Package 'fabR'

April 15, 2023

Type Package

Title Wrapper Functions Collection Used in Data Pipelines

Version 1.1.1

Description The goal of this package is to provide wrapper functions in the data cleaning and cleansing processes. These function helps in messages and interaction with the user, keep track of information in pipelines, help in the wrangling, munging, assessment and visualization of data frame-like material.

License GPL-3

Depends R (>= 3.4)

Imports dplyr, rlang, lubridate, ggplot2, digest, fs, janitor, tidytext, purrr, readr, readxl, stringr, tidyr, writexl, DT

URL https://github.com/GuiFabre/fabR/

BugReports https://github.com/GuiFabre/fabR/issues

RoxygenNote 7.2.3

Encoding UTF-8

Suggests knitr, rmarkdown, stats, haven, xlsx, viridis,

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Guillaume Fabre [aut, cre], Maelstrom-Research [fnd]

Maintainer Guillaume Fabre <guijoseph.fabre@gmail.com>

Repository CRAN

Date/Publication 2023-04-15 18:40:02 UTC

R topics documented:

add_index . . . 3
as_any_boolean . . . 4
as_any_date . . . 5
as_any_symbol . . . 6
collect_roxygen . . . 7
fabR_help . . . 8
file_index_create . . . 8
file_index_read . . . 9
file_index_search . . . 10
get_all_na_cols . . . 12
get_all_na_rows . . . 12
get_duplicated_cols . . . 13
get_duplicated_rows . . . 14
get_path_list . . . 15
get_unique_value_cols . . . 16
guess_date_format . . . 17
make_name_list . . . 18
message_on_prompt . . . 20
parceval . . . 20
plot_bar . . . 22
plot_box . . . 23
plot_date . . . 24
plot_density . . . 26
plot_histogram . . . 27
plot_main_word . . . 29
plot_pie . . . 30
plot_pie_valid_value . . . 31
read_csv_any_formats . . . 33
read_excel_allsheets . . . 33
silently_run . . . 34
summary_category . . . 35
summary_numerical . . . 36
summary_text . . . 37
template_visual_report . . . 39
which_any_date . . . 40
write_excel_allsheets . . . 41

Index **42**

add_index *Add an index column at the first place of a tibble*

Description

Add an index, possibly by group, at the first place of a data frame or a tibble The name by default is 'index' but can be named. If 'index' already exists, or the given name, the column can be forced to be created, and replace the other one.

Usage

```
add_index(tbl, name_index = "index", start = 1, .force = FALSE)
```

Arguments

tbl	tibble or data frame
name_index	A character string of the name of the column.
start	integer indicating first index number. 1 by default.
.force	TRUE or FALSE, that parameter indicates whether or not the column is created if already exists. FALSE by default.

Value

A tibble or a data frame containing one extra first column 'index' or any given name.

Examples

```
{

##### Example 1 ----------------------------------------------------------------
# add an index for the tibble
add_index(iris, "my_index")

##### Example 2 ----------------------------------------------------------------
# add an index for the grouped tibble
library(tidyr)
library(dplyr)

my_tbl <- tibble(iris) %>% group_by(Species) %>% slice(1:3)
fabR::add_index(my_tbl, "my_index")

}
```

as_any_boolean *Create objects of type "logical".*

Description

Create or test for objects of type "logical", and the basic logical constants. This function is a wrapper of the function `base::as.logical()` and evaluates if the object to be coerced can be interpreted as a boolean. Any object : NA, NA_integer, NA_Date_, (...), 0, 0L, F, FALSE, false, FaLsE, (...), 1, 1L,T, TRUE, true, TrUe, (...), will be converted as NA, FALSE and TRUE. Any other other will return an error.

Usage

```
as_any_boolean(x)
```

Arguments

x	Object to be coerced or tested. Can be a vector.

Value

An logical object of the same size.

See Also

`base::as.logical()`

Examples

```
{

library(dplyr)

as_any_boolean("TRUE")
as_any_boolean(c("1"))
as_any_boolean(0L)
try(as_any_boolean(c('foo')))
as_any_boolean(c(0,1L,0,TRUE,"t","F","FALSE"))
tibble(values = c(0,1L,0,TRUE,"t","F","FALSE")) %>%
  mutate(bool_values = as_any_boolean(values))

}
```

as_any_date *Create objects of class "Date"*

Description

This function takes a character string or a vector. This vector is evaluates one observation after the other, and casts the best matching date format for each of them (independently). The best matching format is tested across seven different formats provided by the lubridate library. The user can specify the wanted matching format (and can be helped using `which_any_date()` for each value or `guess_date_format()` for the values as a whole.

Usage

```
as_any_date(
  x = as.character(),
  format = c("dmy", "dym", "ymd", "ydm", "mdy", "myd", "as_date")
)
```

Arguments

x	object to be coerced.
format	A character identifying the format to apply to the object. That format can be 'ymd','ydm','dym','dmy','mdy' or 'myd'.

Details

Contrary to lubridate library or `base::as.Date()`, the function evaluates the different possibilities for a date. For example, c('02-03-1982') can be either March the 2nd or February the 3rd. The function will cast the value as NA, and a warning, since there is an ambiguity that cannot be solved, unless the user provides the format to apply.

Value

A R Object of class 'Date'.

See Also

`lubridate::ymd()`,`lubridate::ydm()`,`lubridate::dmy()`, `lubridate::dym()`,`lubridate::mdy()`,`lubridate::myd()`,
`lubridate::as_date()`, `guess_date_format()`,`which_any_date()`

Examples

```
{

library(dplyr)
library(tidyr)

##### Example 1 ------------------------------------------------------------
```

```
# Ambiguous dates -------------------------------------------------------
time <-
  tibble(time = c(
  "1983 07-19",
  "2003-01-14",
  "2010-09-29",
  "2023/12/12",
  "2009-09-03",
  "1809-01-01"))

time %>% mutate(new_time = as_any_date(time))
time %>% mutate(new_time = as_any_date(time, format = "ymd"))

##### Example 2 ---------------------------------------------------------
# Non-ambiguous dates ---------------------------------------------------
time <-
  tibble(time = c(
  "1983 07-19",
  "14-01-1925",
  "2010-09-29",
  "12/13/2015",
  "2009-09-13",
  "2025 jan the 30th",
  "1809-01-19"))

time %>% mutate(new_time = as_any_date(time))

}
```

`as_any_symbol` *Create objects of type "symbol"*

Description

Create or test for objects of type "symbol".

Usage

```
as_any_symbol(x)
```

Arguments

`x`	Object to be coerced or tested. Can be a vector, a character string, a symbol.

Value

Object of type "symbol".

Examples

```
{

as_any_symbol(coucou)
as_any_symbol("coucou")

}
```

`collect_roxygen` *Collects and Generates documentation of a package in a tibble format.*

Description

This function crawls and aggregates roxygen documentation into a tibble format. To work properly, elements must be separated with the named fields at title, at description, at ...), each at will be used as column name. The column name will also have 80 character to show the margin limit of each chunk of documentation.

Usage

```
collect_roxygen(folder_r = "R")
```

Arguments

`folder_r`	A character string identifying the folder to index. If not specified, 'R/' is the default.

Value

A tibble where each line represents a function described in a package, and each column is documentation field. Most common fields (title, description, details, param, see also, return and examples are placed ahead).

Examples

```
{

library(tidyr)
try({tibble(collect_roxygen(tempfile()))}, silent = FALSE)

}
```

fabR_help *Call the help center for full documentation*

Description

This feature is a direct call of the documentation in the repository hosting the package. The user accesses the description of the latest version of the package, the vignettes, and the list of functions.

Usage

```
fabR_help()
```

Value

Nothing to be returned. The function opens a web package.

Examples

```
{

# call the help center!
fabR_help()

}
```

file_index_create *Create an index of files in a folder*

Description

Creates a tibble listing files in a specified folder (recursively) with file path name and other useful metadata. This index can be used to quickly find files in the environment. The index also generates script to read files as R objects into the environment. Names for R objects are generated automatically from file names (R objects are not created at this step but the command line is generated and stored in the column to_eval, ready to be evaluated and generate R objects).

Usage

```
file_index_create(folder = getwd(), pattern = "^", negate = FALSE)
```

Arguments

folder	A character string identifying the folder to index. If not specified, the current folder is the default
pattern	A character string defining a pattern to sub-select within folder. Can be useful for excluding certain folders from indexing (matching by regex is supported).
negate	logical. If TRUE, return non-matching elements.

Details

The user must make sure their files are in the folder to be indexed.

Value

A tibble with folder_path, file_path, file_name, extension, file_type columns and a last column to_eval which is R code in a character vector to read the file into the environment.

Examples

```
## Not run:

file_index_create(tempdir())


## End(Not run)
```

`file_index_read` *Read, source and open objects from an index of files*

Description

Reads all files from a file index tibble as R objects to generate in the environment or R scripts to be sourced. Any other file types will be opened in browser (html files) or in environment. If no index tibble is provided, the function creates one from the working directory. (matching by regex is supported).

Usage

```
file_index_read(
  index,
  file_path = "^",
  file_name = "^",
  extension = "^",
  file_type = "^",
  assign = FALSE,
  .envir = parent.frame()
)
```

Arguments

`index`	The index (tibble) of a folder with file locations and metadata, either previously generated by file_index_create() or created from folder.
`file_path`	A character string specifying a file path to search by. Can be the full string or substring (matching by regex is supported)

file_name	A character string a file name to search by. Can be the full string or substring (matching by regex is supported).
extension	A character string a file extension to search by. Can be the full string or substring (matching by regex is supported).
file_type	A character string a file type to search by. Can be the full string or substring (matching by regex is supported).
assign	If TRUE, the name is automatically assigned from the name of the object read.
.envir	The environment to use. parent.frame() by default

Details

for each file selected, xlsx files will be read using the function `read_excel_allsheets()`, csv files will be read using the function `read_csv_any_formats()`, spss and sav files will be read using the function `haven::read_spss()`, dta files will be read using the function `haven::read_dta()`, sas7bdat and sas files will be read using the function `haven::read_sas()`, R scripts, Rmd and md files be read using the function `base::readLines()`, The whole files will be created in a list, which name is the name of the file.

Value

R objects generated in the environment or R scripts. R object names are created automatically from their file names. Otherwise return messages indicating what objects were created, or files opened, and if any troubles occurred.

See Also

`read_excel_allsheets()`, `read_csv_any_formats()`, `haven::read_spss()`, `haven::read_dta()`, `haven::read_sas()`, `base::readLines()`

Examples

```
## Not run:

index <- file_index_create(tempdir())
file_index_read(index, file_name = my_file_name)


## End(Not run)
```

---

file_index_search *Search an index of files*

Description

Searches in file index R object (tibble) based on pattern and other query options and provides a table where all the files in a specified folder and corresponding to the query are listed (recursively). If no index tibble is provided, the function creates one from the working directory.

Usage

```
file_index_search(
  index,
  file_path = "^",
  file_name = "^",
  extension = "^",
  file_type = "^",
  show_tree = FALSE
)
```

Arguments

index	The index (tibble) of a folder with file locations and metadata, either previously generated by file_index_create() or created from folder.
file_path	A character string specifying a file path to search by. Can be the full string or substring (matching by regex is supported)
file_name	A character string a file name to search by. Can be the full string or substring (matching by regex is supported).
extension	A character string a file extension to search by. Can be the full string or substring (matching by regex is supported).
file_type	A character string a file type to search by. Can be the full string or substring (matching by regex is supported).
show_tree	If TRUE, return the file tree of the query.

Details

The function displays the tree of your files. You can enable this functionality with 'show_tree = TRUE'

Value

A tibble with indexed information for files matching the query.

Examples

```
## Not run:

index <- file_index_create(tempdir())
file_index_search(index, file_name = my_file_name)


## End(Not run)
```

get_all_na_cols *Extract columns that are all 'NA' from a tibble*

Description

This helper function extracts the names of the columns in a tibble having NA values for all observations.

Usage

```
get_all_na_cols(tbl)
```

Arguments

tbl	R object(dataframe or tibble) of the input tibble

Value

A vector string indicating either that the tibble does not have empty columns or the names of the empty columns.

Examples

```
{

##### Example 1 ----------------------------------------------------------------
# All columns have observation
get_all_na_cols(iris)

##### Example 2 ----------------------------------------------------------------
# One column doesn't have any observations
library(dplyr)
get_all_na_cols(mutate(iris, new_col = NA))

}
```

get_all_na_rows *Extract columns that are all 'NA' from a tibble*

Description

This helper function extracts the names of the columns in a tibble having NA values for all observations.

Usage

```
get_all_na_rows(tbl, id_col = NULL)
```

Arguments

tbl	R object(dataframe or tibble) of the input tibble
id_col	A character string specifying the column to ignore in identification of repeated observations. If NULL (by default), all of the columns will be taken in account for repeated observation identification. The row number will be used to identify those observations.

Value

A vector string indicating either that the tibble does not have empty columns or the names of the empty columns.

Examples

```
{

##### Example 1 ---------------------------------------------------------------
# All rows have observation
get_all_na_cols(iris)

##### Example 2 ---------------------------------------------------------------
# One row doesn't have any observations
library(dplyr)
get_all_na_rows(bind_rows(iris, tibble(Species = c(NA,NA))))
get_all_na_rows(bind_rows(iris, tibble(Species =  c('an_id', 'another_id'))),
                id_col = 'Species')

}
```

---

get_duplicated_cols *Extract columns that have same values in a tibble*

Description

This helper function extracts the names of the columns in a tibble having identical values for all observations.

Usage

```
get_duplicated_cols(tbl)
```

Arguments

tbl	R object(dataframe or tibble) of the input tibble

Value

A tibble indicating which columns which values is the same in the tibble

Examples

```
{

library(dplyr)
mtcars_duplicated <-
 mtcars %>%
 mutate(
  cyl_2 = cyl,
  cyl_3 = cyl,
  mpg_2 = mpg)

 get_duplicated_cols(mtcars_duplicated)

}
```

`get_duplicated_rows` *Extract observations(rows) that have same values in a tibble*

Description

This helper function extracts the row number (or first column value) in a tibble having identical values for all columns. This function can be used either on the whole columns or excluding the first column (id) (which can be useful to identify repeated observation across different ids)

Usage

```
get_duplicated_rows(tbl, id_col = NULL)
```

Arguments

`tbl`	R object(dataframe or tibble) of the input tibble
`id_col`	A character string specifying the column to ignore in identification of repeated observations. If NULL (by default), all of the columns will be taken in account for repeated observation identification. The row number will be used to identify those observations.

Value

A tibble indicating which row which values is the same in the tibble

Examples

```
{

# the row numbers are returned to identify which observations have repeated
# values
library(dplyr)
```

```
get_duplicated_rows(bind_rows(mtcars,mtcars[1,]))

get_duplicated_rows(
  tbl = bind_rows(mtcars,mtcars[1,]) %>%
        add_index() %>%
        mutate(index = paste0('obs_',index)),
  id_col = 'index')

}
```

`get_path_list` *Get the paths of branches in a list*

Description

Function that recursively go through a list object and store in a tibble the path of each element in the list. The paths can be after that edited and accessed using `parceval()` for example.

Usage

```
get_path_list(list_obj, .map_list = NULL)
```

Arguments

`list_obj`	R list object to be evaluated
`.map_list`	non usable parameter. This parameter is only there to ensure recursivity. Any modification of this object returns NULL

Value

A tibble containing all the paths of each element of the list and the class of each leaf (can be a list, or R objects).

See Also

`parceval()`

Examples

```
{

library(dplyr)
get_path_list(
  list(
    tibble = iris,
    list = list(t1 = mtcars, t2 = tibble(iris)),
    char = "foo"))
```

```
}
```

`get_unique_value_cols` *Extract columns that are all 'NA' from a tibble*

Description

This helper function extracts the names of the columns in a tibble having NA values for all observations.

Usage

```
get_unique_value_cols(tbl)
```

Arguments

`tbl`	R object(dataframe or tibble) of the input tibble

Value

A vector string indicating either that the tibble does not have empty columns or the names of the empty columns.

Examples

```
{

##### Example 1 ----------------------------------------------------------------
# All columns have distinct observation
get_unique_value_cols(iris)

##### Example 2 ----------------------------------------------------------------
# One column doesn't have distinct observations
get_unique_value_cols(iris[1:50,])

}
```

guess_date_format *Evaluate and gives the best match to any date format using lubridate library*

Description

This function takes a tibble and a specific column. This column is evaluated one observation after the other, and finally gives the best matching date format for the whole column. The best matching format is tested across seven different formats provided by the lubridate library. Along with the format, the percentage of matching is given in the output tibble. The information of the best matching format can be used to mutate a column using `as_any_date()`.

Usage

```
guess_date_format(tbl, col = NULL)
```

Arguments

tbl	R object(dataframe or tibble) of the input tbl
col	A character string specifying a column of interest

Details

Contrary to lubridate library or `base::as.Date()`, the function evaluates the column as a whole, and does not cast the column if there is ambiguity between values. For example, ('19-07-1983', '02-03-1982') implies that 02 refers to the day and 03 refers to the month, since that order works for the first element, and doesn't otherwise.

Value

A tibble with information concerning the best matching date format, given an object to be evaluated.

See Also

`lubridate::ymd()`,`lubridate::ydm()`,`lubridate::dmy()`, `lubridate::dym()`,`lubridate::mdy()`,`lubridate::myd()`,
`lubridate::as_date()`, `which_any_date()`,`as_any_date()`

Examples

```
{

library(tidyr)

##### Example 1 ------------------------------------------------------------
# Non-ambiguous dates ------------------------------------------------
time <-
  tibble(time = c(
  "1983-07-19",
```

```
  "2003-01-14",
  "2010-09-29",
  "2023-12-12",
  "2009-09-03",
  "1509-11-30",
  "1809-01-01"))
guess_date_format(time)

##### Example 2 ------------------------------------------------------------
# Ambiguous dates ----------------------------------------------------
time <-
  tibble(time = c(
  "1983-19-07",
  "2003-01-14",
  "2010-09-29",
  "2023-12-12",
  "2009-09-03",
  "1509-11-30",
  "1809-01-01"))
guess_date_format(time)

##### Example 3 ------------------------------------------------------------
# Non date format dates ----------------------------------------------
time <-
  tibble(time = c(
  "198-07-19",
  "200-01-14",
  "201-09-29",
  "202-12-12",
  "200-09-03",
  "150-11-30",
  "180-01-01"))
guess_date_format(time)

}
```

`make_name_list` *Shortcut to create beautiful names in a list*

Description

Generate a name for an element in a list. This function is targeted for functions creations which handle lists. Those lists may need names to go through each elements. This function can works with `stats::setNames()` and allows the user to provide name shorter, more user-friendly in their lists.

Usage

```
make_name_list(args_list, list_elem)
```

Arguments

args_list	A list of character string of same length of list_elem
list_elem	A list of character string of same length of args_list

Value

A character string simplified to be used as names in a list.

See Also

stats::setNames()

Examples

```
{

library(tidyr)

#### Example 1 ------------------------------------------------------------------
# make_name_list generates names that are informative through a line of code
# or function. tibble(iris), iris %>% tibble and
# list(iris = tibble(mytibble) %>% select(Species)) will have 'iris' as name.

list(tibble(iris), tibble(mtcars)) %>%
  setNames(make_name_list(list(tibble(iris), tibble(mtcars)), args_list =
    c("IRIS %>% complicated_code","complicated_function(MTCARS)")))

#### Example 2 ------------------------------------------------------------------
# make_name_list can be used when a function uses arguments provided by the
# user to generate a list. The name is simplified and given to the list
# itself

library(dplyr)
my_function <- function(df){

  .fargs <- as.list(match.call(expand.dots = TRUE))
  list_df <-
    list(df) %>%
    stats::setNames(.,make_name_list(as.character(.fargs['df']),list(df)))
  return(list_df)}

my_function(tibble(iris))
my_function(iris %>% tibble %>% select(Species))

}
```

message_on_prompt *Shortcut to display a message and acceptation on prompt*

Description

Shortcut allowing to provide user a prompt and a message that is to be read and validated before pursuing process. This function is targeted for function creators where user interaction is required.

Usage

```
message_on_prompt(...)
```

Arguments

...	String character to put in a message

Value

Nothing to be returned. The function sends a message as a prompt in the console.

Examples

```
{

message_on_prompt("Do you want to continue? Press `enter` or `esc`")

}
```

parceval *Shortcut to turn String character into R code*

Description

Shortcut to `base::parse()` and `base::eval()` evaluate R expression in a character string, and turns it into actual R code. This function is targeted for interaction with external files (where expression is stored in text format) ; for tidy elements where code expression is generated using `dplyr::mutate()`, combined with `base::paste0()` ; in for while, map, etc. loops where character string expression can be indexed or iteratively generated and evaluated ; objects to be created (using assign, <- or «- obj) where the name of the R object is stored in a string. Some issues may occur when parceval is used in a different environment, such as in a function. Prefer eval(parse(text = ...) instead.

Usage

```
parceval(...)
```

Arguments

...	String character to be parsed and evaluated

Value

Any output generated by the evaluation of the string character.

See Also

base::parse(), base::eval()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# Simple assignation will assign 'b' in parceval environment (which is
# associated to a function and different from .GlobalEnv, by definition).
# Double assignation will put 'b' in .GlobalEnv.
# (similar to assign(x = "b",value = 1,envir = .GlobalEnv))

a <- 1
parceval("print(a)")
my_code <- paste0("a <- a + ",rep(1,3), "; message('value of a: ', a)")
parceval(my_code)

##### Example 2 ------------------------------------------------------------
# use rowwise to directly use parceval in a tibble, or use a for loop.
library(dplyr)
library(tidyr)

tibble(cars) %>%
  mutate(
    to_eval = paste0(speed,"/",dist)) %>%
  rowwise() %>%
  mutate(
    eval = parceval(to_eval))

##### Example 3 ------------------------------------------------------------
# parceval can be parcevaled itself!

code_R <-
  'as_tibble(cars) %>%
  mutate(
    to_eval = paste0(speed,"/",dist)) %>%
  rowwise() %>%
  mutate(
    eval = parceval(to_eval))'

cat(code_R)
parceval(code_R)
```

```
}
```

`plot_bar` *Draw bar plot of one (possibly grouped) column in a tibble*

Description

This function draws a bar plot of the values of a column. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_bar(
  tbl = "dplyr::storms",
  col = "status",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

`tbl`	A character string or tibble specifying the input tibble
`col`	A character string specifying a column of interest
`filter`	A character string specifying the values to filter. (equivalent to 'values in') This determines which values should be retained. It can be applied to both grouped and ungrouped data.
`negate`	If TRUE, return non-matching elements.
`missing_values`	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
`out`	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
`group_by`	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A bar plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ----------------------------------------------------------------
# cat output generated as a template when no argument provided
plot_bar()

##### Example 2 ----------------------------------------------------------------
# graph of Species
plot_bar(tbl = "dplyr::storms", col = "status", out = "ggplot2")

}
```

---

plot_box *Draw box plot of one (possibly grouped) column in a tibble*

Description

This function draws a box plot of the values of a column. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_box(
  tbl = "airquality",
  col = "Month",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string specifying a column of interest
filter	A character string specifying the values to filter. (equivalent to values in'). This determines which values should be retained. It can be applied to both grouped and ungrouped data.

negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
out	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A box plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
plot_box()

##### Example 2 ------------------------------------------------------------
# graph of Petal.Length
plot_box(tbl = iris, col = "Petal.Length", out = "ggplot2")

}
```

---

plot_date *Draw lollipop plot of one (possibly grouped) time-related column in a tibble*

Description

This function draws a lollipop plot of the values of time related column. the 'time' parameter uses lubridate syntax to specify the period of time to consider. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_date(
  tbl = "dplyr::storms",
  col = "annual",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  time = "day",
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string specifying a column of interest
filter	A character string specifying the values to filter. (equivalent to 'values in'). This determines which values should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
time	parameter following lubridate syntax to specify the period of time to consider. Can be ymd, mdy, year, months, etc. See lubridate documentation.
out	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A lollipop plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ---------------------------------------------------------
# cat output generated as a template when no argument provided
plot_date()
```

```
##### Example 2 ------------------------------------------------------------
# graph of number of storms per month
library(dplyr)
annual_storms <-
  dplyr::storms %>% sample_n(100) %>%
  mutate(annual = as_any_date(paste(year,month,day),"ymd"))
plot_date(
 tbl = annual_storms,
 col = "annual",
 time = "month",
 out = "ggplot2")

}
```

---

plot_density *Draw density plot of one (possibly grouped) column in a tibble*

Description

This function draws a density line plot of the values of a column. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_density(
  tbl = "iris",
  col = "Sepal.Length",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string specifying a column of interest
filter	A character string specifying the values to filter. (equivalent to 'values in') This determines which values should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.

out	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A density plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
plot_density()

##### Example 2 ------------------------------------------------------------
# graph of Petal.Length
plot_density(tbl = iris, col = "Petal.Length", out = "ggplot2")

}
```

---

plot_histogram *Draw histogram of one (possibly grouped) column in a tibble*

Description

This function draws a histogram plot of the values of a column. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_histogram(
  tbl = "airquality",
  col = "Ozone",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
```

```
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string specifying a column of interest
filter	A character string specifying the values to filter. (equivalent to 'values in') This determines which values should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
out	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A hist plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
plot_histogram()

##### Example 2 ------------------------------------------------------------
# graph of Petal.Length
plot_histogram(tbl = iris, col = "Petal.Length", out = "ggplot2")

}
```

plot_main_word *Draw bar plot of one (possibly grouped) open-text column in a tibble*

Description

This function draws a bar plot of the values of open text column. This plot shows the x-th first most cited words in a column having open text values using tidytext library. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_main_word(
  tbl = "iris",
  col = "Species",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  max = 10,
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string specifying a column of interest
filter	A character string specifying the values to filter. (equivalent to 'values in') This determines which values should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
max	integer specifying the x-th first most cited words
out	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A bar plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ----------------------------------------------------------------
# cat output generated as a template when no argument provided
plot_main_word()

##### Example 2 ----------------------------------------------------------------
# words contained in Species
plot_main_word(tbl = "iris", col = "Species", out = "ggplot2")

}
```

`plot_pie` *Draw pie chart of one (possibly grouped) column in a tibble*

Description

This function draws a pie plot of the values of column. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_pie(
  tbl = "dplyr::storms",
  col = "status",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

`tbl`	A character string or tibble specifying the input tibble
`col`	A character string specifying a column of interest
`filter`	A character string specifying the values to filter. (equivalent to 'values in'). This determines which values should be retained. It can be applied to both grouped and ungrouped data.

negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
out	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A pie plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
plot_pie()

##### Example 2 ------------------------------------------------------------
# graph of status in storms
plot_pie(tbl = "dplyr::storms", col = "status", out = "ggplot2")

}
```

`plot_pie_valid_value` *Draw pie chart of one column in a tibble (valid and non-valid values)*

Description

This function draws a pie plot of the values of a column separating valid, non-valid and missing values. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose.

Usage

```
plot_pie_valid_value(
  tbl = "dplyr::storms",
  col = "status",
  filter = "c()",
  negate = FALSE,
  missing_values = "'other low'",
  out = "ggplot2-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string specifying a column of interest
filter	A character string specifying the values to filter. (equivalent to 'values in') This determines which values should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
out	parameter that specifies the output expected: can be either 'ggplot2', 'plotly','ggplot2-code', 'plotly-code','ggplot2-cat' or 'plotly-cat'. ggplot2 renders a static plot, plotly a dynamic plot, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A pie plot object

See Also

ggplot2::ggplot()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
plot_pie_valid_value()

##### Example 2 ------------------------------------------------------------
# graph of Species (virginica is as missing values)
plot_pie_valid_value(
  tbl = "dplyr::storms",
```

```
  col = "status",
  missing_values = "'other low'" ,
  out = "ggplot2")

}
```

`read_csv_any_formats` *Read a csv file using read_csv and avoid errors*

Description

The csv file is read twice to detect the number of lines to use in attributing the column type ('guess_max' parameter of read_csv). This avoids common errors when reading csv files.

Usage

```
read_csv_any_formats(filename)
```

Arguments

`filename`	A character string of the path of the csv file.

Value

A tibble corresponding to the csv read.

Examples

```
{

try(read_csv_any_formats(filename = tempfile()),silent = TRUE)

}
```

`read_excel_allsheets` *Read all Excel sheets using* `readxl::read_excel()` *recursively*

Description

The Excel file is read and the values are placed in a list of tibbles, with each sheet in a separate element in the list. If the Excel file has only one sheet, the output is a single tibble.

Usage

```
read_excel_allsheets(filename, sheets = "")
```

Arguments

filename	A character string of the path of the Excel file.
sheets	A vector containing only the sheets to be read.

Value

A list of tibbles corresponding to the sheets read, or a single tibble if the number of sheets is one.

See Also

readxl::read_excel()

Examples

```
{

try(read_excel_allsheets(filename = tempfile()), silent = TRUE)

}
```

---

silently_run *Shortcut to silently run a code chunk avoiding error, messages and warnings*

Description

Shortcut avoiding user to get messages, warnings and being stopped by an error. The usage is very similar to base::suppressWarnings(). This function is targeted for function creators where user experience enhancement is sought.

Usage

```
silently_run(...)
```

Arguments

...	R code

Value

The output of the R code, unless the output is a message, a warning or an error, nothing will be returned in that case.

See Also

base::invisible(), base::suppressWarnings(), base::suppressMessages()

Examples

```
{

as.integer("text")
silently_run(as.integer("text"))

}
```

---

summary_category *Create summary tibble of one (possibly grouped) category-type column*

Description

This function creates datatable of the values of a column separating valid, non-valid and missing values. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose. The user can download the datatable in csv format.

Usage

```
summary_category(
  tbl = "iris",
  col = "col",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  out = "DT-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string of a column of interest
filter	A character string to subset the rows, applying the expressions to the column values to determine which rows should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. Those values will not be excluded from counting, but will be separated from valid values.
out	parameter that specifies the output expected: can be either 'DT', 'DT-code' and 'DT-cat'. DT renders a datatable using DT library, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.

group_by A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A datatable (editable) object or a R script in a character string to create it.

See Also

DT::datatable()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
summary_category()

##### Example 2 ------------------------------------------------------------
# summary table of Petal.Length
summary_category(tbl = iris, col = "Species", out = "DT")

}
```

---

summary_numerical *Create summary tibble of a (possibly grouped) numerical-type column*

Description

This function creates datatable of the values of a column separating valid, non-valid and missing values. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose. The user can download the datatable in csv format.

Usage

```
summary_numerical(
  tbl = "iris",
  col = "col",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  out = "DT-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble
col	A character string of a column of interest
filter	A character string to subset the rows, applying the expressions to the column values to determine which rows should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. Those values will not be excluded from counting, but will be separated from valid values.
out	parameter that specifies the output expected: can be either 'DT', 'DT-code' and 'DT-cat'. DT renders a datatable using DT library, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A datatable (editable) object or a R script in a character string to create it.

See Also

DT::datatable()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
summary_numerical()

##### Example 2 ------------------------------------------------------------
# summary table of Petal.Length
summary_numerical(tbl = iris, col = "Petal.Length", out = "DT")

}
```

---

summary_text *Create summary table of one (possibly grouped) text-type column in a tibble*

Description

This function creates a datatable of the values of a column with separate valid, non-valid and missing values. Missing values can be given as input to non-valid and valid values separately, or grouped by another column. The output can be editable (using plotly library) or static (using ggplot2 library). The R-code is also editable for coding recycling purpose. The user can download the datatable in csv format.

Usage

```
summary_text(
  tbl = "dplyr::storms",
  col = "status",
  filter = "c()",
  negate = FALSE,
  missing_values = "c()",
  out = "DT-cat",
  group_by = NULL
)
```

Arguments

tbl	A character string or tibble specifying the input tibble
col	A character string specifying a column of interest
filter	A character string specifying the values to filter. (equivalent to 'values in'). This determines which values should be retained. It can be applied to both grouped and ungrouped data.
negate	If TRUE, return non-matching elements.
missing_values	Vector listing values to exclude from valid values. These values will not be excluded from counting - but will be displayed separately from valid values.
out	parameter that specifies the output expected: can be either 'DT', 'DT-code' and 'DT-cat'. DT renders a datatable using DT library, code gives the code in a string (usable directly with eval/parse functions) and cat provides indented code in the console.
group_by	A character string of one column in the tbl that can be taken as a grouping column. The visual element will be grouped and displayed by this column.

Value

A datatable (editable) object or a R script in a character string to create it.

See Also

DT::datatable()

Examples

```
{

##### Example 1 ------------------------------------------------------------
# cat output generated as a template when no argument provided
summary_text()

##### Example 2 ------------------------------------------------------------
# summary table of Species
summary_text(tbl = "dplyr::storms", col = "status", out = "DT")

}
```

`template_visual_report` *Create a bookdown template for the visual report*

Description

This helper function creates a template for the visual report bookdown. This template is taken from the following link: https://github.com/jtr13/bookdown-template/archive/refs/heads/master.zip folder

Usage

```
template_visual_report(to)
```

Arguments

`to`	A character string of a path where the bookdown report will be placed

Value

A folder containing all files (Rmd, yml, docs, ...) to generate bookdown report

Examples

```
{

unlink(template_visual_report(tempdir()))

}
```

`which_any_date` *Evaluates and gives the possible format(s) for an object to be evaluated*

Description

This function takes a character string or a vector. This vector is evaluates one observation after the other, and gives the best matching date format for each of them (independently). The best matching format is tested across seven different formats provided by the lubridate library. The information of the best matching format can be used to mutate a column using `as_any_date()`.

Usage

```
which_any_date(
  x,
  format = c("dmy", "dym", "ymd", "ydm", "mdy", "myd", "as_date")
)
```

Arguments

`x`	object to be coerced. Can be a character string or a vector.
`format`	A character identifying the format to apply to the object to test. That format can be 'ymd','ydm','dym','dmy','mdy' or 'myd'.

Details

Contrary to lubridate library or `base::as.Date()`, the function evaluates the different possibilities for a date. For example, c('02-03-1982') can be either March the 2nd or February the 3rd. The function will provide "mdy, dmy" as possible formats. If no format is found, the function returns NA.

Value

A character string of the possible date formats given a parameter to be tested. The length of the vector is the length of the input object.

See Also

`lubridate::ymd()`,`lubridate::ydm()`,`lubridate::dmy()`, `lubridate::dym()`,`lubridate::mdy()`,`lubridate::myd()`, `lubridate::as_date()`, `guess_date_format()`,`as_any_date()`

Examples

```
{

time <- c(
  "1983-07-19",
  "31 jan 2017",
```

```
  "1988/12/17",
  "31-02-2005",
  "02-02-02",
  "2017 october the 2nd",
  "02-07-2012",
  "19-19-1923")

which_any_date(time)

}
```

`write_excel_allsheets` *Write all Excel sheets using* `xlsx::write.xlsx()` *recursively*

Description

The R objects are read and the values are placed in separated sheets. This function is inspired by the function proposed in https://statmethods.wordpress.com/2014/06/19/quickly-export-multiple-r-objects-to-an-excel-workbook/

Usage

```
write_excel_allsheets(list, filename)
```

Arguments

`list`	R objects, coma separated.
`filename`	A character string of the path of the Excel file.

Value

Nothing to be returned. The file is created at the path declared in the environment.

See Also

`xlsx::write.xlsx()`

Examples

```
{

unlink(
  write_excel_allsheets(
    list = list(iris = iris, mtcars = mtcars),
    filename = tempfile()))

}
```

Index

add_index, 3
as_any_boolean, 4
as_any_date, 5
as_any_date(), *17*, *40*
as_any_symbol, 6

base::as.Date(), *5*, *17*, *40*
base::as.logical(), *4*
base::eval(), *20, 21*
base::invisible(), *34*
base::parse(), *20, 21*
base::paste0(), *20*
base::readLines(), *10*
base::suppressMessages(), *34*
base::suppressWarnings(), *34*

collect_roxygen, 7

dplyr::mutate(), *20*
DT::datatable(), *36–38*

fabR_help, 8
file_index_create, 8
file_index_read, 9
file_index_search, 10

get_all_na_cols, 12
get_all_na_rows, 12
get_duplicated_cols, 13
get_duplicated_rows, 14
get_path_list, 15
get_unique_value_cols, 16
ggplot2::ggplot(), *23–25*, *27, 28*, *30–32*
guess_date_format, 17
guess_date_format(), *5*, *40*

haven::read_dta(), *10*
haven::read_sas(), *10*
haven::read_spss(), *10*

lubridate::as_date(), *5*, *17*, *40*
lubridate::dmy(), *5*, *17*, *40*
lubridate::dym(), *5*, *17*, *40*
lubridate::mdy(), *5*, *17*, *40*
lubridate::myd(), *5*, *17*, *40*
lubridate::ydm(), *5*, *17*, *40*
lubridate::ymd(), *5*, *17*, *40*

make_name_list, 18
message_on_prompt, 20

parceval, 20
parceval(), *15*
plot_bar, 22
plot_box, 23
plot_date, 24
plot_density, 26
plot_histogram, 27
plot_main_word, 29
plot_pie, 30
plot_pie_valid_value, 31

read_csv_any_formats, 33
read_csv_any_formats(), *10*
read_excel_allsheets, 33
read_excel_allsheets(), *10*
readxl::read_excel(), *33, 34*

silently_run, 34
stats::setNames(), *18, 19*
summary_category, 35
summary_numerical, 36
summary_text, 37

template_visual_report, 39

which_any_date, 40
which_any_date(), *5*, *17*
write_excel_allsheets, 41

xlsx::write.xlsx(), *41*