	1.8
6.5	3.0	5.8	2.2

n_iris$data[[3]]

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:
1. Group the data frame into groups with **dplyr::group_by()**
2. Use **nest()** to create a nested data frame with one row per group



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
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
versi	6.5	2.8	4.6	1.5
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
virgini	6.5	3.0	5.8	2.2

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
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
versi	6.5	2.8	4.6	1.5
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
virgini	6.5	3.0	5.8	2.2

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

S.L	S.W	P.L	P.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

S.L	S.W	P.L	P.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

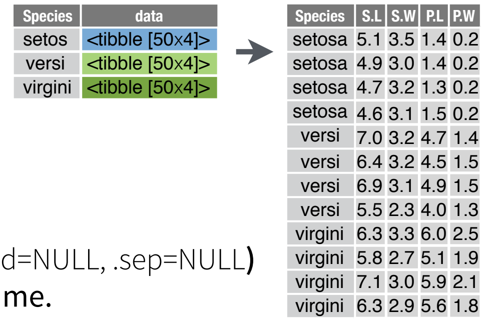
S.L	S.W	P.L	P.W
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
6.3	2.9	5.6	1.8
6.5	3.0	5.8	2.2

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

nest(data, ..., .key = data)

For grouped data, moves groups into cells as data frames.

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



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

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

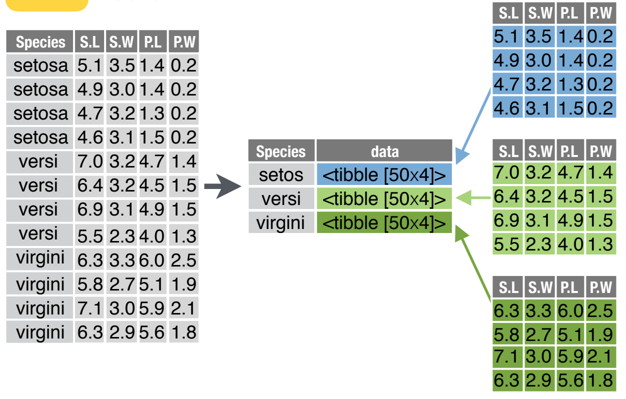
unnest(data, ..., .drop = NA, .id=NULL, .sep=NULL)

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
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

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

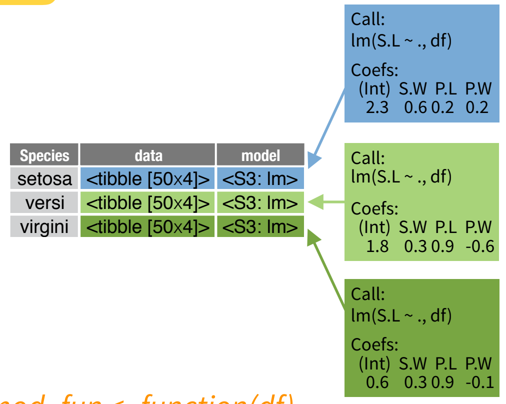
S.L	S.W	P.L	P.W
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2

S.L	S.W	P.L	P.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

S.L	S.W	P.L	P.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

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

2 Work with list columns



Species	data	model
setosa	<tibble [50x4]>	<S3: lm>
versi	<tibble [50x4]>	<S3: lm>
virgini	<tibble [50x4]>	<S3: lm>

```
Call:
lm(S.L ~ ., df)

Coefs:
  (Int)  S.W  P.L  P.W
   2.3   0.6  0.2  0.2
```

```
Call:
lm(S.L ~ ., df)

Coefs:
  (Int)  S.W  P.L  P.W
   1.8   0.3  0.9  -0.6
```

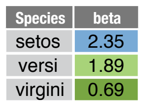
```
Call:
lm(S.L ~ ., df)

Coefs:
  (Int)  S.W  P.L  P.W
   0.6   0.3  0.9  -0.1
```

```
mod_fun <- function(df)
  lm(Sepal.Length ~ ., data = df)

m_iris <- n_iris %>%
  mutate(model = map(data, mod_fun))
```

3 Simplify the list column



Species	beta
setos	2.35
versi	1.89
virgini	0.69

```
b_fun <- function(mod)
  coefficients(mod)[[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
)
```

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

tibble::**tibble**(…)

Saves list input as list columns

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

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::**mutate**(.data, …) Also **transmute()**

Returns list col when result returns list.

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

dplyr::**summarise**(.data, …)

Returns list col when result is wrapped with **list()**

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

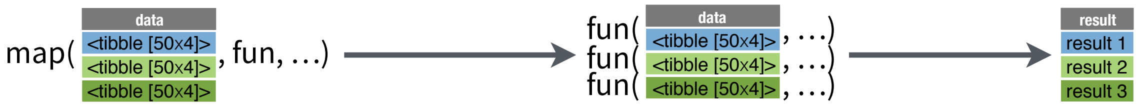
2. Work with list columns

Use **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. Each of these is in the **purrr** package.

purrr::**map**(.x, .f, ...)

Apply .f element-wise to .x as .f(.x)

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

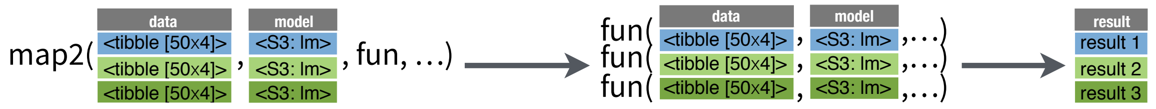


map(data <tibble [50x4]> <tibble [50x4]> <tibble [50x4]> , fun, …)
fun(<tibble [50x4]> , …)
fun(<tibble [50x4]> , …)
fun(<tibble [50x4]> , …)
result
result 1
result 2
result 3

purrr::**map2**(.x, .y, .f, ...)

Apply .f element-wise to .x and .y as .f(.x, .y)

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

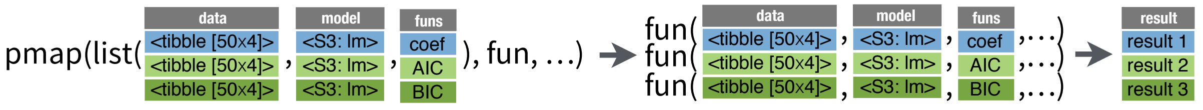


map2(data <tibble [50x4]> <tibble [50x4]> <tibble [50x4]> , model <S3: lm> <S3: lm> <S3: lm> , fun, …)
fun(<tibble [50x4]> , <S3: lm> ,…)
fun(<tibble [50x4]> , <S3: lm> ,…)
fun(<tibble [50x4]> , <S3: lm> ,…)
result
result 1
result 2
result 3

purrr::**pmap**(.l, .f, ...)

Apply .f element-wise to vectors saved in .l

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



pmap(list(data <tibble [50x4]> <tibble [50x4]> <tibble [50x4]> , model <S3: lm> <S3: lm> <S3: lm> , funs coef AIC BIC), fun, …)
fun(<tibble [50x4]> , <S3: lm> , coef ,…)
fun(<tibble [50x4]> , <S3: lm> , AIC ,…)
fun(<tibble [50x4]> , <S3: lm> , BIC ,…)
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()**, and **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_int(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))
```

Learn more at browseVignettes(package = c("tidyr")) • tidyr 0.6.0 • Updated: 11/16