Package: inborutils (via r-universe)

August 17, 2024

Title Collection of Useful R Utilities

Version 0.4.0

Description While working on research projects, typical small functionalities are useful across these projects. Instead of copy-pasting these functions in all individual project repositories/folders, this package collects these functions for reuse by ourself and - if useful - others as well.

License MIT + file LICENSE

URL https://github.com/inbo/inborutils, https://inbo.github.io/inborutils/

BugReports https://github.com/inbo/inborutils/issues

Depends R (>= 3.4.0)

Imports assertable, assertthat, curl, DBI, dplyr, httr, iterators, jsonlite, leaflet, lubridate, purrr, readr, rgbif, rlang, RSQLite, sf, stats, stringr, tibble, tidyr, tidyselect

Suggests drat, ggplot2, glue, htmlwidgets, knitr, lifecycle, mockery, odbc, rmarkdown, sp, testthat, withr

VignetteBuilder knitr

Config/checklist/communities inbo

Config/checklist/keywords helper functions; utilities; coding club

Encoding UTF-8

Language en-GB

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Repository https://inbo.r-universe.dev

RemoteUrl https://github.com/inbo/inborutils

RemoteRef HEAD

RemoteSha 1bfb7405a58022e8a3b7cf9b97a5c14a9a958d7e

Contents

`convertdf_enc` *Convert encoding of character and factor variables in a dataframe*

Description

Convert encoding of character and factor variables in a dataframe

Usage

```
convertdf_enc(x, from = "", to = "UTF-8", sub = NA, colnames = FALSE)
```

Arguments

`x`	An object with the `data.frame` class (such as `data.frame` or `sf`)
`from`	A character string describing the current encoding.
`to`	A character string describing the target encoding.
`sub`	character string. If not `NA` it is used to replace any non-convertible bytes in the input. (This would normally be a single character, but can be more.) If `"byte"`, the indication is `"<xx>"` with the hex code of the byte. If `"Unicode"` and converting from UTF-8, the Unicode point in the form `"<U+xxxx>"`, or if `c99`, a C99-style escape `"\uxxxx"`.
`colnames`	Should column names be converted as well?

Details

Encoding strings: all R platforms support "" (for the encoding of the current locale), "latin1" and "UTF-8". See iconv for more information.

Value

The original object, with character variables (and levels of (character) factor variables) converted to the specified encoding.

See Also

Other Data_handling_utilities: csv_to_sqlite(), df_factors_to_char()

---

coordinate_example *Example* data.frame *with coordinates*

Description

A dataset containing 52 coordinates as latitude and longitude

Usage

```
coordinate_example
```

Format

A data.frame with 52 rows and 3 variables:

- id: resource identifier
- latitude: Latitude of the coordinates
- longitude: Longitude of the coordinates

See Also

Other datasets: rain_knmi_2012, species_example

csv_to_sqlite *Save a delimited text table into a single table* sqlite *database*

Description

The table can be a comma separated (csv) or a tab separated (tsv) or any other delimited text file. The file is read in chunks. Each chunk is copied in the same sqlite table database before the next chunk is loaded into memory. See the INBO tutorial Handling large files in R to learn more.

Usage

```
csv_to_sqlite(
  csv_file,
  sqlite_file,
  table_name,
  delim = ",",
  pre_process_size = 1000,
  chunk_size = 50000,
  show_progress_bar = TRUE,
  ...
)
```

Arguments

csv_file	Name of the text file to convert.
sqlite_file	Name of the newly created sqlite file.
table_name	Name of the table to store the data table in the sqlite database.
delim	Text file delimiter (default ",").
pre_process_size	Number of lines to check the data types of the individual columns (default 1000).
chunk_size	Number of lines to read for each chunk (default 50000).
show_progress_bar	Show progress bar (default TRUE).
...	Further arguments to be passed to read_delim.

Value

a SQLite database

Remark

The callback argument in the read_delim_chunked function call refers to the custom written callback function append_to_sqlite applied to each chunk.

See Also

Other Data_handling_utilities: convertdf_enc(), df_factors_to_char()

Examples

```
## Not run:
library(R.utils)
library(dplyr)
csv.name <- "2016-04-20-processed-logs-big-file-example.csv"
db.name <- "2016-04-20-processed-logs-big-file-example.db"
# download the CSV file example
csv.url <- paste("https://s3-eu-west-1.amazonaws.com/lw-birdtracking-data/",
  csv.name, ".gz",
  sep = ""
)
download.file(csv.url, destfile = paste0(csv.name, ".gz"))
gunzip(paste0(csv.name, ".gz"))
# Make a SQLite database
sqlite_file <- "example2.sqlite"
table_name <- "birdtracks"
csv_to_sqlite(
  csv_file = csv.name,
  sqlite_file = sqlite_file,
  table_name = table_name
)
# Get access to SQLite database
my_db <- src_sqlite(sqlite_file, create = FALSE)
bird_tracking <- tbl(my_db, "birdtracks")
# Example query via dplyr
results <- bird_tracking %>%
  filter(device_info_serial == 860) %>%
  select(date_time, latitude, longitude, altitude) %>%
  filter(date_time < "2014-07-01") %>%
  filter(date_time > "2014-03-01") %>%
  as_tibble()
head(results)

## End(Not run)
```

---

df_factors_to_char *Convert all factors of a dataframe to characters*

Description

The function checks the which columns are factors and converts these to character vectors

Usage

```
df_factors_to_char(rp)
```

Arguments

rp	A dataframe

Details

all credits to Thierry Onkelinx

See Also

Other Data_handling_utilities: convertdf_enc(), csv_to_sqlite()

Examples

```
df_factors_to_char(PlantGrowth) # column group will be chars as well
```

---

download_knmi_data_hour

Download KNMI hourly data to file

Description

More info, zie https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script # nolint

Usage

```
download_knmi_data_hour(
  stations,
  variables,
  start_date,
  end_date,
  output_file = "knmi_download.txt"
)
```

Arguments

stations	list of integers. For an overview, see https://www.daggegevens.knmi.nl/klimatologie/uurgegevens
variables	list of variables, options are: • "WIND = DD:FH:FF:FX"Wind • "TEMP = T:T10N:TD"Temperature • "SUNR = SQ:Q"Sunshine duration and global radiation • "PRCP = DR:RH"Precipitation and potential evaporation • "VICL = VV:N:U"Visibility, cloud cover and relative humidity • "WEER = M:R:S:O:Y:WW"Weather phenomena, weather types • "ALL"all variables

start_date	date interpretable string to define start of the period
end_date	date interpretable string to define end of the period
output_file	path/filename of the output_file_name

Value

response containing, status_code, content,...

See Also

Other download_functions: download_zenodo(), extract_soil_map_data(), read_kmi_data(), read_kml_file(), read_knmi_data(), read_mow_data()

Examples

```
## Not run:
download_knmi_data_hour(c(310, 319), "PRCP", "2011-01-01", "2012-02-17",
  output_file = "knmi_output_download.txt"
)

## End(Not run)
```

---

download_zenodo *Get data from a Zenodo archive*

Description

This function will download an entire archive from Zenodo (https://zenodo.org). It only works for Zenodo created DOI (not when the DOI is for example derived from Zookeys.)

Usage

```
download_zenodo(doi, path = ".", parallel = TRUE, quiet = FALSE)
```

Arguments

doi	a doi pointer to the Zenodo archive starting with '10.5281/zenodo.'. See examples.
path	Path where the data must be downloaded. Defaults to the working directory.
parallel	Logical. If TRUE (the default), files will be downloaded concurrently for multi-file records. Of course, the operation is limited by bandwidth and traffic limitations.
quiet	Logical (FALSE by default). Do you want to suppress informative messages (not warnings)?

Author(s)

Hans Van Calster, <hans.vancalster@inbo.be>

Floris Vanderhaeghe, <floris.vanderhaeghe@inbo.be>

See Also

Other download_functions: download_knmi_data_hour(), extract_soil_map_data(), read_kmi_data(), read_kml_file(), read_knmi_data(), read_mow_data()

Examples

```
## Not run:
# Example download of an archive containing a single zip
download_zenodo(doi = "10.5281/zenodo.1283345")
download_zenodo(doi = "10.5281/zenodo.1283345", quiet = TRUE)
# Example download of an archive containing multiple files
# using parallel download
# (multiple files will be simultaneously downloaded)
download_zenodo(doi = "10.5281/zenodo.1172801", parallel = TRUE)
# Example download of an archive containing a single pdf file
download_zenodo(doi = "10.5281/zenodo.168478")

## End(Not run)
```

---

extract_soil_map_data *Extract soil properties from Flemish soil map*

Description

This function queries the the Flemish soil map # nolint attributes at a given coordinate, by using the affiliated WFS service provided by DOV. The user can pick the properties of interest. See the examples section to see how to obtain a full list of possible properties of interest. Coordinates should be given in the 'Belge 1972 / Belgian Lambert 72' (EPSG:31370) coordinate reference system. When outside the Flemish region, an NA value is given for each of the properties.

Usage

```
extract_soil_map_data(x_lam, y_lam, properties_of_interest)
```

Arguments

x_lam	The numeric value of the X coordinate in CRS 'Belge 1972 / Belgian Lambert 72' (EPSG:31370).
y_lam	The numeric value of the Y coordinate in CRS 'Belge 1972 / Belgian Lambert 72' (EPSG:31370).
properties_of_interest	A vector or properties, as a subset of these provided by the webservice (see the examples section for how to obtain the list). Default Bodemserie, Unibodemtype and Bodemtype.

Value

A `data.frame` with the properties as column headers.

See Also

Other download_functions: `download_knmi_data_hour()`, `download_zenodo()`, `read_kmi_data()`, `read_kml_file()`, `read_knmi_data()`, `read_mow_data()`

Examples

```
## Not run:
library(inborutils)
extract_soil_map_data(173995.67, 212093.44)

# code to obtain list of all possible soil properties
library(ows4R)
library(purrr)
wfs <- "https://www.dov.vlaanderen.be/geoserver/bodemkaart/bodemtypes/wfs"
wfs_client <- WFSClient$new(wfs,
  serviceVersion = "1.1.0"
)
wfs_client$
  getCapabilities()$
  findFeatureTypeByName("bodemkaart:bodemtypes")$
  getDescription() %>%
  map_chr(function(x) {
    x$getName()
  })

extract_soil_map_data(173995.67, 212093.44,
  properties_of_interest = c(
    "Eenduidige_legende",
    "Textuurklasse"
  )
)

## End(Not run)
```

---

gbif_species_name_match

Add species information provided by the GBIF taxonomic backbone API

Description

[Superseded]

Development on `gbif_species_name_match()` is complete, and for new code we strongly recommend switching to `rgbif::name_backbone_checklist()`, which is easier to use, has more

features, and still under active development. This functions extends an existing dataframe with additional columns provided by the GBIF taxonomic backbone and matched on the species (scientific) name, which need to be an available column in the dataframe.

This function is essentially a wrapper around the existing rgbif name_backbone and extends the application to a data.frame. Such extension has been added to rgbif via the function rgbif::name_backbone_checklist. # nolint For more information on the name matching API of GBIF on which rgbif relies, see https://www.gbif.org/developer/species/#searching.

Usage

```
gbif_species_name_match(
  df,
  name = "name",
 gbif_terms = c("usageKey", "scientificName", "rank", "order", "matchType", "phylum",
    "kingdom", "genus", "class", "confidence", "synonym", "status", "family"),
  ...
)
```

Arguments

df	data.frame with species information
name	char column name of the column containing the names used for the name matching with the GBIF taxonomic backbone. Default: "name".
gbif_terms	list of valid GBIF terms to add as additional columns to the data.frame. Default: usageKey, scientificName, rank, order, matchType, phylum, kingdom, genus, class, confidence, synonym, status, family.
...	any parameter to pass to rgbif function name_bakbone. One of: rank, kingdom, phylum, class, order, family, genus, strict, verbose, start, limit, curlopts. See ?name_backbone for more details.

Value

a tibble data.frame with GBIF information as additional columns. If none of the taxa in df is matched, only the columns confidence, matchType and synonym are added. This behaviour is inherited by rgbif::name_backbone.

Examples

```
## Not run:
library(readr)
library(dplyr)
species_list <- read_csv(paste0(
  "https://raw.githubusercontent.com/inbo",
  "/inbo-pyutils/master/gbif/gbif_name_match",
  "/sample.csv"
),
trim_ws = TRUE, col_types = cols()
)
# basic usage
```

```
species_list %>%
  gbif_species_name_match()
# pass optional parameters to name_backbone
species_list %>%
  gbif_species_name_match(name = "name", kingdom = "kingdom", strict = TRUE)
# select GBIF terms
species_list %>%
  gbif_species_name_match(gbif_terms = c("scientificName", "rank"))

## End(Not run)
```

`guess_crs` *Take a dataset and plot in different coordinate reference systems (CRS) on a leaflet map to check the expected CRS*

Description

When retrieving a number of data-points, but the kind of CRS is not provided/known (for any reason), this utility plots the data for different CRS on a map to make comparison possible. The main function is `guess_crs` which creates the map with different options. Custom projections can be provided by the user as well.

Usage

```
guess_crs(
  df,
  col_x,
  col_y,
  belgium = TRUE,
  crs_try = c("EPSG:4326", "EPSG:31370", "EPSG:28992", "EPSG:32631", "EPSG:3812",
    "EPSG:3035")
)
```

Arguments

`df`	data.frame with a x and y coordinate column
`col_x`	(character) name of the x (longitude) column
`col_y`	(character) name of the y (latitude) column
`belgium`	(logical) If `TRUE`, coordinates are expected to be in Belgium
`crs_try`	(list) `EPSG` codes of the different CRS to evaluate By default, the following six CRS are added to the map: • "EPSG:4326"WGS 84, https://spatialreference.org/ref/epsg/wgs-84/ • "EPSG:31370"Belge 1972/Belgian Lambert 72, https://spatialreference.org/ref/epsg/31370/ • "EPSG:28992"Amersfoort/Rijksdriehoek nieuw, https://spatialreference.org/ref/epsg/28992/ • "EPSG:32631"WGS 84 / UTM zone 31N, https://spatialreference.org/ref/epsg/32631/ • "EPSG:3812"ETRS89/Belgian Lambert 2008, https://spatialreference.org/ref/epsg/3812/ • "EPSG:3035"ETRS89 / ETRS-LAEA, https://spatialreference.org/ref/epsg/3035/

Details

For each of the given CRS, the coordinates columns are given the the specified CRS In a next step, these CRS are converted to `wgs84` and plotted on a `openstreetmap` background with leaflet. The interactive map provides the possibility to select/unselect specific layers.

See Also

crsuggest::guess_crs() for a (better) alternative

Other GIS_utilities: `plot_coordinates_on_map()`, `transform_coordinates()`

Examples

```
## Not run:
guess_crs(data, "x", "y")

## End(Not run)
```

`human_filesize` *Human-readable binary file size*

Description

Takes an integer (referring to number of bytes) and returns an optimally human-readable binary-prefixed byte size (KiB, MiB, GiB, TiB, PiB, EiB). The function is vectorised.

Usage

```
human_filesize(x)
```

Arguments

`x`	A positive integer, i.e. the number of bytes (B). Can be a vector of file sizes.

Value

A character vector.

Author(s)

Floris Vanderhaeghe, <floris.vanderhaeghe@inbo.be>

Examples

```
human_filesize(7845691)
v <- c(12345, 456987745621258)
human_filesize(v)
```

inborutils-deprecated *Deprecated functions in inborutils*

Description

These functions still work but will be removed (defunct) in the next version.

plot_coordinates_on_map *Plot x/y coordinates on a map*

Description

This function plots x/y coordinates from a data.frame on a (leaflet) map. The coordinates are first converted to WGS84 in order to map them correctly on the leaflet map (@seealso transform_coordinates()). To do this, the original Coordinate Reference System (CRS) is asked.

Usage

```
plot_coordinates_on_map(df, col_x, col_y, projection, ...)
```

Arguments

df	A data.frame with a x and y coordinate columns.
col_x, col_y	Column names or positions of the x (longitude) and y (latitude) column. They are passed to vars_pull. These arguments are passed by expression and support quasiquotation (you can unquote column names or column positions).
projection	Projection string of class CRS-class (sp objects) or crs-class (sf objects) defining the current projection.
...	Additional arguments passed on to addCircleMarkers to customize points.

Value

Leaflet map with coordinates added as dots.

See Also

Other GIS_utilities: guess_crs(), transform_coordinates()

Examples

```
## Not run:
library(sf)
data_pts <- data.frame(
  id = c(1, 2, 3),
  lat = c(51.23031, 50.76931, 50.21439),
  lon = c(5.083980, 3.829593, 3.289044),
  stringsAsFactors = FALSE
)

# projection is of class CRS-class (sp)
if (requireNamespace("sp")) {
  proj_crs_sp <- CRS("+init=epsg:4269")
  plot_coordinates_on_map(data_pts, "lon", "lat", proj_crs_sp)
}

# projection is of class crs-class (sf)
proj_crs_sf <- st_crs(4269)
plot_coordinates_on_map(data_pts, "lon", "lat", proj_crs_sf)

# customize circles
plot_coordinates_on_map(data_pts, "lon", "lat", proj_crs_sf,
  radius = 5, color = "red", stroke = FALSE, fillOpacity = 0.75
)

## End(Not run)
```

`plot_label_splitter` *Split a long (plot) label into a two-liner*

Description

When a label exceeds the maximum length as defined by the user, the function splits the text/label on the first space after the centre of the string. This is very useful for long labels in a plot

Usage

```
plot_label_splitter(label, maxlength)
```

Arguments

`label`	A character array (the label)
`maxlength`	When the label exceeds this length, it will be split into a two-line label

Details

The function can be easily applied to a `data.frame` by using e.g. `sapply`

Examples

```
plot_label_splitter("Exotic long label", 10)
```

---

rain_knmi_2012 *Example* data.frame *with* KNMI *downloaded data*

Description

A dataset containing the rainfall from January 1st till February 1st for Vlissingen and Westdorpe, as downloaded from KNMI

Usage

```
rain_knmi_2012
```

Format

A data.frame with 1536 rows and 9 variables:

- value: measured value
- datetime: datetime of the measurement
- unit: unit (mm)
- variable_name: precipitation
- longitude: coordinate
- latitude: coordinate
- location_name: station name
- source_filename: filename from which the data was read
- quality_code: empty string as KNMI does not provide this

See Also

Other datasets: coordinate_example, species_example

read_kmi_data *Load* KMI *data as data.frame*

Description

Load KMI data as data.frame

Usage

```
read_kmi_data(filename, n_max = Inf)
```

Arguments

filename	path/filename of the KMI datafile
n_max	integer; shortcut for testing, loading just a section of the data

Value

data.frame with the headers location_name, datetime (UTC), value, unit, variable_name, latitude, longitude and source_filename

See Also

Other download_functions: download_knmi_data_hour(), download_zenodo(), extract_soil_map_data(), read_kml_file(), read_knmi_data(), read_mow_data()

Examples

```
## Not run:
# see vignettes for more examples
file_path <- "kmi_file.txt"
precipation_kmi <- read_kmi_data(file_path)

## End(Not run)
```

read_kml_file *Extract date and coordinate information from Google Maps* kml *file*

Description

Extract date and coordinate information from Google Maps kml file

Usage

```
read_kml_file(filename)
```

Arguments

filename	(char) filename/path of the kml file

Value

data.frame with columns datetime, x (numeric), y (numeric)

See Also

Other download_functions: download_knmi_data_hour(), download_zenodo(), extract_soil_map_data(), read_kmi_data(), read_knmi_data(), read_mow_data()

Examples

```
## Not run:
df <- read_kml_file("BE1002.kml")

## End(Not run)
```

---

read_knmi_data *Read* KNMI *datafile (precipitation) as* data.frame

---

Description

Remark that this function is specifically to extract the precipitation (RH) data, coordinates are extracted from the header of the KNMI data file

Usage

```
read_knmi_data(filename, n_max = Inf)
```

Arguments

filename	path/filename of the KNMI datafile
n_max	integer; shortcut for testing, loading just a section of the data

Value

data.frame with the headers location_name, datetime (UTC), value, unit, variable_name, latitude, longitude and source_filename

See Also

Other download_functions: download_knmi_data_hour(), download_zenodo(), extract_soil_map_data(), read_kmi_data(), read_kml_file(), read_mow_data()

Examples

```
## Not run:
# see vignettes for more examples
file_path <- "knmi_file.txt"
knmi_data <- read_knmi_data(file_path)

## End(Not run)
```

`read_mow_data` *Read MOW data*

Description

Coordinate data is contained in the header of the file

Usage

```
read_mow_data(filename, n_max = Inf)
```

Arguments

`filename`	path/filename of the RWS datafile
`n_max`	integer; shortcut for testing, loading just a section of the data

Value

`data.frame` with the headers `location_name`, `datetime` (UTC), value, `unit`, `variable_name`, `latitude`, `longitude` and `source_filename`

See Also

Other download_functions: `download_knmi_data_hour()`, `download_zenodo()`, `extract_soil_map_data()`, `read_kmi_data()`, `read_kml_file()`, `read_knmi_data()`

Examples

```
## Not run:
# see vignettes for more examples
file_path <- "mow_file.txt"
mow_data <- read_mow_data(file_path)

## End(Not run)
```

setup_codingclub_session *Download files (code and data) for a specific coding club session*

Description

This function will populate the content of `/src` and `/data` directories for a specific coding club session, the date of which is passed as a parameter. Directories are created if needed. Files are downloaded if not already present.

Usage

```
setup_codingclub_session(
  session_date = format(Sys.Date(), "%Y%m%d"),
  root_dir = ".",
  src_rel_path = "src",
  data_rel_path = "data"
)
```

Arguments

`session_date`	The date of the coding-club session, in the `"YYYYMMDD"` format. Default: actual date
`root_dir`	Root directory where source and data subdirectories are located. Default: `./`
`src_rel_path`	Relative path for R script(s). Default: `src`
`data_rel_path`	Relative path for data files. Default: `data`

Examples

```
## Not run:
library(inborutils)

# Download coding club files for session 2020-02-25
# R script(s) in `./src/20200225` and data files in `./data/20200225`
setup_codingclub_session("20200225")

# Specify the folders where you want to save R files and data files
# R files will be downloaded in `./source`
# Data files will be downloaded in `./dataframes`
setup_codingclub_session(
  "20200225",
  src_rel_path = "source",
  data_rel_path = "dataframes"
)

# You can modify the root directory even if it should be normally not needed.
# For example you want to move the root to the parent directory, `../`
setup_codingclub_session("20200225",
```

```
  root_dir = "../",
  src_rel_path = "source",
  data_rel_path = "dataframes"
)

## End(Not run)
```

`species_example` *Example* `data.frame` *with species name column*

Description

A dataset containing 3 taxa to be matched with GBIF Taxonomy Backbone. The variable are as follows:

Usage

```
species_example
```

Format

A data frame with 3 rows and 3 variables

- `speciesName`: name of the species
- `kingdom`: kingdom to which the species belongs
- `euConcernStatus`: level of concern according to EU directives

See Also

Other datasets: `coordinate_example`, `rain_knmi_2012`

`transform_coordinates` *Transform x-y coordinates from* `data.frame` *columns.*

Description

This function extracts x-y coordinates from a `data.frame` by means of the given coordinate reference system (`CRS`), transforms them to the new `CRS` and assign them back to the `data.frame` columns.

Usage

```
transform_coordinates(df, col_x, col_y, crs_input, crs_output)
```

Arguments

df	A data.frame with a x and y coordinate column.
col_x, col_y	Column names or positions of the x and y column. They are passed to vars_pull. These arguments are passed by expression and support quasiquotation (you can unquote column names or column positions).
crs_input	Projection string of class CRS-class (sp compatible) or crs-class (sf compatible) defining the current CRS.
crs_output	Projection string of class CRS-class (sp compatible) or crs-class (sf compatible) defining the CRS to convert to.

Value

A data.frame with the same columns, but transformed coordinates for the x and y column values.

See Also

Other GIS_utilities: guess_crs(), plot_coordinates_on_map()

Examples

```
library(sf)
data_pts <- data.frame(
  id = c(1, 2),
  lat = c(51.23031, 50.76931),
  lon = c(5.083980, 3.829593),
  stringsAsFactors = FALSE
)

# CRS-class (use sp package)
if (requireNamespace("sp")) {
  sp_crs1 <- sp::CRS("+init=epsg:4269")
  sp_crs2 <- sp::CRS("+init=epsg:3857")
  transform_coordinates(data_pts,
    col_x = "lon", col_y = "lat",
    crs_input = sp_crs1, crs_output = sp_crs2
  )
}
# crs-class (use sf package)
sf_crs1 <- st_crs(4269)
sf_crs2 <- st_crs(3857)
transform_coordinates(data_pts,
  col_x = "lon", col_y = "lat",
  crs_input = sf_crs1, crs_output = sf_crs2
)

if (requireNamespace("sp")) {
  # input projection is CRS-class (sp) and output projection crs-class (sf)
  transform_coordinates(data_pts,
    col_x = "lon", col_y = "lat",
    crs_input = sp_crs1, crs_output = sf_crs2
```

```
  )
  # input projection is crs-class (spf and output projection CRS-class (sp)
  transform_coordinates(data_pts,
    col_x = "lon", col_y = "lat",
    crs_input = sf_crs1, crs_output = sp_crs2
  )
}

# use names (character) of x-y columns
transform_coordinates(data_pts,
  col_x = "lon", col_y = "lat",
  crs_input = sf_crs1, crs_output = sf_crs2
)
# use NSE of x-y columns
transform_coordinates(data_pts,
  col_x = lon, col_y = lat,
  crs_input = sf_crs1, crs_output = sf_crs2
)
# use position of x-y columns
transform_coordinates(data_pts,
  col_x = 3, col_y = 2,
  crs_input = sf_crs1, crs_output = sf_crs2
)
```

`vif` *Variance inflation factor*

Description

Variance inflation factor

Variance inflation factor (`lm` model)

Variance inflation factor (data.frame)

Variance inflation factor (default routine)

Usage

```
vif(mod, ...)

## S3 method for class 'lm'
vif(mod, ...)

## S3 method for class 'data.frame'
vif(mod, ...)

## Default S3 method:
vif(mod, ...)
```

Arguments

`mod`	A model object (or data.frame). For the moment only `lm` and a plain dataset is implemented
`...`	not used

Value

a matrix with for each variable de variance inflation factor, the degrees of freedom en the rescaled variance inflation factor based on the degrees of freedom.

Examples

```
{
vif(airquality)
}
```

Index