Package ‘fabR’

April 19, 2024

Type Package

Title Wrapper Functions Collection Used in Data Pipelines

Version 2.1.0

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 (>= 1.1.0), rlang, utils, usethis, stringr, tidyr, purrr, janitor, fs, readr, readxl, writexl, haven, lubridate, 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 <guijoseph.fabre@gmail.com>

Repository CRAN

Date/Publication 2024-04-19 07:32:37 UTC
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

```r
{  
  ##### 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

```r
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')))    # this will fail
  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","my","ym","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','mdy','myd','my','ym','as_date'.
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::mdy()`, `lubridate::myd()`, `lubridate::dym()`, `lubridate::my()` , `lubridate::ym()`, `lubridate::as_date()`, `as.Date()`, `guess_date_format()`, `which_any_date()`

Examples

```r
{
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 <-
time %>% mutate(new_time = as_any_date(time))
}
```
as_any_integer

Create objects of type "integer".

Description
Create or test for objects of type "integer". This function is a wrapper of the function \texttt{as.integer()} and evaluates if the object to be coerced can be interpreted as an integer. Any object: \texttt{NA}, \texttt{NA}\_integer, \texttt{NA}\_Date\_, (...) (...), \texttt{Boolean}, such as \texttt{0}, \texttt{0L}, \texttt{F}, \texttt{FALSE}, \texttt{false}, \texttt{FaLsE}, (...) (...). Any string "1", "+1", "-1", "1.0000" will be converted as \texttt{NA} or integer. Any other other will return an error.

Usage
\texttt{as_any_integer(x)}

Arguments
\texttt{x}
Object to be coerced or tested. Can be a vector.

Value
An integer object of the same size.

See Also
\texttt{as.logical()}

Examples
\{
library(dplyr)

as_any_integer("1")
as_any_integer(c("1.000","2.0","1","+12","-12"))
try(as_any_integer('foo'))
tibble(values = c("1.000","2.0","1","+12","-12")) %>%
  mutate(bool_values = as_any_integer(values))
\}
as\_any\_symbol  

Create objects of type "symbol"

Description

Create or test for objects of type "symbol".

Usage

as\_any\_symbol(x)

Arguments

\textit{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 web-based bookdown folder in a browser

Description

Opens a previously generated HTML bookdown site from files in the specified folder. This is a shortcut function to access 'index.html' in the specified folder.

Usage

bookdown\_open(bookdown\_path)

Arguments

\textit{bookdown\_path}  
A character string identifying the folder path containing the files to open the bookdown site.
null
`bookdown_template`  

**Examples**  

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

---

**bookdown_template** *Create a bookdown template.*

**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**  

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

```r
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**

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

---

**fabR_website**

*Call to online documentation*

**Description**

Direct call to the online documentation for the package, which includes a description of the latest version of the package, vignettes, user guides, and a reference list of functions and help pages.

**Usage**

```r
fabR_website()
```
**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**

```r
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.
## Not run:

```r
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

```r
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 `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

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

## End(Not run)
```

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

```r
file_index_search(
  index,
  file_path = "*",
  file_name = "*",
  extension = "*",
  file_type = "*",
  show_tree = FALSE
)
```
get_all_na_cols

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

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

## End(Not run)
```

Description

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

Usage

```r
get_all_na_cols(tbl)
```
get_all_na_rows

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.
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_rows

get_duplicated_cols(tbl)

get_duplicated_rows(tbl)

get_duplicated_rows(tbl, id_col = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl</td>
<td>R object (dataframe or tibble) of the input tibble</td>
</tr>
<tr>
<td>id_col</td>
<td>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.</td>
</tr>
</tbody>
</table>

Value

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

Examples

```r
# 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**

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**

```r
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**

```r
{
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_date(). The default format is yyyy-mm-dd.
guess_date_format

Usage

```r
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::myd()`, `lubridate::dmy()`, `lubridate::my()`, `lubridate::ym()`, `lubridate::as_date()`, `as.Date()`, `which_any_date()`, `as_any_date()`

Examples

```r
{
  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(  
    "1983-07-19",  
    "1983-07-07",  
    "1983-10-13",  
    "2009-09-03",  
    "1509-11-30")
  )
}
```
make_name_list

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

```r
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 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`")
}

### 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

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

Arguments

```r
...
```

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 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)

}

---

read_csv_any_formats

*Read a csv file using read_csv and avoid errors*

**Description**

*Experimental* 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.
**read_excel_allsheets**

**Usage**

```r
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_csv2()`

**Examples**

```r
{
  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.

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**

```r
read_excel_allsheets(filename, sheets = "", keep_as_list = FALSE)
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)
}

Description

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

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.
which_any_date

See Also

invisible(), suppressWarnings(), suppressMessages()

Examples

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

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 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("ymd", "ydm", "dmy", "myd", "mdy", "dym", "ym", "ym”, "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','dmy','myd','mdy','dym', 'ym', 'my' or 'as_date' in that specific order ("ymd" will be chose as a default format, then 'ymd', etc.).

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::mdy(), lubridate::dym(), lubridate::my(), lubridate::ym(), lubridate::as_date(), as.Date(), guess_date_format(), as_any_date()

Examples

```
{  
  time <- c(
    "1983-07-19",
    "31 Jan 2017",
    "1988/12/17",
    "31-02-05",
    "02-02-02",
    "2017 October the 2nd",
    "02-07-2012",
    "19-07-83",
    "19-19-19")

  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)
write_excel_allsheets(list, filename)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>R objects, coma separated.</td>
</tr>
<tr>
<td>filename</td>
<td>A character string of the path of the Excel file.</td>
</tr>
</tbody>
</table>
Value

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

{ unlink(
    write_excel_allsheets(
        list = list(iris = iris, mtcars = mtcars),
        filename = tempfile()))
}
Index

add_index, 2
as.Date(), 5, 20, 27, 28
as.integer(), 6
as.logical(), 3, 4, 6
as_any_boolean, 3
as_any_date, 4
as_any_date(), 19, 20, 27, 28
as_any_integer, 6
as_any_symbol, 7

bookdown_open, 7
bookdown_open(), 8, 9
bookdown_render, 8
bookdown_render(), 9
bookdown_template, 9
bookdown_template(), 8

collect_roxygen, 10

dplyr::mutate(), 23
eval(), 23

fabR_website, 10
file_index_create, 11
file_index_read, 12
file_index_search, 13

get_all_na_cols, 14
get_all_na_rows, 15
get_duplicated_cols, 16
get_duplicated_rows, 17
get_path_list, 18
get_unique_value_cols, 19
guess_date_format, 19
guess_date_format(), 4, 5, 28

haven::read_dta(), 13
haven::read_sas(), 13
haven::read_spss(), 13

invisible(), 27
lubridate::as_date(), 5, 20, 28
lubridate::dmy(), 5, 20, 28
lubridate::dym(), 5, 20, 28
lubridate::mdy(), 5, 20, 28
lubridate::my(), 5, 20, 28
lubridate::myd(), 5, 20, 28
lubridate::ydm(), 5, 20, 28
lubridate::ym(), 5, 20, 28
lubridate::ymd(), 5, 20, 28

make_name_list, 21
message_on_prompt, 22
parceval, 23
parceval(), 18
parse(), 23
paste0(), 23

read_csv_any_formats, 24
read_csv_any_formats(), 13
read_excel_allsheets, 25
read_excel_allsheets(), 13
readLines(), 13
readr::read_csv(), 25
readr::read_csv2(), 25
readxl::read_excel(), 25, 26

silently_run, 26
stats::setNames(), 21
suppressMessages(), 27
suppressWarnings(), 26, 27

which_any_date, 27
which_any_date(), 4, 5, 20
write_excel_allsheets, 28
writexl::write_xlsx(), 28, 29