Package: nplyr (via r-universe)

March 2, 2025

Title A Grammar of Nested Data Manipulation

Version 0.3.0

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/jibarozzo/nplyr, https://jibarozzo.github.io/nplyr/

BugReports https://github.com/jibarozzo/nplyr/issues

Depends R (>= 3.5.0)

Imports assertthat, dplyr, magrittr, purrr, rlang, tidyr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

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

Config/testthat/edition 3

VignetteBuilder knitr

LazyData true

Config/pak/sysreqs libicu-dev

Repository https://jibarozzo.r-universe.dev

RemoteUrl https://github.com/jibarozzo/nplyr

RemoteRef HEAD

RemoteSha 9afecf1719510c5d23c2459dbe8041c1d6448d6d

Contents

`job_survey` *Example survey data regarding job satisfaction*

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` with*out* 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

`.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`	A character vector of variables to join by or a join specification created with `join_by()`. 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. 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`. 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`. To perform a cross-join, generating all combinations of each object in `.nest_data` and `y`, use `by = character()`.
`copy`	If `.nest_data` and `y` are not from the same data source and `copy = TRUE` then `y` will be copied into the same src as `.nest_data`. *(Need to review this parameter in more detail for applicability with nplyr)*

... One or more unquoted expressions separated by commas. Variable names can be used if they were positions in the data frame, so expressions like `x:y` 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

.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	A character vector of variables to join by or a join specification created with `join_by()`. 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. 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`. 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`. To perform a cross-join, generating all combinations of each object in `.nest_data` and `y`, use `by = character()`.
copy	If `.nest_data` and `y` are not from the same data source and `copy = TRUE` then `y` will be copied into the same src as `.nest_data`. *(Need to review this parameter in more detail for applicability with nplyr)*
suffix	If there are non-joined duplicate variables in `.nest_data` and `y`, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
...	Other parameters passed onto methods. Includes: • `na_matches` : Should two `NA` or two `NaN` values match? – `"na"`, the default, treats two `NA` or two `NaN` values as equal. – `"never"` treats two `NA` or two `NaN` values as different, and will never match them together or to any other values. • `multiple` : Handling of rows in `.nest_data` with multiple matches in `y`. – `"all"` returns every match detected in `y`. – `"any"` returns one match detected in `y`, with no guarantees on which match will be returned. It is often faster than `"first"` and `"last"` if you just need to detect if there is at least one match. – `"first"` returns the first match detected in `y`. – `"last"` returns the last match detected in `y`.

  - "warning" throws a warning if multiple matches are detected, and then falls back to "all".
  - "error" 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` *Arrange rows within a nested data frames by column values*

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 `dplyr::desc()` 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 *Count observations in a nested data frame by group*

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: • 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

`.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
`...`	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.
`.keep_all`	If `TRUE`, keep all variables in `.nest_data`. If a combination of `...` 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_drop_na *Drop rows containing missing values in a column of nested data frames*

Description

nest_drop_na() is used to drop rows from each data frame in a column of nested data frames.

Usage

```
nest_drop_na(.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
...	Columns within .nest_data to inspect for missing values. If empty, all columns within each dataframe in .nest_data are used.

Details

nest_drop_na() is a wrapper for tidyr::drop_na() and maintains the functionality of drop_na() within each nested data frame. For more information on drop_na() please refer to the documentation in 'tidyr'.

Value

An object of the same type as .data. Each object in the column .nest_data will have rows dropped according to the presence of NAs.

See Also

Other tidyr verbs: nest_extract(), nest_fill(), nest_replace_na(), nest_separate(), nest_unite()

Examples

```
gm <- gapminder::gapminder

# randomly insert NAs into the dataframe & nest
set.seed(123)
gm <-
  gm %>%
  dplyr::mutate(pop = dplyr::if_else(runif(nrow(gm)) >= 0.9,
                                     NA_integer_,
                                     pop))

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

# drop rows where an NA exists in column `pop`
gm_nest %>%
  nest_drop_na(country_data, pop)
```

---

nest_extract *Extract a character column into multiple columns using regex groups in a column of nested data frames*

Description

nest_extract() is used to extract capturing groups from a column in a nested data frame using regular expressions into a new column. If the groups don't match, or the input is NA, the output will be NA.

Usage

```
nest_extract(
  .data,
  .nest_data,
  col,
  into,
  regex = "([[:alnum:]]+)",
  remove = TRUE,
  convert = 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
col	Column name or position within .nest_data (must be present within all nested data frames in .nest_data). This is passed to tidyselect::vars_pull(). This argument is passed by expression and supports quasiquotation (you can unquote column names or column positions).
into	Names of new variables to create as character vector. Use NA to omit the variable in the output.
regex	A string representing a regular expression used to extract the desired values. There should be one group (defined by ()) for each element of into.
remove	If TRUE, remove input column from output data frame.
convert	If TRUE, will run type.convert() with as.is = TRUE on new columns. This is useful if the component columns are integer, numeric or logical. NB: this will cause string "NA"s to be converted to NAs.
...	Additional arguments passed on to tidyr::extract() methods.

Details

nest_extract() is a wrapper for tidyr::extract() and maintains the functionality of extract() within each nested data frame. For more information on extract() please refer to the documentation in 'tidyr'.

Value

An object of the same type as .data. Each object in the column .nest_data will have new columns created according to the capture groups specified in the regular expression.

See Also

Other tidyr verbs: nest_drop_na(), nest_fill(), nest_replace_na(), nest_separate(), nest_unite()

Examples

```
set.seed(123)
gm <- gapminder::gapminder

gm <-
  gm %>%
  dplyr::mutate(comb = sample(c(NA, "a-b", "a-d", "b-c", "d-e"),
                              size = nrow(gm),
                              replace = TRUE))

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

gm_nest %>%
```

```
nest_extract(country_data,
             col = comb,
             into = c("var1","var2"),
             regex = "([[:alnum:]]+)-([[:alnum:]]+)")
```

`nest_fill` *Fill missing values in a column of nested data frames*

Description

`nest_fill()` is used to fill missing values in selected columns of nested data frames using the next or previous entries in a column of nested data frames.

Usage

```
nest_fill(
  .data,
  .nest_data,
  ...,
  .direction = c("down", "up", "downup", "updown")
)
```

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
`...`	<tidy-select> Columns to fill.
`.direction`	Direction in which to fill missing values. Currently either "down" (the default), "up", "downup" (i.e. first down and then up) or "updown" (first up and then down).

Details

`nest_fill()` is a wrapper for `tidyr::fill()` and maintains the functionality of `fill()` within each nested data frame. For more information on `fill()` please refer to the documentation in 'tidyr'.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will have the chosen columns filled in the direction specified by `.direction`.

See Also

Other tidyr verbs: `nest_drop_na()`, `nest_extract()`, `nest_replace_na()`, `nest_separate()`, `nest_unite()`

Examples

```
set.seed(123)
gm <-
  gapminder::gapminder %>%
  dplyr::mutate(pop = dplyr::if_else(runif(dplyr::n()) >= 0.9,
                                     NA_integer_,
                                     pop))

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

gm_nest %>%
  nest_fill(country_data, pop, .direction = "down")
```

`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

`.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
`...`	Expressions that return a logical value, and are defined in terms of the variables in `.nest_data`. If multiple expressions are included, they are combined with the `&` operator. Only rows for which all conditions evaluate to `TRUE` are kept.
`.preserve`	Relevant when `.nest_data` is grouped. If `.preserve = FALSE` (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

.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
...	In nest_group_by(), 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 mutate() step after the group_by(). In nest_ungroup(), variables to remove from the grouping.
.add	When FALSE (the default), nest_group_by() will override the existing groups. To add to the existing groups, use .add = TRUE.
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when .nest_data has been previously grouped with .drop = FALSE. See dplyr::group_by_drop_default() 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: • 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 *Nested nest join*

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	A character vector of variables to join by or a join specification created with join_by(). 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. 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. 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. To perform a cross-join, generating all combinations of each object in .nest_data and y, use by = character().
copy	If .nest_data and y are not from the same data source and copy = TRUE then y will be copied into the same src as .nest_data. *(Need to review this parameter in more detail for applicability with nplyr)*
keep	Should the join keys from both .nest_data and y be preserved in the output?
name	The name of the list column nesting joins create. If NULL, the name of y 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 x:y 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 *Change column order within a nested data frame*

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

.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
...	Columns to move.
.before, .after	Destination of columns selected by .... 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

`.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
`...`	For `nest_rename()`: Use `new_name = old_name` to rename selected variables. For `nest_rename_with()`: additional arguments passed onto `.fn`.
`.fn`	A function used to transform the selected `.cols`. Should return a character vector the same length as the input.
`.cols`	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_replace_na *Replace NAs with specified values in a column of nested data frames*

Description

`nest_replace_na()` is used to replace missing values in selected columns of nested data frames using values specified by column.

Usage

```
nest_replace_na(.data, .nest_data, replace, ...)
```

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
`replace`	A list of values, with one value for each column in that has `NA` values to be replaced.
`...`	Additional arguments for `tidyr::replace_na()` methods. Currently unused.

Details

`nest_replace_na()` is a wrapper for `tidyr::replace_na()` and maintains the functionality of `replace_na()` within each nested data frame. For more information on `replace_na()` please refer to the documentation in 'tidyr'.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will have NAs replaced in the specified columns.

See Also

Other tidyr verbs: `nest_drop_na()`, `nest_extract()`, `nest_fill()`, `nest_separate()`, `nest_unite()`

Examples

```
set.seed(123)
gm <-
  gapminder::gapminder %>%
  dplyr::mutate(pop = dplyr::if_else(runif(dplyr::n()) >= 0.9,
                                     NA_integer_,
                                     pop))

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

gm_nest %>%
  nest_replace_na(.nest_data = country_data,
                  replace = list(pop = -500))
```

`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

`.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
`...`	One or more unquoted expressions separated by commas. Variable names can be used if they were positions in the data frame, so expressions like `x:y` 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, dplyr::where(is.numeric))
```

---

nest_separate *Separate a character column into multiple columns in a column of nested data frames*

Description

nest_separate() is used to separate a single character column into multiple columns using a regular expression or a vector of character positions in a list of nested data frames.

Usage

```
nest_separate(
  .data,
  .nest_data,
  col,
  into,
  sep = "[^[:alnum:]]+",
  remove = TRUE,
  convert = FALSE,
```

```
  extra = "warn",
  fill = "warn",
  ...
)
```

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
col	Column name or position within. Must be present in all data frames in .nest_data. This is passed to tidyselect::vars_pull(). This argument is passed by expression and supports quasiquotation (you can unquote column names or column positions).
into	Names of new variables to create as character vector. Use NA to omit the variable in the output.
sep	Separator between columns. If character, sep is interpreted as a regular expression. The default value is a regular expression that matches any sequence of non-alphanumeric values. If numeric, sep is interpreted as character positions to split at. Positive values start at 1 at the far-left of the string; negative value start at -1 at the far-right of the string. The length of sep should be one less than into.
remove	If TRUE, remove input column from output data frame.
convert	If TRUE, will run type.convert() with as.is = TRUE on new columns. This is useful if the component columns are integer, numeric or logical. NB: this will cause string "NA"s to be converted to NAs.
extra	If sep is a character vector, this controls what happens when there are too many pieces. There are three valid options: • "warn" (the default): emit a warning and drop extra values. • "drop": drop any extra values without a warning. • "merge": only splits at most length(into) times
fill	If sep is a character vector, this controls what happens when there are not enough pieces. There are three valid options: • "warn" (the default): emit a warning and fill from the right • "right": fill with missing values on the right • "left": fill with missing values on the left
...	Additional arguments passed on to tidyr::separate() methods.

Details

nest_separate() is a wrapper for tidyr::separate() and maintains the functionality of separate() within each nested data frame. For more information on separate() please refer to the documentation in 'tidyr'.

Value

An object of the same type as `.data`. Each object in the column `.nest_data` will have the specified column split according to the regular expression or the vector of character positions.

See Also

Other tidyr verbs: `nest_drop_na()`, `nest_extract()`, `nest_fill()`, `nest_replace_na()`, `nest_unite()`

Examples

```
set.seed(123)
gm <-
  gapminder::gapminder %>%
  dplyr::mutate(comb = paste(continent, year, sep = "-"))

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

gm_nest %>%
  nest_separate(country_data,
                col = comb,
                into = c("var1","var2"),
                sep = "-")
```

---

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
...	For nest_slice(): Integer row values. 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. For nest_slice_helpers(), these arguments are passed on to methods. Additionally: • n,prop Provide either n, the number of rows, or prop, the proportion of rows to select. If neither are supplied, n = 1 will be used. If a negative value of n or prop is provided, the specified number or proportion of rows will be removed. If n is greater than the number of rows in the group (or prop > 1), 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 prop*nrow(.nest_data) is rounded down.
.preserve	Relevant when .nest_data is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping data is kept as is.
order_by	Variable or function of variables to order by.
with_ties	Should ties be kept together? The default, TRUE, may return more rows than you request. Use FALSE to ignore ties and return the first n rows.
weight_by	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.
replace	Should sampling be performed with (TRUE) or without (FALSE, 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 = dplyr::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 = dplyr::n(),
    median_pop = median(pop)
  )
```

---

nest_unite *Unite multiple columns into one in a column of nested data frames*

Description

`nest_unite()` is used to combine multiple columns into one in a column of nested data frames.

Usage

```
nest_unite(
  .data,
  .nest_data,
  col,
  ...,
  sep = "_",
  remove = TRUE,
  na.rm = 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
col	The name of the new column, as a string or symbol. This argument is passed by expression and supports quasiquotation (you can unquote strings and symbols). The name is captured from the expression with rlang::ensym() (note that this kind of interface where symbols do not represent actual objects is now discouraged in the tidyverse; we support it here for backward compatibility).
...	Columns to unite.
sep	Separator to use between values.
remove	If TRUE, remove input columns from output data frame.
na.rm	If TRUE, missing values will be removed prior to uniting each value.

Details

nest_unite() is a wrapper for tidyr::unite() and maintains the functionality of unite() within each nested data frame. For more information on unite() please refer to the documentation in 'tidyr'.

Value

An object of the same type as .data. Each object in the column .nest_data will have a new column created as a combination of existing columns.

See Also

Other tidyr verbs: nest_drop_na(), nest_extract(), nest_fill(), nest_replace_na(), nest_separate()

Examples

```
set.seed(123)
gm <- gapminder::gapminder
gm_nest <- gm %>% tidyr::nest(country_data = -continent)

gm_nest %>%
  nest_unite(country_data,
             col = comb,
             year,
             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