Package 'instatClimatic'

December 3, 2024

Title A set of climatic functions used in R-Instat

Version 0.2.0

Description This package contains a set of climatic functions used in R-Instat. This includes functions written specifically for R-Instat, as well functions written elsewhere that are modified for use in R-Instat.

License LGPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports climdex.pcic,
colorRamps,
data.table,
distillery,
dplyr,
extRemes,
fields,
forcats,
ggplot2,
lubridate,
magrittr,
maps,
methods,
ncdf4,
ncdf4.helpers,
openair,
patchwork,
PCICt,
plyr,
purrr,
raster,
reshape2,
rlang,
sf,
sp,
stringr,
texmex,
tibble,
tidyr,

tidyselect,
zoo

Remotes pacificclimate/climdex.pcic

Contents

`add_nc` *Modify a path*

Description

: This function takes a path as input and returns a modified path by appending "data.nc" to it.

Usage

```
add_nc(path)
```

Arguments

`path`	A character string representing the file path.

Value

: A character string representing the modified file path with "data.nc" appended to it.

Examples

```
add_nc("my_folder/")  # Returns "my_folder/data.nc"
add_nc("/path/to/file.txt")  # Returns "/path/to/file.txtdata.nc"
add_nc("")  # Returns "data.nc"
```

`add_t_range` *add time range*

Description

This function generates a string representing a time range to be added to a given path.

Usage

```
add_t_range(path, min_date, max_date, dim_t = "T")
```

Arguments

`path`	The base path to which the time range will be added.
`min_date`	The minimum date of the time range.
`max_date`	The maximum date of the time range.
`dim_t`	The dimension of time to be included in the range (default: "T").

Value

A string representing the path with the time range added.

Examples

```
# Example 1: Generate a time range string with default dimension
#path <- "http://example.com/"
#min_date <- lubridate::ymd("2023-01-01")
#max_date <- lubridate::ymd("2023-12-31")
#t_range <- add_t_range(path, min_date, max_date)
```

---

add_xy_area_range *Add xy area range*

Description

A function that generates a string representing an XY area range to be added to a given path.

Usage

```
add_xy_area_range(
  path,
  min_lon,
  max_lon,
  min_lat,
  max_lat,
  dim_x = "X",
  dim_y = "Y"
)
```

Arguments

path	The base path to which the XY area range will be added.
min_lon	The minimum longitude of the area range.
max_lon	The maximum longitude of the area range.
min_lat	The minimum latitude of the area range.
max_lat	The maximum latitude of the area range.
dim_x	The dimension for longitude (default: "X").
dim_y	The dimension for latitude (default: "Y").

Value

A string representing the path with the XY area range added.

Examples

```
#Example: Generate an XY area range string with custom dimensions
#path <- "http://example.com"
#min_lon <- -90
#max_lon <- -80
#min_lat <- 30
#max_lat <- 40
#xy_range <- add_xy_area_range(path, min_lon, max_lon, min_lat,
#                              max_lat,dim_x = "LON", dim_y = "LAT")
```

`add_xy_point_range` *add xy area range*

Description

This function generates a file path for XY point range data based on the provided parameters.

Usage

```
add_xy_point_range(path, min_lon, min_lat, dim_x = "X", dim_y = "Y")
```

Arguments

`path`	(character) The base path where the data file will be stored.
`min_lon`	(numeric) The minimum longitude value for the range.
`min_lat`	(numeric) The minimum latitude value for the range.
`dim_x`	(character) The name of the X dimension. (Default: "X")
`dim_y`	(character) The name of the Y dimension. (Default: "Y")

Value

The generated file path for the XY point range data as a character string.

Examples

```
add_xy_point_range("data", -90, 30, "X", "Y")
```

climatic_details *Climatic Details*

Description

This function extracts climatic details from a given dataset based on specified parameters such as date, elements, stations, and level. It provides information about the duration and frequency of missing values for each element at different levels (day, month, year).

Usage

```
climatic_details(
  data,
  date,
  elements = ...,
  stations,
  order = TRUE,
  day = FALSE,
  month = TRUE,
  year = FALSE,
  level = FALSE
)
```

Arguments

data	A dataset containing climatic data.
date	The date variable in the dataset.
elements	A character vector specifying the climatic elements to analyse.
stations	A character vector specifying the stations to analyse.
order	A logical value indicating whether the resulting tables should be ordered.
day	A logical value indicating whether to include information at the day level.
month	A logical value indicating whether to include information at the month level.
year	A logical value indicating whether to include information at the year level.
level	A logical value indicating whether to include the level information in the output.

Value

A data frame containing climatic details such as start and end dates of missing values and the count of missing values for each element at the specified levels.

Examples

```
#climatic_details(data = climatic_data,
#                 date = "Date",
#                 elements = c("Temperature", "Precipitation"),
#                 stations = c("Station1", "Station2"),
#                 order = TRUE,
#                 day = TRUE,
#                 month = FALSE,
#                 year = TRUE,
#                 level = TRUE)
```

`climatic_missing` *Summarise missing climatic values*

Description

This function calculates the summary of missing data in climatic variables for different stations and elements within a specified time period. It provides information on the start and end dates of missing data, the total number of missing values, and the percentage of missing values for each station and element. Additionally, it counts the number of complete years (without any missing values) for each station and element.

Usage

```
climatic_missing(
  data,
  date,
  elements = ...,
  stations,
  start = TRUE,
  end = FALSE
)
```

Arguments

`data`	A data frame containing climatic data.
`date`	A variable representing the date or time in the data frame.
`elements`	A character vector specifying the climatic elements to be analyzed. Defaults to all elements in the data frame.
`stations`	A character vector specifying the stations to be analyzed. If not provided, the analysis will be performed for all stations in the data frame.
`start`	A logical value indicating whether to determine the start date of missing data for each station and element. If set to TRUE, the function finds the first date with missing values. If set to FALSE, the function uses the first date in the specified time period.
`end`	A logical value indicating whether to determine the end date of missing data for each station and element. If set to TRUE, the function finds the last date with missing values. If set to FALSE, the function uses the last date in the specified time period.

Value

A data frame containing the summary of missing data, including the start and end dates, the total number of missing values, the percentage of missing values, and the number of complete years for each station and element.

Examples

```
# TODO
```

`climdex` *Calculate climate indices*

Description

This function calculates climate indices based on temperature and precipitation data.

Usage

```
climdex(
  data,
  station,
  date,
  year,
  month,
  prec,
  tmax,
  tmin,
  indices,
  freq = "annual",
  base.range = c(1961, 1990),
  n = 5,
  northern.hemisphere = TRUE,
  quantiles = NULL,
  temp.qtiles = c(0.1, 0.9),
  prec.qtiles = c(0.95, 0.99),
  max.missing.days = c(annual = 15, monthly = 3),
  min.base.data.fraction.present = 0.1,
  spells.can.span.years = FALSE,
  gsl.mode = "GSL",
  threshold = 1
)
```

Arguments

`data`	A data frame containing the temperature and precipitation data.
`station`	The name of the column in the data frame that represents the station.
`date`	The name of the column in the data frame that represents the date.
`year`	The name of the column in the data frame that represents the year.
`month`	The name of the column in the data frame that represents the month.
`prec`	The name of the column in the data frame that represents precipitation.
`tmax`	The name of the column in the data frame that represents maximum temperature.
`tmin`	The name of the column in the data frame that represents minimum temperature.
`indices`	A character vector specifying the climate indices to be calculated.
`freq`	The frequency at which the indices should be calculated. Can be "annual" or "monthly". Defaults to "annual".
`base.range`	A numeric vector specifying the base range for calculating indices. Defaults to c(1961, 1990).

`n` The number of years to be considered for calculating percentile-based indices. Defaults to 5.

`northern.hemisphere` A logical value indicating whether the data is from the Northern Hemisphere. Defaults to TRUE.

`quantiles` A numeric vector specifying the quantiles for calculating percentile-based indices. Defaults to NULL.

`temp.qtiles` A numeric vector specifying the quantiles for temperature-based indices. Defaults to c(0.1, 0.9).

`prec.qtiles` A numeric vector specifying the quantiles for precipitation-based indices. Defaults to c(0.95, 0.99).

`max.missing.days` A numeric vector specifying the maximum number of missing days allowed for each frequency. Defaults to c(annual = 15, monthly = 3).

`min.base.data.fraction.present` The minimum fraction of base data required for calculating indices. Defaults to 0.1.

`spells.can.span.years` A logical value indicating whether climate spells can span across years. Defaults to FALSE.

`gsl.mode` The method to calculate the growing season length. Can be "GSL" or "ASL". Defaults to "GSL".

`threshold` The threshold for calculating wet and dry spell indices. Defaults to 1.

Value

A data frame containing the calculated climate indices.

Examples

```
# data <- read.csv("climate_data.csv")
# indices <- c("fd", "su", "id", "tr", "wsdi", "csdi", "gsl", "sdii",
# "r10mm", "r20mm", "rnnmm", "cdd", "cwd", "r95ptot", "r99ptot", "prcptot")
# climdex(data, station = "station_name", date = "date", year = "year",
# month = "month", prec = "precipitation",
#          tmax = "max_temperature",
#          tmin = "min_temperature",
#          indices = indices)
```

`climdex_single_station` *climdex_single_station*

Description

This function calculates climate indices for a single weather station based on the provided parameters. The function utilizes the climdex.pcic package to compute various climate indices. The indices can be calculated either on an annual or monthly basis, depending on the specified frequency.

Usage

```
climdex_single_station(
  ci,
  freq = "annual",
  indices,
  year,
  month,
  spells.can.span.years = FALSE,
  gsl.mode = gsl.mode,
  threshold = 1
)
```

Arguments

ci	The climate data for the weather station.
freq	The frequency at which the climate indices should be calculated. It can be either "annual" or "monthly" (default: "annual").
indices	A character vector specifying the climate indices to be calculated.
year	The column name or index of the year in the climate data.
month	The column name or index of the month in the climate data. Required only if freq = "monthly".
spells.can.span.years	A logical value indicating whether the spells can span multiple years. This parameter is used by certain indices that involve consecutive days, such as "wsdi", "csdi", "cdd", and "cwd" (default: FALSE).
gsl.mode	The mode used to calculate growing season length (GSL) index. It is used by the "gsl" index. Please refer to the documentation of climdex.pcic::climdex.gsl for details.
threshold	A threshold value used by the "rnnmm" index to calculate the number of consecutive wet days above the threshold (default: 1).

Value

A data frame containing the calculated climate indices for the specified station and time period.

Examples

```
#data <- read.csv("climate_data.csv")
#climdex_single_station(data, freq = "annual", indices = c("fd", "su"),
# year = "Year")
```

---

convert_SST *Convert SST Data*

Description

This function converts SST data from a given file into a structured format. It extracts information such as start year, end year, duration, longitude, latitude, and SST values from the data file. The function then creates a data frame with the extracted information and returns it along with the latitude-longitude data frame.

Usage

```
convert_SST(datafile, data_from = 5)
```

Arguments

datafile	A data file containing SST data.
data_from	The row index from where the SST data starts in the data file. Default is 5.

Value

A list containing the converted data frame and the latitude-longitude data frame.

Examples

```
# Example usage:
# Assuming the datafile is "sst_data.csv" and the SST data starts from row 5
#converted_data <- convert_SST("sst_data.csv", data_from = 5)

# Access the converted data frame and latitude-longitude data frame
#my_data <- converted_data[[1]]
#lat_lon_df <- converted_data[[2]]

# Print the converted data frame
```

---

convert_to_character_matrix

Convert Data to Character Matrix

Description

This function converts the input data to a character matrix, with options for formatting decimal places and handling missing values.

Usage

```
convert_to_character_matrix(
  data,
  format_decimal_places = TRUE,
  decimal_places,
  is_scientific = FALSE,
  return_data_frame = TRUE,
  na_display = NULL,
  check.names = TRUE
)
```

Arguments

data	The input data to be converted.
format_decimal_places	A logical value indicating whether decimal places should be formatted. Default is TRUE.

decimal_places	A numeric vector specifying the number of decimal places for each column. If not provided, it uses the function get_default_significant_figures to determine the number of decimal places.
is_scientific	A logical vector indicating whether scientific notation should be used for each column. Default is FALSE.
return_data_frame	A logical value indicating whether the result should be returned as a data frame. Default is TRUE.
na_display	A character value specifying how missing values should be displayed. If not provided, missing values are left as is.
check.names	A logical value indicating whether column names should be checked and modified if necessary. Default is 'TRUE'.

Value

The converted data as a character matrix or data frame, depending on the value of return_data_frame.

See Also

get_default_significant_figures

Examples

```
data <- data.frame(A = c(1.234, 5.678), B = c(10.123, 20.456))
converted <- convert_to_character_matrix(data)
print(converted)
```

---

convert_to_dec_deg *convert to decimal degrees*

Description

This function converts coordinates given in degrees, minutes, and seconds format to decimal degrees.

Usage

```
convert_to_dec_deg(dd, mm = 0, ss = 0, dir)
```

Arguments

dd	The degrees component of the coordinate.
mm	The minutes component of the coordinate (default: 0).
ss	The seconds component of the coordinate (default: 0).
dir	The direction of the coordinate, indicating whether it is east (E), west (W), north (N), or south (S).

Value

The converted coordinate in decimal degrees.

Examples

```
convert_to_dec_deg(45, 30, 30, "N") # Returns 45.508333
convert_to_dec_deg(122, 10, 0, "W") # Returns -122.166667
```

---

convert_yy_to_yyyy *Convert Two-Digit Year to Four-Digit Year*

Description

This function converts a two-digit year to a four-digit year based on a given base year. It adds 2000 to the two-digit year if it is less than or equal to the base year, otherwise, it adds 1900 to the two-digit year.

Usage

```
convert_yy_to_yyyy(x, base)
```

Arguments

x	A numeric vector of two-digit years.
base	The base year used as a reference for the conversion.

Value

A numeric vector of four-digit years.

Examples

```
# Example usage:
# Convert two-digit years to four-digit years based on the base year 1990
years <- c(92, 98, 04, 88)
converted_years <- convert_yy_to_yyyy(years, base = 1990)
print(converted_years)
# Output: 1992 1998 2004 1988
```

---

cumulative_inventory *Cumulative Inventory*

Description

Adds a cumulative count of missing values by period (and station if specified) to the data.

Usage

```
cumulative_inventory(data, station = NULL, from, to)
```

Arguments

data	The input data frame.
station	An optional column name indicating the station. Defaults to NULL.
from	The start period column name.
to	The end period column name.

Value

A data frame with added cumulative counts.

Examples

```
## Not run:
  cumulative_inventory(my_data, "station", "start_date", "end_date")

## End(Not run)
```

---

dd_to_dms *Convert decimal degrees to DMS format*

Description

Converts decimal degrees to degrees, minutes, and seconds (DMS) format.

Usage

```
dd_to_dms(x, lat)
```

Arguments

x	Numeric value representing the decimal degree to be converted.
lat	Logical value indicating whether the conversion is for latitude (TRUE) or longitude (FALSE).

Value

A character string representing the input decimal degree in DMS format. The format is "DD MM SS Dir", where DD represents degrees, MM represents minutes, SS represents seconds, and Dir represents the direction (N, S, E, or W). The degrees, minutes, and seconds are zero-padded to two digits each.

Examples

```
dd_to_dms(37.7749, 'TRUE')
# Output: "37 46 29 N"

dd_to_dms(-122.4194, 'FALSE')
# Output: "122 25 10 W"
```

dekad *Get the dekad component of a date-time object*

Description

Convert a date or date-time object to a yearly dekad (10-day period)

Usage

```
dekad(date)
```

Arguments

date	A date-time object

Value

a numerical vector of dekad objects corresponding to date variable.

Examples

```
dekad(as.Date("2020/12/25"))
dekad(as.Date("1999/01/01"))
```

fourier_series *Fourier Series*

Description

This function calculates the Fourier series for a given input x with a specified number of terms n and period period.

Usage

```
fourier_series(x, n, period)
```

Arguments

x	The input value for which the Fourier series is calculated.
n	The number of terms in the Fourier series.
period	The period of the Fourier series.

Value

The Fourier series expression as a character string.

Examples

```
fourier_series(3, 5, 2)
```

get_default_significant_figures *Get the default number of significant figures*

Description

This function gets the default number of significant figures when given a numeric vector.

Usage

```
get_default_significant_figures(data)
```

Arguments

data	numeric(1) A numerical vector

Value

If the data is numeric, "3", otherwise NA.

Examples

```
x <- 1:8
get_default_significant_figures(x)
```

get_lat_from_data *Get Latitude values from a Data file*

Description

This function takes a data file as input and extracts the latitude values from it. The latitude values should be present in the first column of the data file, starting from the 5th row. The function removes any missing or non-numeric values and returns a vector of unique latitude values.

Usage

```
get_lat_from_data(datafile)
```

Arguments

datafile	A data file containing latitude values. The latitude values should be present in the first column, starting from the 5th row.

Value

A vector of unique latitude values extracted from the data file.

Examples

```
# Example data file: mydata.csv
# Get latitude values from the data file
# data_file <- read.csv("mydata.csv")
# get_lat_from_data(data_file)
```

---

get_lon_from_data *Get Longitude coordinates from Data*

Description

This function takes a data file as input and extracts the longitude values from it. The longitude values should be present in the second column of the fifth row. The function removes any missing or non-numeric values and returns a vector of unique longitude values. The function returns a vector of unique longitude values extracted from the data file. It removes any missing or non-numeric values using the `na.omit()` function and transposes the extracted values using the `t()` function. The function is also exported, making it available for use outside the current package or script.

Usage

```
get_lon_from_data(datafile)
```

Arguments

`datafile`	A data file containing longitude values. The longitude values should be present in the second column of the fifth row.

Value

A vector of unique longitude values extracted from the data file.

Examples

```
# Example data file: mydata.csv
# Get longitude values from the data file
# data_file <- read.csv("mydata.csv")
# get_lon_from_data(data_file)
```

---

get_quarter_label *Get Quarter of the year Label*

Description

This function returns a three-letter string representing a specific quarter in a year.

Usage

```
get_quarter_label(quarter, start_month)
```

Arguments

quarter	The quarter number. Must be in the range of 1 to 4.
start_month	The starting month of the year. Must be in the range of 1 to 12.

Value

A factor object containing three-letter strings representing the specified quarters.

Examples

```
# Get quarter label starting from January
# get_quarter_label(quarter = 1, start_month = 1)

# Get quarter label starting from April
# get_quarter_label(quarter = 2, start_month = 4)

# Get quarter label starting from October
# get_quarter_label(quarter = 4, start_month = 10)
```

---

get_years_from_data *Get year from data*

Description

Extracts unique years from a data file.

Usage

```
get_years_from_data(datafile)
```

Arguments

datafile	A data file or data frame.

Value

A vector of unique years extracted from the data file.

Examples

```
# # Sample data file
# datafile <- data.frame(
#   Name = c("John", "Alice", "Bob"),
#   Age = c(25, 30, 35),
#   "2019" = c(100, 150, 200),
#   "2020" = c(200, 300, 400),
#   "2021" = c(300, 450, 600)
# )
#
# # Get years from data file
# years <- get_years_from_data(datafile)
# years
#
```

ggwalter_lieth *Generate Walter-Lieth Plot*

Description

Generates Walter-Lieth climate diagrams using ggplot2.

Usage

```
ggwalter_lieth(
  data,
  month,
  station = NULL,
  p_mes,
  tm_max,
  tm_min,
  ta_min,
  station_name = "",
  alt = NA,
  per = NA,
  pcol = "#002F70",
  tcol = "#ff0000",
  pfcol = "#9BAEE2",
  sfcol = "#3C6FC4",
  shem = FALSE,
  p3line = FALSE,
  ...
)
```

Arguments

data	The input data frame.
month	The column name for months.
station	An optional column name for station. Defaults to NULL.
p_mes	The column name for precipitation measurements.
tm_max	The column name for maximum temperature.
tm_min	The column name for minimum temperature.
ta_min	The column name for average temperature.
station_name	The name of the station. Defaults to an empty string.
alt	The altitude. Defaults to NA.
per	The period. Defaults to NA.
pcol	The color for precipitation lines. Defaults to "#002F70".
tcol	The color for temperature lines. Defaults to "#ff0000".
pfcol	The fill color for probable freeze areas. Defaults to "#9BAEE2".
sfcol	The fill color for sure freeze areas. Defaults to "#3C6FC4".
shem	Logical indicating whether the station is in the southern hemisphere. Defaults to FALSE.
p3line	Logical indicating whether to plot the precipitation/3 line. Defaults to FALSE.
...	Additional arguments to be passed to ggplot2 functions.

Value

A ggplot2 object with the Walter-Lieth climate diagram.

Examples

```
## Not run:
  ggwalter_lieth(my_data, "Month", NULL, "Precipitation",
  "MaxTemp", "MinTemp", "AvgTemp")

## End(Not run)
```

---

import_from_iri *Import Data from IRI (International Research Institute for Climate and Society)*

Description

This function imports data from various sources at the International Research Institute for Climate and Society (IRI) based on the specified parameters.

Usage

```
import_from_iri(download_from, data_file, path, X1, X2, Y1, Y2, get_area_point)
```

Arguments

download_from	The source from which to download the data. Supported values are "CHIRPS_V2P0", "TAMSAT", "NOAA_ARC2", "NOAA_RFE2", "NOAA_CMORPH_DAILY", "NOAA_CMORPH_3HOURLY", "NOAA_CMORPH_DAILY_CALCULATED", "NASA_TRMM_3B42".
data_file	The specific data file to download from the selected source.
path	The directory path where the downloaded file will be saved. If empty, the current working directory is used.
X1	The starting longitude or X-coordinate of the desired data range.
X2	The ending longitude or X-coordinate of the desired data range.
Y1	The starting latitude or Y-coordinate of the desired data range.
Y2	The ending latitude or Y-coordinate of the desired data range.
get_area_point	Specifies whether the data should be downloaded for an "area" or a single "point".

Value

A list containing two elements:

- The imported data as a data frame.
- A data frame containing unique latitude and longitude values from the imported data.

Examples

```
# Import area data from CHIRPS_V2P0 source
# import_from_iri(download_from = "CHIRPS_V2P0", data_file = "daily_0p05",
#                 path = "data", X1 = -10, X2 = 10, Y1 = -10, Y2 = 10,
#                 get_area_point = "area")

# Import point data from TAMSAT source
# import_from_iri(download_from = "TAMSAT", data_file = "rainfall_estimates",
#                 path = "", X1 = -1, Y1 = 50, get_area_point = "point")
```

---

lat_lon_dataframe *Create a dataframe with latitude and longitude information from a data file.*

Description

This function creates a data frame with latitude and longitude information from a data file.

Usage

```
lat_lon_dataframe(datafile)
```

Arguments

datafile	The path or name of the data file.

Value

A data frame containing latitude, longitude, and station information.

Examples

```
# Create a latitude-longitude data frame
# datafile <- "data.csv"
# lat_lon_dataframe(datafile)
```

---

make_factor *Make factor*

Description

Creates a factor variable from the input data, with optional specification of ordering.

Usage

```
make_factor(x, ordered = is.ordered(x))
```

Arguments

x	The input data to be converted into a factor variable.
ordered	Logical value indicating whether the resulting factor should be ordered or not.

Value

A factor variable generated from the input data. If the input data is already a factor, and the ordered parameter is consistent with the existing ordering, the input data is returned as is. Otherwise, the input data is converted into a factor variable with the specified ordering.

Examples

```
# Create a factor from a numeric vector
#make_factor(c(1, 2,3,3,2,2,1,1,1, 3, 2), ordered = TRUE)
# Output: 1 2 3 2
# Levels: 1 < 2 < 3

# Create a factor from a logical vector
#make_factor(c(TRUE, FALSE,FALSE,TRUE,TRUE, TRUE), ordered = FALSE)
# Output: TRUE FALSE TRUE
# Levels: FALSE TRUE

# Create a factor from a character vector
#make_factor(c("apple", "banana", "apple", "orange"), ordered = FALSE)
# Output: apple banana apple orange
# Levels: apple banana orange

# Create a factor from an unsupported data type
#make_factor(Sys.time(), ordered = FALSE)
# Output: Error: The input data type is not supported for factor creation.
```

---

multiple_nc_as_data_frame

Convert multiple netCDF files to a single data frame

Description

This function reads multiple netCDF files from a specified path and converts them into a single data frame. It allows the user to specify the variables of interest, whether to keep the raw time values, and include metadata. Additionally, the function provides options for subsetting the data based on a boundary, specific lon/lat points, or an ID variable. The resulting data frame contains the merged data from all netCDF files.

Usage

```
multiple_nc_as_data_frame(
  path,
  vars,
  keep_raw_time = TRUE,
  include_metadata = TRUE,
  boundary = NULL,
  lon_points = NULL,
```

```
  lat_points = NULL,
  id_points = NULL,
  show_requested_points = TRUE,
  great_circle_dist = TRUE,
  id = "id"
)
```

Arguments

path	The path to the directory containing the netCDF files.
vars	The names of the variables of interest in the netCDF files.
keep_raw_time	If TRUE, keeps the raw time values as a separate column in the data frame. Default is TRUE.
include_metadata	If TRUE, includes metadata information in the data frame. Default is TRUE.
boundary	An optional boundary to subset the data. It should be a list with elements "lon_min", "lon_max", "lat_min", and "lat_max".
lon_points	An optional vector of specific longitudes to subset the data.
lat_points	An optional vector of specific latitudes to subset the data.
id_points	An optional vector of specific ID points to subset the data.
show_requested_points	If TRUE, includes a column indicating whether the requested lon/lat points are within the data. Default is TRUE.
great_circle_dist	If TRUE, uses great circle distance calculation for subsetting based on lon/lat points. Default is TRUE.
id	The name of the ID column in the merged data frame. Default is "id".

Value

The merged data frame containing the data from all netCDF files.

Examples

```
# Example usage
# path <- "path/to/netcdf/files"
# vars <- c("temperature", "precipitation")
# data <- multiple_nc_as_data_frame(path, vars)

# Example usage with additional parameters
# boundary <- list(lon_min = -180, lon_max = 180,
#                  lat_min = -90, lat_max = 90)
# lon_points <- c(-120, -100, -80)
# lat_points <- c(30, 40, 50)
# id_points <- c("A", "B", "C")
# data <- multiple_nc_as_data_frame(path, vars, keep_raw_time = FALSE,
#         include_metadata = FALSE, boundary = boundary,
#         lon_points = lon_points, lat_points = lat_points,
#         id_points = id_points, show_requested_points = FALSE,
#         great_circle_dist = FALSE, id = "station_id")
```

na_check *Check Missing Values Based on Conditions*

Description

Evaluates a vector against specified conditions for missing values.

Usage

```
na_check(
  x,
  na_type = c(),
  na_consecutive_n = NULL,
  na_max_n = NULL,
  na_max_prop = NULL,
  na_min_n = NULL,
  na_FUN = NULL,
  ...
)
```

Arguments

x	A vector to check for missing values.
na_type	A character vector specifying the types of checks to perform. Options include: • "n": Total number of missing values (<= na_max_n). • "prop": Proportion of missing values (<= na_max_prop in percentage). • "n_non_miss": Minimum number of non-missing values (>= na_min_n). • "FUN": A custom function to evaluate missing values. • "con": Maximum consecutive missing values (<= na_consecutive_n).
na_consecutive_n	Optional. Maximum allowed consecutive missing values.
na_max_n	Optional. Maximum allowed missing values.
na_max_prop	Optional. Maximum allowed proportion of missing values (in percentage).
na_min_n	Optional. Minimum required non-missing values.
na_FUN	Optional. A custom function to evaluate missing values.
...	Additional arguments passed to the custom function na_FUN.

Value

Logical. Returns TRUE if all specified checks pass, otherwise FALSE.

`nc_as_data_frame` *Convert netCDF data to a data frame*

Description

This function converts netCDF data into a data frame. It allows the user to specify the variables of interest, whether to keep the raw time values, and include metadata. Additionally, the function provides options for subsetting the data based on a boundary, specific lon/lat points, or an ID variable. The resulting data frame contains the selected variables and corresponding values.

Usage

```
nc_as_data_frame(
  nc,
  vars,
  keep_raw_time = TRUE,
  include_metadata = TRUE,
  boundary = NULL,
  lon_points = NULL,
  lat_points = NULL,
  id_points = NULL,
  show_requested_points = TRUE,
  great_circle_dist = TRUE
)
```

Arguments

`nc`	The netCDF object.
`vars`	The names of the variables of interest in the netCDF object.
`keep_raw_time`	If TRUE, keeps the raw time values as a separate column in the data frame. Default is TRUE.
`include_metadata`	If TRUE, includes metadata information in the data frame. Default is TRUE.
`boundary`	An optional boundary to subset the data. It should be a list with elements "lon_min", "lon_max", "lat_min", and "lat_max".
`lon_points`	An optional vector of specific longitudes to subset the data.
`lat_points`	An optional vector of specific latitudes to subset the data.
`id_points`	An optional vector of specific ID points to subset the data.
`show_requested_points`	If TRUE, includes a column indicating whether the requested lon/lat points are within the data. Default is TRUE.
`great_circle_dist`	If TRUE, uses great circle distance calculation for subsetting based on lon/lat points. Default is TRUE.

Value

The data frame containing the selected variables and their values from the netCDF data.

Examples

```
# Example usage
# nc <- ncdf4::nc_open("path/to/netcdf/file.nc")
# vars <- c("temperature", "precipitation")
# data <- nc_as_data_frame(nc, vars)

# Example usage with additional parameters
# boundary <- list(lon_min = -180, lon_max = 180,
#                  lat_min = -90, lat_max = 90)
# lon_points <- c(-120, -100, -80)
# lat_points <- c(30, 40, 50)
# id_points <- c("A", "B", "C")
# data <- nc_as_data_frame(nc, vars, keep_raw_time = FALSE,
#         include_metadata = FALSE, boundary = boundary,
#         lon_points = lon_points, lat_points = lat_points,
#         id_points = id_points, show_requested_points = FALSE,
#         great_circle_dist = FALSE)
```

---

nc_get_dim_min_max *Retrieve minimum and maximum values of a dimension from a NetCDF file.*

Description

Retrieves the minimum and maximum values of a dimension from a NetCDF file.

Usage

```
nc_get_dim_min_max(nc, dimension, time_as_date = TRUE)
```

Arguments

nc	A NetCDF file object.
dimension	The name of the dimension for which to retrieve the minimum and maximum values.
time_as_date	A logical value indicating whether to treat time dimension values as dates. Default is TRUE.

Value

A numeric vector containing the minimum and maximum values of the dimension.

Examples

```
# nc_file <- nc_open("path/to/netcdf/file.nc")
# min_max <- nc_get_dim_min_max(nc_file, "time", time_as_date = TRUE)
# nc_close(nc_file)
```

other_rose_plots *OTHER ROSE PLOTS*

Description

This function creates a wrapper around functions from openair package for generating various types of rose plots.

Usage

```
other_rose_plots(
  data,
  type1_col_name,
  type2_col_name,
  date_col_name,
  wd_col_name,
  ws_col_name,
  main_method,
  single_pollutant,
  multiple_pollutant,
  ...
)
```

Arguments

data	A data frame containing the required variables.
type1_col_name	The column name for the first type variable.
type2_col_name	The column name for the second type variable.
date_col_name	The column name for the date variable.
wd_col_name	The column name for the wind direction variable.
ws_col_name	The column name for the wind speed variable.
main_method	The main method to be used for generating the rose plots. Valid options are "percentile_rose", "polar_plot", "polar_annulus", "polar_cluster", and "polar_frequency".
single_pollutant	The name of the pollutant variable to be used for single-pollutant plots.
multiple_pollutant	The name(s) of the pollutant variable(s) to be used for multiple-pollutant plots.
...	Additional arguments to be passed to the openair package functions.

Value

The function generates the desired rose plot based on the specified parameters.

Examples

```
# This example generates a percentile rose plot using the "percentile_rose"
# method from the openair package, using the "PM2.5" pollutant for the plot
# and including "NO2" and "SO2" as additional pollutants in the plot.
# data <- read.csv("data.csv")
```

```
# other_rose_plots(data, type1_col_name, type2_col_name, date_col_name,
#                  wd_col_name, ws_col_name, "percentile_rose", "PM2.5",
#                  c("NO2", "SO2"))
```

`output_CPT` *Generate CPT data for visualization*

Description

This function generates a CPT (Color Palette Table) data frame for visualization purposes. It prepares the data by rearranging and merging the necessary variables and information. The resulting CPT data frame can be used for visualizing spatial data using CPT-based software or libraries.

Usage

```
output_CPT(
  data,
  lat_lon_data,
  station_latlondata,
  latitude,
  longitude,
  station,
  year,
  element,
  long.data = TRUE,
  na_code = -999
)
```

Arguments

`data`	The input data frame containing the raw data.
`lat_lon_data`	The data frame containing latitude and longitude information.
`station_latlondata`	The column name representing the station in the lat_lon_data data frame.
`latitude`	The column name representing latitude in the data frame.
`longitude`	The column name representing longitude in the data frame.
`station`	The column name representing the station in the data frame.
`year`	The column name representing the year in the data frame.
`element`	The column name representing the element in the data frame.
`long.data`	If TRUE, the data frame contains all the required variables. If FALSE, the data frame contains multiple years stacked as columns.
`na_code`	The code to be used for missing values. Default is -999.

Value

The CPT data frame for visualization.

Examples

```
# Example usage
# data <- read.csv("data.csv")
# lat_lon_data <- read.csv("lat_lon_data.csv")
# output_CPT(data, lat_lon_data, "station_latlon", "latitude", "longitude",
#            "station", "year", "element")

# Example usage with additional parameters
# output_CPT(data, lat_lon_data, "station_latlon", "latitude", "longitude",
#            "station", "year", "element", long.data = FALSE, na_code = -1)
```

---

pentad *Calculate the pentad for a given date*

Description

This function calculates the pentad (5-day period) for a given date. It determines which pentad the date falls into based on the day of the month.

Usage

```
pentad(date)
```

Arguments

date	The input date.

Value

The pentad number corresponding to the input date.

Examples

```
# Example usage
# pentad(as.Date("2023-07-17"))
```

---

plot.region *Generate a map of the selected product*

Description

This function generates a map of the selected product. It can plot either a certain year/month from a data file with multiple time steps or a single 2D field. The function requires the data to be prepared using the R script "Prep.Data.R" or "Apply.Function.R".

Usage

```
## S3 method for class 'region'
plot(
  lon,
  lat,
  product,
  time,
  time_point = as.Date("2002-01-01"),
  add2title = "CM SAF, ",
  lonmin = NA,
  lonmax = NA,
  latmin = NA,
  latmax = NA,
  height = 600,
  width = 600,
  plot.ano = FALSE,
  set.col.breaks = FALSE,
  brk.set = seq(240, 310, 5),
  colmin0 = NA,
  colmax0 = NA,
  ncol = 14,
  plotHighRes = FALSE,
  plotCoastline = TRUE,
  plotCountries = TRUE,
  plotRivers = FALSE,
  contour.thick = 2,
  plotCities = TRUE,
  pch.cities = 2,
  cex.cities = 1,
  label.cities = TRUE,
  plotCapitals = 1,
  cex.label.cities = 0.5,
  dlat = 0.25,
  plotOwnLocations = FALSE,
  loc_lon = c(),
  loc_lat = c(),
  loc_name = c(""),
  label_pos = 1,
  variable = "Tm",
  level = 5,
  CTY.type = 4,
  ...
)
```

Arguments

lon	Numeric vector representing the longitudes.
lat	Numeric vector representing the latitudes.
product	The data product to be plotted.
time	A vector representing the time steps of the data.
time_point	The specific time point to be plotted. Default is "2002-01-01".

add2title	Additional text to be added to the plot title. Default is "CM SAF, ".
lonmin	The minimum longitude value for the plot. Default is NA.
lonmax	The maximum longitude value for the plot. Default is NA.
latmin	The minimum latitude value for the plot. Default is NA.
latmax	The maximum latitude value for the plot. Default is NA.
height	The height of the plot in pixels. Default is 600.
width	The width of the plot in pixels. Default is 600.
plot.ano	If TRUE, plot the anomalies. Default is FALSE.
set.col.breaks	If TRUE, set custom color breaks. Default is FALSE.
brk.set	A numeric vector representing custom color breaks. Default is seq(240,310,5).
colmin0	The minimum color value for the plot. Default is NA.
colmax0	The maximum color value for the plot. Default is NA.
ncol	The number of colors in the color scale. Default is 14.
plotHighRes	If TRUE, plot the map in high resolution. Default is FALSE.
plotCoastline	If TRUE, plot the coastline. Default is TRUE.
plotCountries	If TRUE, plot the country borders. Default is TRUE.
plotRivers	If TRUE, plot the rivers. Default is FALSE.
contour.thick	The thickness of the contour lines. Default is 2.
plotCities	If TRUE, plot the cities. Default is TRUE.
pch.cities	The symbol type for plotting cities. Default is 2.
cex.cities	The size of the city symbols. Default is 1.
label.cities	If TRUE, label the cities. Default is TRUE.
plotCapitals	The type of capitals to plot. Default is 1.
cex.label.cities	The size of the city labels. Default is 0.5.
dlat	The latitude spacing for plotting. Default is 0.25.
plotOwnLocations	If TRUE, plot user-defined locations. Default is FALSE.
loc_lon	Numeric vector representing the longitudes of the user-defined locations. Default is c().
loc_lat	Numeric vector representing the latitudes of the user-defined locations. Default is c().
loc_name	Character vector representing the names of the user-defined locations. Default is c("").
label_pos	The position of the city labels. Default is 1.
variable	The variable name. Default is "Tm".
level	The level of the data. Default is 5.
CTY.type	The data type for the "CTY" variable. Default is 4.
...	Additional parameters to read in.

Examples

```
# Example usage
# lon <- read.csv("lon.csv")
# lat <- read.csv("lat.csv")
# product <- read.csv("product.csv")
# time <- read.csv("time.csv")
# plot.region(lon, lat, product, time)

# Example usage with additional parameters
# plot.region(lon, lat, product, time, time_point = as.Date("2002-01-01"),
#             add2title = "CM SAF, ", lonmin = -10, lonmax = 10, latmin = 35,
#             latmax = 45, height = 800, width = 800, plot.ano = TRUE,
#             set.col.breaks = TRUE, brk.set = seq(240,310,10),
#             colmin0 = 240, colmax0 = 310, ncol = 10, plotHighRes = TRUE,
#             plotCoastline = FALSE, plotCountries = FALSE,
#             plotRivers = TRUE, contour.thick = 3, plotCities = FALSE,
#             pch.cities = 16, cex.cities = 1.5, label.cities = FALSE,
#             plotCapitals = 2, cex.label.cities = 0.8, dlat = 0.5,
#             plotOwnLocations = TRUE, loc_lon = c(0, 2, 4),
#             loc_lat = c(40, 42, 44), loc_name = c("Location A",
#             "Location B", "Location C"), label_pos = 3,
#             variable = "Precipitation", level = 2, CTY.type = 2)
```

---

plot_declustered *Generate declustering plots*

Description

This function generates declustering plots based on the provided data and parameters. It uses the texmex package for declustering analysis.

Usage

```
plot_declustered(
  data,
  station_col_name,
  element_col_name,
  threshold,
  r = NULL,
  xlab = NULL,
  ylab = NULL,
  ncol = 1,
  print_summary = FALSE
)
```

Arguments

data	A data frame containing the input data.
station_col_name	The name of the column in the data frame that represents the stations.
element_col_name	The name of the column in the data frame that represents the elements.

threshold	The threshold value used for declustering.
r	The run length parameter used for declustering. Default is NULL.
xlab	The label for the x-axis of the plot. Default is NULL.
ylab	The label for the y-axis of the plot. Default is NULL.
ncol	The number of columns in the plot grid. Default is 1.
print_summary	If TRUE, it prints the declustering summary for each station. Default is FALSE.

Value

If print_summary is FALSE, it returns a plot object(s). Otherwise, it returns NULL.

Examples

```
# Example usage
# data <- read.csv("data.csv")
# plot_declustered(data, "station", "element", threshold = 0.5)

# Example usage with additional parameters
# plot_declustered(data, "station", "element", threshold = 0.5, r = 10,
#                  xlab = "Time", ylab = "Value", ncol = 2,
#                  print_summary = TRUE)
```

---

plot_mrl *Plot Mean Residual Life*

Description

This function generates a plot of the mean excess for a given threshold in a dataset.

Usage

```
plot_mrl(
  data,
  station_name,
  element_name,
  umin,
  umax,
  ncol = 1,
  xlab = "Threshold",
  ylab = "Mean excess",
  fill = "red",
  col = "black",
  rug = TRUE,
  addNexcesses = TRUE,
  textsize = 4
)
```

Arguments

data	A data frame containing the dataset.
station_name	The name of the column in data representing the stations.
element_name	The name of the column in data representing the elements.
umin	The lower threshold value. If not provided, the minimum value in element_name will be used.
umax	The upper threshold value. If not provided, the maximum value in element_name will be used.
ncol	The number of columns for arranging the subplots. Default is 1.
xlab	The label for the x-axis. Default is "Threshold".
ylab	The label for the y-axis. Default is "Mean excess".
fill	The fill color for the plot. Default is "red".
col	The color for the plot. Default is "black".
rug	A logical value indicating whether to include rug plot. Default is TRUE.
addNexcesses	A logical value indicating whether to add the number of excesses. Default is TRUE.
textsize	The size of the text in the plot. Default is 4.

Value

If station_name is provided, a grid of plots is returned. Otherwise, a single plot is returned.

Examples

```
# Example 1: Single plot
# Generate a data frame
# data <- data.frame(
#  station = c("A", "A", "B", "B", "C", "C"),
#  element = c(10, 15, 20, 25, 30, 35)
# )

# Generate a plot for the element column with default settings
# plot_mrl(data, element_name = "element")

# Example 2: Grid of plots
# Generate a data frame
# data <- data.frame(
#  station = c("A", "A", "B", "B", "C", "C"),
#  element = c(10, 15, 20, 25, 30, 35)
# )

# Generate a grid of plots for each station
# plot_mrl(data, station_name = "station", element_name = "element",
#          ncol = 2)
```

plot_multiple_threshold *Multiple Threshold plots*

Description

This function produces multiple threshold plots for various stations at a time.

Usage

```
plot_multiple_threshold(
  data,
  station_col_name,
  element_col_name,
  r,
  type = c("GP", "PP", "Exponential"),
  nint = 10,
  alpha = 0.05,
  ncol = 1,
  xlb = "",
  main = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

data	The input data frame containing the data for threshold plots.
station_col_name	The name of the column in 'data' representing the station identifier.
element_col_name	The name of the column in 'data' representing the element to plot.
r	The threshold value for the element.
type	The type of threshold plot to generate. Possible values: "GP" (Generalized Pareto), "PP" (Point Process), "Exponential".
nint	The number of intervals for plotting.
alpha	The significance level for threshold selection.
ncol	The number of columns for arranging the plots.
xlb	The label for the x-axis.
main	The main title for the plot.
verbose	Boolean value indicating whether to display verbose output.
...	Additional parameters to be passed to the threshold_Plot function.

Value

Returns a threshold plot

Examples

```
# TODO
```

`prepare_walter_lieth` *Prepare Data for Walter-Lieth Plot*

Description

Prepares data for Walter-Lieth climate diagrams.

Usage

```
prepare_walter_lieth(data, month, tm_min, ta_min)
```

Arguments

`data`	The input data frame.
`month`	The column name for months.
`tm_min`	The column name for minimum temperature.
`ta_min`	The column name for average temperature.

Value

A list of data frames and plots prepared for Walter-Lieth climate diagrams.

Examples

```
## Not run:
  prepare_walter_lieth(my_data, "Month", "MinTemp", "AvgTemp")

## End(Not run)
```

`spei_input` *Calculate SPEI Input Data*

Description

This function calculates the Standardized Precipitation-Evapotranspiration Index (SPEI) input data from a given data frame. The function sorts the data by year and month, checks for data completeness, and prepares the data for further SPEI calculations.

#' @details The function expects the input data to be in a data frame format. It calculates the SPEI input data by performing the following steps:

1. Sorts the data frame by the year and month columns in ascending order.
2. Verifies if the sorted data frame matches the original data frame, ensuring correct sorting for SPEI/SPI calculations. If they differ, an error is raised.

3. Checks if there are multiple values per month (per station) in the data frame. If multiple values are detected, an error is raised as SPEI/SPI requires monthly data with one value per month.
4. If the `station` parameter is provided, the function performs additional checks for each unique station:
   - Filters the data frame to include only rows with the current station.
   - Generates a sequence of dates from the first date in the filtered data to the last date, incrementing by one month.
   - Compares the length of the generated date sequence with the number of rows in the filtered data. If they differ, an error is raised, indicating missing months in the data for the current station.
5. Constructs the `cols` variable by combining the `id_cols` and `element` columns.
6. Determines the start year and month based on the first values in the `year` and `month` columns.
7. If the `station` parameter is provided, the function transforms the data frame into a "wide" format using `pivot_wider`, organizing the data by year and month with station-specific columns. Missing values are filled with `NA`.
8. Converts the resulting data frame to a time series object (`ts`) using `as.matrix`. The frequency is set to 12 for monthly data, and the start year and month are determined from the data.
9. If the `station` parameter is not provided, the function directly converts the `element` column to a time series object (`ts`) using `as.matrix`, with a frequency of 12 and the determined start year and month.
10. Returns the calculated time series data as the output.

#' @keywords SPEI, precipitation, evapotranspiration, time series, data preparation

Usage

```
spei_input(data, station, year, month, element)
```

Arguments

`data`	The data.frame to calculate from.
`station`	The name of the station column in `data`, if the data are for multiple station.
`year`	The name of the year column in `data`.
`month`	The name of the month column in `data`.
`element`	The name of the column(s) in `data` to apply the condition to.

Value

A time series object (`ts`) containing the calculated SPEI input data.

Examples

```
# TODO
```

spei_output *Extract SPEI/SPI Column from 'spei' Object*

Description

This function extracts the Standardized Precipitation-Evapotranspiration Index (SPEI) or Standardized Precipitation Index (SPI) column from a 'spei' object. It is designed to work with the original data and handle multiple stations if present. The function removes NA values introduced when unstacking the data to return a vector of the correct length.

Usage

```
spei_output(x, data, station, year, month)
```

Arguments

x	An object of class 'spei'.
data	The data.frame to calculate from.
station	The name of the station column in data, if the data are for multiple station.
year	The name of the year column in data.
month	The name of the month column in data.

Value

A vector containing the extracted SPEI/SPI column from the 'spei' object 'x'.

Examples

```
# TODO:
```

spells *Running spell length*

Description

Calculates the running number of consecutive non-zero values of a vector

Usage

```
spells(x, initial_value = NA_real_)
```

Arguments

x	A numeric vector
initial_value	The initial value of the spell length

Details

The `spells` function calculates the running number of consecutive non-zero values in a numeric vector `x`. It assigns a spell length value to each element in `x` based on the consecutive non-zero values encountered.

The function takes the following parameters:

- `x`: A numeric vector for which the spell lengths need to be calculated.
- `initial_value`: The initial value of the spell length. Default is `NA_real_`.

The function uses a loop to iterate over the elements of `x` and determine the spell length. If an element of `x` is non-zero, the spell length is incremented by 1. If an element is zero, the spell length is reset to 0. The result is returned as a vector of the same length as `x`, where each element represents the running number of consecutive non-zero values encountered in `x`.

Value

a vector of length `x` of the running number of consecutive non-zero values of `x`

See Also

The calculated running spell lengths can be useful for analyzing patterns of non-zero values in a vector and identifying consecutive sequences.

Examples

```
# TODO:
```

`summary_mean` *Calculate Mean of Data*

Description

Computes the mean or weighted mean of a dataset.

Usage

```
summary_mean(
  x,
  add_cols,
  weights = NULL,
  na.rm = FALSE,
  trim = 0,
  na_type = "",
  ...
)
```

Arguments

x	A numeric vector.
add_cols	Additional columns (not used directly in calculation).
weights	Optional weights for the data.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
trim	Numeric. Fraction of observations to trim from each end before computing the mean.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The mean or weighted mean of the data.

---

summary_sum *Calculate Sum of Data*

Description

Computes the sum or weighted sum of a dataset.

Usage

```
summary_sum(x, weights = NULL, na.rm = FALSE, na_type = "", ...)
```

Arguments

x	A numeric vector.
weights	Optional weights for the data.
na.rm	Logical. Should missing values be removed? Defaults to FALSE.
na_type	Character string indicating the type of NA check to perform.
...	Additional arguments passed to na_check.

Value

The sum or weighted sum of the data.

threshold_Plot *Threshold Plot*

Description

This function generates threshold plots for a given dataset.

Usage

```
threshold_Plot(
  x,
  r,
  type = c("GP", "PP", "Exponential"),
  nint = 10,
  alpha = 0.05,
  na.action = stats::na.omit,
  xlb = "",
  main = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

x	Numeric vector of data values.
r	Numeric vector of quantiles defining the range of thresholds. Default is to use the 75th and 99th percentiles of x.
type	Character string specifying the type of extreme value distribution to fit. Possible values are "GP" , "PP" , or "Exponential". Default is "GP".
nint	Integer specifying the number of intervals between the lower and upper thresholds. Default is 10.
alpha	Numeric value specifying the significance level for confidence intervals. Default is 0.05.
na.action	Function to handle missing values. Default is na.omit.
xlb	Character string specifying the x-axis label for the plot. Default is an empty string.
main	Character string specifying the main title for the plot. Default is NULL.
verbose	Logical value indicating whether to print additional output. Default is FALSE.
...	Additional arguments to be passed to the fitting and confidence interval functions.

Value

A list of threshold plots, including location, scale, and shape, depending on the specified type.

Examples

```
# Generate threshold plots for a dataset
data <- c(2.1, 2.2, 3.4, 4.5, 5.6, 6.7, 7.8, 8.9, 9.0, 10.1)
threshold_Plot(data, type = "GP")
```

WB_evaporation *Calculate Evaporation for Water Balance*

Description

Calculates the evaporation based on water balance, fraction of capacity, and other parameters.

Usage

```
WB_evaporation(water_balance, frac, capacity, evaporation_value, rain)
```

Arguments

water_balance	The current water balance.
frac	The fraction of capacity.
capacity	The total capacity.
evaporation_value	The value of evaporation.
rain	The amount of rain.

Value

The calculated evaporation.

Examples

```
## Not run:
  WB_evaporation(100, 0.5, 200, 10, 5)

## End(Not run)
```

wind_pollution_rose *Wind pollution Rose*

Description

This function creates a wrapper around windRose and pollutionRose functions from openair package

Usage

```
wind_pollution_rose(
  mydata,
  date_name,
  pollutant,
  type1_col_name,
  type2_col_name,
  ...
)
```

Arguments

mydata	A data frame containing the data for the plot.
date_name	The name of the column that contains the date information.
pollutant	The name of the column that contains the pollutant information.
type1_col_name	The name of the first type column.
type2_col_name	The name of the second type column.
...	Additional arguments to be passed to the underlying plotting functions.

Value

The generated wind or pollution rose plot.

Examples

```
# Example 1: Creating a wind rose plot
# wind_pollution_rose(mydata = wind_data, date_name = "date")
```

---

write_weather_data *Write Weather Data to File*

Description

Writes weather data to a text file with specified missing value codes.

Usage

```
write_weather_data(
  year,
  month,
  day,
  rain,
  mn_tmp,
  mx_tmp,
  missing_code,
  output_file
)
```

Arguments

year	A vector of years.
month	A vector of months.
day	A vector of days.
rain	A vector of rainfall values.
mn_tmp	A vector of minimum temperatures.
mx_tmp	A vector of maximum temperatures.
missing_code	The code to use for missing values.
output_file	The name of the output file.

Value

None. Writes the data to a file.

Examples

```
## Not run:
  write_weather_data(2020, 1:12, 1:31, rain_data, min_temp, max_temp,
  -99, "weather.txt")

## End(Not run)
```

`wwr_export` *Reshape data into formats required by WMO for submission of climatic data*

Description

The function is meant to reshape data into formats required by WMO for submission of climatic data.This gives Yearly data records with monthly and annual data for a particular year:

Usage

```
wwr_export(
  data,
  year,
  month,
  mean_station_pressure,
  mean_sea_level_pressure,
  mean_temp,
  total_precip,
  mean_max_temp,
  mean_min_temp,
  mean_rel_hum,
  link,
  link_by,
  station_data,
  wmo_number,
  latitude,
  longitude,
  country_name,
  station_name,
  height_station,
  height_barometer,
  wigos_identifier,
  folder
)
```

Arguments

`data`	The input data frame containing the climatic data.
`year`	The name of the column in 'data' representing the year.

`month` The name of the column in 'data' representing the month.

`mean_station_pressure` The name of the column in 'data' representing the mean station pressure.

`mean_sea_level_pressure` The name of the column in 'data' representing the mean sea level pressure.

`mean_temp` The name of the column in 'data' representing the mean daily air temperature.

`total_precip` The name of the column in 'data' representing the total precipitation.

`mean_max_temp` The name of the column in 'data' representing the mean daily maximum air temperature.

`mean_min_temp` The name of the column in 'data' representing the mean daily minimum air temperature.

`mean_rel_hum` The name of the column in 'data' representing the mean of the daily relative humidity.

`link` The name of the column in 'data' representing the link between the data and station information.

`link_by` The method for linking the data and station information. Possible values: "wmo_number", "station_name".

`station_data` The data frame containing the station information.

`wmo_number` The name of the column in 'station_data' representing the WMO number.

`latitude` The name of the column in 'station_data' representing the latitude.

`longitude` The name of the column in 'station_data' representing the longitude.

`country_name` The name of the column in 'station_data' representing the country name.

`station_name` The name of the column in 'station_data' representing the station name.

`height_station` The name of the column in 'station_data' representing the station height.

`height_barometer` The name of the column in 'station_data' representing the barometer height.

`wigos_identifier` The name of the column in 'station_data' representing the WIGOS station identifier.

`folder` The folder where the output files will be saved.

Value

TODO

Examples

```
# TODO
```

yday_366 *Get the 366-based day of the year of a date*

Description

#' yday_366 returns an integer between 1 and 366 representing the day of the year number of x.

In contrast to lubridate::yday, yday_366 considers the length of all years to be 366 days. In non leap years, there is no day 60 (29 February) and 1 March is always day 61.

This convention is often used for weather data and other applications. It's advantage is that all days have the same day number regardless of the year e.g. 31 December is always day 366, even in non leap years. This may be desirable, for example, when comparing day of year numbers across years that contain leap years and non leap years.

Usage

```
yday_366(date)
```

Arguments

date	A Date object

Value

An integer between 1 and 366 representing the day of the year number of x.

Examples

```
# yday_366(as.Date("1999-03-01"))
# yday_366(as.Date("2000-12-31"))
# yday_366(as.Date("2005-12-31"))
```

Index