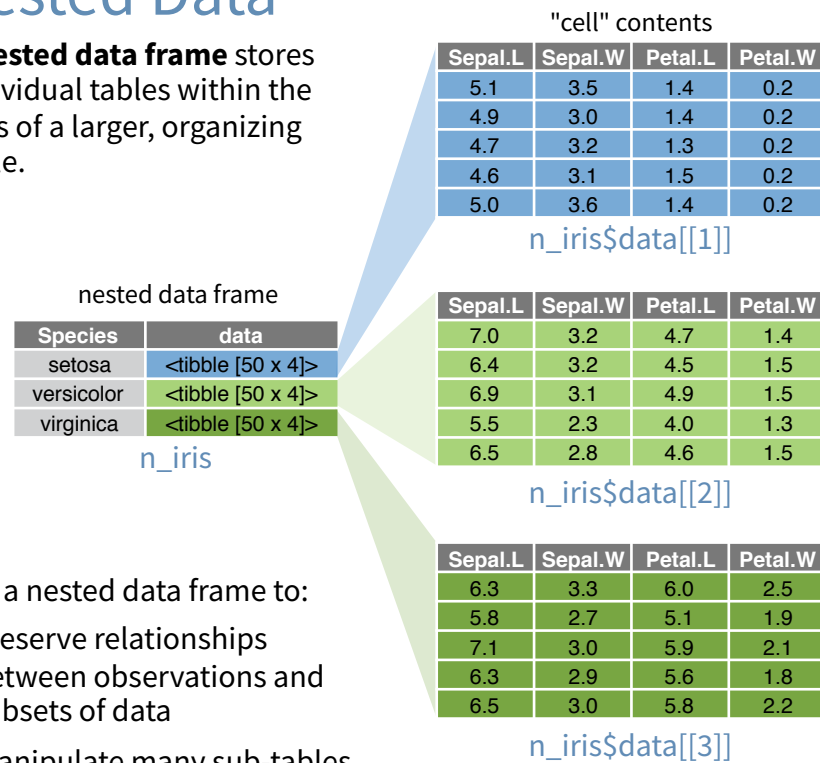


RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at [purrr.tidyverse.org](https://purrr.tidyverse.org) • purrr 0.2.3 • Updated: 2017-09

# Nested Data

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



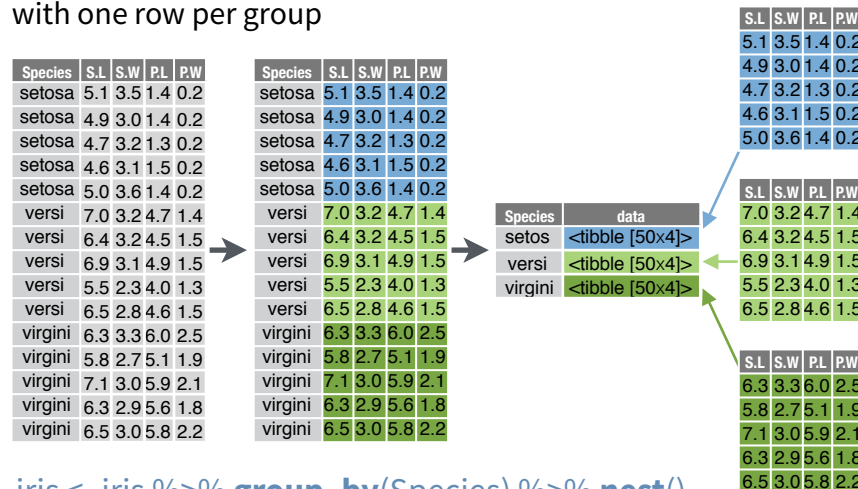
Use a nested data frame to:

- 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



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

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

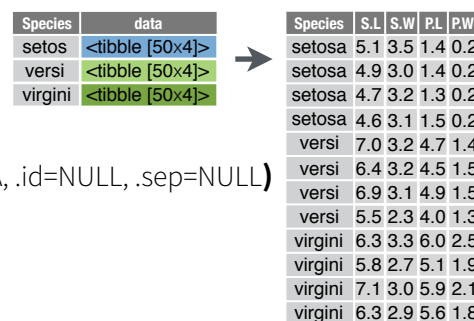
For grouped data, moves groups into cells as data frames.

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

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

**tidyr::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 |
| setosa    | 5.0 | 3.6 | 1.4 | 0.2 |
| versicol  | 7.0 | 3.2 | 4.7 | 1.4 |
| versicol  | 6.4 | 3.2 | 4.5 | 1.5 |
| versicol  | 6.9 | 3.1 | 4.9 | 1.5 |
| versicol  | 5.5 | 2.3 | 4.0 | 1.3 |
| virginica | 6.3 | 3.3 | 6.0 | 2.5 |
| virginica | 5.8 | 2.7 | 5.1 | 1.9 |
| virginica | 7.1 | 3.0 | 5.9 | 2.1 |
| virginica | 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> |
| versicol  | <tibble [50x4]> | <S3: lm> |
| virginica | <tibble [50x4]> | <S3: lm> |

```
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 |
|-----------|------|
| setosa    | 2.35 |
| versicol  | 1.89 |
| virginica | 0.69 |

```
b_fun <- function(mod)
  coefficients(mod)[1,1]

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 the **purrr** functions **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.

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

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

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

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

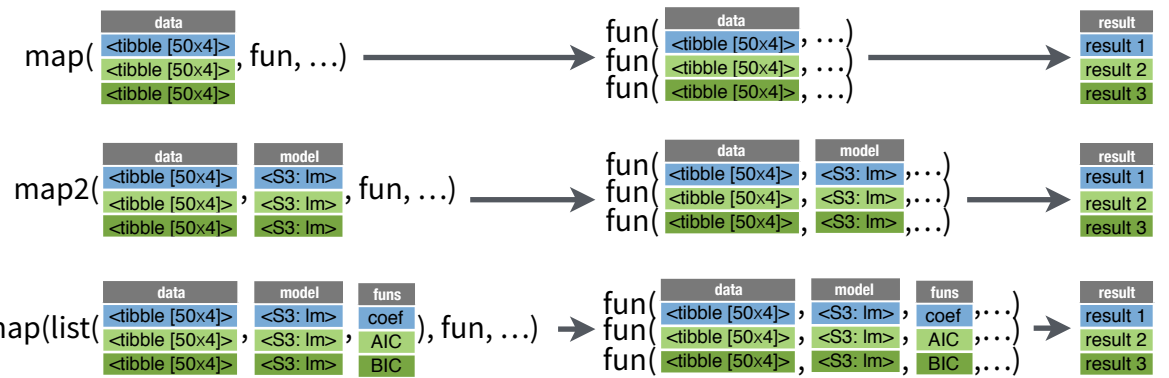
Apply .f element-wise to .x and .y as .f(.x, .y)

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

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

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

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



**3. SIMPLIFY THE LIST COLUMN** (into a regular column)

Use the **purrr** functions **map\_lgl()**, **map\_int()**, **map\_dbl()**, **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_dbl(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))
```