

# Package ‘fabR’

November 2, 2023

**Type** Package

**Title** Wrapper Functions Collection Used in Data Pipelines

**Version** 2.0.1

**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, utils, usethis, stringr, tidyr, purrr, janitor, fs, readr, readxl, writexl, haven, lubridate, digest, bookdown, xfun, lifecycle

**Suggests** knitr, stats, Matrix

**URL** <https://github.com/GuiFabre/fabR>

**BugReports** <https://github.com/GuiFabre/fabR/issues>

**RoxygenNote** 7.2.3

**Encoding** UTF-8

**VignetteBuilder** knitr

**Language** en-US

**NeedsCompilation** no

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

**Maintainer** Guillaume Fabre <gui.joseph.fabre@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-11-02 22:30:05 UTC

**R topics documented:**

add_index . . . . .	2
as_any_boolean . . . . .	3
as_any_date . . . . .	4
as_any_symbol . . . . .	5
bookdown_open . . . . .	6
bookdown_render . . . . .	7
bookdown_template . . . . .	8
collect_roxygen . . . . .	8
fabR_help . . . . .	9
file_index_create . . . . .	10
file_index_read . . . . .	11
file_index_search . . . . .	12
get_all_na_cols . . . . .	13
get_all_na_rows . . . . .	14
get_duplicated_cols . . . . .	15
get_duplicated_rows . . . . .	16
get_path_list . . . . .	17
get_unique_value_cols . . . . .	18
guess_date_format . . . . .	18
make_name_list . . . . .	20
message_on_prompt . . . . .	21
parceval . . . . .	22
read_csv_any_formats . . . . .	23
read_excel_allsheets . . . . .	24
silently_run . . . . .	25
which_any_date . . . . .	26
write_excel_allsheets . . . . .	27
<b>Index</b>	<b>28</b>

---

add_index	<i>Add an index column at the first place of a tibble</i>
-----------	---

---

**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)
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 `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**

[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 evaluated 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 `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\(\)](#), [as.Date\(\)](#), [guess\\_date\\_format\(\)](#), [which\\_any\\_date\(\)](#)

**Examples**

```
{
  library(dplyr)
  library(tidyr)

  ##### Example 1 -----
  # Ambiguous dates -----
  as_any_date('19 02 12')
  as_any_date('19 02 12', format = "ymd")
  as_any_date('19 02 12', format = "dym")

  ##### Example 2 -----
  # Non-ambiguous dates -----
  time <-
  tibble(time = c(
    "1983 07-19",
    "14-01-1925",
    "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")  
  
}
```

---

bookdown\_open

*Open a bookdown site in a browser*

---

**Description**

Opens a previously generated HTML bookdown site This is a shortcut function to access 'index.html'.

**Usage**

```
bookdown_open(bookdown_path)
```

**Arguments**

bookdown\_path    A character string specifying the path of the bookdown site to be opened.

**Value**

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

**See Also**

[bookdown\\_template\(\)](#), [bookdown\\_open\(\)](#)

**Examples**

```
{  
  
  bookdown_path = tempdir()  
  bookdown_template(bookdown_path, overwrite = TRUE)  
  bookdown_render(bookdown_path, overwrite = TRUE)  
  bookdown_open(bookdown_path)  
  
}
```

---

bookdown_render	<i>Render a bookdown into a bookdown site</i>
-----------------	---

---

**Description**

This helper function renders an existing bookdown folder (containing at least 'index.Rmd file)

**Usage**

```
bookdown_render(bookdown_path, overwrite = FALSE)
```

**Arguments**

`bookdown_path` A character string identifying the folder path where the bookdown report files are.

`overwrite` whether to overwrite existing files. FALSE by default.

**Value**

A folder containing html files (in docs, ...) generated from a bookdown report.

**See Also**

[bookdown\\_template\(\)](#), [bookdown\\_open\(\)](#)

**Examples**

```
{  
  
  bookdown_path = tempdir()  
  bookdown_template(bookdown_path, overwrite = TRUE)  
  bookdown_render(bookdown_path, overwrite = TRUE)  
  
}
```

---

bookdown_template	<i>Create a bookdown template.</i>
-------------------	------------------------------------

---

**Description**

This helper function creates a template for a bookdown.

**Usage**

```
bookdown_template(bookdown_path, overwrite = FALSE)
```

**Arguments**

bookdown_path	A character string identifying the folder path where the bookdown will be generated.
overwrite	whether to overwrite existing files. FALSE by default.

**Value**

A folder containing all files (Rmd, yml, css) to generate the bookdown.

**See Also**

[bookdown\\_render\(\)](#), [bookdown\\_open\(\)](#)

**Examples**

```
{  
  
  bookdown_path = tempdir()  
  bookdown_template(bookdown_path, overwrite = TRUE)  
  
}
```

---

collect_roxygen	<i>Collects and Generates documentation of a package in a tibble format.</i>
-----------------	--

---

**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	<i>Read, source and open objects from an index of files</i>
-----------------	---

---

### 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 <code>file_index_create()</code> 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. <code>parent.frame()</code> 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 `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\(\)](#), [readLines\(\)](#)

**Examples**

```
## Not run:  
  
index <- file_index_create(tempdir())  
file_index_read(index, file_name = my_file_name)  
  
## End(Not run)
```

---

file_index_search	<i>Search an index of files</i>
-------------------	---------------------------------

---

**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 <code>file_index_create()</code> 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 observations (rows) that have all NA values in a tibble*


---

**Description**

This helper function extracts the row number(s) having NA value for all columns.

**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 observation or the row number of the empty observations.

**Examples**

```
{
  ##### Example 1 -----
  # All rows have observation
  get_all_na_rows(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(
  tbl = bind_rows(iris, tibble(Species = c('id_151', 'id_152'))),
  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)
  tbl <-
    mtcars %>%
    mutate(
      cyl_2 = cyl,
      cyl_3 = cyl,
      mpg_2 = mpg)

  get_duplicated_cols(tbl)
}
```

---

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(tbl = bind_rows( tbl = 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	<i>Get the paths of branches in a list</i>
---------------	--

---

## 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 have unique values in a tibble*

---

### Description

This helper function extracts the names of the columns in a tibble having unique value 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(tbl = 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 `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()`, `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(
    "2023-12-12",
    "2009-09-03",
    "1809-01-01"))
guess_date_format(time)
```

```
time <-
  tibble(time = c(
    "1983-19-07",
    "1983-10-13",
    "2009-09-03",
    "1509-11-30"))
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

<code>args_list</code>	A list of character string of same length of <code>list_elem</code>
<code>list_elem</code>	A list of character string of same length of <code>args_list</code>

## Value

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

**See Also**

[stats::setNames\(\)](#)

**Examples**

```
{
  library(tidyr)
  library(stats)

  #### 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) %>%
        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 acceptance 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 `parse()` and `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 `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**

`parse()`, `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)")

##### 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 parcevald itself!

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

cat(code_R)
parceval(code_R)

}

```

---

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.

**See Also**

[readr::read\\_csv\(\)](#), [readr::read\\_delim\(\)](#)

**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 = "", keep_as_list = FALSE)
```

**Arguments**

filename      A character string of the path of the Excel file.  
sheets        A vector containing only the sheets to be read.  
keep\_as\_list   A Boolean to say whether the object should be a list or a tibble, when there is only one sheet provided. FALSE by default.

**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	<i>Shortcut to silently run a code chunk avoiding error, messages and warnings</i>
--------------	--

---

**Description**

Shortcut avoiding user to get messages, warnings and being stopped by an error. The usage is very similar to [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**

[invisible\(\)](#), [suppressWarnings\(\)](#), [suppressMessages\(\)](#)

**Examples**

```
{  
  as.integer("text")  
  silently_run(as.integer("text"))  
}
```

---

which_any_date	<i>Evaluates and gives the possible format(s) for an object to be evaluated</i>
----------------	---

---

### Description

This function takes a character string or a vector. This vector is evaluated 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 `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()`, `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 `writexl::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

[writexl::write\\_xlsx\(\)](#)

### Examples

```
{

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

}
```

# Index

add\_index, 2  
as.Date(), 5, 19, 26  
as.logical(), 3, 4  
as\_any\_boolean, 3  
as\_any\_date, 4  
as\_any\_date(), 18, 19, 26  
as\_any\_symbol, 5  
  
bookdown\_open, 6  
bookdown\_open(), 6–8  
bookdown\_render, 7  
bookdown\_render(), 8  
bookdown\_template, 8  
bookdown\_template(), 6, 7  
  
collect\_roxygen, 8  
  
dplyr::mutate(), 22  
  
eval(), 22  
  
fabR\_help, 9  
file\_index\_create, 10  
file\_index\_read, 11  
file\_index\_search, 12  
  
get\_all\_na\_cols, 13  
get\_all\_na\_rows, 14  
get\_duplicated\_cols, 15  
get\_duplicated\_rows, 16  
get\_path\_list, 17  
get\_unique\_value\_cols, 18  
guess\_date\_format, 18  
guess\_date\_format(), 4, 5, 26  
  
haven::read\_dta(), 11, 12  
haven::read\_sas(), 11, 12  
haven::read\_spss(), 11, 12  
  
invisible(), 25  
  
lubridate::as\_date(), 5, 19, 26  
  
lubridate::dmy(), 5, 19, 26  
lubridate::dym(), 5, 19, 26  
lubridate::mdy(), 5, 19, 26  
lubridate::myd(), 5, 19, 26  
lubridate::ydm(), 5, 19, 26  
lubridate::ymd(), 5, 19, 26  
  
make\_name\_list, 20  
message\_on\_prompt, 21  
  
parceval, 22  
parceval(), 17  
parse(), 22  
paste0(), 22  
  
read\_csv\_any\_formats, 23  
read\_csv\_any\_formats(), 11, 12  
read\_excel\_allsheets, 24  
read\_excel\_allsheets(), 11, 12  
readLines(), 11, 12  
readr::read\_csv(), 24  
readr::read\_delim(), 24  
readxl::read\_excel(), 24, 25  
  
silently\_run, 25  
stats::setNames(), 20, 21  
suppressMessages(), 25  
suppressWarnings(), 25  
  
which\_any\_date, 26  
which\_any\_date(), 4, 5, 19  
write\_excel\_allsheets, 27  
writexl::write\_xlsx(), 27