Package 'janitor'

May 8, 2026

Title Simple Tools for Examining and Cleaning Dirty Data

Version 2.2.1

Description The main janitor functions can: perfectly format data.frame column names; provide quick counts of variable combinations (i.e., frequency tables and crosstabs); and explore duplicate records. Other janitor functions nicely format the tabulation results. These tabulate-and-report functions approximate popular features of SPSS and Microsoft Excel. This package follows the principles of the ``tidyverse'' and works well with the pipe function %>%. janitor was built with beginning-to-intermediate R users in mind and is optimized for user-friendliness.

URL https://github.com/sfirke/janitor, https://sfirke.github.io/janitor/

BugReports https://github.com/sfirke/janitor/issues

Depends R (>= 3.1.2)

Imports dplyr (>= 1.0.0), hms, lifecycle, lubridate, magrittr, purrr, rlang, stringi, stringr, snakecase (>= 0.9.2), tidyselect (>= 1.0.0), tidyr (>= 0.7.0)

License MIT + file LICENSE

RoxygenNote 7.2.3

Suggests dbplyr, knitr, rmarkdown, RSQLite, sf, testthat (>= 3.0.0), tibble, tidygraph

VignetteBuilder knitr

Encoding UTF-8

Config/testthat/edition 3

NeedsCompilation no

Author Sam Firke [aut, cre], Bill Denney [ctb], Chris Haid [ctb], Ryan Knight [ctb], Malte Grosser [ctb], Jonathan Zadra [ctb]

Maintainer Sam Firke <samuel.firke@gmail.com>

Repository CRAN

Date/Publication 2024-12-22 16:30:01 UTC

Contents

add_totals_col *Append a totals column to a data.frame.*

Description

This function is deprecated, use `adorn_totals` instead.

Usage

```
add_totals_col(dat, na.rm = TRUE)
```

Arguments

`dat`	an input data.frame with at least one numeric column.
`na.rm`	should missing values (including NaN) be omitted from the calculations?

Value

Returns a data.frame with a totals column containing row-wise sums.

add_totals_row *Append a totals row to a data.frame.*

Description

This function is deprecated, use `adorn_totals` instead.

Usage

```
add_totals_row(dat, fill = "-", na.rm = TRUE)
```

Arguments

`dat`	an input data.frame with at least one numeric column.
`fill`	if there are more than one non-numeric columns, what string should fill the bottom row of those columns?
`na.rm`	should missing values (including NaN) be omitted from the calculations?

Value

Returns a data.frame with a totals row, consisting of "Total" in the first column and column sums in the others.

adorn_ns *Add underlying Ns to a tabyl displaying percentages.*

Description

This function adds back the underlying Ns to a `tabyl` whose percentages were calculated using `adorn_percentages()`, to display the Ns and percentages together. You can also call it on a non-tabyl data.frame to which you wish to append Ns.

Usage

```
adorn_ns(
  dat,
  position = "rear",
  ns = attr(dat, "core"),
  format_func = function(x) {
     format(x, big.mark = ",")
 },
  ...
)
```

Arguments

`dat`	a data.frame of class `tabyl` that has had `adorn_percentages` and/or `adorn_pct_formatting` called on it. If given a list of data.frames, this function will apply itself to each data.frame in the list (designed for 3-way `tabyl` lists).
`position`	should the N go in the front, or in the rear, of the percentage?
`ns`	the Ns to append. The default is the "core" attribute of the input tabyl `dat`, where the original Ns of a two-way `tabyl` are stored. However, if your Ns are stored somewhere else, or you need to customize them beyond what can be done with 'format_func', you can supply them here.
`format_func`	a formatting function to run on the Ns. Consider defining with `base::format()`.
`...`	columns to adorn. This takes a tidyselect specification. By default, all columns are adorned except for the first column and columns not of class `numeric`, but this allows you to manually specify which columns should be adorned, for use on a data.frame that does not result from a call to `tabyl`.

Value

a data.frame with Ns appended

Examples

```
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_percentages("col") %>%
```

```
  adorn_pct_formatting() %>%
  adorn_ns(position = "front")

# Format the Ns with a custom format_func:
set.seed(1)
bigger_dat <- data.frame(sex = rep(c("m", "f"), 3000),
                         age = round(runif(3000, 1, 102), 0))
bigger_dat$age_group = cut(bigger_dat$age, quantile(bigger_dat$age, c(0, 1/3, 2/3, 1)))

bigger_dat %>%
  tabyl(age_group, sex, show_missing_levels = FALSE) %>%
  adorn_totals(c("row", "col")) %>%
  adorn_percentages("col") %>%
  adorn_pct_formatting(digits = 1) %>%
  adorn_ns(format_func = function(x) format(x, big.mark = ".", decimal.mark = ","))
# Control the columns to be adorned with the ... variable selection argument
# If using only the ... argument, you can use empty commas as shorthand
# to supply the default values to the preceding arguments:

cases <- data.frame(
  region = c("East", "West"),
  year = 2015,
  recovered = c(125, 87),
  died = c(13, 12)
)

cases %>%
 adorn_percentages("col",,recovered:died) %>%
 adorn_pct_formatting(,,,,,recovered:died) %>%
 adorn_ns(,,,recovered:died)
```

`adorn_pct_formatting` *Format a data.frame of decimals as percentages.*

Description

Numeric columns get multiplied by 100 and formatted as percentages according to user specifications. This function defaults to excluding the first column of the input data.frame, assuming that it contains a descriptive variable, but this can be overridden by specifying the columns to adorn in the `...` argument. Non-numeric columns are always excluded.

The decimal separator character is the result of `getOption("OutDec")`, which is based on the user's locale. If the default behavior is undesirable, change this value ahead of calling the function, either by changing locale or with `options(OutDec = ",")`. This aligns the decimal separator character with that used in `base::print()`.

Usage

```
adorn_pct_formatting(
```

```
  dat,
  digits = 1,
  rounding = "half to even",
  affix_sign = TRUE,
  ...
)
```

Arguments

dat	a data.frame with decimal values, typically the result of a call to adorn_percentages on a tabyl. If given a list of data.frames, this function will apply itself to each data.frame in the list (designed for 3-way tabyl lists).
digits	how many digits should be displayed after the decimal point?
rounding	method to use for rounding - either "half to even", the base R default method, or "half up", where 14.5 rounds up to 15.
affix_sign	should the % sign be affixed to the end?
...	columns to adorn. This takes a tidyselect specification. By default, all numeric columns (besides the initial column, if numeric) are adorned, but this allows you to manually specify which columns should be adorned, for use on a data.frame that does not result from a call to tabyl.

Value

a data.frame with formatted percentages

Examples

```
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_percentages("col") %>%
  adorn_pct_formatting()

# Control the columns to be adorned with the ... variable selection argument
# If using only the ... argument, you can use empty commas as shorthand
# to supply the default values to the preceding arguments:

cases <- data.frame(
  region = c("East", "West"),
  year = 2015,
  recovered = c(125, 87),
  died = c(13, 12)
)

cases %>%
 adorn_percentages("col",,recovered:died) %>%
 adorn_pct_formatting(,,,recovered:died)
```

adorn_percentages *Convert a data.frame of counts to percentages.*

Description

This function defaults to excluding the first column of the input data.frame, assuming that it contains a descriptive variable, but this can be overridden by specifying the columns to adorn in the ... argument.

Usage

```
adorn_percentages(dat, denominator = "row", na.rm = TRUE, ...)
```

Arguments

dat	a tabyl or other data.frame with a tabyl-like layout. If given a list of data.frames, this function will apply itself to each data.frame in the list (designed for 3-way tabyl lists).
denominator	the direction to use for calculating percentages. One of "row", "col", or "all".
na.rm	should missing values (including NaN) be omitted from the calculations?
...	columns to adorn. This takes a tidyselect specification. By default, all numeric columns (besides the initial column, if numeric) are adorned, but this allows you to manually specify which columns should be adorned, for use on a data.frame that does not result from a call to tabyl.

Value

Returns a data.frame of percentages, expressed as numeric values between 0 and 1.

Examples

```
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_percentages("col")

# calculates correctly even with totals column and/or row:
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_totals("row") %>%
  adorn_percentages()

# Control the columns to be adorned with the ... variable selection argument
# If using only the ... argument, you can use empty commas as shorthand
# to supply the default values to the preceding arguments:

cases <- data.frame(
  region = c("East", "West"),
  year = 2015,
```

```
  recovered = c(125, 87),
  died = c(13, 12)
)

cases %>%
 adorn_percentages(,,recovered:died)
```

---

adorn_rounding *Round the numeric columns in a data.frame.*

Description

Can run on any data.frame with at least one numeric column. This function defaults to excluding the first column of the input data.frame, assuming that it contains a descriptive variable, but this can be overridden by specifying the columns to round in the `...` argument.

If you're formatting percentages, e.g., the result of `adorn_percentages()`, use `adorn_pct_formatting()` instead. This is a more flexible variant for ad-hoc usage. Compared to `adorn_pct_formatting()`, it does not multiply by 100 or pad the numbers with spaces for alignment in the results data.frame. This function retains the class of numeric input columns.

Usage

```
adorn_rounding(dat, digits = 1, rounding = "half to even", ...)
```

Arguments

`dat`	a `tabyl` or other data.frame with similar layout. If given a list of data.frames, this function will apply itself to each data.frame in the list (designed for 3-way `tabyl` lists).
`digits`	how many digits should be displayed after the decimal point?
`rounding`	method to use for rounding - either "half to even", the base R default method, or "half up", where 14.5 rounds up to 15.
`...`	columns to adorn. This takes a tidyselect specification. By default, all numeric columns (besides the initial column, if numeric) are adorned, but this allows you to manually specify which columns should be adorned, for use on a data.frame that does not result from a call to `tabyl`.

Value

Returns the data.frame with rounded numeric columns.

Examples

```
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_percentages() %>%
  adorn_rounding(digits = 2, rounding = "half up")

# tolerates non-numeric columns:
library(dplyr)
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_percentages("all") %>%
  mutate(dummy = "a") %>%
  adorn_rounding()

# Control the columns to be adorned with the ... variable selection argument
# If using only the ... argument, you can use empty commas as shorthand
# to supply the default values to the preceding arguments:
cases <- data.frame(
  region = c("East", "West"),
  year = 2015,
  recovered = c(125, 87),
  died = c(13, 12)
)

cases %>%
  adorn_percentages(,,ends_with("ed")) %>%
  adorn_rounding(,,one_of(c("recovered", "died")))
```

---

adorn_title *Add column name to the top of a two-way tabyl.*

Description

This function adds the column variable name to the top of a `tabyl` for a complete display of information. This makes the tabyl prettier, but renders the data.frame less useful for further manipulation.

Usage

```
adorn_title(dat, placement = "top", row_name, col_name)
```

Arguments

`dat`	a data.frame of class `tabyl` or other data.frame with a tabyl-like layout. If given a list of data.frames, this function will apply itself to each data.frame in the list (designed for 3-way `tabyl` lists).
`placement`	whether the column name should be added to the top of the tabyl in an otherwise-empty row `"top"` or appended to the already-present row name variable (`"combined"`). The formatting in the `"top"` option has the look of base R's `table()`; it also

wipes out the other column names, making it hard to further use the data.frame besides formatting it for reporting. The "combined" option is more conservative in this regard.

row_name (optional) default behavior is to pull the row name from the attributes of the input tabyl object. If you wish to override that text, or if your input is not a tabyl, supply a string here.

col_name (optional) default behavior is to pull the column_name from the attributes of the input tabyl object. If you wish to override that text, or if your input is not a tabyl, supply a string here.

Value

the input tabyl, augmented with the column title. Non-tabyl inputs that are of class tbl_df are downgraded to basic data.frames so that the title row prints correctly.

Examples

```
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_title(placement = "top")

# Adding a title to a non-tabyl
library(tidyr); library(dplyr)
mtcars %>%
  group_by(gear, am) %>%
  summarise(avg_mpg = mean(mpg), .groups = "drop") %>%
  spread(gear, avg_mpg) %>%
  adorn_rounding() %>%
  adorn_title("top", row_name = "Gears", col_name = "Cylinders")
```

---

adorn_totals *Append a totals row and/or column to a data.frame.*

Description

This function defaults to excluding the first column of the input data.frame, assuming that it contains a descriptive variable, but this can be overridden by specifying the columns to be totaled in the ... argument. Non-numeric columns are converted to character class and have a user-specified fill character inserted in the totals row.

Usage

```
adorn_totals(dat, where = "row", fill = "-", na.rm = TRUE, name = "Total", ...)
```

Arguments

dat	an input data.frame with at least one numeric column. If given a list of data.frames, this function will apply itself to each data.frame in the list (designed for 3-way tabyl lists).
where	one of "row", "col", or c("row", "col")
fill	if there are non-numeric columns, what should fill the bottom row of those columns? If a string, relevant columns will be coerced to character. If 'NA' then column types are preserved.
na.rm	should missing values (including NaN) be omitted from the calculations?
name	name of the totals row and/or column. If both are created, and name is a single string, that name is applied to both. If both are created and name is a vector of length 2, the first element of the vector will be used as the row name (in column 1), and the second element will be used as the totals column name. Defaults to "Total".
...	columns to total. This takes a tidyselect specification. By default, all numeric columns (besides the initial column, if numeric) are included in the totals, but this allows you to manually specify which columns should be included, for use on a data.frame that does not result from a call to tabyl.

Value

Returns a data.frame augmented with a totals row, column, or both. The data.frame is now also of class tabyl and stores information about the attached totals and underlying data in the tabyl attributes.

Examples

```
mtcars %>%
  tabyl(am, cyl) %>%
  adorn_totals()
```

---

as_tabyl *Add* tabyl *attributes to a data.frame.*

Description

A tabyl is a data.frame containing counts of a variable or co-occurrences of two variables (a.k.a., a contingency table or crosstab). This specialized kind of data.frame has attributes that enable adorn_ functions to be called for precise formatting and presentation of results. E.g., display results as a mix of percentages, Ns, add totals rows or columns, rounding options, in the style of Microsoft Excel PivotTable.

A tabyl can be the result of a call to janitor::tabyl(), in which case these attributes are added automatically. This function adds tabyl class attributes to a data.frame that isn't the result of a call to tabyl but meets the requirements of a two-way tabyl: 1) First column contains values of variable

1 2) Column names 2:n are the values of variable 2 3) Numeric values in columns 2:n are counts of the co-occurrences of the two variables.*

* = this is the ideal form of a tabyl, but janitor's adorn_ functions tolerate and ignore non-numeric columns in positions 2:n.

For instance, the result of dplyr::count() followed by tidyr::spread() can be treated as a tabyl.

The result of calling tabyl() on a single variable is a special class of one-way tabyl; this function only pertains to the two-way tabyl.

Usage

```
as_tabyl(dat, axes = 2, row_var_name = NULL, col_var_name = NULL)
```

Arguments

dat	a data.frame with variable values in the first column and numeric values in all other columns.
axes	is this a two_way tabyl or a one_way tabyl? If this function is being called by a user, this should probably be "2". One-way tabyls are created by tabyl but are a special case.
row_var_name	(optional) the name of the variable in the row dimension; used by adorn_title().
col_var_name	(optional) the name of the variable in the column dimension; used by adorn_title().

Value

Returns the same data.frame, but with the additional class of "tabyl" and the attribute "core".

Examples

```
as_tabyl(mtcars)
```

---

chisq.test *Apply stats::chisq.test to a two-way tabyl*

Description

This generic function overrides stats::chisq.test. If the passed table is a two-way tabyl, it runs it through janitor::chisq.test.tabyl, otherwise it just calls stats::chisq.test.

Usage

```
chisq.test(x, ...)

## Default S3 method:
chisq.test(x, y = NULL, ...)

## S3 method for class 'tabyl'
chisq.test(x, tabyl_results = TRUE, ...)
```

Arguments

x	a two-way tabyl, a numeric vector or a factor
...	other parameters passed to stats::chisq.test
y	if x is a vector, must be another vector or factor of the same length
tabyl_results	if TRUE and x is a tabyl object, also return ‘observed‘, ‘expected‘, ‘residuals‘ and ‘stdres‘ as tabyl

Value

The result is the same as the one of stats::chisq.test. If ‘tabyl_results‘ is TRUE, the returned tables ‘observed‘, ‘expected‘, ‘residuals‘ and ‘stdres‘ are converted to tabyls.

Examples

```
tab <- tabyl(mtcars, gear, cyl)
chisq.test(tab)
chisq.test(tab)$residuals
```

---

clean_names *Cleans names of an object (usually a data.frame).*

Description

Resulting names are unique and consist only of the _ character, numbers, and letters. Capitalization preferences can be specified using the `case` parameter.

Accented characters are transliterated to ASCII. For example, an "o" with a German umlaut over it becomes "o", and the Spanish character "enye" becomes "n".

This function takes and returns a data.frame, for ease of piping with `%>%`. For the underlying function that works on a character vector of names, see `make_clean_names`. `clean_names` relies on the versatile function `to_any_case`, which accepts many arguments. See that function's documentation for ideas on getting the most out of `clean_names`. A few examples are included below.

A common issue is that the micro/mu symbol is replaced by "m" instead of "u". The replacement with "m" is more correct when doing Greek-to-ASCII transliteration but less correct when doing scientific data-to-ASCII transliteration. A warning will be generated if the "m" replacement occurs. To replace with "u", please add the argument `replace=janitor:::mu_to_u` which is a character vector mapping all known mu or micro Unicode code points (characters) to "u".

Usage

```
clean_names(dat, ...)

## Default S3 method:
clean_names(dat, ...)

## S3 method for class 'sf'
clean_names(dat, ...)

## S3 method for class 'tbl_graph'
clean_names(dat, ...)

## S3 method for class 'tbl_lazy'
clean_names(dat, ...)
```

Arguments

dat	the input data.frame.
...	Arguments passed on to make_clean_names

case The desired target case (default is "snake") will be passed to snakecase::to_any_case() with the exception of "old_janitor", which exists only to support legacy code (it preserves the behavior of clean_names() prior to addition of the "case" argument (janitor versions <= 0.3.1). "old_janitor" is not intended for new code. See to_any_case for a wide variety of supported cases, including "sentence" and "title" case.

replace A named character vector where the name is replaced by the value.

ascii Convert the names to ASCII (TRUE, default) or not (FALSE).

use_make_names Should make.names() be applied to ensure that the output is usable as a name without quoting? (Avoiding make.names() ensures that the output is locale-independent but quoting may be required.)

allow_dupes Allow duplicates in the returned names (TRUE) or not (FALSE, the default).

sep_in (short for separator input) if character, is interpreted as a regular expression (wrapped internally into stringr::regex()). The default value is a regular expression that matches any sequence of non-alphanumeric values. All matches will be replaced by underscores (additionally to "_" and " ", for which this is always true, even if NULL is supplied). These underscores are used internally to split the strings into substrings and specify the word boundaries.

parsing_option An integer that will determine the parsing_option.

- 1: "RRRStudio" -> "RRR_Studio"
- 2: "RRRStudio" -> "RRRS_tudio"
- 3: "RRRStudio" -> "RRRSStudio". This will become for example "Rrrstudio" when we convert to lower camel case.
- -1, -2, -3: These parsing_options's will suppress the conversion after non-alphanumeric values.

- 0: no parsing

`transliterations` A character vector (if not `NULL`). The entries of this argument need to be elements of `stringi::stri_trans_list()` (like "Latin-ASCII", which is often useful) or names of lookup tables (currently only "german" is supported). In the order of the entries the letters of the input string will be transliterated via `stringi::stri_trans_general()` or replaced via the matches of the lookup table. When named character elements are supplied as part of 'transliterations', anything that matches the names is replaced by the corresponding value. You should use this feature with care in case of `case = "parsed"`, `case = "internal_parsing"` and `case = "none"`, since for upper case letters, which have transliterations/replacements of length 2, the second letter will be transliterated to lowercase, for example Oe, Ae, Ss, which might not always be what is intended. In this case you can make usage of the option to supply named elements and specify the transliterations yourself.

`numerals` A character specifying the alignment of numerals (`"middle"`, `left`, `right`, `asis` or `tight`). I.e. `numerals = "left"` ensures that no output separator is in front of a digit.

Details

`clean_names()` is intended to be used on `data.frames` and `data.frame`-like objects. For this reason there are methods to support using `clean_names()` on `sf` and `tbl_graph` (from `tidygraph`) objects as well as on database connections through `dbplyr`. For cleaning other named objects like named lists and vectors, use `make_clean_names()`.

Value

Returns the data.frame with clean names.

See Also

Other Set names: `find_header()`, `mu_to_u`, `row_to_names()`

Examples

```
# --- Simple Usage ---
x <- data.frame(caseID = 1, DOB = 2, Other = 3)
clean_names(x)

# or pipe in the input data.frame:
x %>%
  clean_names()

# if you prefer camelCase variable names:
x %>%
  clean_names(., "lower_camel")

# (not run) run clean_names after reading in a spreadsheet:
# library(readxl)
```

```
# read_excel("messy_excel_file.xlsx") %>%
#   clean_names()

# --- Taking advantage of the underlying snakecase::to_any_case arguments ---

# Restore column names to Title Case, e.g., for plotting
mtcars %>%
  clean_names(case = "title")

# Tell clean_names to leave certain abbreviations untouched:
x %>%
  clean_names(case = "upper_camel", abbreviations = c("ID", "DOB"))
```

`compare_df_cols` *Generate a comparison of data.frames (or similar objects) that indicates if they will successfully bind together by rows.*

Description

Generate a comparison of data.frames (or similar objects) that indicates if they will successfully bind together by rows.

Usage

```
compare_df_cols(
  ...,
  return = c("all", "match", "mismatch"),
  bind_method = c("bind_rows", "rbind"),
  strict_description = FALSE
)
```

Arguments

`...`	A combination of data.frames, tibbles, and lists of data.frames/tibbles. The values may optionally be named arguments; if named, the output column will be the name; if not named, the output column will be the data.frame name (see examples section).
`return`	Should a summary of "all" columns be returned, only return "match"ing columns, or only "mismatch"ing columns?
`bind_method`	What method of binding should be used to determine matches? With "bind_rows", columns missing from a data.frame would be considered a match (as in `dplyr::bind_rows()`; with "rbind", columns missing from a data.frame would be considered a mismatch (as in `base::rbind()`.
`strict_description`	Passed to `describe_class`. Also, see the Details section.

Details

Due to the returned "column_name" column, no input data.frame may be named "column_name".

The `strict_description` argument is most typically used to understand if factor levels match or are bindable. Factors are typically bindable, but the behavior of what happens when they bind differs based on the binding method ("bind_rows" or "rbind"). Even when `strict_description` is `FALSE`, data.frames may still bind because some classes (like factors and characters) can bind even if they appear to differ.

Value

A data.frame with a column named "column_name" with a value named after the input data.frames' column names, and then one column per data.frame (named after the input data.frame). If more than one input has the same column name, the column naming will have suffixes defined by sequential use of `base::merge()` and may differ from expected naming. The rows within the data.frame-named columns are descriptions of the classes of the data within the columns (generated by `describe_class`).

See Also

Other Data frame type comparison: `compare_df_cols_same()`, `describe_class()`

Examples

```
compare_df_cols(data.frame(A=1), data.frame(B=2))
# user-defined names
compare_df_cols(dfA=data.frame(A=1), dfB=data.frame(B=2))
# a combination of list and data.frame input
compare_df_cols(listA=list(dfA=data.frame(A=1), dfB=data.frame(B=2)), data.frame(A=3))
```

`compare_df_cols_same` *Do the the data.frames have the same columns & types?*

Description

Check whether a set of data.frames are row-bindable. Calls `compare_df_cols()`and returns TRUE if there are no mis-matching rows. ‘

Usage

```
compare_df_cols_same(
  ...,
  bind_method = c("bind_rows", "rbind"),
  verbose = TRUE
)
```

Arguments

...	A combination of data.frames, tibbles, and lists of data.frames/tibbles. The values may optionally be named arguments; if named, the output column will be the name; if not named, the output column will be the data.frame name (see examples section).
bind_method	What method of binding should be used to determine matches? With "bind_rows", columns missing from a data.frame would be considered a match (as in `dplyr::bind_rows()`; with "rbind", columns missing from a data.frame would be considered a mismatch (as in `base::rbind()`.
verbose	Print the mismatching columns if binding will fail.

Value

`TRUE` if row binding will succeed or `FALSE` if it will fail.

See Also

Other Data frame type comparison: `compare_df_cols()`, `describe_class()`

Examples

```
compare_df_cols_same(data.frame(A=1), data.frame(A=2))
compare_df_cols_same(data.frame(A=1), data.frame(B=2))
compare_df_cols_same(data.frame(A=1), data.frame(B=2), verbose=FALSE)
compare_df_cols_same(data.frame(A=1), data.frame(B=2), bind_method="rbind")
```

---

convert_to_date *Convert many date and datetime formats as may be received from Microsoft Excel*

Description

Convert many date and datetime formats as may be received from Microsoft Excel

Usage

```
convert_to_date(
  x,
  ...,
  character_fun = lubridate::ymd,
  string_conversion_failure = c("error", "warning")
)

convert_to_datetime(
  x,
  ...,
  tz = "UTC",
```

```
  character_fun = lubridate::ymd_hms,
  string_conversion_failure = c("error", "warning")
)
```

Arguments

x	The object to convert
...	Passed to further methods. Eventually may be passed to 'excel_numeric_to_date()', 'base::as.POSIXct()', or 'base::as.Date()'.
character_fun	A function to convert non-numeric-looking, non-NA values in 'x' to POSIXct objects.
string_conversion_failure	If a character value fails to parse into the desired class and instead returns 'NA', should the function return the result with a warning or throw an error?
tz	The timezone for POSIXct output, unless an object is POSIXt already. Ignored for Date output.

Details

Character conversion checks if it matches something that looks like a Microsoft Excel numeric date, converts those to numeric, and then runs convert_to_datetime_helper() on those numbers. Then, character to Date or POSIXct conversion occurs via 'character_fun(x, ...)' or 'character_fun(x, tz=tz, ...)', respectively.

Value

POSIXct objects for 'convert_to_datetime()' or Date objects for 'convert_to_date()'.

Functions

- `convert_to_datetime()`: Convert to a date-time (POSIXct)

See Also

Other Date-time cleaning: `excel_numeric_to_date()`, `sas_numeric_to_date()`

Examples

```
convert_to_date("2009-07-06")
convert_to_date(40000)
convert_to_date("40000.1")
# Mixed date source data can be provided.
convert_to_date(c("2020-02-29", "40000.1"))
convert_to_datetime(
  c("2009-07-06", "40000.1", "40000", NA),
  character_fun=lubridate::ymd_h, truncated=1, tz="UTC"
)
```

convert_to_NA *Convert string values to true* NA *values.*

Description

Converts instances of user-specified strings into NA. Can operate on either a single vector or an entire data.frame.

Usage

```
convert_to_NA(dat, strings)
```

Arguments

dat	vector or data.frame to operate on.
strings	character vector of strings to convert.

Value

Returns a cleaned object. Can be a vector, data.frame, or `tibble::tbl_df` depending on the provided input.

Warning

Deprecated, do not use in new code. Use `dplyr::na_if()` instead.

See Also

janitor_deprecated

describe_class *Describe the class(es) of an object*

Description

Describe the class(es) of an object

Usage

```
describe_class(x, strict_description = TRUE)

## S3 method for class 'factor'
describe_class(x, strict_description = TRUE)

## Default S3 method:
describe_class(x, strict_description = TRUE)
```

Arguments

x	The object to describe
strict_description	Should differing factor levels be treated as differences for the purposes of identifying mismatches? strict_description = `TRUE` is stricter and factors with different levels will be treated as different classes. FALSE is more lenient: for class comparison purposes, the variable is just a "factor".

Details

For package developers, an S3 generic method can be written for describe_class() for custom classes that may need more definition than the default method. This function is called by compare_df_cols.

Value

A character scalar describing the class(es) of an object where if the scalar will match, columns in a data.frame (or similar object) should bind together without issue.

Methods (by class)

- describe_class(factor): Describe factors with their levels and if they are ordered.
- describe_class(default): List all classes of an object.

See Also

Other Data frame type comparison: compare_df_cols_same(), compare_df_cols()

Examples

```
describe_class(1)
describe_class(factor("A"))
describe_class(ordered(c("A", "B")))
describe_class(ordered(c("A", "B")), strict_description=FALSE)
```

---

excel_numeric_to_date *Convert dates encoded as serial numbers to Date class.*

Description

Converts numbers like 42370 into date values like 2016-01-01.

Defaults to the modern Excel date encoding system. However, Excel for Mac 2008 and earlier Mac versions of Excel used a different date system. To determine what platform to specify: if the date 2016-01-01 is represented by the number 42370 in your spreadsheet, it's the modern system. If it's 40908, it's the old Mac system. More on date encoding systems at http://support.office.com/en-us/article/Date-calculations-in-Excel-e7fe7167-48a9-4b96-bb53-5612a800b487.

A list of all timezones is available from `base::OlsonNames()`, and the current timezone is available from `base::Sys.timezone()`.

If your input data has a mix of Excel numeric dates and actual dates, see the more powerful functions `convert_to_date()` and `convert_to_datetime()`.

Usage

```
excel_numeric_to_date(
  date_num,
  date_system = "modern",
  include_time = FALSE,
  round_seconds = TRUE,
  tz = Sys.timezone()
)
```

Arguments

`date_num`	numeric vector of serial numbers to convert.
`date_system`	the date system, either `"modern"` or `"mac pre-2011"`.
`include_time`	Include the time (hours, minutes, seconds) in the output? (See details)
`round_seconds`	Round the seconds to an integer (only has an effect when `include_time` is `TRUE`)?
`tz`	Time zone, used when `include_time = TRUE` (see details for more information on timezones).

Details

When using `include_time=TRUE`, days with leap seconds will not be accurately handled as they do not appear to be accurately handled by Windows (as described in https://support.microsoft.com/en-us/help/2722715/support-for-the-leap-second).

Value

Returns a vector of class Date if `include_time` is `FALSE`. Returns a vector of class POSIXlt if `include_time` is `TRUE`.

See Also

Other Date-time cleaning: `convert_to_date()`, `sas_numeric_to_date()`

Examples

```
excel_numeric_to_date(40000)
excel_numeric_to_date(40000.5) # No time is included
excel_numeric_to_date(40000.5, include_time = TRUE) # Time is included
excel_numeric_to_date(40000.521, include_time = TRUE) # Time is included
excel_numeric_to_date(40000.521, include_time = TRUE,
  round_seconds = FALSE) # Time with fractional seconds is included
```

`find_header` *Find the header row in a data.frame*

Description

Find the header row in a data.frame

Usage

```
find_header(dat, ...)
```

Arguments

`dat`	The input data.frame
`...`	See details

Details

If ... is missing, then the first row with no missing values is used.

When searching for a specified value or value within a column, the first row with a match will be returned, regardless of the completeness of the rest of that row. If ... has a single character argument, then the first column is searched for that value. If ... has a named numeric argument, then the column whose position number matches the value of that argument is searched for the name (see the last example below). If more than one row is found matching a value that is searched for, the number of the first matching row will be returned (with a warning).

Value

The row number for the header row

See Also

Other Set names: `clean_names()`, `mu_to_u`, `row_to_names()`

Examples

```
# the first row
find_header(data.frame(A="B"))
# the second row
find_header(data.frame(A=c(NA, "B")))
# the second row since the first has an empty value
find_header(data.frame(A=c(NA, "B"), B=c("C", "D")))
# The third row because the second column was searched for the text "E"
find_header(data.frame(A=c(NA, "B", "C", "D"), B=c("C", "D", "E", "F")), "E"=2)
```

fisher.test *Apply stats::fisher.test to a two-way tabyl*

Description

This generic function overrides stats::fisher.test. If the passed table is a two-way tabyl, it runs it through janitor::fisher.test.tabyl, otherwise it just calls stats::fisher.test.

Usage

```
fisher.test(x, ...)

## Default S3 method:
fisher.test(x, y = NULL, ...)

## S3 method for class 'tabyl'
fisher.test(x, ...)
```

Arguments

x	a two-way tabyl, a numeric vector or a factor
...	other parameters passed to stats::fisher.test
y	if x is a vector, must be another vector or factor of the same length

Value

The result is the same as the one of stats::fisher.test.

Examples

```
tab <- tabyl(mtcars, gear, cyl)
fisher.test(tab)
```

get_dupes *Get rows of a* data.frame *with identical values for the specified variables.*

Description

For hunting duplicate records during data cleaning. Specify the data.frame and the variable combination to search for duplicates and get back the duplicated rows.

Usage

```
get_dupes(dat, ...)
```

Arguments

dat	The input data.frame.
...	Unquoted variable names to search for duplicates. This takes a tidyselect specification.

Value

Returns a data.frame with the full records where the specified variables have duplicated values, as well as a variable dupe_count showing the number of rows sharing that combination of duplicated values. If the input data.frame was of class tbl_df, the output is as well.

Examples

```
get_dupes(mtcars, mpg, hp)

# or called with the magrittr pipe %>% :
mtcars %>% get_dupes(wt)

# You can use tidyselect helpers to specify variables:
mtcars %>% get_dupes(-c(wt, qsec))
mtcars %>% get_dupes(starts_with("cy"))
```

---

get_one_to_one *Find the list of columns that have a 1:1 mapping to each other*

Description

Find the list of columns that have a 1:1 mapping to each other

Usage

```
get_one_to_one(dat)
```

Arguments

dat	A data.frame or similar object

Value

A list with one element for each group of columns that map identically to each other.

Examples

```
foo <- data.frame(
  Lab_Test_Long=c("Cholesterol, LDL", "Cholesterol, LDL", "Glucose"),
  Lab_Test_Short=c("CLDL", "CLDL", "GLUC"),
  LOINC=c(12345, 12345, 54321),
  Person=c("Sam", "Bill", "Sam"),
  stringsAsFactors=FALSE
)
get_one_to_one(foo)
```

`janitor_deprecated` *Deprecated Functions in Package janitor*

Description

These functions have already become defunct or may be defunct as soon as the next release.

Details

- `adorn_crosstab`
- `crosstab`
- `use_first_valid_of`
- `convert_to_NA`
- `add_totals_col`
- `add_totals_row`
- `remove_empty_rows`
- `remove_empty_cols`

`make_clean_names` *Cleans a vector of text, typically containing the names of an object.*

Description

Resulting strings are unique and consist only of the _ character, numbers, and letters. By default, the resulting strings will only consist of ASCII characters, but non-ASCII (e.g. Unicode) may be allowed by setting `ascii=FALSE`. Capitalization preferences can be specified using the `case` parameter.

For use on the names of a data.frame, e.g., in a `` `%>%` `` pipeline, call the convenience function `clean_names`.

When `ascii=TRUE` (the default), accented characters are transliterated to ASCII. For example, an "o" with a German umlaut over it becomes "o", and the Spanish character "enye" becomes "n".

The order of operations is: make replacements, (optional) ASCII conversion, remove initial spaces and punctuation, apply `base::make.names()`, apply `snakecase::to_any_case`, and add numeric suffixes to resolve any duplicated names.

This function relies on `snakecase::to_any_case` and can take advantage of its versatility. For instance, an abbreviation like "ID" can have its capitalization preserved by passing the argument `abbreviations = "ID"`. See the documentation for `snakecase::to_any_case` for more about how to use its features.

On some systems, not all transliterators to ASCII are available. If this is the case on your system, all available transliterators will be used, and a warning will be issued once per session indicating that results may be different when run on a different system. That warning can be disabled with `options(janitor_warn_transliterators=FALSE)`.

If the objective of your call to `make_clean_names()` is only to translate to ASCII, try the following instead: `stringi::stri_trans_general(x, id="Any-Latin;Greek-Latin;Latin-ASCII")`.

Usage

```
make_clean_names(
  string,
  case = "snake",
  replace = c(`'` = "", `"` = "", `%` = "_percent_", `#` = "_number_"),
  ascii = TRUE,
  use_make_names = TRUE,
  allow_dupes = FALSE,
  sep_in = "\\.",
  transliterations = "Latin-ASCII",
  parsing_option = 1,
  numerals = "asis",
  ...
)
```

Arguments

`string`	A character vector of names to clean.
`case`	The desired target case (default is `"snake"`) will be passed to `snakecase::to_any_case()` with the exception of "old_janitor", which exists only to support legacy code (it preserves the behavior of `clean_names()` prior to addition of the "case" argument (janitor versions <= 0.3.1). "old_janitor" is not intended for new code. See `to_any_case` for a wide variety of supported cases, including "sentence" and "title" case.
`replace`	A named character vector where the name is replaced by the value.
`ascii`	Convert the names to ASCII (`TRUE`, default) or not (`FALSE`).
`use_make_names`	Should `make.names()` be applied to ensure that the output is usable as a name without quoting? (Avoiding `make.names()` ensures that the output is locale-independent but quoting may be required.)
`allow_dupes`	Allow duplicates in the returned names (`TRUE`) or not (`FALSE`, the default).

`sep_in` (short for separator input) if character, is interpreted as a regular expression (wrapped internally into `stringr::regex()`). The default value is a regular expression that matches any sequence of non-alphanumeric values. All matches will be replaced by underscores (additionally to `"_"` and `" "`, for which this is always true, even if `NULL` is supplied). These underscores are used internally to split the strings into substrings and specify the word boundaries.

`transliterations`
A character vector (if not `NULL`). The entries of this argument need to be elements of `stringi::stri_trans_list()` (like "Latin-ASCII", which is often useful) or names of lookup tables (currently only "german" is supported). In the order of the entries the letters of the input string will be transliterated via `stringi::stri_trans_general()` or replaced via the matches of the lookup table. When named character elements are supplied as part of 'transliterations', anything that matches the names is replaced by the corresponding value. You should use this feature with care in case of `case = "parsed"`, `case = "internal_parsing"` and `case = "none"`, since for upper case letters, which have transliterations/replacements of length 2, the second letter will be transliterated to lowercase, for example Oe, Ae, Ss, which might not always be what is intended. In this case you can make usage of the option to supply named elements and specify the transliterations yourself.

`parsing_option` An integer that will determine the parsing_option.

- 1: `"RRRStudio" -> "RRR_Studio"`
- 2: `"RRRStudio" -> "RRRS_tudio"`
- 3: `"RRRStudio" -> "RRRSStudio"`. This will become for example `"Rrrstudio"` when we convert to lower camel case.
- -1, -2, -3: These `parsing_options`'s will suppress the conversion after non-alphanumeric values.
- 0: no parsing

`numerals` A character specifying the alignment of numerals (`"middle"`, `left`, `right`, `asis` or `tight`). I.e. `numerals = "left"` ensures that no output separator is in front of a digit.

`...` Arguments passed on to `snakecase::to_any_case`

`abbreviations` character. (Case insensitive) matched abbreviations are surrounded by underscores. In this way, they can get recognized by the parser. This is useful when e.g. `parsing_option` 1 is needed for the use case, but some abbreviations but some substrings would require `parsing_option` 2. Furthermore, this argument also specifies the formatting of abbreviations in the output for the cases title, mixed, lower and upper camel. E.g. for upper camel the first letter is always in upper case, but when the abbreviation is supplied in upper case, this will also be visible in the output.
Use this feature with care: One letter abbreviations and abbreviations next to each other are hard to read and also not easy to parse for further processing.

`sep_out` (short for separator output) String that will be used as separator. The defaults are `"_"` and `""`, regarding the specified `case`. When `length(sep_out) > 1`, the last element of `sep_out` gets recycled and separators are incorporated per string according to their order.

unique_sep A string. If not NULL, then duplicated names will get a suffix integer in the order of their appearance. The suffix is separated by the supplied string to this argument.

empty_fill A string. If it is supplied, then each entry that matches "" will be replaced by the supplied string to this argument.

prefix prefix (string).

postfix postfix (string).

Value

Returns the "cleaned" character vector.

See Also

to_any_case()

Examples

```
# cleaning the names of a vector:
x <- structure(1:3, names = c("name with space", "TwoWords", "total $ (2009)"))
x
names(x) <- make_clean_names(names(x))
x # now has cleaned names

# if you prefer camelCase variable names:
make_clean_names(names(x), "small_camel")

# similar to janitor::clean_names(poorly_named_df):
# not run:
# make_clean_names(names(poorly_named_df))
```

---

mu_to_u *Constant to help map from mu to u*

Description

This is a character vector with names of all known Unicode code points that look like the Greek mu or the micro symbol and values of "u". This is intended to simplify mapping from mu or micro in Unicode to the character "u" with clean_names() and make_clean_names().

Usage

```
mu_to_u
```

Format

An object of class character of length 10.

Details

See the help in `clean_names()` for how to use this.

See Also

Other Set names: `clean_names()`, `find_header()`, `row_to_names()`

`remove_constant` *Remove constant columns from a data.frame or matrix.*

Description

Remove constant columns from a data.frame or matrix.

Usage

```
remove_constant(dat, na.rm = FALSE, quiet = TRUE)
```

Arguments

`dat`	the input data.frame or matrix.
`na.rm`	should `NA` values be removed when considering whether a column is constant? The default value of `FALSE` will result in a column not being removed if it's a mix of a single value and `NA`.
`quiet`	Should messages be suppressed (`TRUE`) or printed (`FALSE`) indicating the summary of empty columns or rows removed?

See Also

`remove_empty()` for removing empty columns or rows.

Other remove functions: `remove_empty()`

Examples

```
remove_constant(data.frame(A=1, B=1:3))

# To find the columns that are constant
data.frame(A=1, B=1:3) %>%
  dplyr::select_at(setdiff(names(.), names(remove_constant(.)))) %>%
  unique()
```

remove_empty *Remove empty rows and/or columns from a data.frame or matrix.*

Description

Removes all rows and/or columns from a data.frame or matrix that are composed entirely of NA values.

Usage

```
remove_empty(dat, which = c("rows", "cols"), cutoff = 1, quiet = TRUE)
```

Arguments

dat	the input data.frame or matrix.
which	one of "rows", "cols", or c("rows", "cols"). Where no value of which is provided, defaults to removing both empty rows and empty columns, declaring the behavior with a printed message.
cutoff	What fraction (>0 to <=1) of rows or columns must be empty to be removed?
quiet	Should messages be suppressed (TRUE) or printed (FALSE) indicating the summary of empty columns or rows removed?

Value

Returns the object without its missing rows or columns.

See Also

remove_constant() for removing constant columns.

Other remove functions: remove_constant()

Examples

```
# not run:
# dat %>% remove_empty("rows")
# addressing a common untidy-data scenario where we have a mixture of
# blank values in some (character) columns and NAs in others:
library(dplyr)
dd <- tibble(x=c(LETTERS[1:5],NA,rep("",2)),
             y=c(1:5,rep(NA,3)))
# remove_empty() drops row 5 (all NA) but not 6 and 7 (blanks + NAs)
dd %>% remove_empty("rows")
# solution: preprocess to convert whitespace/empty strings to NA,
# _then_ remove empty (all-NA) rows
dd %>% mutate(across(is.character,~na_if(trimws(.),""))) %>%
   remove_empty("rows")
```

remove_empty_cols *Removes empty columns from a data.frame.*

Description

This function is deprecated, use `remove_empty("cols")` instead.

Usage

```
remove_empty_cols(dat)
```

Arguments

`dat`	the input data.frame.

Value

Returns the data.frame with no empty columns.

Examples

```
# not run:
# dat %>% remove_empty_cols
```

remove_empty_rows *Removes empty rows from a data.frame.*

Description

This function is deprecated, use `remove_empty("rows")` instead.

Usage

```
remove_empty_rows(dat)
```

Arguments

`dat`	the input data.frame.

Value

Returns the data.frame with no empty rows.

Examples

```
# not run:
# dat %>% remove_empty_rows
```

round_half_up — *Round a numeric vector; halves will be rounded up, ala Microsoft Excel.*

Description

In base R `round()`, halves are rounded to even, e.g., 12.5 and 11.5 are both rounded to 12. This function rounds 12.5 to 13 (assuming `digits = 0`). Negative halves are rounded away from zero, e.g., -0.5 is rounded to -1.

This may skew subsequent statistical analysis of the data, but may be desirable in certain contexts. This function is implemented exactly from https://stackoverflow.com/a/12688836; see that question and comments for discussion of this issue.

Usage

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

Arguments

`x`	a numeric vector to round.
`digits`	how many digits should be displayed after the decimal point?

Examples

```
round_half_up(12.5)
round_half_up(1.125, 2)
round_half_up(1.125, 1)
round_half_up(-0.5, 0) # negatives get rounded away from zero
```

round_to_fraction — *Round to the nearest fraction of a specified denominator.*

Description

Round a decimal to the precise decimal value of a specified fractional denominator. Common use cases include addressing floating point imprecision and enforcing that data values fall into a certain set.

E.g., if a decimal represents hours and values should be logged to the nearest minute, `round_to_fraction(x, 60)` would enforce that distribution and 0.57 would be rounded to 0.566667, the equivalent of 34/60. 0.56 would also be rounded to 34/60.

Set `denominator = 1` to round to whole numbers.

The `digits` argument allows for rounding of the subsequent result.

Usage

```
round_to_fraction(x, denominator, digits = Inf)
```

Arguments

x	A numeric vector
denominator	The denominator of the fraction for rounding (a scalar or vector positive integer).
digits	Integer indicating the number of decimal places to be used after rounding to the fraction. This is passed to base::round()). Negative values are allowed (see Details). (Inf indicates no subsequent rounding)

Details

If digits is Inf, x is rounded to the fraction and then kept at full precision. If digits is "auto", the number of digits is automatically selected as ceiling(log10(denominator)) + 1.

Value

the input x rounded to a decimal value that has an integer numerator relative to denominator (possibly subsequently rounded to a number of decimal digits).

Examples

```
round_to_fraction(1.6, denominator = 2)
round_to_fraction(pi, denominator = 7) # 22/7
round_to_fraction(c(8.1, 9.2), denominator = c(7, 8))
round_to_fraction(c(8.1, 9.2), denominator = c(7, 8), digits = 3)
round_to_fraction(c(8.1, 9.2, 10.3), denominator = c(7, 8, 1001), digits = "auto")
```

---

row_to_names *Elevate a row to be the column names of a data.frame.*

Description

Elevate a row to be the column names of a data.frame.

Usage

```
row_to_names(dat, row_number, ..., remove_row = TRUE, remove_rows_above = TRUE)
```

Arguments

dat	The input data.frame
row_number	The row of dat containing the variable names or the string "find_header" to use find_header(dat=dat, ...) to find the row_number.
...	Sent to find_header(), if row_number = "find_header". Otherwise, ignored.

remove_row Should the row row_number be removed from the resulting data.frame?

remove_rows_above
If row_number != 1, should the rows above row_number - that is, between 1:(row_number-1) - be removed from the resulting data.frame?

Value

A data.frame with new names (and some rows removed, if specified)

See Also

Other Set names: clean_names(), find_header(), mu_to_u

Examples

```
x <- data.frame(X_1 = c(NA, "Title", 1:3),
                X_2 = c(NA, "Title2", 4:6))
x %>%
  row_to_names(row_number = 2)

x %>%
  row_to_names(row_number = "find_header")
```

---

sas_numeric_to_date *Convert a SAS date, time or date/time to an R object*

Description

Convert a SAS date, time or date/time to an R object

Usage

```
sas_numeric_to_date(date_num, datetime_num, time_num, tz = "UTC")
```

Arguments

date_num	numeric vector of serial numbers to convert.
datetime_num	numeric vector of date/time numbers (seconds since midnight 1960-01-01) to convert
time_num	numeric vector of time numbers (seconds since midnight on the current day) to convert
tz	Time zone, used when include_time = TRUE (see details for more information on timezones).

Value

If a date and time or datetime are provided, a POSIXct object. If a date is provided, a Date object. If a time is provided, an hms::hms object

References

SAS Date, Time, and Datetime Values reference (retrieved on 2022-03-08): https://v8doc.sas.com/sashtml/lrcon/zenid-63.htm

See Also

Other Date-time cleaning: `convert_to_date()`, `excel_numeric_to_date()`

Examples

```
sas_numeric_to_date(date_num=15639) # 2002-10-26
sas_numeric_to_date(datetime_num=1217083532, tz="UTC") # 1998-07-26T14:45:32Z
sas_numeric_to_date(date_num=15639, time_num=3600, tz="UTC") # 2002-10-26T01:00:00Z
sas_numeric_to_date(time_num=3600) # 01:00:00
```

---

signif_half_up *Round a numeric vector to the specified number of significant digits; halves will be rounded up.*

Description

In base R `signif()`, halves are rounded to even, e.g., `signif(11.5, 2)` and `signif(12.5, 2)` are both rounded to 12. This function rounds 12.5 to 13 (assuming `digits = 2`). Negative halves are rounded away from zero, e.g., `signif(-2.5, 1)` is rounded to -3.

This may skew subsequent statistical analysis of the data, but may be desirable in certain contexts. This function is implemented from https://stackoverflow.com/a/1581007/; see that question and comments for discussion of this issue.

Usage

```
signif_half_up(x, digits = 6)
```

Arguments

`x`	a numeric vector to round.
`digits`	integer indicating the number of significant digits to be used.

Examples

```
signif_half_up(12.5, 2)
signif_half_up(1.125, 3)
signif_half_up(-2.5, 1) # negatives get rounded away from zero
```

single_value *Ensure that a vector has only a single value throughout.*

Description

Missing values are replaced with the single value, and if all values are missing, the first value in `missing` is used throughout.

Usage

```
single_value(x, missing = NA, warn_if_all_missing = FALSE, info = NULL)
```

Arguments

`x`	The vector which should have a single value
`missing`	The vector of values to consider missing in `x`
`warn_if_all_missing`	Generate a warning if all values are missing?
`info`	If more than one value is found, append this to the warning or error to assist with determining the location of the issue.

Value

`x` as the scalar single value found throughout (or an error if more than one value is found).

Examples

```
# A simple use case with vectors of input

single_value(c(NA, 1))
# Multiple, different values of missing can be given
single_value(c(NA, "a"), missing = c(NA, "a"))

# A typical use case with a grouped data.frame used for input and the output
# (`B` is guaranteed to have a single value and only one row, in this case)
data.frame(A = rep(1:3, each = 2),
           B = c(rep(4:6, each = 2))) %>%
  dplyr::group_by(A) %>%
  dplyr::summarize(
    B = single_value(B)
  )

try(
# info is useful to give when multiple values may be found to see what
# grouping variable or what calculation is causing the error
data.frame(A = rep(1:3, each = 2),
           B = c(rep(1:2, each = 2), 1:2)) %>%
  dplyr::group_by(A) %>%
```

```
  dplyr::mutate(
    C = single_value(B, info = paste("Calculating C for group A=", A))
  )
)
```

---

tabyl *Generate a frequency table (1-, 2-, or 3-way).*

Description

A fully-featured alternative to `table()`. Results are data.frames and can be formatted and enhanced with janitor's family of `adorn_` functions.

Specify a data.frame and the one, two, or three unquoted column names you want to tabulate. Three variables generates a list of 2-way tabyls, split by the third variable.

Alternatively, you can tabulate a single variable that isn't in a data.frame by calling `tabyl` on a vector, e.g., `tabyl(mtcars$gear)`.

Usage

```
tabyl(dat, ...)

## Default S3 method:
tabyl(dat, show_na = TRUE, show_missing_levels = TRUE, ...)

## S3 method for class 'data.frame'
tabyl(dat, var1, var2, var3, show_na = TRUE, show_missing_levels = TRUE, ...)
```

Arguments

`dat`	a data.frame containing the variables you wish to count. Or, a vector you want to tabulate.
`...`	the arguments to tabyl (here just for the sake of documentation compliance, as all arguments are listed with the vector- and data.frame-specific methods)
`show_na`	should counts of `NA` values be displayed? In a one-way tabyl, the presence of `NA` values triggers an additional column showing valid percentages(calculated excluding `NA` values).
`show_missing_levels`	should counts of missing levels of factors be displayed? These will be rows and/or columns of zeroes. Useful for keeping consistent output dimensions even when certain factor levels may not be present in the data.
`var1`	the column name of the first variable.
`var2`	(optional) the column name of the second variable (the rows in a 2-way tabulation).
`var3`	(optional) the column name of the third variable (the list in a 3-way tabulation).

Value

Returns a data.frame with frequencies and percentages of the tabulated variable(s). A 3-way tabulation returns a list of data.frames.

Examples

```
tabyl(mtcars, cyl)
tabyl(mtcars, cyl, gear)
tabyl(mtcars, cyl, gear, am)

# or using the %>% pipe
mtcars %>%
  tabyl(cyl, gear)

# illustrating show_na functionality:
my_cars <- rbind(mtcars, rep(NA, 11))
my_cars %>% tabyl(cyl)
my_cars %>% tabyl(cyl, show_na = FALSE)

# Calling on a single vector not in a data.frame:
val <- c("hi", "med", "med", "lo")
tabyl(val)
```

---

top_levels *Generate a frequency table of a factor grouped into top-n, bottom-n, and all other levels.*

Description

Get a frequency table of a factor variable, grouped into categories by level.

Usage

```
top_levels(input_vec, n = 2, show_na = FALSE)
```

Arguments

input_vec	the factor variable to tabulate.
n	number of levels to include in top and bottom groups
show_na	should cases where the variable is NA be shown?

Value

Returns a data.frame (actually a `tbl_df`) with the frequencies of the grouped, tabulated variable. Includes counts and percentages, and valid percentages (calculated omitting `NA` values, if present in the vector and `show_na = TRUE`.)

Examples

```
top_levels(as.factor(mtcars$hp), 2)
```

---

untabyl *Remove* tabyl *attributes from a data.frame.*

Description

Strips away all tabyl-related attributes from a data.frame.

Usage

```
untabyl(dat)
```

Arguments

dat	a data.frame of class tabyl.

Value

Returns the same data.frame, but without the tabyl class and attributes.

Examples

```
mtcars %>%
  tabyl(am) %>%
  untabyl() %>%
  attributes() # tabyl-specific attributes are gone
```

---

use_first_valid_of *Returns first non-NA value from a set of vectors.*

Description

At each position of the input vectors, iterates through in order and returns the first non-NA value. This is a robust replacement of the common ifelse(!is.na(x), x, ifelse(!is.na(y), y, z)). It's more readable and handles problems like ifelse's inability to work with dates in this way.

Usage

```
use_first_valid_of(..., if_all_NA = NA)
```

Arguments

...	the input vectors. Order matters: these are searched and prioritized in the order they are supplied.
if_all_NA	what value should be used when all of the vectors return NA for a certain index? Default is NA.

Value

Returns a single vector with the selected values.

Warning

Deprecated, do not use in new code. Use `dplyr::coalesce()` instead.

See Also

janitor_deprecated

Index