Package 'tidytidbits'

October 14, 2022

Type Package

Title A Collection of Tools and Helpers Extending the Tidyverse

Version 0.3.2

Author Marcel Wiesweg [aut, cre]

Maintainer Marcel Wiesweg <marcel.wiesweg@uk-essen.de>

Description
A selection of various tools to extend a data analysis workflow based on the 'tidyverse' packages. This includes high-
level data frame editing methods (in the style of 'mutate'/'mutate_at'), some methods in the style of 'purrr' and 'forcats', 'lookup' methods for dict-
like lists, a generic method for lumping a data frame by a given count,
various low-level methods for special treatment of 'NA' values, 'python'-style tuple-assignment and 'truthy'/'falsy' checks,
saving to PDF and PNG from a pipe and various small utilities.

License GPL-3

Encoding UTF-8

Imports utils, stats, grDevices, methods, rlang (>= 0.4.0), dplyr (>= 1.0.0), forcats, grid, purrr, stringr, tibble, tidyr, tidyselect, extrafont, magrittr

Suggests survival

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2022-03-16 22:30:06 UTC

R topics documented:

`add_prop_test` *Add results of prop.test to data frame*

Description

Adds prop.test results as columns to data frame based on data in columns For use with a tibble in a pipe: Using one-group prop.test, adds confidence intervals (with given conf.level) for the proportion of x positive results in n trials, and the p value that the proportion is equal to p (default: 0.5) (to add the estimated proportion itself, use `count_by`)

Usage

```
add_prop_test(
  .df,
  x,
  n,
  p = NULL,
  CI_lower_name = "CI_lower",
  CI_upper_name = "CI_upper",
  p_name = "p",
  alternative = c("two.sided", "less", "greater"),
  conf.level = 0.95,
  correct = TRUE
)
```

Arguments

`.df`	A data frame
`x`	The column/vector with the number of positive results
`n`	The column/vector/constant with the number of trials
`p`	Assumed proportion: Will add a p-value that the proportion is equal to p (default: 0.5)
`CI_lower_name, CI_upper_name, p_name`	Column names of the added columns
`alternative, conf.level, correct`	As for `prop.test`

Value

Data frame with columns added

See Also

count_by()

Examples

```
library(magrittr)
if (requireNamespace("survival", quietly = TRUE))
{
   survival::aml %>%
   count_by(x) %>%
   add_prop_test(n, sum(n), rel)
}
```

---

all_or_all_na *All() giving NA only if all values are NA*

Description

All() giving NA only if all values are NA

Usage

```
all_or_all_na(...)
```

Arguments

...	Values

Value

NA if and only if all ... are NA, else all(...), ignoring NA values

---

any_or_all_na *Any() giving NA only if all values are NA*

Description

Any() giving NA only if all values are NA

Usage

```
any_or_all_na(...)
```

Arguments

...	Values

Value

NA if and only if all ... are NA, else any(...), ignoring NA values

---

`append_object` *Appending in a pipe, never unlisting*

Description

Append to a given list, while considering as a single object and not unlisting as base::append does. Argument order is reversed compared to base::append to allow a different pattern of use in a pipe.

Usage

```
append_object(x, .l, name = NULL)
```

Arguments

`x`	Object to append. If the object is a list, then it is appended as-is, and not unlisted.
`.l`	The list to append to. Special case handling applies if .l does not exist: then an empty list is used. This alleviates the need for an initial mylist <- list()
`name`	Will be used as name of the object in the list

Value

The list .l with x appended

Examples

```
library(magrittr)
results <- list(first=c(3,4), second=list(5,6))
list(7,8) %>%
  append_object(results, "third result") ->
results
# results has length 1, containing one list named "first"
```

are_true *Vectorised conversion to logical, treating NA as False*

Description

Vectorised conversion to logical, treating NA as False

Usage

```
are_true(x)
```

Arguments

x	A vector

Value

A logical vector of same size as x which is true where x is true (`rlang::as_logical`) and not NA

as_formatted_number *Format numeric value for output*

Description

Vectorised conversion

Usage

```
as_formatted_number(x, decimal_places = 1, remove_trailing_zeroes = T)
```

Arguments

x	Numeric vector
decimal_places	Decimal places to display
remove_trailing_zeroes	If the required decimal places are less than decimal places, should resulting trailing zeros be removed?

Value

Character vector

Examples

```
as_formatted_number(0.74167, 2) # gives "0.74"
```

`as_formatted_p_value` *Formatting p values*

Description

Vectorised conversion

Usage

```
as_formatted_p_value(
  x,
  decimal_places = 3,
  prefix = "p",
  less_than_cutoff = 0.001,
  remove_trailing_zeroes = T,
  alpha = 0.05,
  ns_replacement = NULL
)
```

Arguments

`x`	Numeric vector
`decimal_places`	Decimal places to display
`prefix`	Prefix to prepend (default "p=")
`less_than_cutoff`	Cut-off for small p values. Values smaller than this will be displayed like "p<..."
`remove_trailing_zeroes`	If the required decimal places are less than decimal places, should resulting trailing zeros be removed?
`alpha`	Cut-off for assuming significance, usually 0.05
`ns_replacement`	If p value is not significant (is > alpha), it will be replace by this string (e.g. "n.s.") If NULL (default), no replacement is performed.

Vectorised (in parallel) over x, prefix, less_than_cutoff, alpha and ns_replacement.

Value

Character vector

Examples

```
as_formatted_p_value(0.02) # "p=0.02"
as_formatted_p_value(0.00056) # "p<0.001"
```

as_percentage_label *Format as percentage for output*

Description

Vectorised conversion

Usage

```
as_percentage_label(x, decimal_places = 1, include_plus_sign = F)
```

Arguments

x	Numeric vector
decimal_places	Decimal places to display
include_plus_sign	prepend a "+" to the output if positive (if negative, a "-" must be prepended of course)

Value

Character vector

Examples

```
as_percentage_label(0.746) # gives "74.6%"
```

categorical_test_by *Categorical test in a pipe*

Description

Performs classical categorical tests on two columns of a data frame. Per default, will perform chisq.test or fisher.test on the contingency table created by var1 and var2.

Usage

```
categorical_test_by(
  .tbl,
  var1,
  var2,
  na.rm = T,
  test_function_generator = NULL,
  ...
)
```

Arguments

`.tbl`	A data frame
`var1`	First column to count by
`var2`	Second column to count by
`na.rm`	Shall NA values be removed prior to counting?
`test_function_generator`	A function receiving the matrix to test and returning a named vector with the test function to use. The default uses fisher.test if one count is 5 or lower, otherwise chisq.test. Test functions must return a value with at least one component named "p.value".
`...`	Passed on to the test function

Details

Returns a one-line data frame as result and thus plays nicely with for example `map_dfr`.

Value

A one-row data frame with the columns:

- "var1,var2": The tested variables
- "test": Label of the test function (default: fisher or chisq)
- "p-value": P value
- "result": List column with full result object (default: htest)
- "contingency_table": List column with contingency table data frame as return by `contingency_table_by`

Examples

```
library(magrittr)
if (requireNamespace("datasets", quietly = TRUE))
{
   mtcars %>% categorical_test_by(cyl >= 6, gear)
}
```

`contingency_table_as_matrix` *Convert contingency table to classical R matrix*

Description

Converts the result of `contingency_table_by` to a classical matrix

Usage

```
contingency_table_as_matrix(table_frame)
```

Arguments

table_frame	Result of contingency_table_by

Value

A matrix

---

contingency_table_by *Create data frame formed like a contingency-table*

Description

Counts by the specified two variables and the pivots the count data frame wider to a two-dimensional contingency table. Please note that the resulting data frame is suitable for convenient output or use with functions that work on matrix-like data, but does not fulfill the tidy data criteria.

Usage

```
contingency_table_by(.tbl, var1, var2, na.rm = F, add_margins = F)
```

Arguments

.tbl	A data frame
var1	First column to count by
var2	Second column to count by
na.rm	Shall NA values be removed prior to counting?
add_margins	Add row- and column wise margins as extra column and row

Value

A data frame

Examples

```
library(magrittr)
if (requireNamespace("datasets", quietly = TRUE))
{
   mtcars %>% contingency_table_by(cyl, gear)
}
```

count_at *Count by multiple variables*

Description

Count by multiple variables

Usage

```
count_at(
  .tbl,
  .vars,
  .grouping = vars(),
  label_style = "long",
  long_label_column_names = c("variable", "category"),
  column_names = c("n", "rel", "percent"),
  na_label = "missing",
  percentage_label_decimal_places = 1,
  add_grouping = T,
  na.rm = F
)
```

Arguments

.tbl	A data frame
.vars	A list of variables (created using vars()) for which count_by is to be called
.grouping	Additional grouping to apply prior to counting
label_style	Character vector containing one of "wide" and "long" or both. • "wide": Include labels in wide format, i.e., for each variable one column named as variable and giving the label for the corresponding count, but NA for all rows from different variables • "long": Include two meta columns, one giving the variable that is counted (value from .vars), the second giving the label (which value/category of the variable is counted?).
long_label_column_names	Character vector of size 2: If label_style contains "long", the names for the additional meta columns for variable and category
column_names	vector if size 1 to 3, giving the names of (in order if unnamed, or named with n, rel, percent) the column containing the count, the relative proportion, and the latter formatted as a percent label. If a name is not contained, it will not be added (requires named vector).
na_label	If na.rm=F, label to use for counting NA values
percentage_label_decimal_places	Decimal precision of the percent label

add_grouping	Shall a pre-existing grouping be preserved for counting (adding the newly specified grouping)? Default is yes, which differs from group_by.
na.rm	Shall NA values be removed prior to counting?

Value

A data frame concatenated from individual count_by results, with labels as per label_style.

Examples

```
library(magrittr)
library(datasets)
library(dplyr)
mtcars %>% count_at(vars(gear, cyl))
```

---

count_by *Count according to grouping*

Description

Similar to dplyr::count(), but also adds the relative proportion and a percent-formatted string of the relative proportion, and allows to specify the column names.

Usage

```
count_by(
  .tbl,
  ...,
  column_names = c("n", "rel", "percent"),
  percentage_label_decimal_places = 1,
  add_grouping = T,
  na.rm = F
)
```

Arguments

.tbl	A data frame
...	Columns / expressions by which to group / which shall be used for counting.
column_names	vector if size 1 to 3, giving the names of (in order if unnamed, or named with n, rel, percent) the column containing the count, the relative proportion, and the latter formatted as a percent label. If a name is not contained, it will not be added (requires named vector).
percentage_label_decimal_places	Decimal precision of the percent label
add_grouping	Shall a pre-existing grouping be preserved for counting (adding the newly specified grouping)? Default is yes, which differs from group_by.
na.rm	Shall NA values be removed prior to counting?

Value

The counted data frame

Examples

```
library(magrittr)
if (requireNamespace("survival", quietly = TRUE))
{
   survival::aml %>%
   count_by(x)
}
```

`dina` *The DIN A paper formats*

Description

The DIN A paper formats

Usage

```
dinAFormat()

dinA_format()

dinA(n)

dinAWidth(n)

dinA_width(n)

dinAHeight(n)

dinA_height(n)
```

Arguments

`n`	DIN A paper format index (0-10)

Value

A named list (0-10) of named vectors (long, short) of unit objects with the size in inches of the DIN A paper formats

named unit vector (long, short) with the size in inches of the requested DIN A paper format

the long side / width in landscape as a unit object in inches

the short side / height in landscape as a unit object in inches

See Also

unit

`equal_including_na` *Compare vectors, treating NA like a value*

Description

Compare vectors, treating NA like a value

Usage

```
equal_including_na(v1, v2)
```

Arguments

`v1, v2`	Vectors of equal size

Value

Returns a logical vector of the same size as v1 and v2, TRUE wherever elements are the same. NA is treated like a value level, i.e., NA == NA is true, NA == 1 is false.

`eval_unquoted` *Execute code after tidy evaluation*

Description

This function takes R code as arguments and executes this code in the calling environment. All quoted variables (using rlang's quasiquotation, !! or !!!) will be unquoted prior to evaluation. This results in executed in code in which the variable is replaced verbatim by its value, as if you had typed the variable's value. This is particularly useful for functions using base R's substitute() approach, such as functions taking formulas, and you have built the formula dynamically. It is unnecessary for all functions based on tidy_eval (dplyr).

Usage

```
eval_unquoted(...)
```

Arguments

`...`	R code snippets

Value

The value of the last evaluated expression.

Examples

```
library(rlang)
# Note that evaluation takes place in the calling environment!
l <- quo(l <- 1) # l is a quosure in our env
eval_unquoted(!!l)
l == 1 # TRUE: l is now a vector
```

`expression_list` *Extract symbols from an expression of symbols and operators*

Description

Extract symbols from an expression of symbols and operators

Usage

```
expression_list(expr, seps = "+")

quosure_list(expr, seps = "+", env = caller_env())

symbol_string_list(expr, seps = "+")
```

Arguments

`expr`	A language expression
`seps`	Operators to consider as separators
`env`	Environment for the created quosure

Value

A list of all symbols in the expression, as symbol, quosure or text.

Examples

```
expression_list(a+b+c+d)
```

first_non_nas *Row-wise first value which is not NA*

Description

This is useful in conjunction with dplyr's mutate to condense multiple columns to one, where in each sample typically only one of n columns has a value, while the others are NA. Returns one vector of the same length as each input vector containing the result. Note that factors will be converted to character vectors (with a warning).

Usage

```
first_non_nas(...)
```

Arguments

...	multiple vectors of same type and size, regarded as columns

Value

Returns a vector of type and size as any of the given vectors (vectors regarded a column, number of rows is size of each vectors) For each "row", returns the first value that is not NA, or NA iff all values in the row are NA.

Examples

```
library(tibble)
library(magrittr)
library(dplyr)
# Creates a column containing (4, 2, 2)
tibble(a=c(NA, NA, 2), b=c(4, NA, 5), c=c(1, 2, 3)) %>%
  mutate(essence=first_non_nas(a, b, c))
```

first_non_nas_at *Row-wise first value that is not NA*

Description

Row-wise first value that is not NA

Usage

```
first_non_nas_at(.tbl, ...)
```

Arguments

.tbl	A data frame
...	A column selection, as for dplyr::select

Value

A vector of length nrow(.tbl) containing the first found non-na value

---

first_not *First argument that does not equal a given value*

Description

First argument that does not equal a given value

Usage

```
first_not(not, ...)
```

Arguments

not	Value: we look for the first value not equal to this one
...	Values

Value

The first value that does not equal "not", or NA iff all equal "not"

Examples

```
# 5
first_not(1, 1,1,1,5)
```

---

first_not_na *First argument that is not NA*

Description

First argument that is not NA

Usage

```
first_not_na(...)
```

Arguments

...	Values

Value

The first argument that is not NA, or NA iff all are NA

---

first_which_non_na_at *Row-wise first index of column that is not NA*

Description

Row-wise first index of column that is not NA

Usage

```
first_which_non_na_at(.tbl, ...)
```

Arguments

.tbl	A data frame
...	A column selection, as for dplyr::select

Value

A numeric vector of length nrow(.tbl) containing the index of the first found non-na value in the given columns. Possible values are NA (all values in that row are NA), and 1 ... number of columns in selection

---

first_which_not_na *First which() is not na*

Description

First which() is not na

Usage

```
first_which_not_na(...)
```

Arguments

...	Values; concatenated as given. Intended use is with one vector of length > 1 or multiple single arguments.

Value

The index of the first value which is not NA, or NA iff all elements are NA.

Examples

```
# 4
first_which_not_na(NA, NA, NA, 56)
```

`format_numbers_at` *Format numeric columns for display*

Description

Combines `mutate_at()` and `as_formatted_number()`

Usage

```
format_numbers_at(.tbl, .vars, decimal_places = 1, remove_trailing_zeroes = T)
```

Arguments

`.tbl`	A data frame
`.vars`	A vars() list of symbolic columns
`decimal_places`	Decimal places to display
`remove_trailing_zeroes`	If the required decimal places are less than decimal places, should resulting trailing zeros be removed?

Value

Value of mutate_at

See Also

`format_p_values_at`

Examples

```
library(tibble)
library(magrittr)
library(dplyr)
tibble(a=c(0.1, 0.238546)) %>%
    format_numbers_at(vars(a))
```

`format_p_values_at` *Format numeric columns for display*

Description

Combines `mutate_at()` and `as_formatted_p_value()`

Usage

```
format_p_values_at(
  .tbl,
  .vars,
  decimal_places = 3,
  prefix = "p",
  less_than_cutoff = 0.001,
  remove_trailing_zeroes = T,
  alpha = 0.05,
  ns_replacement = NULL
)
```

Arguments

`.tbl`	A data frame
`.vars`	A vars() list of symbolic columns
`decimal_places`	Decimal places to display
`prefix`	Prefix to prepend (default "p=")
`less_than_cutoff`	Cut-off for small p values. Values smaller than this will be displayed like "p<..."
`remove_trailing_zeroes`	If the required decimal places are less than decimal places, should resulting trailing zeros be removed?
`alpha`	Cut-off for assuming significance, usually 0.05
`ns_replacement`	If p value is not significant (is > alpha), it will be replace by this string (e.g. "n.s.") If NULL (default), no replacement is performed.
	Vectorised (in parallel) over x, prefix, less_than_cutoff, alpha and ns_replacement.

Value

Value of mutate_at

See Also

`format_numbers_at`

Examples

```
library(tibble)
library(magrittr)
library(dplyr)
tibble(p=c(0.05, 0.0001)) %>%
    format_numbers_at(vars(p))
```

---

identity_order *Ordering function: identity order*

Description

This can be used in a place where a function with a signature like `order` is required. It simply retains the original order.

Usage

```
identity_order(x, ...)
```

Arguments

x	a vector
...	Effectively ignored

Value

An integer vector

---

invalid *A notion of valid and invalid*

Description

An object is valid if it is not null, not missing (NA), and is not an empty vector. Note that this is per se not vectorised, because a non-empty list or vector is valid as such.

Usage

```
invalid(x)

valid(x)
```

Arguments

x	Any object, value or NULL

Value

logical

Functions

- `valid`: x is not invalid

Examples

```
invalid(NULL) # TRUE
invalid(NA) # TRUE
invalid(list()) # TRUE
invalid("a") # FALSE
invalid(c(1,2,3)) # FALSE
```

`invert_value_and_names` *Inverting name and value*

Description

Inverting name and value

Usage

```
invert_value_and_names(v)
```

Arguments

v	A named vector

Value

A vector where names(v) are the values and the values of v are the names

`local_variables` *"Variable generating" functions*

Description

A pair of functions that allows a "variable generating" function and read this function's local vars into the environment of the caller.

Usage

```
local_variables(env = parent.frame())

localVariables(env = parent.frame())

source_variables(localVars)

sourceVariables(localVars)
```

Arguments

`env`	Parent environment
`localVars`	Result of function call exporting an environment

Value

Named vector of created local variables

The updated environment

Examples

```
myVariableGeneratingFunction <- function()
{
  x <- 1
  y <- 2
  local_variables()
}
myMainFunction <- function()
{
  source_variables(myVariableGeneratingFunction())
  print(c(x, y))
}
```

lookup *Lookup in a dictionary*

Description

Looks up all values as keys of the dictionary and returns the values.

Usage

```
lookup(dict, ..., default = NA, dict_key_is_regex = F, key_is_regex = F)

lookup_int(dict, ..., default = NA, dict_key_is_regex = F, key_is_regex = F)

lookup_chr(dict, ..., default = NA, dict_key_is_regex = F, key_is_regex = F)

lookup_lgl(dict, ..., default = NA, dict_key_is_regex = F, key_is_regex = F)

lookup_dbl(dict, ..., default = NA, dict_key_is_regex = F, key_is_regex = F)

lookup_num(dict, ..., default = NA, dict_key_is_regex = F, key_is_regex = F)
```

Arguments

dict	A dictionaryish vector (named: key -> value)
...	Keys to lookup in the dictionary
default	Default value to return if key is not found. Can be a value or function (called with the key). Note: default is to return NA; another very intuitive case is to return the key itself. To achieve this, pass `default = identity`.
dict_key_is_regex	Should the dictionary keys, the names of dict, be regarded as regular expressions? (excludes key_is_regex)
key_is_regex	Should the keys to lookup be regarded as regular expressions? (excludes dict_key_is_regex)

Value

A list of the same size as ..., containing the lookup results. For the type-specific functions, returns a vector typed as requested, requiring all lookup results to have matching type.

Examples

```
a <- list("x", "y", "z")
dict <- c(x="xc", y="yv")
# returns c("xc", "yv", na_chr)
lookup_chr(dict, a)#'
# returns c("xc", "yv", "z")
lookup_chr(dict, "x", "y", "z", default=identity)
```

lookup_function_from_dict *Creating a lookup function from dictionary*

Description

Creating a lookup function from dictionary

Usage

```
lookup_function_from_dict(dict, default = identity, dict_key_is_regex = F)
```

Arguments

dict	A dictionaryish character vector (named: key -> value)
default	Value to return if key is not found, or function to evaluate with key as argument
dict_key_is_regex	If True, treats dictionary keys are regular expressions when matching

Value

A function which can be called with keys and performs the described lookup, returning the value (string)

lump *Generic lumping*

Description

Takes levels (labels, factor levels) and corresponding counts and "lumps" according to specified criteria (either n or prop), i.e. preserves some rows and summarises the rest in a single "Other" row

Usage

```
lump(
  levels,
  count,
  n,
  prop,
  other_level = "Other",
  ties.method = c("min", "average", "first", "last", "random", "max")
)
```

Arguments

levels	Vector of levels
count	Vector of corresponding counts
n	If specified, n rows shall be preserved.
prop	If specified, rows shall be preserved if their count >= prop
other_level	Name of the "other" level to be created from lumped rows
ties.method	Method to apply in case of ties

Value

A dictionary (named vector) of levels -> new levels

---

lump_rows *Lump rows of a tibble*

Description

A verb for a dplyr pipeline: In the given data frame, take the .level column as a set of levels and the .count column as corresponding counts. Return a data frame where the rows are lumped according to levels/counts using the parameters n, prop, other_level, ties.method like for `lump()`. The resulting row for other_level has `level=other level`, `count=sum(count of all lumped rows)`. For the remaining columns, either a default concatenation is used, or you can provide custom summarising statements via the summarising_statements parameter. Provide a list named by the column you want to summarize, giving statements wrapped in quo(), using syntax as you would for a call to summarise().

Usage

```
lump_rows(
  .df,
  .level,
  .count,
  summarising_statements = quos(),
  n,
  prop,
  remaining_levels,
  other_level = "Other",
  ties.method = c("min", "average", "first", "last", "random", "max")
)
```

Arguments

.df	A data frame
.level	Column name (symbolic) containing a set of levels
.count	Column name (symbolic) containing counts of the levels
summarising_statements	The "lumped" rows need to have all their columns summarised into one row. This parameter is a vars() list of arguments as if used in a call to summarise(), name is column name, value is language. If not provided for a column, a default summary will be used which takes the sum if numeric, concatenates text, or uses any() if logical.
n	If specified, n rows shall be preserved.
prop	If specified, rows shall be preserved if their count >= prop
remaining_levels	Levels that should explicitly not be lumped
other_level	Name of the "other" level to be created from lumped rows
ties.method	Method to apply in case of ties

Value

The lumped data frame

See Also

lump

---

named_palette *Named color palette*

Description

Returns the palette named by names. This is useful to pick only a few specific colors from a larger palette.

Usage

```
named_palette(palette, names, color_order = NULL)
```

Arguments

palette	Colors
names	Names
color_order	If specified, will reorder palette by this ordering vector

Value

A named palette. If the palette is longer than names, will only use the first n entries. If names is longer than palette, will recycle colors.

orderer_function_from_sorted_vectors *Orderer function for complex sorting*

Description

If you want to order by multiple features and have sorted vectors for each feature which describe the intended order

Usage

```
orderer_function_from_sorted_vectors(...)
```

Arguments

...	k sorted vectors, in order of priority

Value

A function which takes (at least) k vectors This function will return an order for these vectors determined by the sorted vectors

order_factor_by *Reorder a factor*

Description

Makes f a factor ordered according to ... (which is passed to order)

Usage

```
order_factor_by(.f, ...)
```

Arguments

.f	A factor
...	Passed to `order()`. Should be vectors of the same size as .f.

Details

This is a thin wrapper around `forcats::fct_reorder()`, which is unintuitive in conjunction with order().

Value

Reordered factor

See Also

rename_reorder_factor, rename_factor, forcats::fct_reorder

---

pluck_vector *Pluck with simplified return value*

Description

Like purrr::pluck(), but will return simplify()'ed as a vector

Usage

```
pluck_vector(.x, ..., .default = NULL)
```

Arguments

.x	Container object
...	Accessor specification
.default	Default value

Value

Result of purrr::pluck(), transformed y purrr::simplify()

---

prepare_directory *Directory creation*

Description

Creates directory if it does not yet exist

Usage

```
prepare_directory(folder)
```

Arguments

folder	Folder path

Value

Folder path

prepare_path *Directory creation and file path concatenation*

Description

Given a folder, file base name and suffix, ensures the directory exists, and returns the ready file path.

Usage

```
prepare_path(folder, fileBaseName, fileSuffix)
```

Arguments

folder	Folder path, without trailing slash
fileBaseName	File base name, excluding trailing dot
fileSuffix	File suffix without leading dot (e.g., "png", "pdf")

Value

Complete file path

prepend_object *Prepending in a pipe, never unlisting*

Description

Prepend to a given list, while considering as a single object and not unlisting. Argument order is reversed compared to base::append or purrr::prepend to allow a different pattern of use in a pipe.

Usage

```
prepend_object(x, .l, name = NULL, before = 1)
```

Arguments

x	Object to prepend. If the object is a list, then it is appended as-is, and not unlisted.
.l	The list to append to. Special case handling applies if .l does not exist: then an empty list is used. This alleviates the need for an initial mylist <- list()
name	Will be used as name of the object in the list
before	Prepend before this index

Value

The list .l with x prepended

Examples

```
#' library(tibble)
library(magrittr)
library(dplyr)
results <- list(second=list(1,2), third=list(3))
list(-1, 1) %>%
  prepend_object(results, "first") ->
results
# results has length 3, containing three lists
```

---

print_deparsed *Print deparsed language*

Description

Prints deparsed R language tree of given expression

Usage

```
print_deparsed(language)
```

Arguments

language	R language

Value

Invisible null

---

rename_factor *Rename a factor.*

Description

Renames the levels of a factor.

Usage

```
rename_factor(.f, ..., reorder = F)
```

Arguments

.f	A factor or vector (if .f is not yet a factor, it is made one)
...	Dictionaryish arguments, named by old level, value is new level ("old level" = "new level"). You can pass single named arguments, or named vectors or named lists, which will be spliced.
reorder	Logical: If True, the levels will additionally be reordered in the order of first appearance in the arguments

Value

A renamed and reordered factor

See Also

rename_reorder_factor, order_factor_by, forcats::fct_recode, forcats::fct_relevel

---

rename_reorder_factor *Rename and reorder a factor.*

Description

The factor will be recoded according to value_label_dict and, if requested, also reordered by the order of this vector. Secondly, the vector will be reordered according to reorder_vector, if given.

Usage

```
rename_reorder_factor(
  .f,
  value_label_dict,
  reorder_vector,
  reorder_by_value_label_dict = T
)
```

Arguments

.f	A factor or vector (if .f is not yet a factor, it is made one)
value_label_dict	a dictionary (named list or vector) of old->new factor levels
reorder_vector	vector of factor levels (the new levels according to value_label_dict). It need not contain all levels, only those found will be reorderer first
reorder_by_value_label_dict	Should the factor also be reordered following the order of value_label_dict?

Value

A renamed and reordered factor

See Also

rename_factor, order_factor_by, forcats::fct_recode, forcats::fct_relevel

replace_sequential_duplicates *Replace sequential duplicates*

Description

Replace sequential duplicates

Usage

```
replace_sequential_duplicates(strings, replace_with = "", ordering = NULL)
```

Arguments

strings	Character vector
replace_with	Replacement string
ordering	Optional: treat strings as if ordered like strings[ordering], or, if a function, strings[ordering(strings)]

Value

A character vector with strings identical to the previous string replaced with replace_with

Examples

```
# returns c("a", "", "b", "", "", "a")
replace_sequential_duplicates(c("a", "a", "b", "b", "b", "a"))
```

save_pdf *Save plot as PDF*

Description

Save plot as PDF

Usage

```
save_pdf(plot, folder, fileBaseName, width, height, ...)
```

Arguments

plot	A plot object that can be printed, e.g. result of ggplot2, plot_grid
folder	Destination folder (will be created if it does not exist)
fileBaseName	File base name (suffix ".pdf" will be added)
width, height	PDF width and height in inches or as grid::unit. If missing and the plot object has a "papersize" attribute c(width, height), this will be used.
...	Further arguments which will be passed to cairo_pdf, e.g. family

save_png *Save plot as PNG*

Description

Save plot as PNG

Usage

```
save_png(
  plot,
  folder,
  fileBaseName,
  width,
  height,
  dpi = 300,
  background = c("white", "transparent"),
  ...
)
```

Arguments

plot	A plot object that can be printed, e.g. result of ggplot2, plot_grid
folder	Destination folder (will be created if it does not exist)
fileBaseName	File base name (suffix ".png" will be added)
width, height	PNG width and height in inches or as grid::unit. If missing and the plot object has a "papersize" attribute c(width, height), this will be used.
dpi	Resolution (determines file size in pixels, as size is given in inches)
background	Initial background color, "white" or "transparent"
...	Further arguments which will be passed to png, e.g. family

Value

invisible NULL

sequential_duplicates *Detect sequential duplicates*

Description

Detect sequential duplicates

Usage

```
sequential_duplicates(strings, ordering = NULL)
```

Arguments

strings	Character vector
ordering	Optional: treat strings as if ordered like strings[ordering], or, if a function, strings[ordering(strings)]

Value

A logical vector which indicates if a string is identical to the previous string.

Examples

```
# return c(F, T, F, T, T, F)
sequential_duplicates(c("a", "a", "b", "b", "b", "a"))
```

---

str_locate_match *Combine str_match and str_locate*

Description

For every pattern, return the index of the first match of pattern in strings

Usage

```
str_locate_match(patterns, strings)
```

Arguments

patterns	Character vector of patterns
strings	Character vector of strings

Value

Integer vector of length(patterns) where entry i gives the index in strings where pattern i first matched

symbol_as_quosure *Make quosure from symbol*

Description

Make quosure from symbol

Usage

```
symbol_as_quosure(x, env = caller_env())
```

Arguments

x	Symbol
env	Environment for the created quosure

Value

Quosure containing the symbol

syntactically_safe *Syntactically safe names*

Description

Makes the names syntactically safe by wrapping them in “ if necessary

Usage

```
syntactically_safe(expr_strings)
```

Arguments

expr_strings	Strings to convert to syntactically safe form

Value

Strings converted to syntactically safe form

true_or_na *Test for logical true or NA*

Description

Test for logical true or NA

Usage

```
true_or_na(x)
```

Arguments

x	Logical

Value

True if and only if x is TRUE or x is NA, False otherwise.

truthy *A python / javascript-like "truthy" notion*

Description

Values are truthy that are not null, NA, empty, 0, or FALSE.

Usage

```
truthy(x)

falsy(x)
```

Arguments

x	Any object, value or NULL

Details

Note that this is per se not vectorised, because a non-empty list or vector is "truthy" as such.

Value

logical

Functions

- `falsy`: x is not truthy

tuple_assignment *Infix operator for python-style tuple assignment*

Description

Infix operator for python-style tuple assignment

Usage

```
l %=% r

g(...)
```

Arguments

l	left-hand side: "tuple" or variables created by g()
r	right-hand side: Vector to assign to left-hand side variable
...	Left-hand side variables to group

Value

Last assigned value

Examples

```
g(a,b) %=% c(1,2) # equivalent to a <- 1; b <- 2
```

which_non_na *Get indices of non-NA values*

Description

Get indices of non-NA values

Usage

```
which_non_na(...)
```

Arguments

...	k vectors of the same length n, regarded as k columns with each n rows

Value

A list of n numerical vectors. Each numerical vector has a size between 0 and k and contains the indices of the vectors whose elements are not na in the corresponding row.

Examples

```
library(tibble)
library(magrittr)
library(dplyr)
# Creates a list column containing (2,3);(3);(1,2,3)
tibble(a=c(NA, NA, 2), b=c(4, NA, 5), c=c(1, 2, 3)) %>%
  mutate(non_na_idc=which_non_na(a, b, c))
```

`with_name` *Slice by name*

Description

Slices of a vector with elements of given name, or containing given patterns. Analogous accessor functions for `purrr::pluck`

Usage

```
with_name(v, name)

with_name_containing(v, pattern)

named(name)

name_contains(pattern)
```

Arguments

`v`	A vector
`name`	Name of entry to pluck
`pattern`	Pattern passed to `stringr::str_detect`

Value

A slice from v containing all elements in v with the given name, or the name of which contains pattern

`with_value_containing` *Slice by value*

Description

Slices of a vector with elements containing given patterns. Analogous accessor function for `purrr::pluck`

Usage

```
with_value_containing(v, pattern)

value_contains(pattern)
```

Arguments

`v`	A vector
`pattern`	Pattern passed to `stringr::str_detect`

Value

A slice from v containing all elements in v with the given name, or the name of which contains pattern

Index