

Package ‘nplyr’

July 18, 2022

Title A Grammar of Nested Data Manipulation

Version 0.1.0

Description Provides functions for manipulating nested data frames in a list-column using 'dplyr' <<https://dplyr.tidyverse.org/>> syntax. Rather than unnesting, then manipulating a data frame, 'nplyr' allows users to manipulate each nested data frame directly. 'nplyr' is a wrapper for 'dplyr' functions that provide tools for common data manipulation steps: filtering rows, selecting columns, summarising grouped data, among others.

License MIT + file LICENSE

URL <https://github.com/markjrieke/nplyr>,
<https://markjrieke.github.io/nplyr/>

BugReports <https://github.com/markjrieke/nplyr/issues>

Depends dplyr, R (>= 2.10)

Imports assertthat, purrr, rlang

Encoding UTF-8

RoxygenNote 7.2.0

Suggests gapminder, knitr, readr, rmarkdown, stringr, testthat (>= 3.0.0), tibble, tidyr

Config/testthat/edition 3

VignetteBuilder knitr

LazyData true

NeedsCompilation no

Author Mark Rieke [aut, cre]

Maintainer Mark Rieke <markjrieke@gmail.com>

Repository CRAN

Date/Publication 2022-07-18 12:10:02 UTC

R topics documented:

job_survey	2
nest-filter-joins	3
nest-mutate-joins	4
nest_arrange	8
nest_count	9
nest_distinct	10
nest_filter	11
nest_group_by	12
nest_mutate	14
nest_nest_join	16
nest_relocate	17
nest_rename	18
nest_select	20
nest_slice	21
nest_summarise	23
personal_survey	25
Index	26

job_survey	<i>Example survey data regarding job satisfaction</i>
------------	---

Description

A toy dataset containing 500 responses to a job satisfaction survey. The responses were randomly generated using the Qualtrics survey platform.

Usage

```
job_survey
```

Format

A data frame with 500 rows and 6 variables:

survey_name name of survey

Q1 respondent age

Q2 city the respondent resides in

Q3 field that the respondent that works in

Q4 respondent's job satisfaction (on a scale from extremely satisfied to extremely dissatisfied)

Q5 respondent's annual salary, in thousands of dollars

nest-filter-joins *Nested filtering joins*

Description

Nested filtering joins filter rows from `.nest_data` based on the presence or absence of matches in `y`:

- `nest_semi_join()` returns all rows from `.nest_data` with a match in `y`.
- `nest_anti_join()` returns all rows from `.nest_data` *without* a match in `y`.

Usage

```
nest_semi_join(.data, .nest_data, y, by = NULL, copy = FALSE, ...)
```

```
nest_anti_join(.data, .nest_data, y, by = NULL, copy = FALSE, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>y</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>by</code>	<p>A character vector of variables to join by or a join specification created with <code>join_by()</code>.</p> <p>If <code>NULL</code>, the default, <code>nest_*_join()</code> will perform a natural join, using all variables in common across each object in <code>.nest_data</code> and <code>y</code>. A message lists the variables so you can check they're correct; suppress the message by supplying <code>by</code> explicitly.</p> <p>To join on different variables between the objects in <code>.nest_data</code> and <code>y</code>, use a named vector. For example, <code>by = c("a" = "b")</code> will match <code>.nest_data\$a</code> to <code>y\$b</code> for each object in <code>.nest_data</code>.</p> <p>To join by multiple variables, use a vector with length >1. For example, <code>by = c("a", "b")</code> will match <code>.nest_data\$a</code> to <code>y\$a</code> and <code>.nest_data\$b</code> to <code>y\$b</code> for each object in <code>.nest_data</code>. Use a named vector to match different variables in <code>.nest_data</code> and <code>y</code>. For example, <code>by = c("a" = "b", "c" = "d")</code> will match <code>.nest_data\$a</code> to <code>y\$b</code> and <code>.nest_data\$c</code> to <code>y\$d</code> for each object in <code>.nest_data</code>.</p> <p>To perform a cross-join, generating all combinations of each object in <code>.nest_data</code> and <code>y</code>, use <code>by = character()</code>.</p>
<code>copy</code>	If <code>.nest_data</code> and <code>y</code> are not from the same data source and <code>copy = TRUE</code> then <code>y</code> will be copied into the same src as <code>.nest_data</code> . (<i>Need to review this parameter in more detail for applicability with <code>nplyr</code></i>)
<code>...</code>	One or more unquoted expressions separated by commas. Variable names can be used if they were positions in the data frame, so expressions like <code>x:y</code> can be used to select a range of variables.

Details

`nest_semi_join()` and `nest_anti_join()` are largely wrappers for `dplyr::semi_join()` and `dplyr::anti_join()` and maintain the functionality of `semi_join()` and `anti_join()` within each nested data frame. For more information on `semi_join()` or `anti_join()`, please refer to the documentation in [dplyr](#).

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. Each object in `.nest_data` has the following properties:

- Rows are a subset of the input, but appear in the same order.
- Columns are not modified.
- Data frame attributes are preserved.
- Groups are taken from `.nest_data`. The number of groups may be reduced.

See Also

Other joins: [nest-mutate-joins](#), [nest_nest_join\(\)](#)

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)
gm_codes <- gapminder::country_codes %>% dplyr::slice_sample(n = 10)

gm_nest %>% nest_semi_join(country_data, gm_codes, by = "country")
gm_nest %>% nest_anti_join(country_data, gm_codes, by = "country")
```

nest-mutate-joins *Nested Mutating joins*

Description

Nested mutating joins add columns from `y` to each of the nested data frames in `.nest_data`, matching observations based on the keys. There are four nested mutating joins:

Inner join:

`nest_inner_join()` only keeps observations from `.nest_data` that have a matching key in `y`.

The most important property of an inner join is that unmatched rows in either input are not included in the result.

Outer joins:

There are three outer joins that keep observations that appear in at least one of the data frames:

- `nest_left_join()` keeps all observations in `.nest_data`.
- `nest_right_join()` keeps all observations in `y`.
- `nest_full_join()` keeps all observations in `.nest_data` and `y`.

Usage

```
nest_inner_join(  
  .data,  
  .nest_data,  
  y,  
  by = NULL,  
  copy = FALSE,  
  suffix = c(".x", ".y"),  
  ...,  
  keep = FALSE  
)
```

```
nest_left_join(  
  .data,  
  .nest_data,  
  y,  
  by = NULL,  
  copy = FALSE,  
  suffix = c(".x", ".y"),  
  ...,  
  keep = FALSE  
)
```

```
nest_right_join(  
  .data,  
  .nest_data,  
  y,  
  by = NULL,  
  copy = FALSE,  
  suffix = c(".x", ".y"),  
  ...,  
  keep = FALSE  
)
```

```
nest_full_join(  
  .data,  
  .nest_data,  
  y,  
  by = NULL,  
  copy = FALSE,  
  suffix = c(".x", ".y"),  
  ...,  
  keep = FALSE  
)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>y</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>by</code>	<p>A character vector of variables to join by or a join specification created with <code>join_by()</code>.</p> <p>If NULL, the default, <code>nest_*_join()</code> will perform a natural join, using all variables in common across each object in <code>.nest_data</code> and <code>y</code>. A message lists the variables so you can check they're correct; suppress the message by supplying <code>by</code> explicitly.</p> <p>To join on different variables between the objects in <code>.nest_data</code> and <code>y</code>, use a named vector. For example, <code>by = c("a" = "b")</code> will match <code>.nest_data\$a</code> to <code>y\$b</code> for each object in <code>.nest_data</code>.</p> <p>To join by multiple variables, use a vector with length >1. For example, <code>by = c("a", "b")</code> will match <code>.nest_data\$a</code> to <code>y\$a</code> and <code>.nest_data\$b</code> to <code>y\$b</code> for each object in <code>.nest_data</code>. Use a named vector to match different variables in <code>.nest_data</code> and <code>y</code>. For example, <code>by = c("a" = "b", "c" = "d")</code> will match <code>.nest_data\$a</code> to <code>y\$b</code> and <code>.nest_data\$c</code> to <code>y\$d</code> for each object in <code>.nest_data</code>.</p> <p>To perform a cross-join, generating all combinations of each object in <code>.nest_data</code> and <code>y</code>, use <code>by = character()</code>.</p>
<code>copy</code>	If <code>.nest_data</code> and <code>y</code> are not from the same data source and <code>copy = TRUE</code> then <code>y</code> will be copied into the same src as <code>.nest_data</code> . (<i>Need to review this parameter in more detail for applicability with <code>nplyr</code></i>)
<code>suffix</code>	If there are non-joined duplicate variables in <code>.nest_data</code> and <code>y</code> , these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
<code>...</code>	<p>Other parameters passed onto methods. Includes:</p> <ul style="list-style-type: none"> • <code>na_matches</code> : Should two NA or two NaN values match? <ul style="list-style-type: none"> – <code>"na"</code>, the default, treats two NA or two NaN values as equal. – <code>"never"</code> treats two NA or two NaN values as different, and will never match them together or to any other values. • <code>multiple</code> : Handling of rows in <code>.nest_data</code> with multiple matches in <code>y</code>. <ul style="list-style-type: none"> – <code>"all"</code> returns every match detected in <code>y</code>. – <code>"any"</code> returns one match detected in <code>y</code>, with no guarantees on which match will be returned. It is often faster than <code>"first"</code> and <code>"last"</code> if you just need to detect if there is at least one match. – <code>"first"</code> returns the first match detected in <code>y</code>. – <code>"last"</code> returns the last match detected in <code>y</code>. – <code>"warning"</code> throws a warning if multiple matches are detected, and then falls back to <code>"all"</code>. – <code>"error"</code> throws an error if multiple matches are detected.

- `unmatched` : How should unmatched keys that would result in dropped rows be handled?
 - “drop” drops unmatched keys from the result.
 - “error” throws an error if unmatched keys are detected.

`keep` Should the join keys from both `.nest_data` and `y` be preserved in the output?

Details

`nest_inner_join()`, `nest_left_join()`, `nest_right_join()`, and `nest_full_join()` are largely wrappers for `dplyr::inner_join()`, `dplyr::left_join()`, `dplyr::right_join()`, and `dplyr::full_join()` and maintain the functionality of these verbs within each nested data frame. For more information on `inner_join()`, `left_join()`, `right_join()`, or `full_join()`, please refer to the documentation in [dplyr](#).

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. The order of the rows and columns of each object in `.nest_data` is preserved as much as possible. Each object in `.nest_data` has the following properties:

- For `nest_inner_join()`, a subset of rows in each object in `.nest_data`. For `nest_left_join()`, all rows in each object in `.nest_data`. For `nest_right_join()`, a subset of rows in each object in `.nest_data`, followed by unmatched `y` rows. For `nest_full_join()`, all rows in each object in `.nest_data`, followed by unmatched `y` rows.
- Output columns include all columns from each `.nest_data` and all non-key columns from `y`. If `keep = TRUE`, the key columns from `y` are included as well.
- If non-key columns in any object in `.nest_data` and `y` have the same name, suffixes are added to disambiguate. If `keep = TRUE` and key columns in `.nest_data` and `y` have the same name, suffixes are added to disambiguate these as well.
- If `keep = FALSE`, output columns included in by are coerced to their common type between the objects in `.nest_data` and `y`.
- Groups are taken from `.nest_data`.

See Also

Other joins: [nest-filter-joins](#), [nest_nest_join\(\)](#)

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)
gm_codes <- gapminder::country_codes

gm_nest %>% nest_inner_join(country_data, gm_codes, by = "country")
gm_nest %>% nest_left_join(country_data, gm_codes, by = "country")
gm_nest %>% nest_right_join(country_data, gm_codes, by = "country")
gm_nest %>% nest_full_join(country_data, gm_codes, by = "country")
```

nest_arrange	<i>Arrange rows within a nested data frames by column values</i>
--------------	--

Description

nest_arrange() orders the rows of nested data frames by the values of selected columns.

Usage

```
nest_arrange(.data, .nest_data, ..., .by_group = FALSE)
```

Arguments

.data	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from dbplyr or dtplyr).
.nest_data	A list-column containing data frames
...	Variables, or functions of variables. Use <code>dplyr::desc()</code> to sort a variable in descending order.
.by_group	If TRUE, will sort first by grouping variable. Applies to grouped data frames only.

Details

nest_arrange() is largely a wrapper for `dplyr::arrange()` and maintains the functionality of `arrange()` within each nested data frame. For more information on `arrange()`, please refer to the documentation in `dplyr`.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will be also of the same type as the input. Each object in `.nest_data` has the following properties:

- All rows appear in the output, but (usually) in a different place.
- Columns are not modified.
- Groups are not modified.
- Data frame attributes are preserved.

See Also

Other single table verbs: `nest_filter()`, `nest_mutate()`, `nest_rename()`, `nest_select()`, `nest_slice()`, `nest_summarise()`

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)

gm_nest %>%
  nest_arrange(country_data, pop)

gm_nest %>%
  nest_arrange(country_data, desc(pop))
```

nest_count	<i>Count observations in a nested data frame by group</i>
------------	---

Description

nest_count() lets you quickly count the unique values of one or more variables within each nested data frame. nest_count() results in a summary with one row per each set of variables to count by. nest_add_count() is equivalent with the exception that it retains all rows and adds a new column with group-wise counts.

Usage

```
nest_count(.data, .nest_data, ..., wt = NULL, sort = FALSE, name = NULL)

nest_add_count(.data, .nest_data, ..., wt = NULL, sort = FALSE, name = NULL)
```

Arguments

.data	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from dbplyr or dtplyr).
.nest_data	A list-column containing data frames
...	Variables to group by.
wt	Frequency weights. Can be NULL or a variable: <ul style="list-style-type: none"> • If NULL (the default), counts the number of rows in each group. • If a variable, computes sum(wt) for each group.
sort	If TRUE, will show the largest groups at the top.
name	The name of the new column in the output.

Details

nest_count() and nest_add_count() are largely wrappers for [dplyr::count\(\)](#) and [dplyr::add_count\(\)](#) and maintain the functionality of count() and add_count() within each nested data frame. For more information on count() and add_count(), please refer to the documentation in [dplyr](#).

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. `nest_count()` and `nest_add_count()` group each object in `.nest_data` transiently, so the output returned in `.nest_data` will have the same groups as the input.

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = ~continent)

# count the number of times each country appears in each nested tibble
gm_nest %>% nest_count(country_data, country)
gm_nest %>% nest_add_count(country_data, country)

# count the sum of population for each country in each nested tibble
gm_nest %>% nest_count(country_data, country, wt = pop)
gm_nest %>% nest_add_count(country_data, country, wt = pop)
```

 nest_distinct

Subset distinct/unique rows within a nested data frame

Description

`nest_distinct()` selects only unique/distinct rows in a nested data frame.

Usage

```
nest_distinct(.data, .nest_data, ..., .keep_all = FALSE)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>...</code>	Optional variables to use when determining uniqueness. If there are multiple rows for a given combination of inputs, only the first row will be preserved. If omitted, will use all variables.
<code>.keep_all</code>	If <code>TRUE</code> , keep all variables in <code>.nest_data</code> . If a combination of <code>...</code> is not distinct, this keeps the first row of values.

Details

`nest_distinct()` is largely a wrapper for `dplyr::distinct()` and maintains the functionality of `distinct()` within each nested data frame. For more information on `distinct()`, please refer to the documentation in `dplyr`.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. Each object in `.nest_data` has the following properties:

- Rows are a subset of the input but appear in the same order.
- Columns are not modified if `...` is empty or `.keep_all` is `TRUE`. Otherwise, `nest_distinct()` first calls `dplyr::mutate()` to create new columns within each object in `.nest_data`.
- Groups are not modified.
- Data frame attributes are preserved.

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)

gm_nest %>% nest_distinct(country_data, country)
gm_nest %>% nest_distinct(country_data, country, year)
```

 nest_filter

Subset rows in nested data frames using column values.

Description

`nest_filter()` is used to subset nested data frames, retaining all rows that satisfy your conditions. To be retained, the row must produce a value of `TRUE` for all conditions. Note that when a condition evaluates to `NA` the row will be dropped, unlike base subsetting with `[]`.

`nest_filter()` subsets the rows within `.nest_data`, applying the expressions in `...` to the column values to determine which rows should be retained. It can be applied to both grouped and ungrouped data.

Usage

```
nest_filter(.data, .nest_data, ..., .preserve = FALSE)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>...</code>	Expressions that return a logical value, and are defined in terms of the variables in <code>.nest_data</code> . If multiple expressions are included, they are combined with the <code>&</code> operator. Only rows for which all conditions evaluate to <code>TRUE</code> are kept.
<code>.preserve</code>	Relevant when <code>.nest_data</code> is grouped. If <code>.preserve = FALSE</code> (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

Details

nest_filter() is largely a wrapper for `dplyr::filter()` and maintains the functionality of filter() within each nested data frame. For more information on filter(), please refer to the documentation in [dplyr](#).

Value

An object of the same type as .data. Each object in the column .nest_data will also be of the same type as the input. Each object in .nest_data has the following properties:

- Rows are a subset of the input, but appear in the same order.
- Columns are not modified.
- The number of groups may be reduced (if .preserve is not TRUE).
- Data frame attributes are preserved.

See Also

Other single table verbs: [nest_arrange\(\)](#), [nest_mutate\(\)](#), [nest_rename\(\)](#), [nest_select\(\)](#), [nest_slice\(\)](#), [nest_summarise\(\)](#)

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)

# apply a filter
gm_nest %>%
  nest_filter(country_data, year > 1972)

# apply multiple filters
gm_nest %>%
  nest_filter(country_data, year > 1972, pop < 10000000)

# apply a filter on grouped data
gm_nest %>%
  nest_group_by(country_data, country) %>%
  nest_filter(country_data, pop > mean(pop))
```

 nest_group_by

Group nested data frames by one or more variables

Description

nest_group_by() takes a set of nested tbls and converts it to a set of nested grouped tbls where operations are performed "by group". nest_ungroup() removes grouping.

Usage

```
nest_group_by(.data, .nest_data, ..., .add = FALSE, .drop = TRUE)

nest_ungroup(.data, .nest_data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>...</code>	In <code>nest_group_by()</code> , variables or computations to group by. Computations are always done on the ungrouped data frames. To perform computations on the grouped data, you need to use a separate <code>mutate()</code> step after the <code>group_by()</code> . In <code>nest_ungroup()</code> , variables to remove from the grouping.
<code>.add</code>	When <code>FALSE</code> (the default), <code>nest_group_by()</code> will override the existing groups. To add to the existing groups, use <code>.add = TRUE</code> .
<code>.drop</code>	Drop groups formed by factor levels that don't appear in the data? The default is <code>TRUE</code> except when <code>.nest_data</code> has been previously grouped with <code>.drop = FALSE</code> . See <code>dplyr::group_by_drop_default()</code> for details.

Details

`nest_group_by()` and `nest_ungroup()` are largely wrappers for `dplyr::group_by()` and `dplyr::ungroup()` and maintain the functionality of `group_by()` and `ungroup()` within each nested data frame. For more information on `group_by()` or `ungroup()`, please refer to the documentation in `dplyr`.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will be returned as a grouped data frame with class `grouped_df`, unless the combination of `...` and `.add` yields an empty set of grouping columns, in which case a tibble will be returned.

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)

# grouping doesn't change .nest_data, just .nest_data class:
gm_nest_grouped <-
  gm_nest %>%
  nest_group_by(country_data, year)

gm_nest_grouped

# It changes how it acts with other nplyr verbs:
gm_nest_grouped %>%
  nest_summarise(
    country_data,
    lifeExp = mean(lifeExp),
    pop = mean(pop),
```

```

      gdpPercap = mean(gdpPercap)
    )

# ungrouping removes variable groups:
gm_nest_grouped %>% nest_ungroup(country_data)

```

 nest_mutate

Create, modify, and delete columns in nested data frames

Description

nest_mutate() adds new variables to and preserves existing ones within the nested data frames in .nest_data. nest_transmute() adds new variables to and drops existing ones from the nested data frames in .nest_data.

Usage

```

nest_mutate(.data, .nest_data, ...)

nest_transmute(.data, .nest_data, ...)

```

Arguments

.data	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from dbplyr or dtplyr).
.nest_data	A list-column containing data frames
...	Name-value pairs. The name gives the name of the column in the output. The value can be: <ul style="list-style-type: none"> • A vector of length 1, which will be recycled to the correct length. • NULL, to remove the column. • A data frame or tibble, to create multiple columns in the output.

Details

nest_mutate() and nest_transmute() are largely wrappers for [dplyr::mutate\(\)](#) and [dplyr::transmute\(\)](#) and maintain the functionality of mutate() and transmute() within each nested data frame. For more information on mutate() or transmute(), please refer to the documentation in [dplyr](#).

Value

An object of the same type as .data. Each object in the column .nest_data will also be of the same type as the input. Each object in .nest_data has the following properties:

- For nest_mutate():
 - Columns from each object in .nest_data will be preserved according to the .keep argument.

- Existing columns that are modified by ... will always be returned in their original location.
- New columns created through ... will be placed according to the .before and .after arguments.
- For nest_transmute():
 - Columns created or modified through ... will be returned in the order specified by
 - Unmodified grouping columns will be placed at the front.
- The number of rows is not affected.
- Columns given the value NULL will be removed.
- Groups will be recomputed if a grouping variable is mutated.
- Data frame attributes will be preserved.

See Also

Other single table verbs: [nest_arrange\(\)](#), [nest_filter\(\)](#), [nest_rename\(\)](#), [nest_select\(\)](#), [nest_slice\(\)](#), [nest_summarise\(\)](#)

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)

# add or modify columns:
gm_nest %>%
  nest_mutate(
    country_data,
    lifeExp = NULL,
    gdp = gdpPercap * pop,
    pop = pop/1000000
  )

# use dplyr::across() to apply transformation to multiple columns
gm_nest %>%
  nest_mutate(
    country_data,
    across(c(lifeExp:gdpPercap), mean)
  )

# nest_transmute() drops unused columns when mutating:
gm_nest %>%
  nest_transmute(
    country_data,
    country = country,
    year = year,
    pop = pop/1000000
  )
```

nest_nest_join	<i>Nested nest join</i>
----------------	-------------------------

Description

nest_nest_join() returns all rows and columns in .nest_data with a new nested-df column that contains all matches from y. When there is no match, the list contains a 0-row tibble.

Usage

```
nest_nest_join(
  .data,
  .nest_data,
  y,
  by = NULL,
  copy = FALSE,
  keep = FALSE,
  name = NULL,
  ...
)
```

Arguments

.data	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from dbplyr or dtplyr).
.nest_data	A list-column containing data frames
y	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from dbplyr or dtplyr).
by	<p>A character vector of variables to join by or a join specification created with join_by().</p> <p>If NULL, the default, nest_*_join() will perform a natural join, using all variables in common across each object in .nest_data and y. A message lists the variables so you can check they're correct; suppress the message by supplying by explicitly.</p> <p>To join on different variables between the objects in .nest_data and y, use a named vector. For example, by = c("a" = "b") will match .nest_data\$a to y\$b for each object in .nest_data.</p> <p>To join by multiple variables, use a vector with length >1. For example, by = c("a", "b") will match .nest_data\$a to y\$a and .nest_data\$b to y\$b for each object in .nest_data. Use a named vector to match different variables in .nest_data and y. For example, by = c("a" = "b", "c" = "d") will match .nest_data\$a to y\$b and .nest_data\$c to y\$d for each object in .nest_data.</p> <p>To perform a cross-join, generating all combinations of each object in .nest_data and y, use by = character().</p>

copy	If <code>.nest_data</code> and <code>y</code> are not from the same data source and <code>copy = TRUE</code> then <code>y</code> will be copied into the same <code>src</code> as <code>.nest_data</code> . (<i>Need to review this parameter in more detail for applicability with <code>nplyr</code></i>)
keep	Should the join keys from both <code>.nest_data</code> and <code>y</code> be preserved in the output?
name	The name of the list column nesting joins create. If <code>NULL</code> , the name of <code>y</code> is used.
...	One or more unquoted expressions separated by commas. Variable names can be used if they were positions in the data frame, so expressions like <code>x:y</code> can be used to select a range of variables.

Details

`nest_nest_join()` is largely a wrapper around `dplyr::nest_join()` and maintains the functionality of `nest_join()` within east nested data frame. For more information on `nest_join()`, please refer to the documentation in [dplyr](#).

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input.

See Also

Other joins: [nest-filter-joins](#), [nest-mutate-joins](#)

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)
gm_codes <- gapminder::country_codes

gm_nest %>% nest_nest_join(country_data, gm_codes, by = "country")
```

nest_relocate	<i>Change column order within a nested data frame</i>
---------------	---

Description

`nest_relocate()` changes column positions within a nested data frame, using the same syntax as [nest_select\(\)](#) or [dplyr::select\(\)](#) to make it easy to move blocks of columns at once.

Usage

```
nest_relocate(.data, .nest_data, ..., .before = NULL, .after = NULL)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>...</code>	Columns to move.
<code>.before</code> , <code>.after</code>	Destination of columns selected by <code>...</code> . Supplying neither will move columns to the left-hand side; specifying both is an error.

Details

`nest_relocate()` is largely a wrapper for `dplyr::relocate()` and maintains the functionality of `relocate()` within each nested data frame. For more information on `relocate()`, please refer to the documentation in `dplyr`.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. Each object in `.nest_data` has the following properties:

- Rows are not affected.
- The same columns appear in the output, but (usually) in a different place.
- Data frame attributes are preserved.
- Groups are not affected.

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = ~continent)

gm_nest %>% nest_relocate(country_data, year)
gm_nest %>% nest_relocate(country_data, pop, .after = year)
```

nest_rename

Rename columns in nested data frames

Description

`nest_rename()` changes the names of individual variables using `new_name = old_name` syntax; `nest_rename_with()` renames columns using a function.

Usage

```
nest_rename(.data, .nest_data, ...)
```

```
nest_rename_with(.data, .nest_data, .fn, .cols = dplyr::everything(), ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>...</code>	For <code>nest_rename()</code> : Use <code>new_name = old_name</code> to rename selected variables. For <code>nest_rename_with()</code> : additional arguments passed onto <code>.fn</code> .
<code>.fn</code>	A function used to transform the selected <code>.cols</code> . Should return a character vector the same length as the input.
<code>.cols</code>	Columns to rename; defaults to all columns.

Details

`nest_rename()` and `nest_rename_with()` are largely wrappers for `dplyr::rename()` and `dplyr::rename_with()` and maintain the functionality of `rename()` and `rename_with()` within each nested data frame. For more information on `rename()` or `rename_with()`, please refer to the documentation in [dplyr](#).

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. Each object in `.nest_data` has the following properties:

- Rows are not affected.
- Column names are changed; column order is preserved.
- Data frame attributes are preserved.
- Groups are updated to reflect new names.

See Also

Other single table verbs: [nest_arrange\(\)](#), [nest_filter\(\)](#), [nest_mutate\(\)](#), [nest_select\(\)](#), [nest_slice\(\)](#), [nest_summarise\(\)](#)

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)

gm_nest %>% nest_rename(country_data, population = pop)
gm_nest %>% nest_rename_with(country_data, stringr::str_to_lower)
```

`nest_select`*Subset columns in nested data frames using their names and types*

Description

`nest_select()` selects (and optionally renames) variables in nested data frames, using a concise mini-language that makes it easy to refer to variables based on their name (e.g., `a:f` selects all columns from `a` on the left to `f` on the right). You can also use predicate functions like [is.numeric](#) to select variables based on their properties.

Usage

```
nest_select(.data, .nest_data, ...)
```

Arguments

<code>.data</code>	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from <code>dbplyr</code> or <code>dtplyr</code>).
<code>.nest_data</code>	A list-column containing data frames
<code>...</code>	One or more unquoted expressions separated by commas. Variable names can be used if they were positions in the data frame, so expressions like <code>x:y</code> can be used to select a range of variables.

Details

`nest_select()` is largely a wrapper for `dplyr::select()` and maintains the functionality of `select()` within each nested data frame. For more information on `select()`, please refer to the documentation in [dplyr](#).

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. Each object in `.nest_data` has the following properties:

- Rows are not affect.
- Output columns are a subset of input columns, potentially with a different order. Columns will be renamed if `new_name = old_name` form is used.
- Data frame attributes are preserved.
- Groups are maintained; you can't select off grouping variables.

See Also

Other single table verbs: [nest_arrange\(\)](#), [nest_filter\(\)](#), [nest_mutate\(\)](#), [nest_rename\(\)](#), [nest_slice\(\)](#), [nest_summarise\(\)](#)

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = -continent)

gm_nest %>% nest_select(country_data, country, year, pop)
gm_nest %>% nest_select(country_data, where(is.numeric))
```

 nest_slice

Subset rows in nested data frames using their positions.

Description

nest_slice() lets you index rows in nested data frames by their (integer) locations. It allows you to select, remove, and duplicate rows. It is accompanied by a number of helpers for common use cases:

- nest_slice_head() and nest_slice_tail() select the first or last rows of each nested data frame in .nest_data.
- nest_slice_sample() randomly selects rows from each data frame in .nest_data.
- nest_slice_min() and nest_slice_max() select the rows with the highest or lowest values of a variable within each nested data frame in .nest_data.

If .nest_data is a grouped data frame, the operation will be performed on each group, so that (e.g.) nest_slice_head(df, nested_dfs, n = 5) will return the first five rows in each group for each nested data frame.

Usage

```
nest_slice(.data, .nest_data, ..., .preserve = FALSE)

nest_slice_head(.data, .nest_data, ...)

nest_slice_tail(.data, .nest_data, ...)

nest_slice_min(.data, .nest_data, order_by, ..., with_ties = TRUE)

nest_slice_max(.data, .nest_data, order_by, ..., with_ties = TRUE)

nest_slice_sample(.data, .nest_data, ..., weight_by = NULL, replace = FALSE)
```

Arguments

.data	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from dbplyr or dtplyr).
.nest_data	A list-column containing data frames

...	<p>For <code>nest_slice()</code>: Integer row values.</p> <p>Provide either positive values to keep, or negative values to drop. The values provided must be either all positive or all negative. Indices beyond the number of rows in the input are silently ignored.</p> <p>For <code>nest_slice_helpers()</code>, these arguments are passed on to methods.</p> <p>Additionally:</p> <ul style="list-style-type: none"> • <code>n,prop</code> Provide either <code>n</code>, the number of rows, or <code>prop</code>, the proportion of rows to select. If neither are supplied, <code>n = 1</code> will be used. If a negative value of <code>n</code> or <code>prop</code> is provided, the specified number or proportion of rows will be removed. If <code>n</code> is greater than the number of rows in the group (or <code>prop > 1</code>), the result will be silently truncated to the group size. If the proportion of a group size does not yield an integer number of rows, the absolute value of <code>prop*nrow(.nest_data)</code> is rounded down.
<code>.preserve</code>	Relevant when <code>.nest_data</code> is grouped. If <code>.preserve = FALSE</code> (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping data is kept as is.
<code>order_by</code>	Variable or function of variables to order by.
<code>with_ties</code>	Should ties be kept together? The default, <code>TRUE</code> , may return more rows than you request. Use <code>FALSE</code> to ignore ties and return the first <code>n</code> rows.
<code>weight_by</code>	Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
<code>replace</code>	Should sampling be performed with (<code>TRUE</code>) or without (<code>FALSE</code> , the default) replacement?

Details

`nest_slice()` and its helpers are largely wrappers for `dplyr::slice()` and its helpers and maintains the functionality of `slice()` and its helpers within each nested data frame. For more information on `slice()` or its helpers, please refer to the documentation in [dplyr](#).

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will also be of the same type as the input. Each object in `.nest_data` has the following properties:

- Each row may appear 0, 1, or many times in the output.
- Columns are not modified.
- Groups are not modified.
- Data frame attributes are preserved.

See Also

Other single table verbs: [nest_arrange\(\)](#), [nest_filter\(\)](#), [nest_mutate\(\)](#), [nest_rename\(\)](#), [nest_select\(\)](#), [nest_summarise\(\)](#)

Examples

```

gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = ~continent)

# select the 1st, 3rd, and 5th rows in each data frame in country_data
gm_nest %>% nest_slice(country_data, 1, 3, 5)

# or select all but the 1st, 3rd, and 5th rows:
gm_nest %>% nest_slice(country_data, -1, -3, -5)

# first and last rows based on existing order:
gm_nest %>% nest_slice_head(country_data, n = 5)
gm_nest %>% nest_slice_tail(country_data, n = 5)

# rows with minimum and maximum values of a variable:
gm_nest %>% nest_slice_min(country_data, lifeExp, n = 5)
gm_nest %>% nest_slice_max(country_data, lifeExp, n = 5)

# randomly select rows with or without replacement:
gm_nest %>% nest_slice_sample(country_data, n = 5)
gm_nest %>% nest_slice_sample(country_data, n = 5, replace = TRUE)

```

nest_summarise

Summarise each group in nested data frames to fewer rows

Description

nest_summarise() creates a new set of nested data frames. Each will have one (or more) rows for each combination of grouping variables; if there are no grouping variables, the output will have a single row summarising all observations in .nest_data. Each nested data frame will contain one column for each grouping variable and one column for each of the summary statistics that you have specified.

nest_summarise() and nest_summarize() are synonyms.

Usage

```
nest_summarise(.data, .nest_data, ..., .groups = NULL)
```

```
nest_summarize(.data, .nest_data, ..., .groups = NULL)
```

Arguments

.data	A data frame, data frame extension (e.g., a tibble), or a lazy data frame (e.g., from dbplyr or dtplyr).
.nest_data	A list-column containing data frames
...	Name-value pairs of functions. The name will be the name of the variable in the result. The value can be:

- A vector of length 1, e.g. `min(x)`, `n()`, or `sum(is.na(y))`.
 - A vector of length `n`, e.g., `quantile()`.
 - A data frame, to add multiple columns from a single expression.
- `.groups` **[Experimental]** Grouping structure of the result. Refer to `dplyr::summarise()` for more up-to-date information.

Details

`nest_summarise()` is largely a wrapper for `dplyr::summarise()` and maintains the functionality of `summarise()` within each nested data frame. For more information on `summarise()`, please refer to the documentation in `dplyr`.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will *usually* be of the same type as the input. Each object in `.nest_data` has the following properties:

- The rows come from the underlying `group_keys()`
- The columns are a combination of the grouping keys and the summary expressions that you provide.
- The grouping structure is controlled by the `.groups` argument, the output may be another `grouped_df`, a tibble, or a rowwise data frame.
- Data frame attributes are **not** preserved, because `nest_summarise()` fundamentally creates a new data frame for each object in `.nest_data`.

See Also

Other single table verbs: `nest_arrange()`, `nest_filter()`, `nest_mutate()`, `nest_rename()`, `nest_select()`, `nest_slice()`

Examples

```
gm_nest <- gapminder::gapminder %>% tidyr::nest(country_data = ~continent)

# a summary applied to an ungrouped tbl returns a single row
gm_nest %>%
  nest_summarise(
    country_data,
    n = n(),
    median_pop = median(pop)
  )

# usually, you'll want to group first
gm_nest %>%
  nest_group_by(country_data, country) %>%
  nest_summarise(
    country_data,
    n = n(),
    median_pop = median(pop)
  )
```

`personal_survey`*Example survey data regarding personal life satisfaction*

Description

A toy dataset containing 750 responses to a personal satisfaction survey. The responses were randomly generated using the Qualtrics survey platform.

Usage`personal_survey`**Format**

A data frame with 750 rows and 6 variables

survey_name name of survey

Q1 respondent age

Q2 city the respondent resides in

Q3 field that the respondent that works in

Q4 respondent's personal life satisfaction (on a scale from extremely satisfied to extremely dissatisfied)

Q5 open text response elaborating on personal life satisfaction

Index

- * **datasets**
 - job_survey, 2
 - personal_survey, 25
- * **grouping functions**
 - nest_group_by, 12
- * **joins**
 - nest-filter-joins, 3
 - nest-mutate-joins, 4
 - nest_nest_join, 16
- * **single table verbs**
 - nest_arrange, 8
 - nest_filter, 11
 - nest_mutate, 14
 - nest_rename, 18
 - nest_select, 20
 - nest_slice, 21
 - nest_summarise, 23

dplyr::add_count(), 9
dplyr::anti_join(), 4
dplyr::arrange(), 8
dplyr::count(), 9
dplyr::desc(), 8
dplyr::distinct(), 10
dplyr::filter(), 12
dplyr::full_join(), 7
dplyr::group_by(), 13
dplyr::group_by_drop_default(), 13
dplyr::inner_join(), 7
dplyr::left_join(), 7
dplyr::mutate(), 14
dplyr::nest_join(), 17
dplyr::relocate(), 18
dplyr::rename(), 19
dplyr::rename_with(), 19
dplyr::right_join(), 7
dplyr::select(), 17, 20
dplyr::semi_join(), 4
dplyr::slice(), 22
dplyr::summarise(), 24

dplyr::transmute(), 14
dplyr::ungroup(), 13

is.numeric, 20

job_survey, 2

nest-filter-joins, 3
nest-mutate-joins, 4
nest_add_count (nest_count), 9
nest_anti_join (nest-filter-joins), 3
nest_arrange, 8, 12, 15, 19, 20, 22, 24
nest_count, 9
nest_distinct, 10
nest_filter, 8, 11, 15, 19, 20, 22, 24
nest_full_join (nest-mutate-joins), 4
nest_group_by, 12
nest_inner_join (nest-mutate-joins), 4
nest_left_join (nest-mutate-joins), 4
nest_mutate, 8, 12, 14, 19, 20, 22, 24
nest_nest_join, 4, 7, 16
nest_relocate, 17
nest_rename, 8, 12, 15, 18, 20, 22, 24
nest_rename_with (nest_rename), 18
nest_right_join (nest-mutate-joins), 4
nest_select, 8, 12, 15, 19, 20, 22, 24
nest_select(), 17
nest_semi_join (nest-filter-joins), 3
nest_slice, 8, 12, 15, 19, 20, 21, 24
nest_slice_head (nest_slice), 21
nest_slice_max (nest_slice), 21
nest_slice_min (nest_slice), 21
nest_slice_sample (nest_slice), 21
nest_slice_tail (nest_slice), 21
nest_summarise, 8, 12, 15, 19, 20, 22, 23
nest_summarize (nest_summarise), 23
nest_transmute (nest_mutate), 14
nest_ungroup (nest_group_by), 12

personal_survey, 25