# Package 'cards'

May 28, 2026

**Title** Analysis Results Data

**Version** 0.8.0

**Description** Construct CDISC (Clinical Data Interchange Standards Consortium) compliant Analysis Results Data objects. These objects are used and re-used to construct summary tables, visualizations, and written reports. The package also exports utilities for working with these objects and creating new Analysis Results Data objects.

**License** Apache License 2.0

**URL** https://github.com/insightsengineering/cards, https://insightsengineering.github.io/cards/

**BugReports** https://github.com/insightsengineering/cards/issues

**Depends** R (>= 4.1)

**Imports** cli (>= 3.6.5), dplyr (>= 1.1.4), glue (>= 1.8.0), lifecycle (>= 1.0.4), rlang (>= 1.1.6), tidyr (>= 1.3.1), tidyselect (>= 1.2.1)

**Suggests** testthat (>= 3.2.3), withr (>= 3.0.0)

**Config/Needs/coverage** hms

**Config/Needs/website** rmarkdown, jsonlite, yaml, gtsummary, tfrmt, cardx, gt, fontawesome, insightsengineering/crane, insightsengineering/nesttemplate

**Config/roxygen2/version** 8.0.0

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**NeedsCompilation** no

**Author** Daniel D. Sjoberg [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0862-2018>),
Becca Krouse [aut],
Emily de la Rua [aut] (ORCID: <https://orcid.org/0009-0000-8738-5561>),
Malan Bosman [aut] (ORCID: <https://orcid.org/0000-0002-3020-195X>),
F. Hoffmann-La Roche AG [cph, fnd],
GlaxoSmithKline Research & Development Limited [cph]

**Maintainer** Daniel D. Sjoberg <danield.sjoberg@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-05-28 07:10:08 UTC

# Contents

---

adam *Example ADaM Data*

---

## Description

Data frame imported from the CDISC SDTM/ADaM Pilot Project

## Usage

```
ADSL

ADAE

ADTTE

ADLB
```

## Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 254 rows and 49 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1191 rows and 56 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 254 rows and 26 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 5784 rows and 46 columns.

`add_calculated_row` *Add Calculated Row*

## Description

Use this function to add a new statistic row that is a function of the other statistics in an ARD.

## Usage

```
add_calculated_row(
  x,
  expr,
  stat_name,
  by = c(all_ard_groups(), all_ard_variables(), any_of("context")),
  stat_label = stat_name,
  fmt_fun = NULL,
  fmt_fn = deprecated()
)
```

## Arguments

| | |
|---|---|
| `x` | (`card`)<br>data frame of class `'card'` |
| `expr` | (`expression`)<br>an expression |
| `stat_name` | (`string`)<br>string naming the new statistic |
| `by` | (`tidy-select`)<br>Grouping variables to calculate statistics within |
| `stat_label` | (`string`)<br>string of the statistic label. Default is the `stat_name`. |
| `fmt_fun` | (`integer`, `function`, `string`)<br>a function of an integer or string that can be converted to a function with `alias_as_fmt_fun()`. |
| `fmt_fn` | **[Deprecated]** |

## Value

an ARD data frame of class 'card'

## Examples

```
ard_summary(mtcars, variables = mpg) |>
  add_calculated_row(expr = max - min, stat_name = "range")

ard_summary(mtcars, variables = mpg) |>
  add_calculated_row(
```

```
    expr =
      dplyr::case_when(
        mean > median ~ "Right Skew",
        mean < median ~ "Left Skew",
        .default = "Symmetric"
      ),
    stat_name = "skew"
  )
```

---

`alias_as_fmt_fun` *Convert Alias to Function*

---

## Description

Accepted aliases are non-negative integers and strings.

The integers are converted to functions that round the statistics to the number of decimal places to match the integer.

The formatting strings come in the form `"xx"`, `"xx.x"`, `"xx.x%"`, etc. The number of `x`s that appear after the decimal place indicate the number of decimal places the statistics will be rounded to. The number of `x`s that appear before the decimal place indicate the leading spaces that are added to the result. If the string ends in `"%"`, results are scaled by 100 before rounding.

## Usage

```
alias_as_fmt_fun(x, variable, stat_name)
```

## Arguments

| | |
|---|---|
| `x` | (`integer`, `string`, or `function`)<br>a non-negative integer, string alias, or function |
| `variable` | (`character`)<br>the variable whose statistic is to be formatted |
| `stat_name` | (`character`)<br>the name of the statistic that is to be formatted |

## Value

a function

## Examples

```
alias_as_fmt_fun(1)
alias_as_fmt_fun("xx.x")
```

## apply_fmt_fun *Apply Formatting Functions*

### Description

Apply the formatting functions to each of the raw statistics. Function aliases are converted to functions using `alias_as_fmt_fun()`.

### Usage

```
apply_fmt_fun(x, replace = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>an ARD data frame of class 'card' |
| `replace` | (scalar `logical`)<br>logical indicating whether to replace values in the `'stat_fmt'` column (if present). Default is `FALSE`. |

### Value

an ARD data frame of class 'card'

### Examples

```
ard_summary(ADSL, variables = "AGE") |>
  apply_fmt_fun()
```

## ard_attributes *ARD Attributes*

### Description

Add variable attributes to an ARD data frame.

- The `label` attribute will be added for all columns, and when no label is specified and no label has been set for a column using the `label=` argument, the column name will be placed in the label statistic.
- The `class` attribute will also be returned for all columns.
- Any other attribute returned by `attributes()` will also be added, e.g. factor levels.

## Usage

```
ard_attributes(data, ...)

## S3 method for class 'data.frame'
ard_attributes(data, variables = everything(), label = NULL, ...)

## Default S3 method:
ard_attributes(data, ...)
```

## Arguments

| | |
|---|---|
| data | (data.frame)<br>a data frame |
| ... | These dots are for future extensions and must be empty. |
| variables | (tidy-select)<br>variables to include |
| label | (named list)<br>named list of variable labels, e.g. list(cyl = "No. Cylinders"). Default is NULL |

## Value

an ARD data frame of class 'card'

## Examples

```
df <- dplyr::tibble(var1 = letters, var2 = LETTERS)
attr(df$var1, "label") <- "Lowercase Letters"

ard_attributes(df, variables = everything())
```

---

# ard_formals *Argument Values ARD*

---

## Description

Place default and passed argument values to a function into an ARD structure.

## Usage

```
ard_formals(fun, arg_names, passed_args = list(), envir = parent.frame())
```

### Arguments

| | |
|---|---|
| `fun` | (`function`)<br>a function passed to `formals(fun)` |
| `arg_names` | (`character`)<br>character vector of argument names to return |
| `passed_args` | (named `list`)<br>a named list of user-passed arguments. Default is `list()`, which returns all default values from a function |
| `envir` | (`environment`)<br>an environment passed to `formals(envir)` |

### Value

an partial ARD data frame of class 'card'

### Examples

```
# Example 1 ----------------------------------
# add the `mcnemar.test(correct)` argument to an ARD structure
ard_formals(fun = mcnemar.test, arg_names = "correct")

# Example 2 ----------------------------------
# S3 Methods need special handling to access the underlying method
ard_formals(
  fun = asNamespace("stats")[["t.test.default"]],
  arg_names = c("mu", "paired", "var.equal", "conf.level"),
  passed_args = list(conf.level = 0.90)
)
```

---

## `ard_hierarchical` *Hierarchical ARD Statistics*

---

### Description

*Functions* `ard_hierarchical()` *and* `ard_hierarchical_count()` *are primarily helper functions for* `ard_stack_hierarchical()` *and* `ard_stack_hierarchical_count()`*, meaning that it will be rare a user needs to call* `ard_hierarchical()`/`ard_hierarchical_count()` *directly.*

Performs hierarchical or nested tabulations, e.g. tabulates AE terms nested within AE system organ class.

- `ard_hierarchical()` includes summaries for the last variable listed in the `variables` argument, nested within the other variables included.
- `ard_hierarchical_count()` includes summaries for *all* variables listed in the `variables` argument each summary nested within the preceding variables, e.g. `variables=c(AESOC, AEDECOD)` summarizes `AEDECOD` nested in `AESOC`, and also summarizes the counts of `AESOC`.

## Usage

```
ard_hierarchical(data, ...)

ard_hierarchical_count(data, ...)

## S3 method for class 'data.frame'
ard_hierarchical(
  data,
  variables,
  by = dplyr::group_vars(data),
  statistic = everything() ~ c("n", "N", "p"),
  denominator = NULL,
  fmt_fun = NULL,
  stat_label = everything() ~ default_stat_labels(),
  id = NULL,
  fmt_fn = deprecated(),
  ...
)

## S3 method for class 'data.frame'
ard_hierarchical_count(
  data,
  variables,
  by = dplyr::group_vars(data),
  fmt_fun = NULL,
  stat_label = everything() ~ default_stat_labels(),
  fmt_fn = deprecated(),
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `...` | Arguments passed to methods. |
| `variables` | (`tidy-select`)<br>variables to perform the nested/hierarchical tabulations within. |
| `by` | (`tidy-select`)<br>variables to perform tabulations by. All combinations of the variables specified here appear in results. Default is `dplyr::group_vars(data)`. |
| `statistic` | (`formula-list-selector`)<br>a named list, a list of formulas, or a single formula where the list element one or more of `c("n", "N", "p", "n_cum", "p_cum")` (on the RHS of a formula). |
| `denominator` | (`data.frame`, `integer`)<br>used to define the denominator and enhance the output. The argument is required for `ard_hierarchical()` and optional for `ard_hierarchical_count()`. |

- the univariate tabulations of the by variables are calculated with `denominator`, when a data frame is passed, e.g. tabulation of the treatment assignment counts that may appear in the header of a table.
- the `denominator` argument must be specified when `id` is used to calculate the event rates.

`fmt_fun` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) round(x, digits`

`stat_label` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. `everything() ~ list(n = "n", p = "pct")` or `everything() ~ list(n ~ "n", p ~ "pct")`.

`id` (tidy-select)
an optional argument used to assert there are no duplicates within the `c(id, variables)` columns.

`fmt_fn` **[Deprecated]**

## Value

an ARD data frame of class 'card'

## Examples

```
ard_hierarchical(
  data = ADAE |>
    dplyr::slice_tail(n = 1L, by = c(USUBJID, TRTA, AESOC, AEDECOD)),
  variables = c(AESOC, AEDECOD),
  by = TRTA,
  id = USUBJID,
  denominator = ADSL
)

ard_hierarchical_count(
  data = ADAE,
  variables = c(AESOC, AEDECOD),
  by = TRTA
)
```

---

# ard_identity *ARD Identity*

---

## Description

Function ingests pre-calculated statistics and returns the identical results, but in an ARD format.

## Usage

```
ard_identity(x, variable, context = "identity")
```

### Arguments

| | |
|---|---|
| x | (named list/data.frame)<br>named list of results or a data frame. Names are the statistic names, and the values are the statistic values. These comprise the "stat_name" and "stat" columns in the returned ARD. |
| variable | (string)<br>string of a variable name that is assigned to the "variable" column in the ARD. |
| context | (string)<br>string to be added to the "context" column. Default is "identity". |

### Value

a ARD

### Examples

```
t.test(formula = AGE ~ 1, data = ADSL)[c("statistic", "parameter", "p.value")] |>
  ard_identity(variable = "AGE", context = "onesample_t_test")
```

---

## ard_missing *Missing ARD Statistics*

---

### Description

Compute Analysis Results Data (ARD) for statistics related to data missingness.

### Usage

```
ard_missing(data, ...)

## S3 method for class 'data.frame'
ard_missing(
  data,
  variables,
  by = dplyr::group_vars(data),
 statistic = everything() ~ c("N_obs", "N_miss", "N_nonmiss", "p_miss", "p_nonmiss"),
  fmt_fun = NULL,
  stat_label = everything() ~ default_stat_labels(),
  fmt_fn = deprecated(),
  ...
)
```

## Arguments

| | |
|---|---|
| data | (data.frame)<br>a data frame |
| ... | Arguments passed to methods. |
| variables | (tidy-select)<br>columns to include in summaries. |
| by | (tidy-select)<br>results are tabulated by **all combinations** of the columns specified. |
| statistic | (formula-list-selector)<br>a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) mean(x)))`.<br>The value assigned to each variable must also be a named list, where the names are used to reference a function and the element is the function object. Typically, this function will return a scalar statistic, but a function that returns a named list of results is also acceptable, e.g. `list(conf.low = -1, conf.high = 1)`. However, when errors occur, the messaging will be less clear in this setting. |
| fmt_fun | (formula-list-selector)<br>a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) round(x, digits` |
| stat_label | (formula-list-selector)<br>a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. `everything() ~ list(mean = "Mean", sd = "SD")` or `everything() ~ list(mean ~ "Mean", sd ~ "SD")`. |
| fmt_fn | **[Deprecated]** |

## Value

an ARD data frame of class 'card'

## Examples

```
ard_missing(ADSL, by = "ARM", variables = "AGE")

ADSL |>
  dplyr::group_by(ARM) |>
  ard_missing(
    variables = "AGE",
    statistic = ~"N_miss"
  )
```

`ard_mvsummary` *Multivariate ARD Summaries*

## Description

Function is similar to `ard_summary()`, but allows for more complex, multivariate summaries. While `ard_summary(statistic)` only allows for a univariable function, `ard_mvsummary(statistic)` can handle more complex data summaries.

## Usage

```
ard_mvsummary(data, ...)

## S3 method for class 'data.frame'
ard_mvsummary(
  data,
  variables,
  by = dplyr::group_vars(data),
  strata = NULL,
  statistic,
  fmt_fun = NULL,
  stat_label = everything() ~ default_stat_labels(),
  fmt_fn = deprecated(),
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `...` | Arguments passed to methods. |
| `variables` | (`tidy-select`)<br>columns to include in summaries. |
| `by`, `strata` | (`tidy-select`)<br>columns to tabulate by/stratify by for summary statistic calculation. Arguments are similar, but with an important distinction:<br>`by`: results are calculated for **all combinations** of the columns specified, including unobserved combinations and unobserved factor levels.<br>`strata`: results are calculated for **all *observed* combinations** of the columns specified.<br>Arguments may be used in conjunction with one another. |
| `statistic` | (`formula-list-selector`)<br>The form of the statistics argument is identical to `ard_summary(statistic)` argument, except the summary function *must* accept the following arguments:<br>• `x`: a vector |

- `data`: the data frame that has been subset such that the by/`strata` columns and rows in which `"variable"` is `NA` have been removed.
- `full_data`: the full data frame
- `by`: character vector of the by variables
- `strata`: character vector of the `strata` variables

It is unlikely any one function will need *all* of the above elements, and it's recommended the function passed accepts `...` so that any unused arguments will be properly ignored. The `...` also allows this function to perhaps be updated in the future with more passed arguments. For example, if one needs a second variable from the data frame, the function inputs may look like: `foo(x, data, ...)`

`fmt_fun` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) round(x, digits`

`stat_label` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. `everything() ~ list(mean = "Mean", sd = "SD")` or `everything() ~ list(mean ~ "Mean", sd ~ "SD")`.

`fmt_fn` **[Deprecated]**

## Value

an ARD data frame of class 'card'

## Examples

```
# example how to mimic behavior of `ard_summary()`
ard_mvsummary(
  ADSL,
  by = "ARM",
  variables = "AGE",
  statistic = list(AGE = list(mean = \(x, ...) mean(x)))
)

# return the grand mean and the mean within the `by` group
grand_mean <- function(data, full_data, variable, ...) {
  list(
    mean = mean(data[[variable]], na.rm = TRUE),
    grand_mean = mean(full_data[[variable]], na.rm = TRUE)
  )
}

ADSL |>
  dplyr::group_by(ARM) |>
  ard_mvsummary(
    variables = "AGE",
    statistic = list(AGE = list(means = grand_mean))
  )
```

`ard_pairwise` *Pairwise ARD*

## Description

Utility to perform pairwise comparisons.

## Usage

```
ard_pairwise(data, variable, .f, include = NULL)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `variable` | (`tidy-select`)<br>Column to perform pairwise analyses for. |
| `.f` | (`function`)<br>a function that creates ARDs. The function accepts a single argument and a subset of `data` will be passed including the two levels of `variable` for the pairwise analysis. |
| `include` | (`vector`)<br>a vector of levels of the `variable` column to include in comparisons. Pairwise comparisons will only be performed for pairs that have a level specified here. Default is `NULL` and all pairwise computations are included. |

## Value

list of ARDs

## Examples

```
ard_pairwise(
  ADSL,
  variable = ARM,
  .f = \(df) {
    ard_mvsummary(
      df,
      variables = AGE,
    statistic = ~ list(ttest = \(x, data, ...) t.test(x ~ data$ARM)[c("statistic", "p.value")])
    )
  },
  include = "Placebo" # only include comparisons to the "Placebo" group
)
```

ard_stack *Stack ARDs*

## Description

Stack multiple ARD calls sharing common input `data` and `by` variables. Optionally incorporate additional information on represented variables, e.g. overall calculations, rates of missingness, attributes, or transform results with `shuffle_ard()`.

If the `ard_stack(by)` argument is specified, a univariate tabulation of the by variable will also be returned.

## Usage

```
ard_stack(
  data,
  ...,
  .by = NULL,
  .overall = FALSE,
  .missing = FALSE,
  .attributes = FALSE,
  .total_n = FALSE,
  .shuffle = FALSE,
  .by_stats = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `...` | (dynamic-dots)<br>Series of ARD function calls to be run and stacked |
| `.by` | (tidy-select)<br>columns to tabulate by in the series of ARD function calls. Any rows with `NA` or `NaN` values are removed from all calculations. |
| `.overall` | (`logical`)<br>logical indicating whether overall statistics should be calculated (i.e. re-run all `ard_*()` calls with `by=NULL`). Default is `FALSE`. |
| `.missing` | (`logical`)<br>logical indicating whether to include the results of `ard_missing()` for all variables represented in the ARD. Default is `FALSE`. |
| `.attributes` | (`logical`)<br>logical indicating whether to include the results of `ard_attributes()` for all variables represented in the ARD. Default is `FALSE`. |
| `.total_n` | (`logical`)<br>logical indicating whether to include of `ard_total_n()` in the returned ARD. |

| | |
|---|---|
| .shuffle | **[Deprecated]** support for .shuffle = TRUE has been removed. |
| .by_stats | (logical)<br>logical indicating whether to include overall stats of the by variables in the returned ARD. |

## Value

an ARD data frame of class 'card'

## Examples

```
ard_stack(
  data = ADSL,
  ard_tabulate(variables = "AGEGR1"),
  ard_summary(variables = "AGE"),
  .by = "ARM",
  .overall = TRUE,
  .attributes = TRUE
)

ard_stack(
  data = ADSL,
  ard_tabulate(variables = "AGEGR1"),
  ard_summary(variables = "AGE"),
  .by = "ARM"
)
```

---

# ard_stack_hierarchical

*Stacked Hierarchical ARD Statistics*

---

## Description

Use these functions to calculate multiple summaries of nested or hierarchical data in a single call.

- ard_stack_hierarchical(): Calculates *rates* of events (e.g. adverse events) utilizing the denominator and id arguments to identify the rows in data to include in each rate calculation.
- ard_stack_hierarchical_count(): Calculates *counts* of events utilizing all rows for each tabulation.

## Usage

```
ard_stack_hierarchical(
  data,
  variables,
  by = dplyr::group_vars(data),
  id,
```

```
  denominator,
  include = everything(),
  statistic = everything() ~ c("n", "N", "p"),
  overall = FALSE,
  over_variables = FALSE,
  attributes = FALSE,
  total_n = FALSE,
  shuffle = FALSE,
  by_stats = TRUE
)

ard_stack_hierarchical_count(
  data,
  variables,
  by = dplyr::group_vars(data),
  denominator = NULL,
  include = everything(),
  overall = FALSE,
  over_variables = FALSE,
  attributes = FALSE,
  total_n = FALSE,
  shuffle = FALSE,
  by_stats = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `variables` | (tidy-select)<br>Specifies the nested/hierarchical structure of the data. The variables that are specified here and in the `include` argument will have summary statistics calculated. |
| `by` | (tidy-select)<br>variables to perform tabulations by. All combinations of the variables specified here appear in results. Default is `dplyr::group_vars(data)`. |
| `id` | (tidy-select)<br>argument used to subset `data` to identify rows in `data` to calculate event rates in `ard_stack_hierarchical()`. See details below. |
| `denominator` | (`data.frame`, `integer`)<br>used to define the denominator and enhance the output. The argument is required for `ard_stack_hierarchical()` and optional for `ard_stack_hierarchical_count()`.<br>• the univariate tabulations of the by variables are calculated with `denominator`, when a data frame is passed, e.g. tabulation of the treatment assignment counts that may appear in the header of a table.<br>• the `denominator` argument must be specified when `id` is used to calculate the event rates. |

- if `total_n=TRUE`, the `denominator` argument is used to return the total N

`include` (`tidy-select`)
Specify the subset a columns indicated in the `variables` argument for which summary statistics will be returned. Default is `everything()`.

`statistic` (`formula-list-selector`)
a named list, a list of formulas, or a single formula where the list element one or more of `c("n", "N", "p", "n_cum", "p_cum")` (on the RHS of a formula).

`overall` (scalar `logical`)
logical indicating whether overall statistics should be calculated (i.e. repeat the operations with `by=NULL` in *most cases*, see below for details). Default is `FALSE`.

`over_variables` (scalar `logical`)
logical indicating whether summary statistics should be calculated over or across the columns listed in the `variables` argument. Default is `FALSE`.

`attributes` (scalar `logical`)
logical indicating whether to include the results of `ard_attributes()` for all variables represented in the ARD. Default is `FALSE`.

`total_n` (scalar `logical`)
logical indicating whether to include of `ard_total_n(denominator)` in the returned ARD.

`shuffle` **[Deprecated]** support for `.shuffle = TRUE` has been removed.

`by_stats` (`logical`)
logical indicating whether to include overall stats of the by variables in the returned ARD.

## Value

an ARD data frame of class 'card'

## Subsetting Data for Rate Calculations

To calculate event rates, the `ard_stack_hierarchical()` function identifies rows to include in the calculation. First, the primary data frame is sorted by the columns identified in the `id`, `by`, and `variables` arguments.

As the function cycles over the variables specified in the `variables` argument, the data frame is grouped by `id`, `intersect(by, names(denominator))`, and `variables` utilizing the last row within each of the groups.

For example, if the call is `ard_stack_hierarchical(data = ADAE, variables = c(AESOC, AEDECOD), id = USUBJID)`, then we'd first subset ADAE to be one row within the grouping `c(USUBJID, AESOC, AEDECOD)` to calculate the event rates in `'AEDECOD'`. We'd then repeat and subset ADAE to be one row within the grouping `c(USUBJID, AESOC)` to calculate the event rates in `'AESOC'`.

## Overall Argument

When we set `overall=TRUE`, we wish to re-run our calculations removing the stratifying columns. For example, if we ran the code below, we results would include results with the code chunk being re-run with `by=NULL`.

```
ard_stack_hierarchical(
  data = ADAE,
  variables = c(AESOC, AEDECOD),
  by = TRTA,
  denominator = ADSL,
  id = USUBJID,
  overall = TRUE
)
```

But there is another case to be aware of: when the by argument includes columns that are not present in the denominator, for example when tabulating results by AE grade or severity in addition to treatment assignment. In the example below, we're tabulating results by treatment assignment and AE severity. By specifying overall=TRUE, we will re-run the to get results with by = AESEV and again with by = NULL.

```
ard_stack_hierarchical(
  data = ADAE,
  variables = c(AESOC, AEDECOD),
  by = c(TRTA, AESEV),
  denominator = ADSL,
  id = USUBJID,
  overall = TRUE
)
```

## Examples

```
ard_stack_hierarchical(
  ADAE,
  variables = c(AESOC, AEDECOD),
  by = TRTA,
  denominator = ADSL,
  id = USUBJID
)

ard_stack_hierarchical_count(
  ADAE,
  variables = c(AESOC, AEDECOD),
  by = TRTA,
  denominator = ADSL
)
```

---

# ard_strata *Stratified ARD*

---

## Description

General function for calculating ARD results within subgroups.

While the examples below show use with other functions from the cards package, this function would primarily be used with the statistical functions in the cardx functions.

## Usage

```
ard_strata(.data, .by = NULL, .strata = NULL, .f, ...)
```

## Arguments

| | |
|---|---|
| `.data` | (`data.frame`)<br>a data frame |
| `.by`, `.strata` | (`tidy-select`)<br>columns to tabulate by/stratify by for calculation. Arguments are similar, but with an important distinction:<br>`.by`: results are tabulated by **all combinations** of the columns specified, including unobserved combinations and unobserved factor levels.<br>`.strata`: results are tabulated by **all *observed* combinations** of the columns specified.<br>These argument *should not* include any columns that appear in the `.f` argument. |
| `.f` | (`function`, `formula`)<br>a function or a formula that can be coerced to a function with `rlang::as_function()` (similar to `purrr::map(.f)`) |
| `...` | Additional arguments passed on to the `.f` function. |

## Value

an ARD data frame of class 'card'

## Examples

```
# Example 1 ---------------------------------
ard_strata(
  ADSL,
  .by = ARM,
  .f = ~ ard_summary(.x, variables = AGE)
)

# Example 2 ---------------------------------
df <- data.frame(
  USUBJID = 1:12,
  PARAMCD = rep(c("PARAM1", "PARAM2"), each = 6),
  AVALC = c(
    "Yes", "No", "Yes", # PARAM1
    "Yes", "Yes", "No", # PARAM1
    "Low", "Medium", "High", # PARAM2
    "Low", "Low", "Medium" # PARAM2
  )
)

ard_strata(
  df,
  .strata = PARAMCD,
  .f = \(.x) {
```

```
    lvls <-
      switch(.x[["PARAMCD"]][1],
        "PARAM1" = c("Yes", "No"),
        "PARAM2" = c("Zero", "Low", "Medium", "High")
      )

    .x |>
      dplyr::mutate(AVALC = factor(AVALC, levels = lvls)) |>
      ard_tabulate(variables = AVALC)
  }
)
```

---

ard_summary *Univariate ARD Statistics*

---

## Description

Compute Analysis Results Data (ARD) for simple continuous summary statistics.

## Usage

```
ard_summary(data, ...)

## S3 method for class 'data.frame'
ard_summary(
  data,
  variables,
  by = dplyr::group_vars(data),
  strata = NULL,
  statistic = everything() ~ continuous_summary_fns(),
  fmt_fun = NULL,
  stat_label = everything() ~ default_stat_labels(),
  fmt_fn = deprecated(),
  ...
)
```

## Arguments

| | |
|---|---|
| data | (data.frame)<br>a data frame |
| ... | Arguments passed to methods. |
| variables | (tidy-select)<br>columns to include in summaries. |
| by, strata | (tidy-select)<br>columns to tabulate by/stratify by for summary statistic calculation. Arguments are similar, but with an important distinction: |

by: results are calculated for **all combinations** of the columns specified, including unobserved combinations and unobserved factor levels.

strata: results are calculated for **all *observed* combinations** of the columns specified.

Arguments may be used in conjunction with one another.

`statistic` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) mean(x)))`.

The value assigned to each variable must also be a named list, where the names are used to reference a function and the element is the function object. Typically, this function will return a scalar statistic, but a function that returns a named list of results is also acceptable, e.g. `list(conf.low = -1, conf.high = 1)`. However, when errors occur, the messaging will be less clear in this setting.

`fmt_fun` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) round(x, digits`

`stat_label` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. `everything() ~ list(mean = "Mean", sd = "SD")` or `everything() ~ list(mean ~ "Mean", sd ~ "SD")`.

`fmt_fn` **[Deprecated]**

## Value

an ARD data frame of class 'card'

## Examples

```
ard_summary(ADSL, by = "ARM", variables = "AGE")

# if a single function returns a named list, the named
# results will be placed in the resulting ARD
ADSL |>
  dplyr::group_by(ARM) |>
  ard_summary(
    variables = "AGE",
    statistic =
      ~ list(conf.int = \(x) t.test(x)[["conf.int"]] |>
        as.list() |>
        setNames(c("conf.low", "conf.high")))
  )
```

`ard_tabulate` *Tabulate ARD*

## Description

Compute Analysis Results Data (ARD) for categorical summary statistics.

## Usage

```
ard_tabulate(data, ...)

## S3 method for class 'data.frame'
ard_tabulate(
  data,
  variables,
  by = dplyr::group_vars(data),
  strata = NULL,
  statistic = everything() ~ c("n", "p", "N"),
  denominator = "column",
  fmt_fun = NULL,
  stat_label = everything() ~ default_stat_labels(),
  fmt_fn = deprecated(),
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `...` | Arguments passed to methods. |
| `variables` | (`tidy-select`)<br>columns to include in summaries. Default is `everything()`. |
| `by, strata` | (`tidy-select`)<br>columns to use for grouping or stratifying the table output. Arguments are similar, but with an important distinction:<br>`by`: results are tabulated by **all combinations** of the columns specified, including unobserved combinations and unobserved factor levels.<br>`strata`: results are tabulated by **all *observed* combinations** of the columns specified.<br>Arguments may be used in conjunction with one another. |
| `statistic` | (`formula-list-selector`)<br>a named list, a list of formulas, or a single formula where the list element one or more of `c("n", "N", "p", "n_cum", "p_cum")` (on the RHS of a formula). |
| `denominator` | (`string`, `data.frame`, `integer`)<br>Specify this argument to change the denominator, e.g. the `"N"` statistic. Default is `'column'`. See below for details. |

`fmt_fun` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) round(x, digits`

`stat_label` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. `everything() ~ list(n = "n", p = "pct")` or `everything() ~ list(n ~ "n", p ~ "pct")`.

`fmt_fn` **[Deprecated]**

## Value

an ARD data frame of class 'card'

## Denominators

By default, the `ard_tabulate()` function returns the statistics `"n"`, `"N"`, and `"p"`, where little `"n"` are the counts for the variable levels, and big `"N"` is the number of non-missing observations. The calculation for the proportion is `p = n/N`.

However, it is sometimes necessary to provide a different `"N"` to use as the denominator in this calculation. For example, in a calculation of the rates of various observed adverse events, you may need to update the denominator to the number of enrolled subjects.

In such cases, use the `denominator` argument to specify a new definition of `"N"`, and subsequently `"p"`. The argument expects one of the following inputs:

- a string: one of `"column"`, `"row"`, or `"cell"`.
  - `"column"`, the default, returns percentages where the sum is equal to one within the variable after the data frame has been subset with `by`/`strata`.
  - `"row"` gives 'row' percentages where `by`/`strata` columns are the 'top' of a cross table, and the variables are the rows. This is well-defined for a single `by` or `strata` variable, and care must be taken when there are more to ensure the the results are as you expect.
  - `"cell"` gives percentages where the denominator is the number of non-missing rows in the source data frame.
- a data frame. Any columns in the data frame that overlap with the `by`/`strata` columns will be used to calculate the new `"N"`.
- an integer. This single integer will be used as the new `"N"`
- a structured data frame. The data frame will include columns from `by`/`strata`. The last column must be named `"...ard_N..."`. The integers in this column will be used as the updated `"N"` in the calculations.

When the `p` statistic is returned, the proportion is returned—bounded by `[0, 1]`. The default function to format the statistic scales the proportion by 100 and the percentage is returned which matches the default statistic label of `'%'`. To get the formatted values, pass the ARD to `apply_fmt_fun()`.

## Examples

```
ard_tabulate(ADSL, by = "ARM", variables = "AGEGR1")
```

```
ADSL |>
  dplyr::group_by(ARM) |>
  ard_tabulate(
    variables = "AGEGR1",
    statistic = everything() ~ "n"
  )
```

---

`ard_tabulate_rows` *Row Tabulate ARD*

---

## Description

Tabulate the number of rows in a data frame.

## Usage

```
ard_tabulate_rows(
  data,
  colname = "..row_count..",
  by = dplyr::group_vars(data),
  strata = NULL,
  fmt_fun = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `colname` | (`string`)<br>name of the column that will be returned along with the row tabulation. |
| `by`, `strata` | (`tidy-select`)<br>columns to use for grouping or stratifying the table output. Arguments are similar, but with an important distinction:<br>`by`: results are tabulated by **all combinations** of the columns specified, including unobserved combinations and unobserved factor levels.<br>`strata`: results are tabulated by **all *observed* combinations** of the columns specified.<br>Arguments may be used in conjunction with one another. |
| `fmt_fun` | (`formula-list-selector`)<br>a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) round(x, digits` |

## Value

an ARD data frame of class 'card'

## Examples

```
ard_tabulate_rows(ADSL, by = TRTA)
```

---

`ard_tabulate_value` *Tabulate Value ARD*

---

## Description

Tabulate an Analysis Results Data (ARD) for dichotomous or a specified value.

## Usage

```
ard_tabulate_value(data, ...)

## S3 method for class 'data.frame'
ard_tabulate_value(
  data,
  variables,
  by = dplyr::group_vars(data),
  strata = NULL,
  value = maximum_variable_value(data[variables]),
  statistic = everything() ~ c("n", "N", "p"),
  denominator = NULL,
  fmt_fun = NULL,
  stat_label = everything() ~ default_stat_labels(),
  fmt_fn = deprecated(),
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `...` | Arguments passed to methods. |
| `variables` | (tidy-select)<br>columns to include in summaries. Default is `everything()`. |
| `by, strata` | (tidy-select)<br>columns to use for grouping or stratifying the table output. Arguments are similar, but with an important distinction:<br>`by`: results are tabulated by **all combinations** of the columns specified, including unobserved combinations and unobserved factor levels.<br>`strata`: results are tabulated by **all *observed* combinations** of the columns specified.<br>Arguments may be used in conjunction with one another. |

`value` (named `list`)
named list of values to tabulate. Default is `maximum_variable_value(data)`, which returns the largest/last value after a sort.

`statistic` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element one or more of `c("n", "N", "p", "n_cum", "p_cum")` (on the RHS of a formula).

`denominator` (`string`, `data.frame`, `integer`)
Specify this argument to change the denominator, e.g. the `"N"` statistic. Default is `'column'`. See below for details.

`fmt_fun` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is a named list of functions (or the RHS of a formula), e.g. `list(mpg = list(mean = \(x) round(x, digits`

`stat_label` (formula-list-selector)
a named list, a list of formulas, or a single formula where the list element is either a named list or a list of formulas defining the statistic labels, e.g. `everything() ~ list(n = "n", p = "pct")` or `everything() ~ list(n ~ "n", p ~ "pct")`.

`fmt_fn` **[Deprecated]**

## Value

an ARD data frame of class 'card'

## Denominators

By default, the `ard_tabulate()` function returns the statistics `"n"`, `"N"`, and `"p"`, where little `"n"` are the counts for the variable levels, and big `"N"` is the number of non-missing observations. The calculation for the proportion is `p = n/N`.

However, it is sometimes necessary to provide a different `"N"` to use as the denominator in this calculation. For example, in a calculation of the rates of various observed adverse events, you may need to update the denominator to the number of enrolled subjects.

In such cases, use the `denominator` argument to specify a new definition of `"N"`, and subsequently `"p"`. The argument expects one of the following inputs:

- a string: one of `"column"`, `"row"`, or `"cell"`.
  - `"column"`, the default, returns percentages where the sum is equal to one within the variable after the data frame has been subset with `by`/`strata`.
  - `"row"` gives 'row' percentages where `by`/`strata` columns are the 'top' of a cross table, and the variables are the rows. This is well-defined for a single `by` or `strata` variable, and care must be taken when there are more to ensure the the results are as you expect.
  - `"cell"` gives percentages where the denominator is the number of non-missing rows in the source data frame.
- a data frame. Any columns in the data frame that overlap with the `by`/`strata` columns will be used to calculate the new `"N"`.
- an integer. This single integer will be used as the new `"N"`

- a structured data frame. The data frame will include columns from by/strata. The last column must be named "...ard_N...". The integers in this column will be used as the updated "N" in the calculations.

When the p statistic is returned, the proportion is returned—bounded by [0, 1]. The default function to format the statistic scales the proportion by 100 and the percentage is returned which matches the default statistic label of '%'. To get the formatted values, pass the ARD to apply_fmt_fun().

## Examples

```
ard_tabulate_value(mtcars, by = vs, variables = c(cyl, am), value = list(cyl = 4))

mtcars |>
  dplyr::group_by(vs) |>
  ard_tabulate_value(
    variables = c(cyl, am),
    value = list(cyl = 4),
    statistic = ~"p"
  )
```

---

# ard_total_n *ARD Total N*

---

## Description

Returns the total N for the data frame. The placeholder variable name returned in the object is "..ard_total_n.."

## Usage

```
ard_total_n(data, ...)

## S3 method for class 'data.frame'
ard_total_n(data, ...)
```

## Arguments

| | |
|---|---|
| data | (data.frame)<br>a data frame |
| ... | Arguments passed to methods. |

## Value

an ARD data frame of class 'card'

## Examples

```
ard_total_n(ADSL)
```

`as_card` *Data Frame as ARD*

## Description

Convert data frames to ARDs of class 'card'.

## Usage

```
as_card(x, check = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>a data frame |
| `check` | (scalar `logical`)<br>Whether the input data frame should be checked for standard ARD features |

## Value

an ARD data frame of class 'card'

## Examples

```
data.frame(
  stat_name = c("N", "mean"),
  stat_label = c("N", "Mean"),
  stat = c(10, 0.5)
) |>
  as_card(check = FALSE)
dplyr::tibble(
  variable = "AGE",
  stat_name = c("N", "mean"),
  stat_label = c("N", "Mean"),
  stat = list(10, 0.5),
  fmt_fun = replicate(2, list()),
  warning = replicate(2, list()),
  error = replicate(2, list())
) |>
  as_card()
```

as_cards_fn *As card function*

## Description

Add attributes to a function that specify the expected results. It is used when `ard_summary()` or `ard_mvsummary()` errors and constructs an ARD with the correct structure when the results cannot be calculated.

## Usage

```
as_cards_fn(f, stat_names)

is_cards_fn(f)

get_cards_fn_stat_names(f)
```

## Arguments

| | |
|---|---|
| `f` | (`function`)<br>a function |
| `stat_names` | (`character`)<br>a character vector of the expected statistic names returned by function `f` |

## Value

an ARD data frame of class 'card'

## Examples

```
# When there is no error, everything works as if we hadn't used `as_card_fn()`
ttest_works <-
  as_cards_fn(
    \(x) t.test(x)[c("statistic", "p.value")],
    stat_names = c("statistic", "p.value")
  )
ard_summary(
  mtcars,
  variables = mpg,
  statistic = ~ list(ttest = ttest_works)
)

# When there is an error and we use `as_card_fn()`,
#   we will see the same structure as when there is no error
ttest_error <-
  as_cards_fn(
    \(x) {
      t.test(x)[c("statistic", "p.value")]
      stop("Intentional Error")
```

```
    },
    stat_names = c("statistic", "p.value")
  )
ard_summary(
  mtcars,
  variables = mpg,
  statistic = ~ list(ttest = ttest_error)
)

# if we don't use `as_card_fn()` and there is an error,
#   the returned result is only one row
ard_summary(
  mtcars,
  variables = mpg,
  statistic = ~ list(ttest = \(x) {
    t.test(x)[c("statistic", "p.value")]
    stop("Intentional Error")
  })
)
```

---

`as_nested_list` *ARD as Nested List*

---

## Description

**[Experimental]**
Convert ARDs to nested lists.

## Usage

```
as_nested_list(x)
```

## Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>an ARD data frame of class 'card' |

## Value

a nested list

## Examples

```
ard_summary(mtcars, by = "cyl", variables = c("mpg", "hp")) |>
  as_nested_list()
```

bind_ard *Bind ARDs*

## Description

Wrapper for `dplyr::bind_rows()` with additional checks for duplicated statistics.

## Usage

```
bind_ard(
  ...,
  .distinct = TRUE,
  .update = FALSE,
  .order = FALSE,
  .quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `...` | (dynamic-dots)<br>ARDs to combine. Each argument can either be an ARD, or a list of ARDs. Columns are matched by name, and any missing columns will be filled with `NA`. |
| `.distinct` | (`logical`)<br>logical indicating whether to remove non-distinct values from the ARD. Duplicates are checked across grouping variables, primary variables, context (if present), the **statistic name and the statistic value**. Default is `TRUE`. If a statistic name and value is repeated and `.distinct=TRUE`, the more recently added statistics will be retained, and the other(s) omitted. |
| `.update` | (`logical`)<br>logical indicating whether to update ARD and remove duplicated named statistics. Duplicates are checked across grouping variables, primary variables, and the **statistic name**. Default is `FALSE`. If a statistic name is repeated and `.update=TRUE`, the more recently added statistics will be retained, and the other(s) omitted. |
| `.order` | (`logical`)<br>logical indicating whether to order the rows of the stacked ARDs, allowing statistics that share common group and variable values to appear in consecutive rows. Default is `FALSE`. Ordering will be based on the order of the group/variable values prior to stacking. |
| `.quiet` | (`logical`)<br>logical indicating whether to suppress any messaging. Default is `FALSE` |

## Value

an ARD data frame of class 'card'

## Examples

```
ard <- ard_tabulate(ADSL, by = "ARM", variables = "AGEGR1")

bind_ard(ard, ard, .update = TRUE)
```

---

`cards.options` *Options in {cards}*

---

## Description

See below for options available in the {cards} package

### cards.round_type

There are two types of rounding types in the {cards} package that are implemented in `label_round()`, `alias_as_fmt_fun()`, and `apply_fmt_fun()` functions.

- `'round-half-up'` (*default*): rounding method where values exactly halfway between two numbers are rounded to the larger in magnitude number. Rounding is implemented via `round5()`.
- `'round-to-even'`: base R's default IEC 60559 rounding standard. See `round()` for details.

To change the default rounding to use IEC 60559, this option must be set **both** when the ARDs are created and when `apply_fmt_fun()` is run. This ensures that any *default* formatting functions created with `label_round()` utilize the specified rounding method and the method is used what aliases are converted into functions (which occurs in `apply_fmt_fun()` when it calls `alias_as_fmt_fun()`).

---

`check_ard_structure` *Check ARD Structure*

---

## Description

Function tests the structure and returns notes when object does not conform to expected structure.

## Usage

```
check_ard_structure(
  x,
  column_order = TRUE,
  method = TRUE,
  error_on_fail = FALSE
)
```

### Arguments

| | |
|---|---|
| x | (data.frame)<br>an ARD data frame of class 'card' |
| column_order | (scalar logical)<br>check whether ordering of columns adheres to to cards::tidy_ard_column_order(). |
| method | (scalar logical)<br>check whether a "stat_name" equal to "method" appears in results. |
| error_on_fail | (scalar logical)<br>Error if a check is failed? FALSE by default. |

### Value

an ARD data frame of class 'card' (invisible)

### Examples

```
ard_summary(ADSL, variables = "AGE") |>
  dplyr::select(-warning, -error) |>
  check_ard_structure()
```

---

## compare_ard *Compare ARDs*

---

### Description

**[Experimental]**
compare_ard() compares columns of two ARDs row-by-row using a shared set of key columns. Rows where the column values differ are returned.

The is_ard_equal() function accepts a compare_ard() object, and returns TRUE or FALSE depending on whether the comparison reported difference. check_ard_equal() returns as error if not equal.

### Usage

```
compare_ard(
  x,
  y,
 keys = c(all_ard_groups(), all_ard_variables(), any_of(c("variable", "variable_level",
    "stat_name"))),
  columns = any_of(c("stat_label", "stat", "stat_fmt")),
  tolerance = sqrt(.Machine$double.eps),
  check.attributes = TRUE
)

is_ard_equal(x)

check_ard_equal(x)
```

## Arguments

| | |
|---|---|
| x | (card)<br>first ARD to compare. |
| y | (card)<br>second ARD to compare. |
| keys | (tidy-select)<br>columns identifying unique records. The intersection of the selected columns in both ARDs is used. Default is `c(all_ard_groups(), all_ard_variables(), any_of(c("variable", "variable_level", "stat_name")))`. |
| columns | (tidy-select)<br>columns to compare between the two ARDs. Default is `any_of(c("stat_label", "stat", "stat_fmt"))`. |
| tolerance | (numeric(1))<br>numeric tolerance passed to `all.equal()` for numeric comparisons. Default is `sqrt(.Machine$double.eps)`. |
| check.attributes | (logical(1))<br>logical passed to `all.equal()` indicating whether object attributes (e.g. names) should be compared. Default is `TRUE`. |

## Value

a named list of class `"ard_comparison"` containing:

- `rows_in_x_not_y`: data frame of rows present in `x` but not in `y` (based on key columns)
- `rows_in_y_not_x`: data frame of rows present in `y` but not in `x` (based on key columns)
- `compare`: a named list where each element is a data frame containing the key columns, the compared column values from both ARDs, and a `difference` column with the `all.equal()` description for rows where values differ

## Examples

```
base <- ard_summary(ADSL, by = ARM, variables = AGE)
compare <- ard_summary(dplyr::mutate(ADSL, AGE = AGE + 1),
  by = ARM,
  variables = AGE
)

compare_ard(base, compare)$compare$stat
```

## default_stat_labels *Defaults for Statistical Arguments*

### Description

Returns a named list of statistics labels

### Usage

```
default_stat_labels()
```

### Value

named list

### Examples

```
# stat labels
default_stat_labels()
```

## deprecated *Deprecated functions*

### Description

**[Deprecated]**
Some functions have been deprecated and are no longer being actively supported.

**Renamed functions**

- `ard_categorical()` to `ard_tabulate()`
- `ard_continuous()` to `ard_summary()`
- `ard_complex()` to `ard_mvsummary()`
- `apply_fmt_fn()` to `apply_fmt_fun()`
- `alias_as_fmt_fn()` to `alias_as_fmt_fun()`
- `update_ard_fmt_fn()` to `update_ard_fmt_fun()`

**Deprecated functions**

- `shuffle_ard()`

**[Deprecated]**

This function ingests an ARD object and shuffles the information to prepare for analysis. Helpful for streamlining across multiple ARDs. Combines each group/group_level into 1 column, back fills missing grouping values from the variable levels where possible, and optionally trims statistics-level metadata.

## Usage

```
ard_continuous(data, ...)

ard_categorical(data, ...)

ard_complex(data, ...)

ard_dichotomous(data, ...)

## S3 method for class 'data.frame'
ard_continuous(data, ...)

## S3 method for class 'data.frame'
ard_categorical(data, ...)

## S3 method for class 'data.frame'
ard_complex(data, ...)

## S3 method for class 'data.frame'
ard_dichotomous(data, ...)

apply_fmt_fn(...)

alias_as_fmt_fn(...)

update_ard_fmt_fn(...)

shuffle_ard(x, trim = TRUE)
```

## Arguments

| | |
|---|---|
| data, ... | **[Deprecated]** |
| x | (data.frame)<br>an ARD data frame of class 'card' |
| trim | (logical)<br>logical representing whether or not to trim away statistic-level metadata and filter only on numeric statistic values. |

## Value

a tibble

## Examples

```
bind_ard(
  ard_tabulate(ADSL, by = "ARM", variables = "AGEGR1"),
  ard_tabulate(ADSL, variables = "ARM")
) |>
  shuffle_ard()
```

`eval_capture_conditions` *Evaluate and Capture Conditions*

## Description

`eval_capture_conditions()`

Evaluates an expression while also capturing error and warning conditions. Function always returns a named list `list(result=, warning=, error=)`. If there are no errors or warnings, those elements will be `NULL`. If there is an error, the result element will be `NULL`.

Messages are neither saved nor printed to the console.

Evaluation is done via `rlang::eval_tidy()`. If errors and warnings are produced using the `{cli}` package, the messages are processed with `cli::ansi_strip()` to remove styling from the message.

`captured_condition_as_message()`/`captured_condition_as_error()`

These functions take the result from `eval_capture_conditions()` and return errors or warnings as either messages (via `cli::cli_inform()`) or errors (via `cli::cli_abort()`). These functions handle cases where the condition messages may include curly brackets, which would typically cause issues when processed with the `cli::cli_*()` functions.

Functions return the `"result"` from `eval_capture_conditions()`.

## Usage

```
eval_capture_conditions(expr, data = NULL, env = caller_env())

captured_condition_as_message(
  x,
  message = c("The following {type} occured:", x = "{condition}"),
  type = c("error", "warning"),
  envir = rlang::current_env()
)

captured_condition_as_error(
  x,
  message = c("The following {type} occured:", x = "{condition}"),
  type = c("error", "warning"),
  call = get_cli_abort_call(),
  envir = rlang::current_env()
)
```

## Arguments

| | |
|---|---|
| `expr` | An expression or quosure to evaluate. |

| | |
|---|---|
| data | A data frame, or named list or vector. Alternatively, a data mask created with as_data_mask() or new_data_mask(). Objects in data have priority over those in env. See the section about data masking. |
| env | The environment in which to evaluate expr. This environment is not applicable for quosures because they have their own environments. |
| x | (captured_condition)<br>a captured condition created by eval_capture_conditions(). |
| message | (character)<br>message passed to cli::cli_inform() or cli::cli_abort(). The condition being printed is saved in an object named condition, which should be included in this message surrounded by curly brackets. |
| type | (string)<br>the type of condition to return. Must be one of 'error' or 'warning'. |
| envir | Environment to evaluate the glue expressions in. |
| call | (environment)<br>Execution environment of currently running function. Default is get_cli_abort_call(). |

## Value

a named list

## Examples

```
# function executes without error or warning
eval_capture_conditions(letters[1:2])

# an error is thrown
res <- eval_capture_conditions(stop("Example Error!"))
res
captured_condition_as_message(res)

# if more than one warning is returned, all are saved
eval_capture_conditions({
  warning("Warning 1")
  warning("Warning 2")
  letters[1:2]
})

# messages are not printed to the console
eval_capture_conditions({
  message("A message!")
  letters[1:2]
})
```

`filter_ard_hierarchical` *Filter Stacked Hierarchical ARDs*

## Description

**[Experimental]**

This function is used to filter stacked hierarchical ARDs.

For the purposes of this function, we define a "variable group" as a combination of ARD rows grouped by the combination of all their variable levels, but excluding any by variables.

## Usage

```
filter_ard_hierarchical(
  x,
  filter,
  var = NULL,
  keep_empty = FALSE,
  quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `x` | (`card`)<br>a stacked hierarchical ARD of class `'card'` created using `ard_stack_hierarchical()` or `ard_stack_hierarchical_count()`. |
| `filter` | (`expression`)<br>an expression that is used to filter variable groups of the hierarchical ARD. See the Details section below. |
| `var` | (`tidy-select`)<br>hierarchy variable from `x` to perform filtering on. If `NULL`, the last hierarchy variable from `x` (`dplyr::last(attributes(x)$args$variables)`) will be used. |
| `keep_empty` | (scalar `logical`)<br>Logical argument indicating whether to retain summary rows corresponding to hierarchy sections that have had all rows filtered out. Default is `FALSE`. |
| `quiet` | (`logical`)<br>logical indicating whether to suppress any messaging. Default is `FALSE`. |

## Details

The `filter` argument can be used to filter out variable groups of a hierarchical ARD which do not meet the requirements provided as an expression. Variable groups can be filtered on the values of any of the possible statistics (`n`, `p`, and `N`) provided they are included at least once in the ARD, as well as the values of any by variables.

Additionally, filters can be applied on individual levels of the by variable via the n_XX, N_XX, and p_XX statistics, where each XX represents the index of the by variable level to select the statistic from. For example, filter = n_1 > 5 will check whether n values for the first level of by are greater than 5 in each row group.

Overall statistics for each row group can be used in filters via the n_overall, N_overall, and p_overall statistics. If the ARD is created with parameter overall=TRUE, then these overall statistics will be extracted directly from the ARD, otherwise the statistics will be derived where possible. If overall=FALSE, then n_overall can only be derived if the n statistic is present in the ARD for the filter variable, N_overall if the N statistic is present for the filter variable, and p_overall if both the n and N statistics are present for the filter variable.

By default, filters will be applied at the level of the innermost hierarchy variable, i.e. the last variable supplied to variables. If filters should instead be applied at the level of one of the outer hierarchy variables, the var parameter can be used to select a different variable to filter on. When var is set to a different (outer) variable and a level of the variable does not meet the filtering criteria then the section corresponding to that variable level and all sub-sections within that section will be removed.

To illustrate how the function works, consider the typical example below where the AE summaries are provided by treatment group.

```
ADAE |>
  dplyr::filter(AESOC == "GASTROINTESTINAL DISORDERS",
                AEDECOD %in% c("VOMITING", "DIARRHOEA")) |>
  ard_stack_hierarchical(
    variables = c(AESOC, AEDECOD),
    by = TRTA,
    denominator = ADSL,
    id = USUBJID
  )
```

| **SOC** / AE | Placebo | Xanomeline High Dose | Xanomeline Low Dose |
|---|---|---|---|
| **GASTROINTESTINAL DISORDERS** | 11 (13%) | 10 (12%) | 8 (9.5%) |
| DIARRHOEA | 9 (10%) | 4 (4.8%) | 5 (6.0%) |
| VOMITING | 3 (3.5%) | 7 (8.3%) | 3 (3.6%) |

Filters are applied to the summary statistics of the innermost variable in the hierarchy by default—AEDECOD in this case. If we wanted to filter based on SOC rates instead of AE rates we could specify var = AESOC instead. If any of the summary statistics meet the filter requirement for any of the treatment groups, the entire row is retained. For example, if filter = n >= 9 were passed, the criteria would be met for DIARRHOEA as the Placebo group observed 9 AEs and as a result the summary statistics for the other treatment groups would be retained as well. Conversely, no treatment groups' summary statistics satisfy the filter requirement for VOMITING so all rows associated with this AE would be removed.

In addition to filtering on individual statistic values, filters can be applied across the treatment groups (i.e. across all by variable values) by using aggregate functions such as sum() and mean(). For simplicity, it is suggested to use the XX_overall statistics in place of sum(XX) in equivalent scenarios. For example, n_overall is equivalent to sum(n). A value of filter = sum(n) >= 18 (or

`filter = n_overall >= 18`) retains AEs where the sum of the number of AEs across the treatment groups is greater than or equal to 18.

If `filter = n_overall >= 18` and `var = AESOC` then all rows corresponding to an SOC with an overall rate less than 18 - including all AEs within that SOC - will be removed.

If `ard_stack_hierarchical(overall=TRUE)` was run, the overall column is **not** considered in any filtering except for `XX_overall` statistics, if specified.

If `ard_stack_hierarchical(over_variables=TRUE)` was run, any overall statistics are kept regardless of filtering.

Some examples of possible filters:

- `filter = n > 5`: keep AEs where one of the treatment groups observed more than 5 AEs
- `filter = n == 2 & p < 0.05`: keep AEs where one of the treatment groups observed exactly 2 AEs *and* one of the treatment groups observed a proportion less than 5%
- `filter = n_overall >= 4`: keep AEs where there were 4 or more AEs observed across the treatment groups
- `filter = mean(n) > 4 | n > 3`: keep AEs where the mean number of AEs is 4 or more across the treatment groups *or* one of the treatment groups observed more than 3 AEs
- `filter = n_2 > 2`: keep AEs where the `"Xanomeline High Dose"` treatment group (second by variable level) observed more than 2 AEs

## Value

an ARD data frame of class 'card'

## See Also

`sort_ard_hierarchical()`

## Examples

```
# create a base AE ARD
ard <- ard_stack_hierarchical(
  ADAE,
  variables = c(AESOC, AEDECOD),
  by = TRTA,
  denominator = ADSL,
  id = USUBJID,
  overall = TRUE
)

# Example 1 ----------------------------------
# Keep AEs from TRTA groups where more than 3 AEs are observed across the group
filter_ard_hierarchical(ard, sum(n) > 3)

# Example 2 ----------------------------------
# Keep AEs where at least one level in the TRTA group has more than 3 AEs observed
filter_ard_hierarchical(ard, n > 3)

# Example 3 ----------------------------------
```

```
# Keep AEs that have an overall prevalence of greater than 5%
filter_ard_hierarchical(ard, sum(n) / sum(N) > 0.05)

# Example 4 ---------------------------------
# Keep AEs that have a difference in prevalence of greater than 3% between reference group with
# `TRTA = "Xanomeline High Dose"` and comparison group with `TRTA = "Xanomeline Low Dose"`
filter_ard_hierarchical(ard, abs(p_2 - p_3) > 0.03)

# Example 5 ---------------------------------
# Keep AEs from SOCs that have an overall prevalence of greater than 20%
filter_ard_hierarchical(ard, p_overall > 0.20, var = AESOC)
```

---

`get_ard_statistics` *ARD Statistics as List*

---

## Description

Returns the statistics from an ARD as a named list.

## Usage

```
get_ard_statistics(x, ..., .column = "stat", .attributes = NULL)
```

## Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>an ARD data frame of class 'card' |
| `...` | (dynamic-dots)<br>optional arguments indicating rows to subset of the ARD. For example, to return only rows where the column `"AGEGR1"` is `"65-80"`, pass `AGEGR1 %in% "65-80"`. |
| `.column` | (`string`)<br>string indicating the column that will be returned in the list. Default is `"statistic"` |
| `.attributes` | (`character`)<br>character vector of column names that will be returned in the list as attributes. Default is `NULL` |

## Value

named list

## Examples

```
ard <- ard_tabulate(ADSL, by = "ARM", variables = "AGEGR1")

get_ard_statistics(
  ard,
  group1_level %in% "Placebo",
```

```
  variable_level %in% "65-80",
  .attributes = "stat_label"
)
```

---

label_round *Generate Formatting Function*

---

## Description

Returns a function with the requested rounding and scaling schema.

## Usage

```
label_round(digits = 1, scale = 1, width = NULL)
```

## Arguments

| | |
|---|---|
| digits | (integer)<br>a non-negative integer specifying the number of decimal places round statistics to |
| scale | (numeric)<br>a scalar real number. Before rounding, the input will be scaled by this quantity |
| width | (integer)<br>a non-negative integer specifying the minimum width of the returned formatted values |

## Value

a function

## Examples

```
label_round(2)(pi)
label_round(1, scale = 100)(pi)
label_round(2, width = 5)(pi)
```

## maximum_variable_value *Maximum Value*

### Description

For each column in the passed data frame, the function returns a named list with the value being the largest/last element after a sort. For factors, the last level is returned, and for logical vectors TRUE is returned.

### Usage

```
maximum_variable_value(data)
```

### Arguments

| | |
|---|---|
| data | (data.frame)<br>a data frame |

### Value

a named list

### Examples

```
ADSL[c("AGEGR1", "BMIBLGR1")] |> maximum_variable_value()
```

## mock *Mock ARDs*

### Description

**[Experimental]**
Create empty ARDs used to create mock tables or table shells. Where applicable, the formatting functions are set to return 'xx' or 'xx.x'.

### Usage

```
mock_categorical(
  variables,
  statistic = everything() ~ c("n", "p", "N"),
  by = NULL
)

mock_continuous(
  variables,
```

```
  statistic = everything() ~ c("N", "mean", "sd", "median", "p25", "p75", "min", "max"),
  by = NULL
)

mock_dichotomous(
  variables,
  statistic = everything() ~ c("n", "p", "N"),
  by = NULL
)

mock_missing(
  variables,
  statistic = everything() ~ c("N_obs", "N_miss", "N_nonmiss", "p_miss", "p_nonmiss"),
  by = NULL
)

mock_attributes(label)

mock_total_n()
```

## Arguments

| | |
|---|---|
| variables | (character or named list)<br>a character vector of variable names for functions mock_continuous(), mock_missing(), and mock_attributes().<br>a named list for functions mock_categorical() and mock_dichotomous(), where the list element is a vector of variable values. For mock_dichotomous(), only a single value is allowed for each variable. |
| statistic | (formula-list-selector)<br>a named list, a list of formulas, or a single formula where the list elements are character vectors of statistic names to appear in the ARD. |
| by | (named list)<br>a named list where the list element is a vector of variable values. |
| label | (named list)<br>named list of variable labels, e.g. list(cyl = "No. Cylinders"). |

## Value

an ARD data frame of class 'card'

## Examples

```
mock_categorical(
  variables =
    list(
      AGEGR1 = factor(c("<65", "65-80", ">80"), levels = c("<65", "65-80", ">80"))
    ),
  by = list(TRTA = c("Placebo", "Xanomeline High Dose", "Xanomeline Low Dose"))
) |>
```

```
  apply_fmt_fun()

mock_continuous(
  variables = c("AGE", "BMIBL"),
  by = list(TRTA = c("Placebo", "Xanomeline High Dose", "Xanomeline Low Dose"))
) |>
  # update the mock to report 'xx.xx' for standard deviations
 update_ard_fmt_fun(variables = c("AGE", "BMIBL"), stat_names = "sd", fmt_fun = \(x) "xx.xx") |>
  apply_fmt_fun()
```

---

`nest_for_ard` *ARD Nesting*

---

## Description

This function is similar to `tidyr::nest()`, except that it retains rows for unobserved combinations (and unobserved factor levels) of by variables, and unobserved combinations of stratifying variables.

The levels are wrapped in lists so they can be stacked with other types of different classes.

## Usage

```
nest_for_ard(
  data,
  by = NULL,
  strata = NULL,
  key = "data",
  rename_columns = TRUE,
  list_columns = TRUE,
  include_data = TRUE,
  include_by_and_strata = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | (`data.frame`)<br>a data frame |
| `by, strata` | (`character`)<br>columns to nest by/stratify by. Arguments are similar, but with an important distinction:<br>`by`: data frame is nested by **all combinations** of the columns specified, including unobserved combinations and unobserved factor levels.<br>`strata`: data frame is nested by **all *observed* combinations** of the columns specified.<br>Arguments may be used in conjunction with one another. |
| `key` | (`string`)<br>the name of the new column with the nested data frame. Default is `"data"`. |

rename_columns (logical)
logical indicating whether to rename the by and strata variables. Default is TRUE.

list_columns (logical)
logical indicating whether to put levels of by and strata columns in a list. Default is TRUE.

include_data (scalar logical)
logical indicating whether to include the data subsets as a list-column. Default is TRUE.

include_by_and_strata
(logical)
When TRUE, the by and strata variables are included in the nested data frames.

## Value

a nested tibble

## Examples

```
nest_for_ard(
  data =
    ADAE |>
      dplyr::left_join(ADSL[c("USUBJID", "ARM")], by = "USUBJID") |>
      dplyr::filter(AOCCSFL %in% "Y"),
  by = "ARM",
  strata = "AESOC"
)
```

---

# print_ard_conditions *Print ARD Condition Messages*

---

## Description

Function parses the errors and warnings observed while calculating the statistics requested in the ARD and prints them to the console as messages.

## Usage

```
print_ard_conditions(x, condition_type = c("inform", "identity"))
```

## Arguments

x (data.frame)
an ARD data frame of class 'card'

condition_type (string)
indicates how warnings and errors are returned. Default is "inform" where all are returned as messages. When "identity", errors are returned as errors and warnings as warnings.

## Value

returns invisible if check is successful, throws all condition messages if not.

## Examples

```
# passing a character variable for numeric summary
ard_summary(ADSL, variables = AGEGR1) |>
  print_ard_conditions()
```

---

`process_selectors` *Process tidyselectors*

---

## Description

Functions process tidyselect arguments passed to functions in the cards package. The processed values are saved to the calling environment, by default.

- `process_selectors()`: the arguments will be processed with tidyselect and converted to a vector of character column names.
- `process_formula_selectors()`: for arguments that expect named lists or lists of formulas (where the LHS of the formula is a tidyselector). This function processes these inputs and returns a named list. If a name is repeated, the last entry is kept.
- `fill_formula_selectors()`: when users override the default argument values, it can be important to ensure that each column from a data frame is assigned a value. This function checks that each column in `data` has an assigned value, and if not, fills the value in with the default value passed here.
- `compute_formula_selector()`: used in `process_formula_selectors()` to evaluate a single argument.
- `check_list_elements()`: used to check the class/type/values of the list elements, primarily those processed with `process_formula_selectors()`.
- `cards_select()`: wraps `tidyselect::eval_select() |> names()`, and returns better contextual messaging when errors occur.

## Usage

```
process_selectors(data, ...)

process_formula_selectors(data, ...)

fill_formula_selectors(data, ...)

## S3 method for class 'data.frame'
process_selectors(data, ..., env = caller_env())

## S3 method for class 'data.frame'
process_formula_selectors(
```

```
  data,
  ...,
  env = caller_env(),
  include_env = FALSE,
  allow_empty = TRUE
)

## S3 method for class 'data.frame'
fill_formula_selectors(data, ..., env = caller_env())

compute_formula_selector(
  data,
  x,
  arg_name = caller_arg(x),
  env = caller_env(),
  strict = TRUE,
  include_env = FALSE,
  allow_empty = TRUE
)

check_list_elements(
  x,
  predicate,
  error_msg = NULL,
  arg_name = rlang::caller_arg(x)
)

cards_select(expr, data, ..., arg_name = NULL)
```

## Arguments

| | |
|---|---|
| data | (data.frame)<br>a data frame |
| ... | (dynamic-dots)<br>named arguments where the value of the argument is processed with tidyselect.<br>• process_selectors(): the values are tidyselect-compatible selectors<br>• process_formula_selectors(): the values are named lists, list of formulas a combination of both, or a single formula. Users may pass ~value as a shortcut for everything() ~ value.<br>• check_list_elements(): named arguments where the name matches an existing list in the env environment, and the value is a predicate function to test each element of the list, e.g. each element must be a string or a function. |
| env | (environment)<br>env to save the results to. Default is the calling environment. |
| include_env | (logical)<br>whether to include the environment from the formula object in the returned named list. Default is FALSE |

| | |
|---|---|
| allow_empty | (logical)<br>Logical indicating whether empty result is acceptable while process formula-list selectors. Default is TRUE. |
| x | • compute_formula_selector(): (formula-list-selector)<br>a named list, list of formulas, or a single formula that will be converted to a named list.<br>• check_list_elements(): (named list)<br>a named list |
| arg_name | (string)<br>the name of the argument being processed. Used in error messaging. Default is caller_arg(x). |
| strict | (logical)<br>whether to throw an error if a variable doesn't exist in the reference data (passed to tidyselect::eval_select()) |
| predicate | (function)<br>a predicate function that returns TRUE or FALSE |
| error_msg | (character)<br>a character vector that will be used in error messaging when mis-specified arguments are passed. Elements "{arg_name}" and "{variable}" are available using glue syntax for messaging. |
| expr | (expression)<br>Defused R code describing a selection according to the tidyselect syntax. |

## Value

process_selectors(), fill_formula_selectors(), process_formula_selectors() and check_list_elements() return NULL. compute_formula_selector() returns a named list.

## Examples

```
example_env <- rlang::new_environment()

process_selectors(ADSL, variables = starts_with("TRT"), env = example_env)
get(x = "variables", envir = example_env)

fill_formula_selectors(ADSL, env = example_env)

process_formula_selectors(
  ADSL,
  statistic = list(starts_with("TRT") ~ mean, TRTSDT = min),
  env = example_env
)
get(x = "statistic", envir = example_env)

check_list_elements(
  get(x = "statistic", envir = example_env),
  predicate = function(x) !is.null(x),
  error_msg = c(
    "Error in the argument {.arg {arg_name}} for variable {.val {variable}}.",
```

```
    "i" = "Value must be a named list of functions."
  )
)

# process one list
compute_formula_selector(ADSL, x = starts_with("U") ~ 1L)
```

---

`rename_ard_columns` *Rename ARD Variables*

---

## Description

Rename the grouping and variable columns to their original column names.

## Usage

```
rename_ard_columns(
  x,
  columns = c(all_ard_groups("names"), all_ard_variables("names")),
  fill = "{colname}",
  fct_as_chr = TRUE,
  unlist = NULL
)
```

## Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>an ARD data frame of class 'card' |
| `columns` | (`tidy-select`)<br>columns to rename, e.g. selecting columns `c('group1', 'group2', 'variable')` will rename `'group1_level'` to the name of the variable found in `'group1'`. When, for example, the `'group1_level'` does not exist, the values of the new column are filled with the values in the `fill` argument. Default is `c(all_ard_groups("names"), all_ard_variables("names"))`. |
| `fill` | (scalar/glue)<br>a scalar to fill column values when the variable does not have levels. If a character is passed, then it is processed with `glue::glue()` where the `colname` element is available to inject into the string, e.g. `'Overall {colname}'` may resolve to `'Overall AGE'` for an AGE column. Default is `'{colname}'`. |
| `fct_as_chr` | (scalar `logical`)<br>When `TRUE`, factor elements will be converted to character before unlisting. When the column being unlisted contains mixed types of classes, the factor elements are often converted to the underlying integer value instead of retaining the label. Default is `TRUE`. |
| `unlist` | **[Deprecated]** |

## Value

data frame

## Examples

```
# Example 1 ----------------------------------
ADSL |>
  ard_tabulate(by = ARM, variables = AGEGR1) |>
  apply_fmt_fun() |>
  rename_ard_columns() |>
  unlist_ard_columns()

# Example 2 ----------------------------------
ADSL |>
  ard_summary(by = ARM, variables = AGE) |>
  apply_fmt_fun() |>
  rename_ard_columns(fill = "Overall {colname}") |>
  unlist_ard_columns()
```

---

# rename_ard_groups *Rename ARD Group Columns*

---

## Description

Functions for renaming group columns names in ARDs.

## Usage

```
rename_ard_groups_shift(x, shift = -1)

rename_ard_groups_reverse(x)
```

## Arguments

| | |
|---|---|
| x | (data.frame)<br>an ARD data frame of class 'card'. |
| shift | (integer)<br>an integer specifying how many values to shift the group IDs, e.g. shift=-1 renames group2 to group1. |

## Value

an ARD data frame of class 'card'

## Examples

```
ard <- ard_summary(ADSL, by = c(SEX, ARM), variables = AGE)

# Example 1 ----------------------------------
rename_ard_groups_shift(ard, shift = -1)

# Example 2 ----------------------------------
rename_ard_groups_reverse(ard)
```

---

`replace_null_statistic`

*Replace NULL Statistics with Specified Value*

---

## Description

When a statistical summary function errors, the `"stat"` column will be `NULL`. It is, however, sometimes useful to replace these values with a non-`NULL` value, e.g. `NA`.

## Usage

```
replace_null_statistic(x, value = NA, rows = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>an ARD data frame of class 'card' |
| `value` | (usually a `scalar`)<br>The value to replace `NULL` values with. Default is `NA`. |
| `rows` | (`data-masking`)<br>Expression that return a logical value, and are defined in terms of the variables in `.data`. Only rows for which the condition evaluates to `TRUE` are replaced. Default is `TRUE`, which applies to all rows. |

## Value

an ARD data frame of class 'card'

## Examples

```
# the quantile functions error because the input is character, while the median function returns NA
data.frame(x = rep_len(NA_character_, 10)) |>
  ard_summary(
    variables = x,
    statistic = ~ continuous_summary_fns(c("median", "p25", "p75"))
  ) |>
  replace_null_statistic(rows = !is.null(error))
```

## round5 *Rounding of Numbers*

### Description

Rounds the values in its first argument to the specified number of decimal places (default 0). Importantly, `round5()` **does not** use Base R's "round to even" default. Standard rounding methods are implemented, for example, `cards::round5(0.5) = 1`, whereas `base::round(0.5) = 0`.

### Usage

```
round5(x, digits = 0)
```

### Arguments

| | |
|---|---|
| `x` | (`numeric`)<br>a numeric vector |
| `digits` | (`integer`)<br>integer indicating the number of decimal places |

### Details

Function inspired by `janitor::round_half_up()`.

### Value

a numeric vector

### Examples

```
x <- 0:4 / 2
round5(x) |> setNames(x)

# compare results to Base R
round(x) |> setNames(x)
```

## selectors *ARD Selectors*

## Description

These selection helpers match variables according to a given pattern.

- `all_ard_groups()`: Function selects grouping columns, e.g. columns named `"group##"` or `"group##_level"`.
- `all_ard_variables()`: Function selects variables columns, e.g. columns named `"variable"` or `"variable_level"`.
- `all_ard_group_n()`: Function selects `n` grouping columns.
- `all_missing_columns()`: Function selects columns that are all `NA` or empty.

## Usage

```
all_ard_groups(types = c("names", "levels"))

all_ard_variables(types = c("names", "levels"))

all_ard_group_n(n, types = c("names", "levels"))

all_missing_columns()
```

## Arguments

| | |
|---|---|
| `types` | (`character`)<br>type(s) of columns to select. `"names"` selects the columns variable name columns, and `"levels"` selects the level columns. Default is `c("names", "levels")`. |
| `n` | (`integer`)<br>integer(s) indicating which grouping columns to select. |

## Value

tidyselect output

## Examples

```
ard <- ard_tabulate(ADSL, by = "ARM", variables = "AGEGR1")

ard |> dplyr::select(all_ard_groups())
ard |> dplyr::select(all_ard_variables())
```

`sort_ard_hierarchical` *Sort Stacked Hierarchical ARDs*

## Description

**[Experimental]**

This function is used to sort stacked hierarchical ARDs.

For the purposes of this function, we define a "variable group" as a combination of ARD rows grouped by the combination of all their variable levels, but excluding any by variables.

## Usage

```
sort_ard_hierarchical(x, sort = everything() ~ "descending")
```

## Arguments

| | |
|---|---|
| `x` | (`card`)<br>a stacked hierarchical ARD of class `'card'` created using `ard_stack_hierarchical()` or `ard_stack_hierarchical_count()`. |
| `sort` | (`formula-list-selector`, `string`)<br>a named list, a list of formulas, a single formula where the list element is a named list of functions (or the RHS of a formula), or a single string specifying the types of sorting to perform at each hierarchy variable level. If the sort method for any variable is not specified then the method will default to `"descending"`. If a single unnamed string is supplied it is applied to all variables. For each variable, the value specified must be one of:<br>• `"alphanumeric"` - at the specified hierarchy level of the ARD, groups are ordered alphanumerically (i.e. A to Z) by `variable_level` text.<br>• `"descending"` - within each variable group of the ARD at the specified hierarchy level, count sums are calculated for each group and groups are sorted in descending order by sum. When `sort` is `"descending"` for a given variable and `n` is included in `statistic` for the variable then `n` is used to calculate variable group sums, otherwise `p` is used. If neither `n` nor `p` are present in `x` for the variable, an error will occur.<br>Defaults to `everything() ~ "descending"`. |

## Value

an ARD data frame of class 'card'

## Note

If overall data is present in `x` (i.e. the ARD was created with `ard_stack_hierarchical(overall=TRUE)`), the overall data will be sorted last within each variable group (i.e. after any other rows with the same combination of variable levels).

## See Also

filter_ard_hierarchical()

## Examples

```
ard_stack_hierarchical(
  ADAE,
  variables = c(AESOC, AEDECOD),
  by = TRTA,
  denominator = ADSL,
  id = USUBJID
) |>
  sort_ard_hierarchical(AESOC ~ "alphanumeric")

ard_stack_hierarchical_count(
  ADAE,
  variables = c(AESOC, AEDECOD),
  by = TRTA,
  denominator = ADSL
) |>
  sort_ard_hierarchical(sort = list(AESOC ~ "alphanumeric", AEDECOD ~ "descending"))
```

---

# summary_functions *Summary Functions*

---

## Description

- `continuous_summary_fns()` returns a named list of summary functions for continuous variables. Some functions include slight modifications to their base equivalents. For example, the `min()` and `max()` functions return `NA` instead of `Inf` when an empty vector is passed. Statistics `"p25"` and `"p75"` are calculated with `quantile(type = 2)`, which matches SAS's default value.

## Usage

```
continuous_summary_fns(
  summaries = c("N", "mean", "sd", "median", "p25", "p75", "min", "max"),
  other_stats = NULL
)
```

## Arguments

| | |
|---|---|
| `summaries` | (`character`)<br>a character vector of results to include in output. Select one or more from 'N', 'mean', 'sd', 'median', 'p25', 'p75', 'min', 'max'. |
| `other_stats` | (named `list`)<br>named list of other statistic functions to supplement the pre-programmed functions. |

## Value

named list of summary statistics

## Examples

```
# continuous variable summaries
ard_summary(
  ADSL,
  variables = "AGE",
  statistic = ~ continuous_summary_fns(c("N", "median"))
)
```

---

`tidy_ard_order` *Standard Order of ARD*

---

## Description

ARD functions for relocating columns and rows to the standard order.

- `tidy_ard_column_order()` relocates columns of the ARD to the standard order.
- `tidy_ard_row_order()` orders rows of ARD according to groups and strata (group 1, then group2, etc), while retaining the column order of the input ARD.

## Usage

```
tidy_ard_column_order(x, group_order = c("ascending", "descending"))

tidy_ard_row_order(x)
```

## Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>an ARD data frame of class 'card' |
| `group_order` | (`string`)<br>specifies the ordering of the grouping variables. Must be one of `c("ascending", "descending")`. Default is `"ascending"`, where grouping variables begin with `"group1"` variables, followed by `"group2"` variables, etc. |

## Value

an ARD data frame of class 'card'

## Examples

```
# order columns
ard <-
  dplyr::bind_rows(
    ard_summary(mtcars, variables = "mpg"),
    ard_summary(mtcars, variables = "mpg", by = "cyl")
  )

tidy_ard_column_order(ard) |>
  tidy_ard_row_order()
```

---

`unlist_ard_columns` *Unlist ARD Columns*

---

## Description

Unlist ARD Columns

## Usage

```
unlist_ard_columns(
  x,
  columns = c(where(is.list), -any_of(c("warning", "error", "fmt_fun"))),
  fill = NA,
  fct_as_chr = TRUE
)
```

## Arguments

| | |
|---|---|
| `x` | (`data.frame`)<br>an ARD data frame of class 'card' or any data frame |
| `columns` | (`tidy-select`)<br>columns to unlist. Default is `c(where(is.list), -any_of(c("warning", "error", "fmt_fun")))`. |
| `fill` | (scalar)<br>scalar to fill NULL values with before unlisting (if they are present). Default is `NA`. |
| `fct_as_chr` | (scalar `logical`)<br>When `TRUE`, factor elements will be converted to character before unlisting. When the column being unlisted contains mixed types of classes, the factor elements are often converted to the underlying integer value instead of retaining the label. Default is `TRUE`. |

## Value

a data frame

## Examples

```
ADSL |>
  ard_tabulate(by = ARM, variables = AGEGR1) |>
  apply_fmt_fun() |>
  unlist_ard_columns()

ADSL |>
  ard_summary(by = ARM, variables = AGE) |>
  apply_fmt_fun() |>
  unlist_ard_columns()
```

---

# update_ard *Update ARDs*

---

## Description

Functions used to update ARD formatting functions and statistic labels.

This is a helper function to streamline the update process. If it does not exactly meet your needs, recall that an ARD is just a data frame and it can be modified directly.

## Usage

```
update_ard_fmt_fun(
  x,
  variables = everything(),
  stat_names,
  fmt_fun,
  filter = TRUE,
  fmt_fn = deprecated()
)

update_ard_stat_label(
  x,
  variables = everything(),
  stat_names,
  stat_label,
  filter = TRUE
)
```

## Arguments

| | |
|---|---|
| x | (data.frame)<br>an ARD data frame of class 'card' |
| variables | (tidy-select)<br>variables in x$variable to apply update. Default is everything(). |

| | |
|---|---|
| stat_names | (character)<br>character vector of the statistic names (i.e. values from x$stat_name) to apply the update. |
| fmt_fun | (function)<br>a function or alias recognized by alias_as_fmt_fun(). |
| filter | (expression)<br>an expression that evaluates to a logical vector identifying rows in x to apply the update to. Default is TRUE, and update is applied to all rows. |
| fmt_fn | **[Deprecated]** |
| stat_label | (function)<br>a string of the updated statistic label. |

## Value

an ARD data frame of class 'card'

## Examples

```
ard_summary(ADSL, variables = AGE) |>
  update_ard_fmt_fun(stat_names = c("mean", "sd"), fmt_fun = 8L) |>
  update_ard_stat_label(stat_names = c("mean", "sd"), stat_label = "Mean (SD)") |>
  apply_fmt_fun()

# same as above, but only apply update to the Placebo level
ard_summary(
  ADSL,
  by = ARM,
  variables = AGE,
  statistic = ~ continuous_summary_fns(c("N", "mean"))
) |>
 update_ard_fmt_fun(stat_names = "mean", fmt_fun = 8L, filter = group1_level == "Placebo") |>
  apply_fmt_fun()
```

# Index