Package ‘tidytidbits’

March 8, 2021

Type Package

Title A Collection of Tools and Helpers Extending the Tidyverse

Version 0.2.3

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

LazyData true

Imports utils, stats, grDevices, methods, rlang (>= 0.2.0), dplyr (>= 0.8.0), forcats, grid, purrr, stringr, tibble, tidyr, tidyselect, extrafont, magrittr

Suggests survival

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-03-08 16:22:06 UTC

R topics documented:

add_prop_test ................................................................. 3
add_summary ................................................................. 4
add_summary_by .............................................................. 5
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_or_all_na</td>
</tr>
<tr>
<td>any_or_all_na</td>
</tr>
<tr>
<td>append_object</td>
</tr>
<tr>
<td>are_true</td>
</tr>
<tr>
<td>as_formatted_number</td>
</tr>
<tr>
<td>as_formatted_p_value</td>
</tr>
<tr>
<td>as_percentage_label</td>
</tr>
<tr>
<td>count_at</td>
</tr>
<tr>
<td>count_by</td>
</tr>
<tr>
<td>cross_tabulate</td>
</tr>
<tr>
<td>dina</td>
</tr>
<tr>
<td>equal_including_na</td>
</tr>
<tr>
<td>eval_unquoted</td>
</tr>
<tr>
<td>execute_if</td>
</tr>
<tr>
<td>execute_in_pipeline</td>
</tr>
<tr>
<td>expression_list</td>
</tr>
<tr>
<td>first_non_nas</td>
</tr>
<tr>
<td>first_non_nas_at</td>
</tr>
<tr>
<td>first_not</td>
</tr>
<tr>
<td>first_not_na</td>
</tr>
<tr>
<td>first_which_non_na_at</td>
</tr>
<tr>
<td>first_which_not_na</td>
</tr>
<tr>
<td>format_numbers_at</td>
</tr>
<tr>
<td>format_p_values_at</td>
</tr>
<tr>
<td>identity_order</td>
</tr>
<tr>
<td>invalid</td>
</tr>
<tr>
<td>invert_value_and_names</td>
</tr>
<tr>
<td>local_variables</td>
</tr>
<tr>
<td>lookup</td>
</tr>
<tr>
<td>lookup_function_from_dict</td>
</tr>
<tr>
<td>lump</td>
</tr>
<tr>
<td>lump_rows</td>
</tr>
<tr>
<td>named_palette</td>
</tr>
<tr>
<td>orderer_function_from_sorted_vectors</td>
</tr>
<tr>
<td>order_factor_by</td>
</tr>
<tr>
<td>pluck_vector</td>
</tr>
<tr>
<td>prepare_directory</td>
</tr>
<tr>
<td>prepare_path</td>
</tr>
<tr>
<td>prepend_object</td>
</tr>
<tr>
<td>print_deparsed</td>
</tr>
<tr>
<td>rename_factor</td>
</tr>
<tr>
<td>rename_reorder_factor</td>
</tr>
<tr>
<td>replace_sequential_duplicates</td>
</tr>
<tr>
<td>save_pdf</td>
</tr>
<tr>
<td>save_png</td>
</tr>
<tr>
<td>sequential_duplicates</td>
</tr>
<tr>
<td>str_locate_match</td>
</tr>
<tr>
<td>symbol_as_quosure</td>
</tr>
</tbody>
</table>
add_prop_test

Description
Title

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

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

---

**add_summary**

Add summary to tibble

**Description**

A verb for a dplyr pipeline: Performs a call to `summarise()`, but does not reduce the data frame to one row per group, instead, adds the resulting fields to every row belonging to that group, returning the original frame with added/changed columns. Effectively, this is like calling `summarise()`, and then calling `mutate()` with all the resulting columns.

**Usage**

```r
add_summary(.data, .language)
```

**Arguments**

- `.data` Data argument, typical "first" argument in dplyr verbs
- `.language` A call to `summarise()`, or another method performing equivalent aggregation (potentially wrapping `summarise()`)

**Value**

The tibble with added columns
Description

A verb for a dplyr pipeline: Groups the frame by ... in addition to the current grouping, then calls add_summary, then returns the frame with the mutated summarising columns in the same grouping state as it was before this function was called.

Usage

add_summary_by(.data, .language, ...)

Arguments

.data Data argument, typical "first" argument in dplyr verbs
.language A call to summarise(), or another method performing equivalent aggregation (potentially wrapping summarise())
... Parameters for group_by

Value

The tibble with added columns

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

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
...

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()

ame
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?
as_formatted_p_value

Value
Character vector

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

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.

Value
Character vector
Examples

```r
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**

```r
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**

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

---

**count_at**  
*Count by multiple variables*

---

**Description**

Count by multiple variables
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.

.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.
count_by

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.
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

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

---

**cross_tabulate**

Create cross table from a tibble

Description

A wrapper of `table()` for convenient use in a dplyr pipeline: Pass the factors to tabulate as symbols or expressions like you would in `mutate()`. `useNA` and `dnn` are passed to `table()`.

Usage

```r
cross_tabulate(.df, ..., useNA = c("no", "ifany", "always"), dnn = NULL)
```

Arguments

- `.df` A data frame
- `...` Factors to tabulate by: symbolic column names / language
- `useNA`, `dnn` passed to `table()`

Value

Result from a call to `table()`

Examples

```r
library(magrittr)
if (requireNamespace("survival", quietly = TRUE))
{
  survival::bladder1 %>%
  cross_tabulate(treatment, recur) %>%
  chisq.test()
}
```
Description

The DIN A paper formats

Usage

\texttt{dinAFormat()}
\texttt{dinA\_format()}
\texttt{dinA(n)}
\texttt{dinAWidth(n)}
\texttt{dinA\_width(n)}
\texttt{dinAHeight(n)}
\texttt{dinA\_height(n)}

Arguments

\texttt{n} \quad \text{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

\texttt{unit}
## equal_including_na

### Description

Compare vectors, treating NA like a value

### Usage

```r
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

### 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

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

### Arguments

- `...` R code snippets

### Value

The value of the last evaluated expression.
execute_if

Examples

```r
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
```

execute_if

Conditional execution in a pipeline

Description

A verb for a magrittr pipeline: execute_if_else: The language is executed only if .predicate is true.

Usage

```r
execute_if(.data, .predicate, .language)
execute_if_else(.data, .predicate, .language_true, .language_false)
```

Arguments

- `.data`: Data argument, typical "first" argument in dplyr verbs
- `.predicate`: Evaluated to boolean. If true, executes and returns language; otherwise, returns untouched `.data`
- `.language`: Language call to execute. Write is just as if you would without the execute_if: Will be used as the right-hand side of "%>%" with all possible options of magrittr.
- `.language_true`: Execute if predicate it TRUE
- `.language_false`: Execute if predicate it FALSE

Value

Result of `.language`

Examples

```r
library(magrittr)
library(dplyr)
library(tibble)
library(stringr)
convert_to_quartiles <- TRUE
tibble(score=c(1,2,3,4,1,2,3,4,2,3,2,3,4,3,3)) %>%
    mutate(do_something=1) %>%
    execute_if(convert_to_quartiles,
        mutate(score = cut(score, 4, labels = str_c("Quartile ", 1:4))) %>%
        filter(score > 2)
```
execute_in_pipeline  Executing language as if in a pipeline

Description

Executes the given language as if it was part of a magrittr pipeline

... .language

while .data is the lhs value provided to .language as parameter by magrittr.

Usage

execute_in_pipeline(.data, .language)

Arguments

.data A data frame

.language Language

Details

Note that language is evaluated as a quosure in its captured environment. This is fine if this method is called as a secondary helper and .language is already a quosure; otherwise you may want to explicitly set the quosure’s environment to your caller’s env.

Value

Result of the executed language

evaluation_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

```r
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 \hspace{1cm} \textit{First which() is not na}

\textbf{Description}

First which() is not na

\textbf{Usage}

\texttt{first\_which\_not\_na(...)}

\textbf{Arguments}

\texttt{...} \hspace{0.5cm} Values; concatenated as given. Intended use is with one vector of length \textgreater{} 1 or multiple single arguments.

\textbf{Value}

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

\textbf{Examples}

\begin{verbatim}
  # 4
  first\_which\_not\_na(NA, NA, NA, 56)
\end{verbatim}

\textbf{format\_numbers\_at} \hspace{1cm} \textit{Format numeric columns for display}

\textbf{Description}

Combines \texttt{mutate\_at()} and \texttt{as\_formatted\_number()}

\textbf{Usage}

\texttt{format\_numbers\_at(.tbl, .vars, decimal\_places = 1, remove\_trailing\_zeroses = T)}

\textbf{Arguments}

\begin{itemize}
  \item \texttt{.tbl} \hspace{0.5cm} A data frame
  \item \texttt{.vars} \hspace{0.5cm} A vars() list of symbolic columns
  \item \texttt{decimal\_places} \hspace{0.5cm} Decimal places to display
  \item \texttt{remove\_trailing\_zeroses} \hspace{0.5cm} If the required decimal places are less than decimal places, should resulting trailing zeros be removed?
\end{itemize}
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

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
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 as regular expressions when matching

Value

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

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

### 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

```r
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

```r
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.
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 by 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**

```r
text <- 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**

```r
text <- 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

```r
# 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**

```r
print_deparsed(language)
```

**Arguments**

- `language`  
  R language

**Value**

Invisible null

---

**rename_factor**  
*Rename a factor.*

**Description**

Renames the levels of a factor.

**Usage**

```r
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
rename_reorder_factor

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**

```r
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**

```r
# 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**

```r
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

Description

Save plot as PNG

Usage

```r
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

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

```r
# 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

**Usage**

\[\text{symbol\_as\_quosure}(x, \text{env} = \text{caller\_env()})\]

**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**

\[\text{syntactically\_safe}(\text{expr\_strings})\]

**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**

```r
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**

```r
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**

```r
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

```r
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

```r
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

-%=%(tuple_assignment), 39
add_prop_test, 3
add_summary, 4, 5
add_summary_by, 5
all_or_all_na, 5
any_or_all_na, 6
append_object, 6
are_true, 7
as_formatted_number, 7, 20
as_formatted_p_value, 8, 21
as_logical, 7
as_percentage_label, 9
ccairo_pdf, 34
count, 11
count_at, 9
count_by, 4, 10, 11
cross_tabulate, 12
dinA(dina), 13
dina, 13
dinA_format(dina), 13
dinA_height(dina), 13
dinA_width(dina), 13
dinAFormat(dina), 13
dinAHeight(dina), 13
dinAWidth(dina), 13
equal_including_na, 14
eval_unquotted, 14
execute_if, 15
execute_if_else(execute_if), 15
execute_in_pipeline, 16
expression_list, 16
falsy(truthy), 38
fct_recode, 33
fct_relevel, 33
fct_reorder, 29, 30
first_non_nas, 17
first_non_nas_at, 18
first_not, 18
first_not_na, 19
first_which_non_na_at, 19
first_which_not_na, 20
format_numbers_at, 20, 22
format_p_values_at, 21, 21
g(tuple_assignment), 39
identity_order, 22
invalid, 23
invert_value_and_names, 23
local_variables, 24
localVariables(local_variables), 24
lookup, 25
lookup_chr(lookup), 25
lookup_dbl(lookup), 25
lookup_function_from_dict, 26
lookup_int(lookup), 25
lookup_lgl(lookup), 25
lookup_num(lookup), 25
lump, 26, 27, 28
lump_rows, 27
mutate_at, 20, 21
name_contains(with_name), 40
named(with_name), 40
named_palette, 28
order, 22, 29
order_factor_by, 29, 33
orderer_function_from_sorted_vectors, 29
pluck, 30, 40, 41
pluck_vector, 30
png, 35
prepare_directory, 30
prepare_path, 31
prepend_object, 31
print_deparsed, 32
prop.test, 3
quosure_list(expression_list), 16
rename_factor, 30, 32, 33
rename_reorder_factor, 30, 33, 33
replace_sequential_duplicates, 34
save_pdf, 34
save_png, 35
select, 18, 19
sequential_duplicates, 35
simplify, 30
source_variables(local_variables), 24
sourceVariables(local_variables), 24
str_detect, 40, 41
str_locate_match, 36
summarise, 28
symbol_as_quosure, 37
symbol_string_list(expression_list), 16
syntactically_safe, 37
table, 12
true_or_na, 38
truthy, 38
tuple_assignment, 39
unit, 13, 34, 35
valid(invalid), 23
value_contains(with_value_containing), 41
which_non_na, 39
with_name, 40
with_name_containing(with_name), 40
with_value_containing, 41