Package: certetoolbox (via r-universe)

September 11, 2024

Title A Certe R Package for Miscellaneous Functions

Version 1.13.7

Description A Certe R Package for miscellaneous functions that do not fit a dedicated package. This package also mitigates the 'vctrs' package by allowing numeric-character coercions. This package is part of the 'certedata' universe.

URL https://certe-medical-epidemiology.github.io/certetoolbox, https://github.com/certe-medical-epidemiology/certetoolbox

Depends R (>= 4.1.0)

Imports certeprojects, certestyle, cleaner (>= 1.5.1), dplyr (>= 1.0.0), flextable (>= 0.6.8), glue (>= 1.6.1), hms (>= 1.1.0), knitr (>= 1.32), lubridate (>= 1.9.0), magrittr (>= 2.0.0), openxlsx2 (>= 1.9.0), progress (>= 1.2.0), readr (>= 2.0.0), rstudioapi (>= 0.10), stringr (>= 1.4.0), tibble (>= 3.0.0), tidyr (>= 1.2.0), vctrs (>= 0.3.8), yaml (>= 2.2.0)

Suggests certeplot2, certestats, certemail (>= 1.4.0), AMR (>= 2.0.0), arrow (>= 7.0.0), AzureGraph (>= 1.3.0), cardx (>= 0.2.0), cbsodataR (>= 0.5.1), clipr (>= 0.7.1), crayon (>= 1.4.2), ggplot2 (>= 3.1.0), gtsummary (>= 1.6.0), haven (>= 2.4.0), htmltools (> 0.5.0), plotly (>= 4.8.0), readxl (>= 1.3.0), rio (>= 0.5.27), showtext (>= 0.9.0), showtextdb (>= 3.0.0), testthat (>= 2.0.0)

License GPL-2

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/testthat/edition 2

Repository https://certe-medical-epidemiology.r-universe.dev

RemoteUrl https://github.com/certe-medical-epidemiology/certetoolbox

RemoteRef HEAD

RemoteSha 1d76fcfa63eb0ca5c56f7f4790ad92bc3eb1e45c

Contents

`as.UTC` *Force Time as UTC*

Description

Force Time as UTC

Usage

```
as.UTC(x, ...)

## S3 method for class 'data.frame'
as.UTC(x, ...)

## S3 method for class 'POSIXct'
as.UTC(x, ...)

## Default S3 method:
as.UTC(x, ...)
```

Arguments

x	a vector of datetime values
...	not used at the moment

Examples

```
Sys.time()
as.UTC(Sys.time())
```

---

as_excel *Create Excel Workbook Object*

Description

The as_excel() function relies on the openxlsx2 package for creating an Excel Workbook object in R. These objects can be saved using save_excel() or export_xlsx().

Usage

```
as_excel(
  ...,
  sheet_names = NULL,
  autofilter = TRUE,
  autowidth = TRUE,
  widths = NULL,
  rows_zebra = TRUE,
  cols_zebra = FALSE,
  freeze_top_row = TRUE,
  digits = 2,
  align = "center",
  table_style = "TableStyleMedium2",
  creator = Sys.info()["user"],
  department = read_secret("department.name"),
  project_number = project_get_current_id(ask = FALSE)
)

save_excel(xl, filename = NULL, overwrite = FALSE)
```

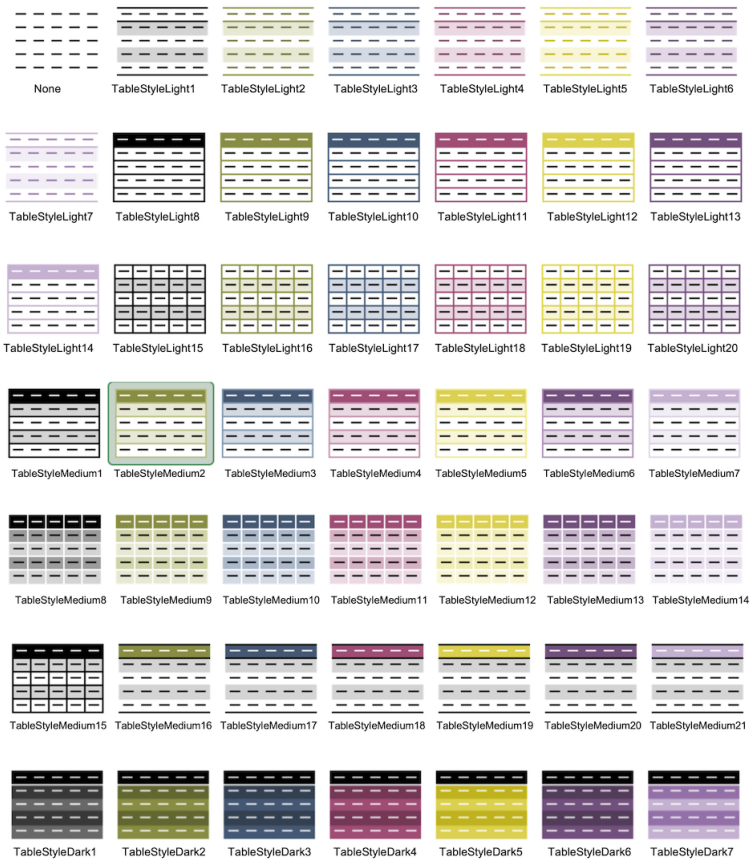
Arguments

...	data sets, use named items for multiple tabs (see *Examples*)
sheet_names	sheet names
autofilter	create autofilter on columns in first row. This can also be a vector with the same length as ....
autowidth	automatically adjust columns widths. This can also be a vector with the same length as ....

widths	width of columns, must be length 1 or ncol() of the data set. If set, overrides autowidth.
rows_zebra	create banded rows. This can also be a vector with the same length as ....
cols_zebra	create banded columns. This can also be a vector with the same length as ....
freeze_top_row	freeze the first row of the sheet. This can also be a vector with the same length as ....
digits	number of digits for numeric values (integer values will always be rounded to whole numbers), defaults to 2
align	horizontal alignment of text
table_style	style(s) for each table, see below. This can also be a vector with the same length as ....
creator	name of the creator of the workbook
department	name of the department of the workbook
project_number	project number, to add project ID as the subject of the workbook
xl	Excel object, as created with as_excel() (or manually with the openxlsx2 package)
filename	file location to save Excel document to, defaults to a random filename in the current folder
overwrite	overwrite existing file

Supported Table Styles

For the argument table_style, use one or more of these table styles as character input. The default is **TableStyleMedium2**.



Examples

```
# creates a Workbook object
xl <- as_excel("this is a sheet" = mtcars,
               "another sheet" = anscombe)
xl

# then save it with save_excel() or export_xlsx()
```

`auto_transform` *Automatically Transform Data Set*

Description

This function transforms a data.frame by guessing the right data classes and applying them, using `readr::parse_guess()` and `cleaner` functions such as `cleaner::clean_Date()`.

Usage

```
auto_transform(
  x,
  datenames = "en",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ".",
  big.mark = "",
  timezone = "",
  na = c("", "NULL", "NA", "<NA>"),
  snake_case = FALSE,
  ...
)
```

Arguments

x	a data.frame
datenames	language of the date names, such as weekdays and months
dateformat	expected date format, will be coerced with format_datetime()
timeformat	expected time format, will be coerced with format_datetime()
decimal.mark	separator for decimal numbers
big.mark	separator for thousands
timezone	expected time zone
na	values to interpret as NA
snake_case	apply snake case to the column names
...	not used as the time, allows for future extension

---

cbs_topics *Download CBS-data*

Description

Download data from CBS Open data Statline.

Usage

```
cbs_topics()

cbs_search(topic, max_print = 25)

cbs_download(identifier, clean_cols = TRUE)

cbs_moreinfo(identifier)
```

Arguments

topic	topics to search for
max_print	maximum number of subjects to print
identifier	tracking number (1 to max_print) in cbs_search()
clean_cols	Clean column names.

Details

cbs_topics() retrieves all topics.

cbs_search() searches for a specific subject.

cbs_download() downloads tables. Input has to be a CBS Identifier (printed in red in cbs_search()), or a tracking number of cbs_search(), or the result of cbs_search().

cbs_moreinfo() gives a detailed explanation for the table. Input can also be a dataset downloaded with cbs_download().

Examples

```
## Not run:
cbs_search("Inwoners")

x <- cbs_download(2) # 2nd hit of cbs_search()
str(x)

cbs_moreinfo(x)
cbs_moreinfo(2)

## End(Not run)
```

---

concat *Paste Character Vectors Together*

Description

The concat() function is at default identical to paste(c(...), sep = "", collapse = "").

The collapse() function is at default identical to paste(x, sep = "", collapse = "").

Usage

```
concat(..., sep = "")

collapse(x, sep = "")
```

Arguments

..., x	element(s) to be pasted together, can also be vectors
sep	separator character, will also be used for collapsing

Examples

```
concat("a", "b", "c")

concat(c("a", "b"), "c")
collapse(c("a", "b"), "c")

concat(letters[1:5], "-")
collapse(letters[1:5], "-")
```

`crosstab` *Create a Crosstab*

Description

Transform a data set into an *n* x *m* table, e.g. to be used in `certestats::confusion_matrix()`.

Usage

```
crosstab(
  df,
  identifier,
  compare,
  outcome,
  positive = "^pos.*",
  negative = "^neg.*",
  ...,
  na.rm = TRUE,
  ignore_case = TRUE
)
```

Arguments

`df`	a data.frame
`identifier`	a column name to use as identifier, such as a patient ID or an order ID
`compare`	a column name for the two axes of the table: the labels between the outcomes must be compared
`outcome`	a column name containing the outcome values to compare
`positive`	a regex to match the values in `outcome` that must be considered as the Positive class, use `FALSE` to not use a Positive class
`negative`	a regex to match the values in `outcome` that must be considered as the Negative class, use `FALSE` to not use a Negative class
`...`	manual regexes for classes if not using `positive` and `negative`, such as `Class1 = "c1", Class2 = "c2"`
`na.rm`	a logical to indicate whether empty values must be removed before forming the table
`ignore_case`	a logical to indicate whether the case in the values of `positive`, `negative` and `...` must be ignored

Examples

```
df <- data.frame(
  order_nr = sort(rep(LETTERS[1:20], 2)),
  test_type = rep(c("Culture", "PCR"), 20),
  result = sample(c("pos", "neg"),
                  size = 40,
                  replace = TRUE,
                  prob = c(0.3, 0.9))
)
head(df)

out <- df |> crosstab(order_nr, test_type, result)
out


df$result <- gsub("pos", "#p", df$result)
df$result <- gsub("neg", "#n", df$result)
head(df)
# gives a warning that pattern matching failed:
df |> crosstab(order_nr, test_type, result)

# define the pattern yourself in such case:
df |> crosstab(order_nr, test_type, result,
               positive = "#p",
               negative = "#n")


# defining classes manually, can be more than 2:
df |> crosstab(order_nr, test_type, result,
               ClassA = "#p", Hello = "#n")

if ("certestats" %in% rownames(utils::installed.packages())) {
  certestats::confusion_matrix(out)
}
```

---

days_around_today *Dates around Today*

Description

These are convenience functions to get certain dates relatively to today.

Usage

```
yesterday(ref = today())

tomorrow(ref = today())

week(ref = today())
```

```
year(ref = today())

last_week(ref = today(), only_start_end = FALSE)

this_week(ref = today(), only_start_end = FALSE)

next_week(ref = today(), only_start_end = FALSE)

last_month(ref = today(), only_start_end = FALSE)

this_month(ref = today(), only_start_end = FALSE)

next_month(ref = today(), only_start_end = FALSE)

last_quarter(ref = today(), only_start_end = FALSE)

this_quarter(ref = today(), only_start_end = FALSE)

next_quarter(ref = today(), only_start_end = FALSE)

last_year(ref = today(), only_start_end = FALSE)

this_year(ref = today(), only_start_end = FALSE)

next_year(ref = today(), only_start_end = FALSE)

last_n_years(n, ref = end_of_last_year(), only_start_end = FALSE)

last_5_years(ref = end_of_last_year(), only_start_end = FALSE)

last_10_years(ref = end_of_last_year(), only_start_end = FALSE)

start_of_last_week(ref = today(), day = 1)

end_of_last_week(ref = today(), day = 7)

start_of_this_week(ref = today(), day = 1)

end_of_this_week(ref = today(), day = 7)

start_of_last_month(ref = today())

end_of_last_month(ref = today())

start_of_this_month(ref = today())

end_of_this_month(ref = today())
```

```
start_of_next_month(ref = today())

end_of_next_month(ref = today())

start_of_last_quarter(ref = today())

end_of_last_quarter(ref = today())

start_of_this_quarter(ref = today())

end_of_this_quarter(ref = today())

start_of_next_quarter(ref = today())

end_of_next_quarter(ref = today())

start_of_last_year(ref = today())

end_of_last_year(ref = today())

start_of_this_year(ref = today())

end_of_this_year(ref = today())

start_of_next_year(ref = today())

end_of_next_year(ref = today())

nth_monday(ref = today(), n = 1)

nth_tuesday(ref = today(), n = 1)

nth_wednesday(ref = today(), n = 1)

nth_thursday(ref = today(), n = 1)

nth_friday(ref = today(), n = 1)

nth_saturday(ref = today(), n = 1)

nth_sunday(ref = today(), n = 1)

week2date(wk, yr = year(today()), day = 1)

week2resp_season(wk, remove_outside_season = FALSE)
```

Arguments

ref	reference date (defaults to today)
only_start_end	logical to indicate whether only the first and last value of the resulting vector should be returned
n	relative number of weeks
day	day to return (0 are 7 are Sunday, 1 is Monday, etc.)
wk	week to search for
yr	year to search for, defaults to current year
remove_outside_season	a logical to remove week numbers in the range 21-39

Details

All functions return a vector of dates, except for yesterday(), today(), tomorrow(), week2date(), and the start_of_*(), end_of_*() and nth_*() functions; these return 1 date.

Week ranges always start on Mondays and end on Sundays.

year() always returns an integer.

The last_n_years(), last_5_years() and last_10_years() functions have their reference date set to end_of_last_year() at default.

week2resp_season() transforms week numbers to an ordered factor, in a range 40-53, 1:39 (or, if remove_outside_season = TRUE, 40-53, 1:20). This function is useful for plotting.

Examples

```
today()
today() %in% this_month()

next_week()
next_week(only_start_end = TRUE)

# 2nd Monday of last month:
last_month() |> nth_monday(2)

# last_*_years() will have 1 Jan to 31 Dec at default:
last_5_years(only_start_end = TRUE)
last_5_years(today(), only_start_end = TRUE)
## Not run:

  # great for certedb functions:
  certedb::get_diver_data(last_5_years(),
                          Bepaling == "ACBDE")

## End(Not run)

df <- data.frame(date = sample(seq.Date(start_of_last_year(),
                                        end_of_this_year(),
                                        by = "day"),
```

```
                                size = 500))
df$time <- as.POSIXct(paste(df$date, "12:00:00"))

library(dplyr, warn.conflicts = FALSE)

# these are equal:
df |>
  filter(date |> between(start_of_last_week(),
                         end_of_last_week()))
df |>
  filter(date %in% last_week())

# but this does not work:
df |>
  filter(time %in% last_week())

# so be sure to transform times to dates in certain filters
df |>
  filter(as.Date(time) %in% last_week())
```

`export` *Export Data Sets and Plots*

Description

These functions can be used to export data sets and plots. They invisibly return the object itself again, allowing for usage in pipes (except for the plot-exporting functions `export_pdf()`, `export_png()` and `export_html()`). The functions work closely together with the `certeprojects` package to support Microsoft Planner project numbers.

Usage

```
export(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  fn = NULL,
  ...
)

export_rds(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  ...
)
```

```
export_xlsx(
  ...,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  sheet_names = NULL,
  autofilter = TRUE,
  rows_zebra = TRUE,
  cols_zebra = FALSE,
  freeze_top_row = TRUE,
  table_style = "TableStyleMedium2",
  align = "center"
)

export_excel(
  ...,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  sheet_names = NULL,
  autofilter = TRUE,
  rows_zebra = TRUE,
  cols_zebra = FALSE,
  freeze_top_row = TRUE,
  table_style = "TableStyleMedium2",
  align = "center"
)

export_csv(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  na = "",
  ...
)

export_csv2(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  na = "",
  ...
)

export_tsv(
```

```
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  na = "",
  ...
)

export_txt(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  sep = "\t",
  na = "",
  ...
)

export_sav(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  ...
)

export_spss(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  ...
)

export_feather(
  object,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  ...
)

export_pdf(
  plot,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  size = "A5",
```

```
  portrait = FALSE,
  ...
)

export_png(
  plot,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  width = 1000,
  height = 800,
  dpi = NULL,
  ...
)

export_html(
  plot,
  filename = NULL,
  project_number = project_get_current_id(ask = FALSE),
  overwrite = NULL,
  ...
)

export_clipboard(
  object,
  sep = "\t",
  na = "",
  header = TRUE,
  quote = FALSE,
  decimal.mark = dec_mark(),
  ...
)

export_teams(
  object,
  filename = NULL,
  full_teams_path = NULL,
  account = connect_teams(),
  ...
)
```

Arguments

object, plot	the R object to export
filename	the full path of the exported file
project_number	a Microsoft Planner project number
overwrite	a logical value to indicate if an existing file must be overwritten. In interactive mode, this will be asked if the file exists. In non-interactive mode, this has a spe-

	cial default behaviour: the original file will be copied to `filename_datetime.ext` before overwriting the file. Exporting with existing files is always non-destructive: if exporting fails, the original, existing file will not be altered.
`fn`	a manual export function, such as `haven::write_sas` to write SAS files. This function has to have the object as first argument and the future file location as second argument.
`...`	arguments passed on to methods
`sheet_names`	sheet names
`autofilter`	create autofilter on columns in first row. This can also be a vector with the same length as `....`
`rows_zebra`	create banded rows. This can also be a vector with the same length as `....`
`cols_zebra`	create banded columns. This can also be a vector with the same length as `....`
`freeze_top_row`	freeze the first row of the sheet. This can also be a vector with the same length as `....`
`table_style`	style(s) for each table, see below. This can also be a vector with the same length as `....`
`align`	horizontal alignment of text
`na`	replacement character for empty values (default: `""`)
`sep`	separator for values in a row (default: tab)
`size`	paper size, defaults to A5. Can be A0 to A7.
`portrait`	portrait mode, defaults to `FALSE` (i.e., landscape mode)
`width`	required width of the PNG file in pixels
`height`	required height of the PNG file in pixels
`dpi`	plot resolution, defaults to DPI set in `showtext` package
`header`	(for `export_clipboard()`) use column names as header (default: `TRUE`)
`quote`	(for `export_clipboard()`) use quotation marks (default: `FALSE`)
`decimal.mark`	(for `export_clipboard()`) character to use for decimal numbers, defaults to `dec_mark()`
`full_teams_path`	path in Teams to export object to. Can be left blank to use interactive folder picking mode in the console.
`account`	a Teams account from Azure or an `AzureAuth` Microsoft 365 token, e.g. retrieved with `certeprojects::connect_teams()`

Details

The `export()` function can export to any file format, also with a manually set export function when passed on to the `fn` argument. This function `fn` has to have the object as first argument and the future file location as second argument. If `fn` is left blank, the `export_*` function will be used based on the filename.

RDS files as created using `export_rds()` are compatible with R3 and R4.

The export_xlsx() and export_excel() functions use save_excel(as_excel(...)) internally. **IMPORTANT**: these two functions can accept more than one data.frame. When naming the data sets, the names will become sheet names in the resulting Excel file. For a complete visual overview of supported table styles, see as_excel(). If the last value in ... is a character of length 1 and filename is NULL, this value is assumed to be the filename.

For export_csv(), export_csv2() and export_tsv(), files will be saved in UTF-8 encoding and NA values will be exported as "" at default. Like other *.csv and *.csv2 functions, csv is comma (,) separated and csv2 is semicolon (;) separated.

The export_txt() function exports to a tab-separated file.

Exporting to an SPSS file using export_sav() or export_spss() requires the haven package to be installed.

Exporting to a Feather file using export_feather() requires the arrow package to be installed. Apache Feather provides efficient binary columnar serialization for data sets, enabling easy sharing data across data analysis languages (such as between Python and R).

Exporting to a PDF file using export_pdf() requires the ggplot2 package to be installed. If the filename is left blank in export_pdf(), export_png() or export_html(), the title of plot will be used if it's available and the certeplot2 package is installed, and a timestamp otherwise. **NOTE:** All export functions invisibly return object again, but the plotting functions invisibly return the file path

Exporting to a PNG file using export_png() requires the ggplot2 and showtext packages to be installed.

Exporting to an HTML file using export_html() requires the ggplot2 and htmltools packages to be installed. The arguments put in ... will be passed on to plotly::layout() if plot is not yet a Plotly object (but rather a ggplot2 object), which of course then requires the plotly package to be installed as well.

Exporting to the clipboard using export_clipboard() requires the clipr package to be installed. The function allows any object (also other than data.frames) to be exported to the clipboard and is only limited to the available amount of RAM memory.

Exporting to Microsoft Teams using export_teams() requires the AzureGraph package to be installed. The function allows any object (also other than data.frames) to be exported to any Team channel. The filename set in filename will determine the exported file type and defaults to an RDS file.

See Also

import()

Examples

```
library(dplyr, warn.conflicts = FALSE)

# export to two files: 'whole_file.rds' and 'first_ten_rows.xlsx'
starwars |>
  export_rds("whole_file") |>
  slice(1:10) |>
  export_xlsx("first_ten_rows")
```

```
# the above is equal to:
# starwars |>
#   export("whole_file.rds") |>
#   slice(1:10) |>
#   export("first_ten_rows.xlsx")


# Apache's Feather format is column-based
# and allow for cross-language specific and fast file reading
starwars |> export_feather()
import("starwars.feather",
       col_select = starts_with("h")) |>
  head()


# (cleanup)
file.remove("whole_file.rds")
file.remove("first_ten_rows.xlsx")
file.remove("starwars.feather")

## Not run:

# ---- Microsoft Teams support -----------------------------------------

# IMPORTING

# import from Teams by picking a folder interactively from any Team
x <- import_teams()

# to NOT pick a Teams folder (e.g. in non-interactive mode), set `full_teams_path`
x <- import_teams(full_teams_path = "MyTeam/MyChannel/MyFolder/MyFile.xlsx")


# EXPORTING

# export to Teams by picking a folder interactively from any Team
mtcars |> export_teams()

# the default is RDS, but you can set `filename` to specify yourself
mtcars |> export_teams("mtcars.xlsx")

# to NOT pick a Teams folder (e.g. in non-interactive mode), set `full_teams_path`
mtcars |> export_teams("mtcars.xlsx", full_teams_path = "MyTeam/MyChannel/MyFolder")
mtcars |> export_teams(full_teams_path = "MyTeam/MyChannel/MyFolder")


## End(Not run)
```

---

generate_identifier *Create Random Identifier*

Description

This function creates unique identifier (IDs) using `sample()`.

Usage

```
generate_identifier(id_length = 6, n = 1, chars = c(0:9, letters[1:6]))
```

Arguments

`id_length`	character length of ID
`n`	number of IDs to generate
`chars`	characters to use for generation, defaults to hexadecimal characters (0-9 and a-f)

Examples

```
generate_identifier(8)
generate_identifier(6, 3)
```

`hospital_name` *Hospitalname*

Description

Hospitalname and/or location, with support for all hospitals in Northern Netherlands, including Meppel, Hardenberg and Zwolle.

Usage

```
hospital_name(x, format = "{naamkort}, {plaats}")
```

Arguments

`x`	text to be transformed
`format`	default is "`{naamkort}, {plaats}`". Attributes like `x` to be returned in '`glue`'-format (in curly brackets).

Examples

```
hospital_name(c("MCL", "MCL", "Martini"))
hospital_name(c("Antonius", "WZA", "Martini"), format = "{naam} te {plaats}")

# special case for GGD
hospital_name(c("Martini", "GGD Groningen", "GGD Drenthe"), format = "{naam}")
hospital_name(c("Martini", "GGD Groningen", "GGD Drenthe"), format = "{naamkort}")
hospital_name("ggd friesland", "{naam}")
```

`import` *Import Data Sets*

Description

These functions can be used to import data, from local or remote paths, or from the internet. They work closely with the `certeprojects` package to support Microsoft Planner project numbers. To support row names and older R versions, `import_*()` functions return plain data.frames, not e.g. tibbles.

Usage

```
import(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  auto_transform = TRUE,
  ...
)

import_rds(filename, project_number = project_get_current_id(ask = FALSE), ...)

import_xlsx(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  sheet = 1,
  range = NULL,
  auto_transform = TRUE,
  datenames = "nl",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = dec_mark(),
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0,
  ...
)

import_excel(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  sheet = 1,
  range = NULL,
  auto_transform = TRUE,
  datenames = "nl",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
```

```
  decimal.mark = dec_mark(),
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0,
  ...
)

import_csv(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  auto_transform = TRUE,
  datenames = "nl",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ".",
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0,
  ...
)

import_csv2(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  auto_transform = TRUE,
  datenames = "nl",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ",",
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0,
  ...
)

import_tsv(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  auto_transform = TRUE,
  datenames = "nl",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ".",
  big.mark = "",
  timezone = "UTC",
```

```
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0,
  ...
)

import_txt(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  auto_transform = TRUE,
  sep = "\t",
  datenames = "nl",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ",",
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0,
  ...
)

import_sav(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  auto_transform = TRUE,
  datenames = "en",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ".",
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  ...
)

import_spss(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  auto_transform = TRUE,
  datenames = "en",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ".",
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  ...
)
```

```
import_feather(
  filename,
  project_number = project_get_current_id(ask = FALSE),
  col_select = everything(),
  ...
)

import_clipboard(
  sep = "\t",
  header = TRUE,
  startrow = 1,
  auto_transform = TRUE,
  datenames = "nl",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = dec_mark(),
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  ...
)

import_mail_attachment(
  search = "hasattachment:yes",
  search_subject = NULL,
  search_from = NULL,
  search_when = NULL,
  search_attachment = NULL,
  folder = certemail::get_inbox_name(account = account),
  n = 5,
  sort = "received desc",
  account = certemail::connect_outlook(),
  auto_transform = TRUE,
  sep = ",",
  ...
)

import_url(
  url,
  auto_transform = TRUE,
  sep = ",",
  datenames = "en",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ".",
  big.mark = "",
  timezone = "UTC",
```

```
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0,
  ...
)

import_teams(
  full_teams_path = NULL,
  account = connect_teams(),
  auto_transform = TRUE,
  sep = ",",
  datenames = "en",
  dateformat = "yyyy-mm-dd",
  timeformat = "HH:MM",
  decimal.mark = ".",
  big.mark = "",
  timezone = "UTC",
  na = c("", "NULL", "NA", "<NA>"),
  skip = 0
)
```

Arguments

filename	the full path of the file to be imported, will be parsed to a character, can also be a remote location (from http/https/ftp/ssh, GitHub/GitLab)
project_number	a Microsoft Planner project number
auto_transform	transform the imported data with auto_transform()
...	arguments passed on to methods
sheet	Excel sheet to import, defaults to first sheet
range	a cell range to read from, allows typical Excel ranges such as "B3:D87" and "Budget!B2:G14"
datenames	language of the date names, such as weekdays and months
dateformat	expected date format, will be coerced with format_datetime()
timeformat	expected time format, will be coerced with format_datetime()
decimal.mark	separator for decimal numbers
big.mark	separator for thousands
timezone	expected time zone
na	values to interpret as NA
skip	number of first rows to skip
sep	character to separate values in a row
col_select	columns to select, supports the tidyselect language)
header	use first row as header
startrow	first row to start importing
search	an ODATA filter, ignores sort and defaults to search only mails with attachments

search_subject a character, equal to search = "subject:(search_subject)", case-insensitive

search_from a character, equal to search = "from:(search_from)", case-insensitive

search_when a Date vector of size 1 or 2, equal to search = "received:date1..date2", see *Examples*

search_attachment a character to use a regular expression for attachment file names

folder email folder name to search in, defaults to Inbox of the current user by calling get_inbox_name()

n maximum number of emails to search

sort initial sorting

account a Teams account from Azure or an AzureAuth Microsoft 365 token, e.g. retrieved with certeprojects::connect_teams()

url remote location of any data set, can also be a (non-raw) GitHub/GitLab link

full_teams_path a full path in Teams, **including the Team name and the channel name**. Leave blank to use interactive mode, which allows file/folder picking from a list in the console.

Details

Importing any unlisted filetype using import() requires the rio package to be installed.

Importing an Excel file using import_xlsx() or import_excel() requires the readxl package to be installed.

Importing an SPSS file using import_sav() or import_spss() requires the haven package to be installed.

Importing a Feather file using import_feather() requires the arrow package to be installed. Apache Feather provides efficient binary columnar serialization for data sets, enabling easy sharing data across data analysis languages (such as between Python and R). Use the col_select argument (which supports the tidyselect language) for specific data selection to improve importing speed.

Importing the clipboard using import_clipboard() requires the clipr package to be installed.

Importing mail attachments using import_mail_attachment() requires the certemail package to be installed. It calls download_mail_attachment() internally and saves the attachment to a temporary folder. For all folder names, run: sapply(certemail::connect_outlook()$list_folders(), function(x) x$properties$displayName).

The import_url() function tries to download the file first, after which it will be imported using the appropriate import_*() function.

The import_teams() function uses certeprojects::teams_download_file() to provide an interactive way to select a file in any Team, to download the file, and to import the file using the appropriate import_*() function.

See Also

export()

Examples

```
export_csv(iris)
import_csv("iris") |> head()

# the above is equal to:
# export(iris, "iris.csv")
# import("iris.csv") |> head()


# row names are also supported
export_csv(mtcars)
import_csv("mtcars") |> head()


# Apache's Feather format is column-based
# and allow for specific and fast file reading
library(dplyr, warn.conflicts = FALSE)
starwars |> export_feather()
import("starwars.feather",
       col_select = starts_with("h")) |>
  head()


# (cleanup)
file.remove("iris.csv")
file.remove("mtcars.csv")
file.remove("starwars.feather")

## Not run:

# ---- Microsoft Teams support ------------------------------------------

# IMPORTING

# import from Teams by picking a folder interactively from any Team
x <- import_teams()

# to NOT pick a Teams folder (e.g. in non-interactive mode), set `full_teams_path`
x <- import_teams(full_teams_path = "MyTeam/MyChannel/MyFolder/MyFile.xlsx")


# EXPORTING

# export to Teams by picking a folder interactively from any Team
mtcars |> export_teams()

# the default is RDS, but you can set `filename` to specify yourself
mtcars |> export_teams("mtcars.xlsx")

# to NOT pick a Teams folder (e.g. in non-interactive mode), set `full_teams_path`
mtcars |> export_teams("mtcars.xlsx", full_teams_path = "MyTeam/MyChannel/MyFolder")
mtcars |> export_teams(full_teams_path = "MyTeam/MyChannel/MyFolder")
```

```
## End(Not run)
```

`like` *Vectorised Pattern Matching with Keyboard Shortcut*

Description

Convenient wrapper around `grepl()` to match a pattern: `x %like% pattern`. It always returns a `logical` vector and is always case-insensitive (use `x %like_case% pattern` for case-sensitive matching). Also, `pattern` can be as long as `x` to compare items of each index in both vectors, or they both can have the same length to iterate over all cases.

Usage

```
like(x, pattern, ignore.case = TRUE)

x %like% pattern

x %unlike% pattern

x %like_case% pattern

x %unlike_case% pattern
```

Arguments

`x`	a character vector where matches are sought, or an object which can be coerced by `as.character()` to a character vector.
`pattern`	a character vector containing regular expressions (or a character string for `fixed = TRUE`) to be matched in the given character vector. Coerced by `as.character()` to a character string if possible.
`ignore.case`	if `FALSE`, the pattern matching is *case sensitive* and if `TRUE`, case is ignored during matching.

Details

These `like()` and `%like%`/`%unlike%` functions:

- Are case-insensitive (use `%like_case%`/`%unlike_case%` for case-sensitive matching)
- Support multiple patterns
- Check if `pattern` is a valid regular expression and sets `fixed = TRUE` if not, to greatly improve speed (vectorised over `pattern`)
- Always use compatibility with Perl unless `fixed = TRUE`, to greatly improve speed

Using RStudio? The %like%/%unlike% functions can also be directly inserted in your code from the Addins menu and can have its own keyboard shortcut like Shift+Ctrl+L or Shift+Cmd+L (see menu Tools > Modify Keyboard Shortcuts...). If you keep pressing your shortcut, the inserted text will be iterated over %like% -> %unlike% -> %like_case% -> %unlike_case%.

Value

A logical vector

Source

Idea from the like function from the data.table package, although altered as explained in *Details*.

See Also

grepl()

Examples

```
a <- "This is a test"
b <- "TEST"
a %like% b
b %like% a

# also supports multiple patterns
a <- c("Test case", "Something different", "Yet another thing")
b <- c(     "case",           "diff",      "yet")
a %like% b
a %unlike% b

a[1] %like% b
a %like% b[1]
```

---

privacy_check *Check Privacy of Plain Files*

Description

Checks if files contain privacy sensitive data and moves them to a 'vault folder' if this is the case.

Usage

```
privacy_check(
  path = getwd(),
  vault = paste0(path, "/vault"),
  log_file = paste0(vault, "/_privacy_log.txt"),
  suspicious_cols = "(zip.*code|postcode|bsn|geboortedatum)"
)
```

Arguments

path	path to check
vault	path to vault
log_file	path to log file where details will be written to
suspicious_cols	regular expression to match suspicious column names

---

p_symbol *P-symbol format as asterisk*

Description

P-symbol format as asterisk

Usage

```
p_symbol(p, emptychar = " ")
```

Arguments

p	numeric value between 0 and 1
emptychar	sign to be displayed for 0.1 < p < 1.0

---

read_secret *Read Certe Secret From File*

Description

This function reads from a local or remote YAML file, as set in the environmental variable "secrets_file".

Usage

```
read_secret(property, file = Sys.getenv("secrets_file"))
```

Arguments

property	the property to read, case-sensitive
file	either a character string naming a file or a connection open for writing

Details

In the secrets file, the property name and value have to be separated with a colon (:), as is intended in YAML files.

The default value for file is the environmental variable "secrets_file".

The file will be read using read_yaml(), which allows almost any local path or remote connection (such as websites).

Examples

```
# for this example, create a temporary 'secrets' file
my_secrets_file <- tempfile(fileext = ".yaml")
Sys.setenv(secrets_file = my_secrets_file)
writeLines(c("tenant_id: 8fb3c03060e02e89",
             "default_users: user_1"),
           my_secrets_file)

read_secret("tenant_id")
read_secret("default_users")
```

---

ref_dir *Return Reference Directory*

Description

Returns the relative reference directory for non-projects.

Usage

```
ref_dir(sub = "")
```

Arguments

sub	relative subfolder or file

Details

This function returns the absolute path using `tools::file_path_as_absolute()`.

---

remember *Temporarily Remember Objects*

Description

Can be used in dplyr-syntax to remember values and objects for later use. Objects are (temporarily) stored in the `certetoolbox` package environment.

Usage

```
remember(.data, ...)

recall(x = NULL, delete = TRUE)
```

Arguments

.data	data.frame
...	value(s) to be remembered
x	value to be recalled
delete	a logical to indicate whether the delete value after recalling

Details

values can be saved with `remember()` and recalled (and deleted) with `recall()`.

Examples

```
library(dplyr, warn.conflicts = FALSE)

x <- mtcars %>% remember(nrow(.))
recall()
recall() # value removed
x <- mtcars %>% remember(n = nrow(.))
recall(n)
recall(n) # value removed

## Not run:
 tbl %>%
   filter(...) %>%
   remember(rows = nrow(.)) %>%
   group_by(...) %>%
   summarise(...) %>%
   plot2(title = "Test",
         subtitle = paste("n =", recall(rows)))

## End(Not run)
```

---

size_humanreadable *Human-readable File Size*

Description

Formats bytes into human-readable units, from "kB" (10^3) to "YB" (10^23).

Usage

```
size_humanreadable(bytes, decimals = 1, decimal.mark = dec_mark())
```

Arguments

bytes	number of bytes
decimals	precision, not used for bytes and kilobytes
decimal.mark	decimal mark to use, defaults to `dec_mark()`

Details

If using `object.size()` on an object, this function is equal to using `format2()` to format the object size.

Examples

```
size_humanreadable(c(12, 1234, 123456, 12345678))

size_humanreadable(1024 ^ c(0:4))
```

`tbl_flextable` *Format Data Set as Flextable*

Description

Format a data.frame as `flextable()` with Certe style, bold headers and Dutch number formats. This function can also transform existing `flextable` and `gtsummary` objects to allow the formatting provided in this `tbl_flextable()` function.

Usage

```
tbl_flextable(
  x,
  row.names = rownames(x),
  row.names.bold = TRUE,
  rows.italic = NULL,
  rows.bold = NULL,
  rows.height = NULL,
  rows.fill = NULL,
  rows.zebra = TRUE,
  row.total = FALSE,
  row.total.name = "Totaal",
  row.total.function = sum,
  row.total.widths = NULL,
  row.total.bold = TRUE,
  row.extra.header = list(values = NULL, widths = 1),
  row.extra.footer = list(values = NULL, widths = 1),
  column.names = colnames(x),
  column.names.bold = TRUE,
  columns.width = NULL,
  columns.percent = NULL,
  columns.italic = NULL,
  columns.bold = NULL,
  columns.fill = NULL,
  columns.zebra = FALSE,
  column.total = FALSE,
  column.total.name = "Totaal",
```

```
  column.total.function = sum,
  column.total.bold = TRUE,
  align = "c",
  align.part = "all",
  caption = "",
  na = "",
  logicals = c("X", ""),
  round.numbers = 2,
  round.percent = 1,
  format.dates = "d mmm yyyy",
  decimal.mark = dec_mark(),
  big.mark = big_mark(),
  font.family = "Source Sans Pro",
  font.size = 9,
  font.size.header = font.size + 1,
  values.colour = NULL,
  values.fill = NULL,
  values.bold = NULL,
  values.italic = NULL,
  autofit = is.null(columns.width) & is.null(rows.height),
  autofit.fullpage = TRUE,
  autofit.fullpage.width = 16,
  vline = NULL,
  vline.part = c("body", "footer"),
  theme = current_markdown_colour(),
 colours = list(rows.fill.even = paste0(theme, "6"), rows.fill.odd = paste0(theme, "5"),
  columns.fill = paste0(theme, "5"), values.fill = paste0(theme, "3"), values.colour =
  theme, vline.colour = theme, hline.colour = theme, header.fill = theme, header.colour
    = "white", vline.header.colour = "white"),
  split.across.pages = NROW(x) > 37,
  print = !interactive(),
  ...
)

## S3 method for class 'certetoolbox_flextable'
print(x, use_knitr = !is_latex_output(), ...)
```

Arguments

x	a data.frame or a flextable object or a gtsummary object
row.names	row names to be displayed. Will be 1:nrow(x) if set to TRUE, but can be a vector of values.
row.names.bold	display row names in bold
rows.italic	column indexes of rows in italics
rows.bold	column indexes of rows in bold
rows.height	height of the rows in centimetres
rows.fill	the column indices of rows to be shaded

`rows.zebra` banded rows in the body - equivalent to `rows.fill = seq(2, nrow(x), 2)`

`row.total` add a row total (at the bottom of the table)

`row.total.name` name of the row total

`row.total.function`
function used to calculate all numeric values per column (non-numeric columns are skipped)

`row.total.widths`
cell width in row total

`row.total.bold` bold formatting of row total

`row.extra.header`
an extra header to be displayed above the table

`row.extra.footer`
an extra footer to show below the table

`column.names` column names to be displayed. Can also be a named vector where the names are existing columns, or indices of columns. When this vector is smaller than `ncol(x)`, only the first `length(column.names)` are replaced. When this vector is longer than `ncol(x)`, all column names are replaced

`column.names.bold`
display column names in bold

`columns.width` width of columns. For `autofit.fullpage = TRUE`, these are proportions to `autofit.fullpage.width`. For `autofit.fullpage = FALSE`, these are centimeters

`columns.percent`
display the column indices as percentages using `format2()` - example: `columns.percent = c(2, 3)`

`columns.italic` column indices of columns to be displayed in italics

`columns.bold` column indices of columns in bold

`columns.fill` the column indices of rows to be shaded

`columns.zebra` banded columns - equivalent to `columns.fill = seq(2, ncol(x), 2)`

`column.total` adding a column total (to the right of the table)

`column.total.name`
name of the column total

`column.total.function`
function used to calculate all numeric values per row

`column.total.bold`
bold formatting of column total

`align` default is "c", which aligns everything centrally. Use "r", "l", "c" and "j"/"u" (justify/align) to change alignment. Can be a vector or a character (like "lrrrcc")

`align.part` part of the table where the alignment should take place ("all", "header", "body", "footer")

`caption` table caption

`na` text for missing values

logicals	vector with two values that replace TRUE and FALSE
round.numbers	number of decimal places to round up for numbers
round.percent	number of decimal places to round to when using columns.percent
format.dates	see format2()
decimal.mark	decimal separator, defaults to dec_mark()
big.mark	thousands separator, defaults to big_mark()
font.family	table font family
font.size	table font size
font.size.header	font size of header
values.colour, values.fill, values.bold, values.italic	values to be formatted
autofit	format table in width automatically. This will apply autofit().
autofit.fullpage	display table across width of page
autofit.fullpage.width	set number of centimetres to width of table
vline	indices of columns to have a vertical line to their right
vline.part	part of the table where the vertical lines should be placed ("all", "header", "body", "footer")
theme	a Certe colour theme, defaults to current_markdown_colour() which determines the Certe colour based on a markdown YAML header and defaults to "certeblauw". Can also be "certeroze", "certegroen", etc. This will set the list in colours and will be ignored if colours is set manually. Can be set to "white" for a clean look.
colours	a list with the following named character values: rows.fill.even, rows.fill.odd, columns.fill, values.fill, and values.colour. All values will be evaluated with colourpicker().
split.across.pages	a logical whether tables are allowed to split across page. This argument only has effect for PDF output.
print	forced printing (required in a for loop), default is TRUE in non-interactive sessions
...	not used
use_knitr	use the knitr package for printing. Ignored when in an interactive session. If FALSE, an internal certetoolbox function will be used to convert the LaTeX longtable that would print across multiple PDF pages. If in a non-interactive session where the output is non-LaTeX, the knitr package will always be used.

Details

Run tbl_markdown() on a flextable object to transform it into markdown for use in Quarto or R Markdown reports. If print = TRUE in non-interactive sessions (Quarto or R Markdown), the flextable object will also be printed in markdown.

The value for `theme` is dependent on whether a colour is set in the markdown YAML header. Otherwise, use `theme` to set a Certe colour theme, defaults to `"certeblauw"`:

```
# from the example below
tbl_flextable(df)
```

text	decimal.numbers	whole.numbers	logical.values	dates
A	4,03	1	X	15 dec 2018
B	8,31	4		8 apr 2020
C	8,48	1	X	19 dec 2018
D	3,66	1		2 mei 2019
E	7,02	4	X	4 jul 2022
F	1,59	2		24 jan 2020
G	7,08	1	X	4 okt 2021
H	5,32	3		20 apr 2020
I	9,15	3	X	22 okt 2022
J	0,33	9		12 jul 2020

```
tbl_flextable(df, theme = "certeroze")
```

text	decimal.numbers	whole.numbers	logical.values	dates
A	4,03	1	X	15 dec 2018
B	8,31	4		8 apr 2020
C	8,48	1	X	19 dec 2018
D	3,66	1		2 mei 2019
E	7,02	4	X	4 jul 2022
F	1,59	2		24 jan 2020
G	7,08	1	X	4 okt 2021
H	5,32	3		20 apr 2020
I	9,15	3	X	22 okt 2022
J	0,33	9		12 jul 2020

```
tbl_flextable(df, theme = "certegeel")
```

text	decimal.numbers	whole.numbers	logical.values	dates
A	4,03	1	X	15 dec 2018
B	8,31	4		8 apr 2020
C	8,48	1	X	19 dec 2018
D	3,66	1		2 mei 2019
E	7,02	4	X	4 jul 2022
F	1,59	2		24 jan 2020
G	7,08	1	X	4 okt 2021
H	5,32	3		20 apr 2020
I	9,15	3	X	22 okt 2022
J	0,33	9		12 jul 2020

```
tbl_flextable(df, theme = "certegroen", vline = c(2:3))
```

text	decimal.numbers	whole.numbers	logical.values	dates
A	4,03	1	X	15 dec 2018
B	8,31	4		8 apr 2020
C	8,48	1	X	19 dec 2018
D	3,66	1		2 mei 2019
E	7,02	4	X	4 jul 2022
F	1,59	2		24 jan 2020
G	7,08	1	X	4 okt 2021
H	5,32	3		20 apr 2020
I	9,15	3	X	22 okt 2022
J	0,33	9		12 jul 2020

```
tbl_flextable(
  df,
  theme = "certelila",
  row.total = TRUE,
  row.total.function = median,
  round.numbers = 4,
  row.extra.header = list(values = LETTERS[1:5])
)
```

A	B	C	D	E
text	decimal.numbers	whole.numbers	logical.values	dates
A	4,0297	1	X	15 dec 2018
B	8,3080	4		8 apr 2020
C	8,4826	1	X	19 dec 2018
D	3,6639	1		2 mei 2019
E	7,0157	4	X	4 jul 2022
F	1,5950	2		24 jan 2020
G	7,0799	1	X	4 okt 2021
H	5,3216	3		20 apr 2020
I	9,1504	3	X	22 okt 2022
J	0,3300	9		12 jul 2020
Totaal	**6,1687**	**2,5**		**14 apr 2020**

Value

flextable object

See Also

`flextable()`

Examples

```
## Not run:

# generate a data.frame
df <- data.frame(text = LETTERS[1:10],
                 `decimal numbers` = runif(10, 0, 10),
                 `whole numbers` = as.integer(runif(10, 0, 10)),
                 `logical values` = as.logical(round(runif(10, 0, 1))),
                 dates = today() - runif(10, 200, 2000),
                 stringsAsFactors = FALSE)

# default
tbl_flextable(df)      # dataset has no row names
tbl_flextable(mtcars)  # dataset has row names

# print in markdown
df |>
  tbl_flextable() |>
  tbl_markdown()

# transform a gtsummary to a flextable
iris |>
```

```
  tbl_gtsummary(Species, add_p = TRUE) |>
  tbl_flextable()

# extra formatting
tbl_flextable(df,
              logicals = c("X", "-"),     # replaces TRUE en FALSE
              values.colour = "X",
              values.fill = "X",
              row.names = "S. aureus",
              columns.italic = 1,
              format.dates = "ddd dd-mm-yy",
              round.numbers = 3)

# row totals
tbl_flextable(df,
              row.total = TRUE,           # add row total
              row.total.function = max,   # instead of sum()
              row.total.name = "Maximum", # also works with dates
              columns.percent = 2,        # 2nd column as percentages
              round.percent = 0)          # rounding percentages

# column names
tbl_flextable(df,
              column.names = c("1" = "Column 1",
                               "2" = "Column 2",
                               dates = "DATES!"))
tbl_flextable(df,
              column.names = LETTERS)

# vertical lines, alignment and row names
tbl_flextable(df,
              align = "lrrcc", # also works: c("l", "r", "r", "c", "c")
              font.size = 12,
              vline = c(2, 4),
              vline.part = "all",
              row.names = paste("Experiment", 1:10))

# width of cells and table
tbl_flextable(data.frame(test1 = "A", test2 = "B"),
              vline = 1,
              autofit.fullpage.width = 16, # default values in cm
              columns.width = c(1, 3))     # ratio; cells become 4 and 12 cm

tbl_flextable(data.frame(test1 = "A", test2 = "B"),
              vline = 1,
              autofit.fullpage = FALSE,    # no fullpage autofit
              columns.width = c(1, 3))     # cells become 1 and 3 cm

# adding extra header or footer
tbl_flextable(data.frame(test1 = "A", test2 = "B"),
              row.extra.header = list(values = c("Header", "Header"),
                                      widths = c(1, 1)),
              row.extra.footer = list(values = c("Footer", "Footer"),
```

```
                                        widths = c(1, 1)))

## End(Not run)
```

---

tbl_gtsummary *Summarise Table as* gtsummary

Description

Summarise a data.frame as gtsummary with Dutch defaults. These objects are based on the gt package by RStudio. To provide Certe style and compatibility with MS Word, use tbl_flextable() to transform the gtsummary object.

Usage

```
tbl_gtsummary(
  x,
  by = NULL,
  label = NULL,
  digits = 1,
  ...,
  language = "nl",
  column1_name = "Eigenschap",
  add_n = FALSE,
  add_p = FALSE,
  add_ci = FALSE,
  add_overall = FALSE,
  decimal.mark = dec_mark(),
  big.mark = big_mark()
)
```

Arguments

x	a data.frame
by	A column name (quoted or unquoted) in data. Summary statistics will be calculated separately for each level of the by variable (e.g. by = trt). If NULL, summary statistics are calculated using all observations. To stratify a table by two or more variables, use tbl_strata()
label	List of formulas specifying variables labels, e.g. list(age ~ "Age", stage ~ "Path T Stage"). If a variable's label is not specified here, the label attribute (attr(data$age, "label")) is used. If attribute label is NULL, the variable name will be used.
digits	List of formulas specifying the number of decimal places to round summary statistics. If not specified, tbl_summary guesses an appropriate number of decimals to round statistics. When multiple statistics are displayed for a single variable, supply a vector rather than an integer. For example, if the statistic being calculated is "{mean} ({sd})" and you want the mean rounded to 1 decimal

	place, and the SD to 2 use `digits = list(age ~ c(1, 2))`. User may also pass a styling function: `digits = age ~ style_sigfig`
`...`	Arguments passed on to `gtsummary::tbl_summary()`
`language`	the language to use, defaults to Dutch
`column1_name`	name to use for the first column
`add_n`	add the overall N using `gtsummary::add_n()`
`add_p`	add the p values `gtsummary::add_p()` (tests will be determined automatcally)
`add_ci`	add the confidence interval using `gtsummary::add_ci()`
`add_overall`	add the overall statistics using `gtsummary::add_overall()`
`decimal.mark`	decimal separator, defaults to `dec_mark()`
`big.mark`	thousands separator, defaults to `big_mark()`

Details

`tbl_gtsummary()` creates a summary table with `gtsummary::tbl_summary()`, to which different extra columns can be added e.g. with `add_p = TRUE` and `add_overall = TRUE`.

Examples

```
# These examples default to the Dutch language

iris |>
  tbl_gtsummary()

iris |>
  tbl_gtsummary(Species, add_p = TRUE)

iris |>
  tbl_gtsummary(Species, add_n = TRUE)

# support strata by providing
iris2 <- iris
iris2$Category <- sample(LETTERS[1:2], size = 150, replace = TRUE)
head(iris2)

iris2 |>
  tbl_gtsummary(c(Category, Species))

# transform to flextable
# (formats to Certe style and allows rendering to Word)
iris |>
  tbl_gtsummary(Species) |>
  tbl_flextable()
```

`tbl_markdown` *Print Table as Markdown, LaTeX of HTML*

Description

Prints a data.frame as Markdown, LaTeX or HTML using `knitr::kable()`, with bold headers and Dutch number formats.

Usage

```
tbl_markdown(
  x,
  row.names = rownames(x),
  column.names = colnames(x),
  align = NULL,
  padding = 2,
  caption = "",
  na = "",
  type = "markdown",
  format.dates = "dd-mm-yyyy",
  decimal.mark = dec_mark(),
  big.mark = big_mark(),
  logicals = c("X", ""),
  columns.percent = NA,
  column.names.bold = TRUE,
  round.numbers = 2,
  round.percent = 1,
  newlines.leading = 0,
  newlines.trailing = 2,
  print = TRUE
)
```

Arguments

`x`	a data.frame or a `flextable` object or a `gtsummary` object
`row.names`	row names to be displayed
`column.names`	column names to be displayed
`align`	alignment of columns (default: numbers to the right, others to the left)
`padding`	extra cell padding
`caption`	caption of table
`na`	text for missing values (default: `""`)
`type`	type of formatting the table - valid options are `"latex"`, `"html"`, `"markdown"`, `"pandoc"` and `"rst"`
`format.dates`	formatting of dates, will be evaluated with `format2()`

decimal.mark	decimal separator, defaults to dec_mark()
big.mark	thousands separator, defaults to big_mark()
logicals	vector with two values that replace TRUE and FALSE
columns.percent	display the column indices as percentages using format2() - example: columns.percent = c(2, 3)
column.names.bold	display column names in bold
round.numbers	number of decimal places to round up for numbers
round.percent	number of decimal places to round to when using columns.percent
newlines.leading	number of white lines to print before the table
newlines.trailing	number of white lines to print after the table
print	only useful when input is a Flextable: force printing

Details

When in an R Markdown rapport a table is printed using this function, column headers only print well if newlines.leading >= 2, or by manually using cat("\\n\\n") before printing the table.

Value

character

See Also

knitr::kable()

Examples

```
tbl_markdown(mtcars[1:6, 1:6], padding = 1)
```

---

update *Update Data Set Based on Row Numbers*

Description

Update a data.frame using specific integers for row numbers or a vectorised filter. Also supports dplyr groups. see *Examples*.

Usage

```
## S3 method for class 'data.frame'
update(object, rows, ...)
```

Arguments

object	a data.frame
rows	row numbers or a logical vector
...	arguments passed on to mutate()

Examples

```
iris |>
  update(3:4, Species = c("A", "B")) |>
  head()

iris |>
  update(Species == "setosa" & Sepal.Length > 5,
         Species = "something else") |>
  head()

if (require("dplyr")) {

  # also supports dplyr groups:
  iris |>
    group_by(Species) |>
    # update every 2nd to 4th row in group
    update(2:4, Species = "test") |>
    # groups will be updated automatically
    count()

}
```

Index