

Package: varunayan (via r-universe)

October 18, 2026

Type Package

Title Download and Process 'ERA5', IMD, 'HadEX3' and 'CRU TS' Data

Version 0.2.0

Description Download and process climate data for custom geographical regions defined by bounding boxes, 'GeoJSON' polygons, or point coordinates. Supports the fifth generation European Centre for Medium-Range Weather Forecasts atmospheric reanalysis ('ERA5'; Hersbach et al. (2020) <[doi:10.1002/qj.3803](https://doi.org/10.1002/qj.3803)>) from the 'Copernicus' Climate Data Store (CDS) <<https://cds.climate.copernicus.eu/>>, India Meteorological Department (IMD) gridded rainfall and temperature (Pai et al. (2014) <[doi:10.54302/mausam.v65i1.851](https://doi.org/10.54302/mausam.v65i1.851)>), Hadley Centre global land climate extremes indices version 3 ('HadEX3'; Dunn et al. (2020) <[doi:10.1029/2019JD032263](https://doi.org/10.1029/2019JD032263)>), and Climatic Research Unit gridded Time Series ('CRU TS'; Harris et al. (2020) <[doi:10.1038/s41597-020-0453-3](https://doi.org/10.1038/s41597-020-0453-3)>). Also computes heat stress indices including heat index, wet-bulb temperature, outdoor wet-bulb globe temperature (Liljegren et al. (2008) <[doi:10.1080/15459620802310770](https://doi.org/10.1080/15459620802310770)>) and the Universal Thermal Climate Index (Bröde et al. (2012) <[doi:10.1007/s00484-011-0454-1](https://doi.org/10.1007/s00484-011-0454-1)>), and heat exposure indicators for health studies: heatwave and warm-night frequency and duration under India Meteorological Department and percentile definitions, the Excess Heat Factor (Nairn and Fawcett (2015) <[doi:10.3390/ijerph120100227](https://doi.org/10.3390/ijerph120100227)>) and diurnal temperature range.

License MIT + file LICENSE

URL <https://varunayan.saketlab.org/R/>,
<https://github.com/saketlab/varunayanR>

BugReports <https://github.com/saketlab/varunayanR/issues>

Depends R (>= 4.1.0)

Imports data.table, digest, dplyr, ecmwfr (>= 2.0.0), httr, jsonlite, lubridate, magrittr, ncdf4, rlang, sf, stars, stats, terra, tools, utils, parallel

Suggests ggplot2, ggpubr, gridExtra, knitr, maps, ncmeta, patchwork,
R.utils, rmarkdown, scales, testthat (>= 3.1.7), tidyr, units,
withr (>= 2.5.0)

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Copyright See file inst/COPYRIGHTS

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libsecret-1-dev libnetcdf-
dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository <https://saketlab.r-universe.dev>

Date/Publication 2026-10-18 03:50:00 UTC

RemoteUrl <https://github.com/saketlab/varunayanR>

RemoteRef HEAD

RemoteSha 95c14023bc1f7560b17d77bae615ae3e5b8c0bb9

Contents

aggregate_by_frequency	5
aggregate_imd_by_frequency	5
aggregate_to_polygons	6
cache_stats	7
calc_heat_indices	8
categorize_variables	8
check_grid_coverage	9
CheckSystemReadiness	9
cleanup_temp_files	10
clear_timings	11
ClearCache	11
combine_netcdf_files	12
compare_grids	12
convert_to_geojson	13
create_geojson_from_bbox	14
create_temp_geojson_file	14
create_temporal_chunks	15
CRUTSBbox	15
CRUTSGeojson	17
DescribeVariables	18
download_era5_pressure	19
download_era5_single	20
download_imd_rainfall	21
download_imd_temperature	22

ensure_output_dir	23
ERA5ifyBbox	23
ERA5ifyPoint	25
excess_heat_factor	27
exclude_cols	28
exposure_indicators	28
extract_point_data	30
filter_dataframe_by_geojson	31
filter_imd_by_bbox	31
filter_imd_by_geojson	32
filter_netcdf_by_bbox	33
filter_netcdf_by_geojson	33
format_date_range	34
format_pressure_levels	34
get_available_datasets	35
get_bbox_from_sf	35
get_cds_credentials	36
get_imd_grid_specs	36
get_pressure_level_variable_names	37
get_pressure_level_variables	37
get_single_level_variable_names	37
get_single_level_variables	38
get_time_values	38
get_timings	39
GetAvailablePressureLevels	39
GetCacheDir	40
GetERA5CountryTemperature	40
GetERA5DailyClimate	42
GetERA5DailyHeatIndexData	43
GetERA5DailyHumidity	46
GetERA5DailyTemperature	47
GetERA5DailyWetBulb	48
GetERA5DailyWind	50
GetERA5MonthlyHeatIndexData	51
GetPackageInfo	53
HadEX3Bbox	54
HadEX3Geojson	55
heat_index	56
heat_index_from_dewpoint	57
heat_index_risk_category	58
heatwave_days	58
humidex	60
imd_temperature_by_region	61
imd_terrain	62
IMDExposureBbox	63
IMDRainfallBbox	64
IMDRainfallGeojson	66
IMDRainfallPoint	67

IMDStations	68
IMDTemperatureBbox	69
IMDTemperatureGeojson	70
india_elevation	71
is_max_var	72
is_min_var	73
is_rate_var	73
is_sum_var	74
is_valid_geojson	74
list_cache	75
list_imd_datasets	75
ListAvailableVariables	76
ListCRUTSVariables	76
ListHadEX3Indices	77
max_vars	77
mean_radiant_temperature	78
min_vars	78
ndbi	79
ndvi	80
parse_date_flexible	80
PopulateSampleCache	81
print_timing_summary	81
process_imd_files	82
process_netcdf_to_dataframe	83
rate_vars	83
read_cdsapirc	84
read_imd_rainfall	84
read_imd_temperature	85
relative_humidity_from_dewpoint	86
SearchVariable	86
SetupCdsCredentials	87
ShowCacheInfo	88
sum_vars	88
time_operation	89
utci	89
utci_category	90
utci_sparse	91
validate_bbox	92
validate_coordinates	92
validate_era5_variables	93
vapor_pressure_from_dewpoint	93
varunayan-aliases	94
visualize_grid_overlap	95
warm_nights	97
wbgt	98
wbgt_from_dewpoint	99
wbgt_liljegren	99
wbgt_risk_category	101

<i>aggregate_by_frequency</i>	5
-------------------------------	---

wbgt_shade	102
wbgt_simple	102
wet_bulb_from_dewpoint	103
wet_bulb_temperature	104

Index	105
--------------	------------

aggregate_by_frequency	
<i>Aggregate data by temporal frequency</i>	

Description

Aggregate data by temporal frequency

Usage

aggregate_by_frequency(data, frequency, verbose = FALSE)

Arguments

data	data.frame with datetime column
frequency	Character string: "daily", "monthly", "yearly"
verbose	Logical. If TRUE, the function prints progress messages

Value

data.frame with temporally aggregated data

aggregate_imd_by_frequency	
<i>Aggregate IMD data by temporal frequency</i>	

Description

Aggregate IMD data by temporal frequency

Usage

aggregate_imd_by_frequency(data, frequency = "daily")

Arguments

data	data.frame with IMD data
frequency	Character string: "daily", "monthly", "yearly"

Value

A data.frame with columns date (first day of each month or year), latitude, longitude and the value column, summed for rainfall and averaged for temperature. Returned unchanged when frequency = "daily".

Examples

```
imd <- data.frame(
  date = rep(as.Date("2023-01-01") + 0:59, each = 2),
  latitude = 20.5, longitude = c(73.5, 79.5), rainfall = 1
)
aggregate_imd_by_frequency(imd, "monthly")
```

aggregate_to_polygons *Aggregate gridded data to polygon regions*

Description

Assigns each grid cell to a polygon, then aggregates the values of the cells in each polygon. The spatial match uses only the distinct cells. Thus, a daily series with many dates does not make the match slower.

Usage

```
aggregate_to_polygons(
  data,
  polygons,
  value_col,
  method = c("nearest_centroid", "point_in_polygon"),
  polygon_id_cols = NULL,
  agg_fun = function(x) mean(x, na.rm = TRUE),
  by = NULL,
  fill_empty = FALSE
)
```

Arguments

data	Data frame with longitude, latitude, and value columns
polygons	sf object with polygon geometries (CRS 4326)
value_col	Name of the column to aggregate, or a character vector of names. Give names to set the result columns, for example c(tmean = "temp_c"). A column without a name keeps its name.
method	Aggregation method: "nearest_centroid" (each polygon gets the cell nearest to a point on its surface) or "point_in_polygon" (each polygon gets the cells in it)

polygon_id_cols	Character vector of column names to use as polygon identifiers. The default is all the columns of polygons except the geometry.
agg_fun	Aggregation function. It gets the values of one group, which can include NA. The default is the mean without the NA values.
by	Optional character vector of more group columns in data, for example "date". The result then has one row for each polygon and each value of by.
fill_empty	For "point_in_polygon": if TRUE, a polygon with no cell in it (smaller than a cell) gets the cell nearest to a point on its surface.

Value

Data frame with the polygon identifiers, the by columns and the aggregated values. It has one row for each polygon and each value of by. A polygon with no cell has NA.

Examples

```
grid <- expand.grid(longitude = seq(72.5, 77.5, by = 1), latitude = seq(18.5, 21.5, by = 1))
grid$temperature <- seq_len(nrow(grid))
cells <- sf::st_make_grid(sf::st_as_sfc(sf::st_bbox(
  c(xmin = 72, ymin = 18, xmax = 78, ymax = 22),
  crs = 4326
)), n = c(2, 1))
polys <- sf::st_sf(region = c("west", "east"), geometry = cells)
aggregate_to_polygons(grid, polys, value_col = "temperature")
```

cache_stats	<i>Get cache statistics</i>
-------------	-----------------------------

Description

Get cache statistics

Usage

```
cache_stats()
```

Value

A list with elements exists (logical), n_files and total_size_mb.

Examples

```
cache_stats()
```

calc_heat_indices	<i>Calculate all heat stress indices</i>
-------------------	--

Description

This function calculates all heat stress indices from temperature and dewpoint.

Usage

```
calc_heat_indices(temp_c, dewpoint_c)
```

Arguments

temp_c	Air temperature in Celsius
dewpoint_c	Dewpoint temperature in Celsius

Value

Data frame with columns: temp_c, dewpoint_c, rh, wet_bulb, heat_index, wbgt, humidex

Examples

```
calc_heat_indices(35, 25)
calc_heat_indices(c(30, 35, 40), c(20, 25, 28))
```

categorize_variables	<i>Sort variables by aggregation method</i>
----------------------	---

Description

Sort variables by aggregation method

Usage

```
categorize_variables(var_names)
```

Arguments

var_names	Character vector of variable names
-----------	------------------------------------

Value

Named list of the variables, sorted by aggregation method

Examples

```
categorize_variables(c("2m_temperature", "total_precipitation"))
```

check_grid_coverage	<i>Calculate the grid coverage statistics for polygons</i>
---------------------	--

Description

Calculate the grid coverage statistics for polygons

Usage

```
check_grid_coverage(grid_data, polygons, polygon_id_cols = NULL)
```

Arguments

grid_data	Data frame with longitude and latitude columns
polygons	sf object with polygon geometries
polygon_id_cols	Column names that identify the polygons

Value

A list with elements total_polygons, total_grid_points, polygons_with_points, polygons_without_points, coverage_percent, points_per_polygon (a data.frame of polygon identifiers and n_points) and summary (a one-line character description).

Examples

```
grid <- expand.grid(longitude = seq(72.5, 77.5, by = 1), latitude = seq(18.5, 21.5, by = 1))
grid$temperature <- seq_len(nrow(grid))
cells <- sf::st_make_grid(sf::st_as_sfc(sf::st_bbox(
  c(xmin = 72, ymin = 18, xmax = 78, ymax = 22),
  crs = 4326
)), n = c(2, 1))
polys <- sf::st_sf(region = c("west", "east"), geometry = cells)
check_grid_coverage(grid, polys)
```

CheckSystemReadiness	<i>Make sure that the system is ready for varunayan operations</i>
----------------------	--

Description

Make sure that the system is ready for varunayan operations

Usage

```
CheckSystemReadiness(...)

check_system_readiness(verbose = TRUE)
```

Arguments

- ... Arguments for the main function (for use by aliases).
- verbose If TRUE, print detailed information

Value

Invisibly, TRUE if all required packages, network access and CDS credentials are available, FALSE otherwise.

Examples

```
check_system_readiness()
```

cleanup_temp_files	<i>Delete temporary files that match a pattern</i>
--------------------	--

Description

Delete temporary files that match a pattern

Usage

```
cleanup_temp_files(pattern = "varunayan.*", directory = tempdir())
```

Arguments

- pattern Regular expression pattern to match files
- directory Directory for the search (default: tempdir())

Value

Number of files removed

clear_timings	<i>Clear all time data</i>
---------------	----------------------------

Description

Clear all time data

Usage

```
clear_timings()
```

Value

NULL, invisibly. Called for its side effect.

Examples

```
clear_timings()
```

ClearCache	<i>Clear cache</i>
------------	--------------------

Description

Clear cache

Usage

```
ClearCache(...)
```

```
clear_cache(source = NULL, older_than = NULL, confirm = TRUE)
```

Arguments

...	Arguments for the main function (for use by aliases).
source	Clear only "era5", "imd", "cru_ts", "hadex3", or NULL for all
older_than	Remove files that are older than this number of days (NULL = all)
confirm	Ask for confirmation (default: TRUE)

Value

Invisibly, the number of files deleted.

Examples

```
# Remove cached ERA5 files older than 30 days
clear_cache(source = "era5", older_than = 30)
```

combine_netcdf_files	<i>Combine NetCDF files into one data.frame</i>
----------------------	---

Description

Combine NetCDF files into one data.frame

Usage

```
combine_netcdf_files(netcdf_files, verbose = FALSE)
```

Arguments

netcdf_files	Character vector of NetCDF file paths
verbose	Logical. If TRUE, the function prints progress messages

Value

data.frame with combined data from all files

compare_grids	<i>Compare two or more grids</i>
---------------	----------------------------------

Description

Compare two or more grids

Usage

```
compare_grids(
  ...,
  polygons = NULL,
  colors = NULL,
  shapes = NULL,
  point_size = 1,
  polygon_color = "#2C3E50",
  title = "Grid Comparison"
)
```

Arguments

<code>...</code>	Named data frames with longitude and latitude columns
<code>polygons</code>	Optional sf object to show polygon boundaries
<code>colors</code>	Named vector of colors for each grid
<code>shapes</code>	Named vector of point shapes for each grid
<code>point_size</code>	Size of grid points
<code>polygon_color</code>	Color for polygon boundaries
<code>title</code>	Plot title

Value

ggplot object

Examples

```
grid <- expand.grid(longitude = seq(72.5, 77.5, by = 1), latitude = seq(18.5, 21.5, by = 1))
grid$temperature <- seq_len(nrow(grid))
cells <- sf::st_make_grid(sf::st_as_sfc(sf::st_bbox(
  c(xmin = 72, ymin = 18, xmax = 78, ymax = 22),
  crs = 4326
)), n = c(2, 1))
polys <- sf::st_sf(region = c("west", "east"), geometry = cells)
coarse <- grid[grid$longitude < 75, ]
compare_grids(fine = grid, coarse = coarse, polygons = polys)
```

convert_to_geojson	<i>Convert data to GeoJSON format</i>
--------------------	---------------------------------------

Description

Convert data to GeoJSON format

Usage

```
convert_to_geojson(data)
```

Arguments

<code>data</code>	Data to convert (must be data.frame with lon/lat columns)
-------------------	---

Value

Data in GeoJSON format

`create_geojson_from_bbox`*Create GeoJSON from bounding box coordinates*

Description

Create GeoJSON from bounding box coordinates

Usage

```
create_geojson_from_bbox(north, south, east, west)
```

Arguments

north	Northern latitude
south	Southern latitude
east	Eastern longitude
west	Western longitude

Value

GeoJSON object that shows the bounding box

`create_temp_geojson_file`*Create temporary GeoJSON file*

Description

Create temporary GeoJSON file

Usage

```
create_temp_geojson_file(geojson_data, request_id)
```

Arguments

geojson_data	GeoJSON data to write
request_id	Request identifier for a unique filename

Value

Path to temporary GeoJSON file

create_temporal_chunks	<i>Divide large date ranges into temporal chunks</i>
------------------------	--

Description

The default chunk size is compute_optimal_chunk_size(frequency). This size changes with the CDS field limit (hourly: 24 fields/day, daily: 1/day, monthly: 12/year). Thus, each chunk fills one CDS job.

Usage

```
create_temporal_chunks(  
  start_date,  
  end_date,  
  frequency,  
  n_variables = 1L,  
  max_days_per_chunk = NA  
)
```

Arguments

start_date	Start date
end_date	End date
frequency	Temporal frequency ("hourly", "daily", "monthly")
n_variables	Number of variables in the request (fields = vars x steps)
max_days_per_chunk	Maximum days per chunk. If NA (default), the function calculates the value from frequency with compute_optimal_chunk_size.

Value

List of date pairs, with one pair for each chunk

CRUTSBbox	<i>Download CRU TS climate data for a bounding box</i>
-----------	--

Description

Downloads CRU TS v4.07 monthly gridded climate data and filters the data to a geographic bounding box. The data is for the global land areas at a resolution of 0.5 degrees, for 1901-2022.

Usage

```
CRUTSBbox(...)

cru_ts_bbox(
  variable,
  start_year,
  end_year,
  north,
  south,
  east,
  west,
  output_dir = tempdir(),
  use_cache = TRUE,
  ...
)
```

Arguments

...	Arguments that the aliases send to the primary function.
variable	CRU TS variable code. See list_cru_ts_variables() . Common values: "tmp" (mean temperature), "pre" (precipitation), "tmx" (max temperature), "tmn" (min temperature).
start_year	Start year (1901-2022).
end_year	End year (1901-2022).
north	North boundary (latitude).
south	South boundary (latitude).
east	East boundary (longitude).
west	West boundary (longitude).
output_dir	Directory for temporary files (default: tempdir()).
use_cache	If TRUE, use the data in the cache (default: TRUE).

Value

A data.frame with the columns: year, month, latitude, longitude, value, variable.

Examples

```
# Monthly mean temperature for India, 2010-2020
data <- cru_ts_bbox(
  variable = "tmp",
  start_year = 2010, end_year = 2020,
  north = 35, south = 8, east = 97, west = 68
)

# Precipitation for Europe, 1981-2010
data <- cru_ts_bbox(
  variable = "pre",
```

```
start_year = 1981, end_year = 2010,  
north = 71, south = 36, east = 40, west = -10  
)
```

CRUTSGeojson*Download CRU TS climate data for a GeoJSON region*

Description

Downloads CRU TS v4.07 monthly gridded climate data and filters the data to a geographic region. A GeoJSON polygon defines the region.

Usage

```
CRUTSGeojson(...)  
  
cru_ts_geojson(  
  variable,  
  start_year,  
  end_year,  
  geojson_file,  
  output_dir = tempdir(),  
  use_cache = TRUE,  
  ...  
)
```

Arguments

...	Arguments that the aliases send to the primary function.
variable	CRU TS variable code. See list_cru_ts_variables() .
start_year	Start year (1901-2022).
end_year	End year (1901-2022).
geojson_file	Path to a GeoJSON file that defines the region.
output_dir	Directory for temporary files (default: <code>tempdir()</code>).
use_cache	If TRUE, use the data in the cache (default: TRUE).

Value

A data.frame with the columns: year, month, latitude, longitude, value, variable.

Examples

```

region <- system.file("extdata", "region.geojson", package = "varunayan")
data <- cru_ts_geojson(
  variable = "tmp",
  start_year = 1981, end_year = 2010,
  geojson_file = region
)

```

DescribeVariables	<i>Describe ERA5 variables in detail</i>
-------------------	--

Description

Describe ERA5 variables in detail

Usage

```

DescribeVariables(...)

describe_variables(variables, dataset_type = "all")

```

Arguments

...	Arguments that the aliases send to the primary function.
variables	Character vector of variable names to describe
dataset_type	Character string that gives the dataset type ("single", "pressure", or "all")

Value

A data.frame with columns variable_name, description, units, dataset_type and category, one row per matched variable.

Examples

```

# Describe specific variables
descriptions <- describe_variables(c("2m_temperature", "total_precipitation"))

# Get descriptions for pressure level variables
pressure_desc <- describe_variables("temperature", dataset_type = "pressure")

```

`download_era5_pressure`*Download ERA5 pressure-level data with ecmwfr*

Description

Download ERA5 pressure-level data with ecmwfr

Usage

```
download_era5_pressure(  
  variables,  
  pressure_levels,  
  start_date,  
  end_date,  
  area = NULL,  
  resolution = 0.25,  
  frequency = "hourly",  
  output_file,  
  use_cache = TRUE,  
  verbose = FALSE  
)
```

Arguments

<code>variables</code>	Character vector of variable names
<code>pressure_levels</code>	Character vector of pressure levels (e.g., <code>c("1000", "850", "500")</code>)
<code>start_date</code>	Date object or character string (YYYY-MM-DD)
<code>end_date</code>	Date object or character string (YYYY-MM-DD)
<code>area</code>	Numeric vector of 4 elements: <code>c(north, west, south, east)</code> in decimal degrees
<code>resolution</code>	Numeric value for grid resolution (default: 0.25 degrees)
<code>frequency</code>	Character string: "hourly", "daily", "monthly"
<code>output_file</code>	Character string path for output NetCDF file
<code>use_cache</code>	If TRUE, use cached data when it is available (default: TRUE)
<code>verbose</code>	Logical. If TRUE, print progress messages

Value

Character string path to downloaded file

download_era5_single *Download ERA5 single-level data with ecmwfr*

Description

Download ERA5 single-level data with ecmwfr

Usage

```
download_era5_single(
  variables,
  start_date,
  end_date,
  area = NULL,
  resolution = 0.25,
  frequency = "hourly",
  output_file,
  timeout = 3600,
  retry_attempts = 5,
  use_cache = TRUE,
  verbose = FALSE
)
```

Arguments

variables	Character vector of variable names
start_date	Date object or character string (YYYY-MM-DD)
end_date	Date object or character string (YYYY-MM-DD)
area	Numeric vector of 4 elements: c(north, west, south, east) in decimal degrees
resolution	Numeric value for grid resolution (default: 0.25 degrees)
frequency	Character string: "hourly", "daily", "monthly"
output_file	Character string path for output NetCDF file
timeout	Numeric value for request timeout in seconds (default: 3600)
retry_attempts	Integer number of retry attempts (default: 5)
use_cache	If TRUE, use cached data when it is available (default: TRUE)
verbose	Logical. If TRUE, print progress messages

Value

Character string path to downloaded file

`download_imd_rainfall` *Download IMD gridded rainfall data*

Description

Downloads daily gridded rainfall data from IMD at the specified resolution.

Usage

```
download_imd_rainfall(  
  start_year,  
  end_year,  
  resolution = 0.25,  
  output_dir = tempdir(),  
  use_cache = TRUE  
)
```

Arguments

<code>start_year</code>	Start year.
<code>end_year</code>	End year.
<code>resolution</code>	Spatial resolution: 0.25 or 1.0 degrees.
<code>output_dir</code>	Directory for the downloaded files.
<code>use_cache</code>	If TRUE, use the cached data when it is available (default: TRUE).

Value

Character vector of the paths to the downloaded files.

Examples

```
# Download 0.25 degree rainfall for 2023-2025  
files <- download_imd_rainfall(  
  start_year = 2023,  
  end_year = 2025,  
  resolution = 0.25  
)
```

`download_imd_temperature`*Download IMD gridded temperature data*

Description

Downloads daily gridded temperature data (maximum or minimum) from IMD. Temperature data is available only at 1.0 degree resolution.

Usage

```
download_imd_temperature(  
  start_year,  
  end_year,  
  var_type = "tmax",  
  output_dir = tempdir(),  
  use_cache = TRUE  
)
```

Arguments

<code>start_year</code>	Start year.
<code>end_year</code>	End year.
<code>var_type</code>	Type of temperature: "tmax" or "tmin".
<code>output_dir</code>	Directory for the downloaded files.
<code>use_cache</code>	If TRUE, use the cached data when it is available (default: TRUE).

Value

Character vector of the paths to the downloaded files.

Examples

```
# Download maximum temperature for 2023-2025  
files <- download_imd_temperature(  
  start_year = 2023,  
  end_year = 2025,  
  var_type = "tmax"  
)
```

ensure_output_dir	Create the output directory if it does not exist
-------------------	--

Description

Create the output directory if it does not exist

Usage

```
ensure_output_dir(output_dir)
```

Arguments

output_dir	Path to output directory
------------	--------------------------

Value

Character string. The path to the directory that the function made or found

ERA5ifyBbox	Download and process ERA5 data for a bounding box
-------------	---

Description

Download and process ERA5 data for a bounding box

Usage

```
ERA5ifyBbox(...)  
  
era5ify_bbox(  
  request_id,  
  variables,  
  start_date,  
  end_date,  
  north,  
  south,  
  east,  
  west,  
  dataset_type = "single",  
  pressure_levels = NULL,  
  frequency = "hourly",  
  resolution = 0.25,  
  save_raw = FALSE,  
  output_dir = tempdir(),  
  use_cache = TRUE,
```

```

    convert_units = TRUE,
    verbose = FALSE
  )

```

Arguments

<code>...</code>	Arguments passed to the main function (used by aliases).
<code>request_id</code>	Unique identifier for the data request.
<code>variables</code>	List of ERA5 variables to download.
<code>start_date</code>	Start date in "YYYY-MM-DD" format.
<code>end_date</code>	End date in "YYYY-MM-DD" format.
<code>north</code>	Northern latitude boundary.
<code>south</code>	Southern latitude boundary.
<code>east</code>	Eastern longitude boundary.
<code>west</code>	Western longitude boundary.
<code>dataset_type</code>	Type of dataset ("single" or "pressure").
<code>pressure_levels</code>	Pressure levels for pressure dataset.
<code>frequency</code>	Temporal frequency ("hourly", "daily", "monthly").
<code>resolution</code>	Spatial resolution in degrees (default: 0.25).
<code>save_raw</code>	If TRUE, save the raw downloaded files.
<code>output_dir</code>	Directory for the saved files (default: <code>tempdir()</code>).
<code>use_cache</code>	If TRUE, use cached data when it is available (default: TRUE).
<code>convert_units</code>	If TRUE, convert units to user-friendly formats (default: TRUE). When TRUE, the function returns temperature variables in Celsius (converted from Kelvin). It returns precipitation variables in mm (converted from meters).
<code>verbose</code>	If TRUE, print detailed progress messages.

Value

data.frame with processed climate data. When `convert_units = TRUE`:

- Temperature variables (e.g., `2m_temperature`): Celsius
- Precipitation variables (e.g., `total_precipitation`): mm
- Other variables: original ERA5 units

Examples

```

setup_cds_credentials(key = "your-cds-api-key")

data <- era5ify_bbox(
  request_id = "my_request",
  variables = c("2m_temperature", "total_precipitation"),
  start_date = "2023-01-01",

```

```

    end_date = "2023-01-31",
    north = 40, south = 35, east = -70, west = -75
  )
  # Temperature is in Celsius. Precipitation is in mm.

```

ERA5ifyPoint

Download and process ERA5 data for a point location

Description

Download and process ERA5 data for a point location

Usage

```

ERA5ifyPoint(...)

era5ify_point(
  request_id,
  variables,
  start_date,
  end_date,
  lat,
  lon,
  dataset_type = "single",
  pressure_levels = NULL,
  frequency = "hourly",
  save_raw = FALSE,
  output_dir = tempdir(),
  use_cache = TRUE,
  convert_units = TRUE,
  verbose = FALSE,
  use_timeseries_api = NA
)

```

Arguments

...	Arguments passed to the main function (used by aliases).
request_id	Unique identifier for the data request.
variables	List of ERA5 variables to download.
start_date	Start date in "YYYY-MM-DD" format.
end_date	End date in "YYYY-MM-DD" format.
lat	Latitude of the point location.
lon	Longitude of the point location.
dataset_type	Type of dataset ("single" or "pressure").

pressure_levels	Pressure levels for pressure dataset.
frequency	Temporal frequency ("hourly", "daily", "monthly").
save_raw	If TRUE, save the raw downloaded files.
output_dir	Directory for the saved files (default: tempdir()).
use_cache	If TRUE, use cached data when it is available (default: TRUE).
convert_units	If TRUE, convert units to user-friendly formats (default: TRUE). When TRUE, the function returns temperature variables in Celsius (converted from Kelvin). It returns precipitation variables in mm (converted from meters).
verbose	If TRUE, print detailed progress messages.
use_timeseries_api	If TRUE, use the CDS time-series endpoint. This endpoint gives the data for a point directly. It is much faster than a subset of a gridded download. The default is NA: use the endpoint only for hourly single-level requests, because the endpoint supports only these requests. Set FALSE to always use the gridded download. If the endpoint is not available, the function automatically uses the gridded download.

Value

data.frame with processed climate data. When `convert_units = TRUE`:

- Temperature variables (e.g., `2m_temperature`): Celsius
- Precipitation variables (e.g., `total_precipitation`): mm
- Other variables: original ERA5 units

Examples

```
setup_cds_credentials(key = "your-cds-api-key")

data <- era5ify_point(
  request_id = "my_request",
  variables = c("2m_temperature", "total_precipitation"),
  start_date = "2023-01-01",
  end_date = "2023-01-31",
  lat = 40.7128,
  lon = -74.0060
)
# Temperature is in Celsius. Precipitation is in mm.
```

excess_heat_factor	<i>Excess Heat Factor</i>
--------------------	---------------------------

Description

Excess Heat Factor (EHF) of Nairn & Fawcett (2015). Daily mean temperature (DMT) pairs each day's maximum with the following night's minimum, $T_i = (tmax_i + tmin_{(i+1)}) / 2$. For the three-day period starting on day i , $EHF_sig = \text{mean}(T_i, T_{i+1}, T_{i+2}) - T95$ and $EHF_accl = \text{mean}(T_i, T_{i+1}, T_{i+2}) - \text{mean}(T_{i-1}, \dots, T_{i-30})$, with $T95$ the 95th percentile of DMT over the baseline years; $EHF = EHF_sig * \max(1, EHF_accl)$. Positive EHF marks heatwave conditions; $EHF85$, the 85th percentile of positive baseline EHF values, is the severe threshold and $3 * EHF85$ the extreme threshold.

Usage

```
excess_heat_factor(date, tmax, tmin, baseline)
```

Arguments

date	Dates of a strictly increasing daily series for one location. Missing days leave the windows that span them NA.
tmax, tmin	Daily maximum and minimum temperature (degrees Celsius).
baseline	Climatology years, <code>c(first, last)</code> .

Value

A data.frame with date, dmt, ehi_sig, ehi_accl and ehf (degrees Celsius squared), indexed by the first day of each three-day period, with attributes t95 and ehf85. NA where the windows are incomplete.

References

Nairn, J.R. & Fawcett, R.J.B. (2015). The Excess Heat Factor: A Metric for Heatwave Intensity and Its Use in Classifying Heatwave Severity. *International Journal of Environmental Research and Public Health*, 12(1), 227-253. doi:10.3390/ijerph120100227

Examples

```
d <- seq(as.Date("2001-01-01"), as.Date("2003-12-31"), by = "day")
doy <- as.POSIXlt(d)$yday
tmax <- 33 + 8 * sin(2 * pi * (doy - 80) / 365)
tmax[d >= as.Date("2003-05-10") & d <= as.Date("2003-05-16")] <- 47
ehf <- excess_heat_factor(d, tmax, tmax - 12, baseline = c(2001, 2002))
subset(ehf, ehf > 0 & date > as.Date("2003-05-01") & date < as.Date("2003-05-31"))
```

exclude_cols	<i>Columns to exclude from aggregation</i>
--------------	--

Description

Aggregation does not apply to these coordinate and metadata columns.

Usage

```
exclude_cols
```

Value

A character vector of variable names.

Examples

```
head(exclude_cols)
```

exposure_indicators	<i>Heat exposure indicators</i>
---------------------	---------------------------------

Description

Summarizes daily temperature (and optionally humidity) into heat exposure indicators per location and year or month, as used in heat-health vulnerability indices: heatwave and warm-night frequency and duration, diurnal temperature range, WBGT, heat index and wet-bulb temperature, plus the ETCCDI indices TX90p, TN90p, WSDI and tropical nights.

Usage

```
exposure_indicators(
  data,
  by = c("latitude", "longitude"),
  period = c("year", "month"),
  months = NULL,
  baseline = c(1991, 2020),
  heatwave_method = c("imd", "percentile", "ehf"),
  warm_night_method = c("imd", "percentile", "absolute"),
  terrain = c("auto", "plains", "coastal", "hilly"),
  hill_elevation = 1200,
  cell_size = NULL
)
```

Arguments

<code>data</code>	A data.frame of daily values; see Details.
<code>by</code>	Columns identifying a location, or NULL for a single series.
<code>period</code>	Summarize by "year" or "month".
<code>months</code>	Months (1-12) to include, e.g. 3:6 for the Indian hot season; NULL for all.
<code>baseline</code>	Climatology years, <code>c(first, last)</code> , for normals and percentiles.
<code>heatwave_method</code>	Passed to <code>heatwave_days()</code> as method.
<code>warm_night_method</code>	Passed to <code>warm_nights()</code> as method.
<code>terrain</code>	Passed to <code>heatwave_days()</code> for the IMD definition. "auto" classifies each location with <code>imd_terrain()</code> from its latitude and longitude columns and adds a terrain column.
<code>hill_elevation, cell_size</code>	Passed to <code>imd_terrain()</code> when <code>terrain = "auto"</code> .

Details

`data` needs columns `date`, `tmax` and `tmin` (degrees Celsius), one row per day per location, covering the baseline years. With a `dewpoint` column (degrees Celsius), afternoon humidity metrics are computed from `tmax` and `dewpoint`: WBGT by `wbgt_simple()`, heat index and wet-bulb temperature. A `wbgt` column, such as daily maxima of `wbgt_liljegren()`, replaces the `wbgt_simple()` estimate.

Spells are found on the full series, so a heatwave spanning two periods counts as an event in the period where it starts and its days count where they fall. `months` filters the summary.

Value

A data.frame with one row per location and period and columns:

n_days days with data

tmax_mean, tmin_mean mean daily maximum and minimum temperature

dtr_mean mean diurnal temperature range (ETCCDI DTR)

hot_days days with Tmax of at least 40 C

tropical_nights nights with Tmin above 20 C (ETCCDI TR)

tx90p, tn90p percent of days above the calendar-day 90th percentile of Tmax and Tmin

wsdi days in runs of at least 6 days with Tmax above its 90th percentile (ETCCDI WSDI)

heatwave_days, heatwave_events, heatwave_longest, severe_heatwave_days heatwave duration (total days), frequency (events starting in the period), longest event and severe days, from `heatwave_days()`

warm_nights, warm_night_spells, warm_night_longest, very_warm_nights the same for `warm_nights()`

With humidity: `wbgt_mean`, `wbgt_max`, `wbgt_days_high` (WBGT of at least 29.4 C, 85 F) and `wbgt_days_extreme` (at least 31.1 C, 88 F), `heat_index_max`, `heat_index_days_danger` (at least 39.4 C, 103 F), `wet_bulb_max` and `wet_bulb_days_31` (wet bulb of at least 31 C, near the 30.55 C mean critical limit of Vecellio et al. 2022).

References

Vecellio, D.J., Wolf, S.T., Cottle, R.M. & Kenney, W.L. (2022). Evaluating the 35 C wet-bulb temperature adaptability threshold for young, healthy subjects (PSU HEAT Project). *Journal of Applied Physiology*, 132(2), 340-345. doi:10.1152/jappphysiol.00738.2021

See Also

[imd_exposure_bbox\(\)](#) to download IMD data and compute these in one call; [ndvi\(\)](#) and [ndbi\(\)](#) for the land-surface indicators.

Examples

```
d <- seq(as.Date("2001-01-01"), as.Date("2004-12-31"), by = "day")
doy <- as.POSIXlt(d)$yday
tmax <- 33 + 8 * sin(2 * pi * (doy - 80) / 365)
tmax[d >= as.Date("2004-05-10") & d <= as.Date("2004-05-14")] <- 46.5
daily <- data.frame(date = d, tmax = tmax, tmin = tmax - 13, dewpoint = 18)
exposure_indicators(daily, by = NULL, baseline = c(2001, 2003), terrain = "plains")

# Jaipur: terrain looked up from the coordinates
daily$latitude <- 26.9
daily$longitude <- 75.8
exposure_indicators(daily, baseline = c(2001, 2003))
```

extract_point_data	<i>Extract data for a specified point location (nearest neighbor)</i>
--------------------	---

Description

Extract data for a specified point location (nearest neighbor)

Usage

```
extract_point_data(netcdf_file, target_lat, target_lon)
```

Arguments

netcdf_file	Path to the NetCDF file, as a character string
target_lat	Target latitude
target_lon	Target longitude

Value

data.frame with data for the nearest point

```
filter_dataframe_by_geojson
```

Filter data.frame by GeoJSON polygon

Description

This function does a test of only the unique (lat, lon) coordinates against the polygon. Then it joins the result to the full dataset with a data.table keyed semi-join.

Usage

```
filter_dataframe_by_geojson(df, geojson_file)
```

Arguments

df	data.frame with latitude and longitude columns
geojson_file	Character string path to GeoJSON file

Value

data.frame with spatially filtered data

```
filter_imd_by_bbox
```

Filter IMD data by spatial boundaries

Description

Filter IMD data by spatial boundaries

Usage

```
filter_imd_by_bbox(data, north, south, east, west)
```

Arguments

data	data.frame with IMD data
north	Latitude of the north boundary
south	Latitude of the south boundary
east	Longitude of the east boundary
west	Longitude of the west boundary

Value

The input data.frame restricted to rows whose latitude/longitude fall inside the region; columns are unchanged.

Examples

```
imd <- data.frame(
  date = rep(as.Date("2023-01-01") + 0:59, each = 2),
  latitude = 20.5, longitude = c(73.5, 79.5), rainfall = 1
)
filter_imd_by_bbox(imd, north = 22, south = 18, east = 76, west = 72)
```

filter_imd_by_geojson *Filter IMD data by GeoJSON polygon*

Description

Filter IMD data by GeoJSON polygon

Usage

```
filter_imd_by_geojson(data, geojson_file)
```

Arguments

data	data.frame with IMD data
geojson_file	Path to the GeoJSON file

Value

The input data.frame restricted to rows whose latitude/longitude fall inside the region; columns are unchanged.

Examples

```
imd <- data.frame(
  date = rep(as.Date("2023-01-01") + 0:59, each = 2),
  latitude = 20.5, longitude = c(73.5, 79.5), rainfall = 1
)
region <- system.file("extdata", "region.geojson", package = "varunayan")
filter_imd_by_geojson(imd, region)
```

filter_netcdf_by_bbox *Filter NetCDF data by spatial boundaries (bounding box)*

Description

Filter NetCDF data by spatial boundaries (bounding box)

Usage

```
filter_netcdf_by_bbox(netcdf_file, north, south, east, west)
```

Arguments

netcdf_file	Path to the NetCDF file, as a character string
north	Northern latitude boundary
south	Southern latitude boundary
east	Eastern longitude boundary
west	Western longitude boundary

Value

data.frame with spatially filtered data

filter_netcdf_by_geojson
Filter NetCDF data by GeoJSON polygon

Description

Filter NetCDF data by GeoJSON polygon

Usage

```
filter_netcdf_by_geojson(netcdf_file, geojson_file)
```

Arguments

netcdf_file	Path to the NetCDF file, as a character string
geojson_file	Character string path to GeoJSON file

Value

data.frame with spatially filtered data

format_date_range	<i>Format date range for CDS API requests</i>
-------------------	---

Description

Format date range for CDS API requests

Usage

```
format_date_range(start_date, end_date)
```

Arguments

start_date	Start date (Date object)
end_date	End date (Date object)

Value

Character vector of date strings in YYYY-MM-DD format

format_pressure_levels	<i>Convert pressure levels to the character format necessary for the CDS API</i>
------------------------	--

Description

Convert pressure levels to the character format necessary for the CDS API

Usage

```
format_pressure_levels(pressure_levels)
```

Arguments

pressure_levels	Numeric or character vector of pressure levels
-----------------	--

Value

Character vector of pressure levels

`get_available_datasets`*Get available dataset types*

Description

Get available dataset types

Usage

```
get_available_datasets()
```

Value

Character vector of available dataset types

Examples

```
get_available_datasets()
```

`get_bbox_from_sf`*Get bounding box from sf object*

Description

Get bounding box from sf object

Usage

```
get_bbox_from_sf(sf_obj)
```

Arguments

`sf_obj` sf object

Value

Named numeric vector with xmin, ymin, xmax, ymax

```
get_cds_credentials
```

Get CDS API credentials from different sources

Description

Checks the CDS_API_KEY environment variable, then the 'ecmwfr' keyring, then ~/.cdsapirc. A key found in the environment variable or in ~/.cdsapirc is registered with 'ecmwfr' via `ecmwfr::wf_set_key()`.

Usage

```
get_cds_credentials(silent = FALSE)
```

Arguments

`silent` If TRUE, do not show information messages

Value

TRUE if credentials are available, FALSE otherwise.

Examples

```
get_cds_credentials()
```

```
get_imd_grid_specs
```

Get IMD grid specifications

Description

Returns the grid specifications for an IMD dataset.

Usage

```
get_imd_grid_specs(dataset)
```

Arguments

`dataset` Dataset identifier: "rainfall_0.25", "rainfall_1.0", "tmax_1.0", "tmin_1.0".

Value

A list with elements `nlat`, `nlon`, `lat_start`, `lat_end`, `lon_start`, `lon_end`, `resolution` (degrees), `unit`, `missing_value`, `format` and `description`.

Examples

```
specs <- get_imd_grid_specs("rainfall_0.25")
```

`get_pressure_level_variable_names`*Get all valid pressure-level variable names*

Description

Get all valid pressure-level variable names

Usage

```
get_pressure_level_variable_names()
```

Value

Character vector of valid pressure-level variable names

`get_pressure_level_variables`*Get the dataset of pressure-level variables*

Description

Get the dataset of pressure-level variables

Usage

```
get_pressure_level_variables()
```

Value

List that contains the pressure-level variable categories

`get_single_level_variable_names`*Get all valid single-level variable names*

Description

Get all valid single-level variable names

Usage

```
get_single_level_variable_names()
```

Value

Character vector of valid single-level variable names

`get_single_level_variables`*Get the dataset of single-level variables*

Description

Get the dataset of single-level variables

Usage

```
get_single_level_variables()
```

Value

List that contains the single-level variable categories

`get_time_values`*Get time values for a frequency*

Description

Get time values for a frequency

Usage

```
get_time_values(frequency)
```

Arguments

frequency Character string that gives the temporal frequency

Value

Character vector of time strings

get_timings	<i>Get all recorded time measurements</i>
-------------	---

Description

Get all recorded time measurements

Usage

```
get_timings()
```

Value

A data.frame with columns label, elapsed_seconds, timestamp and duration (formatted), one row per timed operation; zero rows if nothing has been timed.

Examples

```
get_timings()
```

GetAvailablePressureLevels	<i>Get the standard pressure levels that are available in ERA5</i>
----------------------------	--

Description

Get the standard pressure levels that are available in ERA5

Usage

```
GetAvailablePressureLevels(...)
```

```
get_available_pressure_levels()
```

Arguments

... Arguments that the aliases send to the primary function.

Value

Character vector of pressure levels in hPa

Examples

```
get_available_pressure_levels()
```

GetCacheDir	<i>Get cache directory path</i>
-------------	---------------------------------

Description

Defaults to `tools::R_user_dir("varunayan", "cache")`. Set the option `varunayan.cache_dir` to use a different directory.

Usage

```
GetCacheDir(...)

get_cache_dir(create = TRUE)
```

Arguments

<code>...</code>	Arguments for the main function (for use by aliases).
<code>create</code>	Create the directory when it does not exist (default: TRUE)

Value

Character string with cache directory path

Examples

```
get_cache_dir(create = FALSE)
```

GetERA5CountryTemperature	<i>Get ERA5 temperature data for a country</i>
---------------------------	--

Description

Downloads ERA5 temperature data for a country and aggregates the data. The function finds the country boundary and downloads monthly data with [era5ify_geojson\(\)](#). It returns country-level averages for each year and month, with area weights (cosine of latitude).

Usage

```
GetERA5CountryTemperature(...)

get_era5_country_temperature(
  country,
  start_date,
  end_date,
```

```

    variables = "mean",
    request_id = NULL,
    resolution = 0.25,
    verbose = FALSE,
    ...
  )

```

Arguments

...	More arguments that the function sends to <code>era5ify_geojson()</code> .
country	One of: a country name string (the function uses <code>maps::map()</code> to find it), a path to a GeoJSON file, or an <code>sf</code> object. The function automatically accepts common aliases, for example "USA" and "UK".
start_date	Start date in "YYYY-MM-DD" format.
end_date	End date in "YYYY-MM-DD" format.
variables	Character vector of temperature shorthand names or ERA5 variable names. Short-hands: "mean" (2m_temperature), "max" (maximum_2m_temperature_since_previous_post_processing), "min" (minimum_2m_temperature_since_previous_post_processing). Default: "mean".
request_id	Optional request identifier. If you do not supply it, the function makes it from the country name.
resolution	Spatial resolution in degrees (default: 0.25).
verbose	If TRUE, show detailed progress messages.

Value

A data.frame with the columns year, month, and one column for each requested variable (for example, temperature_mean, temperature_max, temperature_min). The temperature values are in Celsius.

Examples

```

# Monthly mean temperature for India
get_era5_country_temperature("India", "2023-06-01", "2023-06-30")

# Mean and max temperature for the USA
get_era5_country_temperature("USA", "2023-01-01", "2023-12-31",
  variables = c("mean", "max")
)

# Use an sf object directly
library(sf)
poly <- st_read("my_country.geojson")
get_era5_country_temperature(poly, "2023-01-01", "2023-06-30")

```

GetERA5DailyClimate	<i>Download daily ERA5 climate data</i>
---------------------	---

Description

This function downloads ERA5 climate variables (temperature, humidity, wet bulb, wind, solar radiation) for a region.

Usage

```
GetERA5DailyClimate(  
  request_id,  
  start_date,  
  end_date,  
  variables = c("temperature", "humidity"),  
  json_file = NULL,  
  north = NULL,  
  south = NULL,  
  east = NULL,  
  west = NULL,  
  resolution = 0.25,  
  use_cache = TRUE,  
  verbose = FALSE  
)
```

```
get_era5_daily_climate(  
  request_id,  
  start_date,  
  end_date,  
  variables = c("temperature", "humidity"),  
  json_file = NULL,  
  north = NULL,  
  south = NULL,  
  east = NULL,  
  west = NULL,  
  resolution = 0.25,  
  use_cache = TRUE,  
  verbose = FALSE  
)
```

Arguments

request_id	Unique identifier for the data request
start_date	Start date in "YYYY-MM-DD" format
end_date	End date in "YYYY-MM-DD" format

variables	Character vector of variables to download. Options: "temperature", "humidity", "wet_bulb", "wind", "solar"
json_file	Path to the GeoJSON file that defines the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	If TRUE, the function uses cached data (default: TRUE)
verbose	If TRUE, the function prints progress messages (default: FALSE)

Value

data.frame with date, latitude, longitude, and requested variables

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
# Get temperature and humidity
climate <- GetERA5DailyClimate(
  request_id = "india_climate",
  start_date = "2023-05-01",
  end_date = "2023-05-31",
  variables = c("temperature", "humidity"),
  json_file = region
)
```

GetERA5DailyHeatIndexData

Download ERA5 data and calculate heat stress indices

Description

This function downloads ERA5 temperature, dewpoint, and wind data for a region. Then it calculates the heat stress indices.

Usage

```
GetERA5DailyHeatIndexData(
  request_id,
  start_date,
  end_date,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  solar_load = FALSE,
```

```
        resolution = 0.25,  
        use_cache = TRUE,  
        verbose = FALSE  
    )  
  
    get_era5_daily_heat_index_data(  
        request_id,  
        start_date,  
        end_date,  
        json_file = NULL,  
        north = NULL,  
        south = NULL,  
        east = NULL,  
        west = NULL,  
        solar_load = FALSE,  
        resolution = 0.25,  
        use_cache = TRUE,  
        verbose = FALSE  
    )  
  
    GetERA5DailyHeatData(  
        request_id,  
        start_date,  
        end_date,  
        json_file = NULL,  
        north = NULL,  
        south = NULL,  
        east = NULL,  
        west = NULL,  
        solar_load = FALSE,  
        resolution = 0.25,  
        use_cache = TRUE,  
        verbose = FALSE  
    )  
  
    get_era5_daily_heat_data(  
        request_id,  
        start_date,  
        end_date,  
        json_file = NULL,  
        north = NULL,  
        south = NULL,  
        east = NULL,  
        west = NULL,  
        solar_load = FALSE,  
        resolution = 0.25,  
        use_cache = TRUE,  
        verbose = FALSE
```

)

Arguments

request_id	Unique identifier for the data request
start_date	Start date in "YYYY-MM-DD" format
end_date	End date in "YYYY-MM-DD" format
json_file	Path to the GeoJSON file that defines the region (for the geojson method)
north	Northern latitude boundary (for the bbox method)
south	Southern latitude boundary (for the bbox method)
east	Eastern longitude boundary (for the bbox method)
west	Western longitude boundary (for the bbox method)
solar_load	If TRUE, include solar radiation in the calculations (default: FALSE). If TRUE, the function downloads solar radiation data and uses these data for WBGT and UTCI.
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	If TRUE, the function uses cached data when the data is available (default: TRUE)
verbose	If TRUE, the function prints progress messages (default: FALSE)

Value

data.frame with columns: date, latitude, longitude, temp_c, dewpoint_c, wind_speed, rh, wet_bulb, heat_index, wbgt, utci, humidex, wbgt_risk, heat_index_risk, utci_stress. If solar_load=TRUE, the data.frame also contains solar_radiation.

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
heat_data <- GetERA5DailyHeatIndexData(
  request_id = "india_heat_2023",
  start_date = "2023-05-01",
  end_date = "2023-05-31",
  json_file = region
)

# With solar radiation for outdoor WBGT
heat_data <- GetERA5DailyHeatIndexData(
  request_id = "india_heat_2023_solar",
  start_date = "2023-05-01",
  end_date = "2023-05-31",
  json_file = region,
  solar_load = TRUE
)
```

GetERA5DailyHumidity *Download daily ERA5 humidity data*

Description

This function downloads temperature and dewpoint data. Then it calculates the relative humidity.

Usage

```
GetERA5DailyHumidity(
  request_id,
  start_date,
  end_date,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  resolution = 0.25,
  use_cache = TRUE,
  verbose = FALSE
)

get_era5_daily_humidity(
  request_id,
  start_date,
  end_date,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  resolution = 0.25,
  use_cache = TRUE,
  verbose = FALSE
)
```

Arguments

request_id	Unique identifier for the data request
start_date	Start date in "YYYY-MM-DD" format
end_date	End date in "YYYY-MM-DD" format
json_file	Path to the GeoJSON file that defines the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
resolution	Spatial resolution in degrees (default: 0.25)

use_cache If TRUE, the function uses cached data (default: TRUE)
verbose If TRUE, the function prints progress messages (default: FALSE)

Value

data.frame with date, latitude, longitude, temp_c, dewpoint_c, rh

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")  
humidity <- GetERA5DailyHumidity(  
  request_id = "india_humidity",  
  start_date = "2023-05-01",  
  end_date = "2023-05-31",  
  json_file = region  
)
```

GetERA5DailyTemperature

Download daily ERA5 temperature data

Description

Download daily ERA5 temperature data

Usage

```
GetERA5DailyTemperature(  
  request_id,  
  start_date,  
  end_date,  
  json_file = NULL,  
  north = NULL,  
  south = NULL,  
  east = NULL,  
  west = NULL,  
  resolution = 0.25,  
  use_cache = TRUE,  
  verbose = FALSE  
)  
  
get_era5_daily_temperature(  
  request_id,  
  start_date,  
  end_date,  
  json_file = NULL,  
  north = NULL,
```

```

    south = NULL,
    east = NULL,
    west = NULL,
    resolution = 0.25,
    use_cache = TRUE,
    verbose = FALSE
  )

```

Arguments

<code>request_id</code>	Unique identifier for the data request
<code>start_date</code>	Start date in "YYYY-MM-DD" format
<code>end_date</code>	End date in "YYYY-MM-DD" format
<code>json_file</code>	Path to the GeoJSON file that defines the region
<code>north, south, east, west</code>	Bounding box coordinates (alternative to <code>json_file</code>)
<code>resolution</code>	Spatial resolution in degrees (default: 0.25)
<code>use_cache</code>	If TRUE, the function uses cached data (default: TRUE)
<code>verbose</code>	If TRUE, the function prints progress messages (default: FALSE)

Value

data.frame with date, latitude, longitude, temp_c

Examples

```

region <- system.file("extdata", "region.geojson", package = "varunayan")
temp <- GetERA5DailyTemperature(
  request_id = "india_temp",
  start_date = "2023-05-01",
  end_date = "2023-05-31",
  json_file = region
)

```

GetERA5DailyWetBulb *Download daily ERA5 wet bulb temperature data*

Description

This function downloads temperature and dewpoint data. Then it calculates RH and the wet bulb temperature.

Usage

```

GetERA5DailyWetBulb(
  request_id,
  start_date,
  end_date,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  resolution = 0.25,
  use_cache = TRUE,
  verbose = FALSE
)

get_era5_daily_wet_bulb(
  request_id,
  start_date,
  end_date,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  resolution = 0.25,
  use_cache = TRUE,
  verbose = FALSE
)

```

Arguments

<code>request_id</code>	Unique identifier for the data request
<code>start_date</code>	Start date in "YYYY-MM-DD" format
<code>end_date</code>	End date in "YYYY-MM-DD" format
<code>json_file</code>	Path to the GeoJSON file that defines the region
<code>north, south, east, west</code>	Bounding box coordinates (alternative to <code>json_file</code>)
<code>resolution</code>	Spatial resolution in degrees (default: 0.25)
<code>use_cache</code>	If TRUE, the function uses cached data (default: TRUE)
<code>verbose</code>	If TRUE, the function prints progress messages (default: FALSE)

Value

data.frame with date, latitude, longitude, temp_c, dewpoint_c, rh, wet_bulb

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
wetbulb <- GetERA5DailyWetBulb(
  request_id = "india_wetbulb",
  start_date = "2023-05-01",
  end_date = "2023-05-31",
  json_file = region
)
```

GetERA5DailyWind*Download daily ERA5 wind data*

Description

This function downloads the 10m wind components. Then it calculates the wind speed.

Usage

```
GetERA5DailyWind(
  request_id,
  start_date,
  end_date,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  resolution = 0.25,
  use_cache = TRUE,
  verbose = FALSE
)
```

```
get_era5_daily_wind(
  request_id,
  start_date,
  end_date,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
  resolution = 0.25,
  use_cache = TRUE,
  verbose = FALSE
)
```

Arguments

request_id	Unique identifier for the data request
start_date	Start date in "YYYY-MM-DD" format
end_date	End date in "YYYY-MM-DD" format
json_file	Path to the GeoJSON file that defines the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	If TRUE, the function uses cached data (default: TRUE)
verbose	If TRUE, the function prints progress messages (default: FALSE)

Value

data.frame with date, latitude, longitude, wind_speed

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
wind <- GetERA5DailyWind(
  request_id = "india_wind",
  start_date = "2023-05-01",
  end_date = "2023-05-31",
  json_file = region
)
```

GetERA5MonthlyHeatIndexData

Get monthly ERA5 heat stress index data

Description

This function downloads ERA5 heat stress indices and aggregates them to the specified month.

Usage

```
GetERA5MonthlyHeatIndexData(
  request_id,
  year,
  month,
  json_file = NULL,
  north = NULL,
  south = NULL,
  east = NULL,
  west = NULL,
```

```
solar_load = FALSE,  
resolution = 0.25,  
use_cache = TRUE,  
verbose = FALSE  
)  
  
get_era5_monthly_heat_index_data(  
  request_id,  
  year,  
  month,  
  json_file = NULL,  
  north = NULL,  
  south = NULL,  
  east = NULL,  
  west = NULL,  
  solar_load = FALSE,  
  resolution = 0.25,  
  use_cache = TRUE,  
  verbose = FALSE  
)  
  
GetERA5MonthlyHeatData(  
  request_id,  
  year,  
  month,  
  json_file = NULL,  
  north = NULL,  
  south = NULL,  
  east = NULL,  
  west = NULL,  
  solar_load = FALSE,  
  resolution = 0.25,  
  use_cache = TRUE,  
  verbose = FALSE  
)  
  
get_era5_monthly_heat_data(  
  request_id,  
  year,  
  month,  
  json_file = NULL,  
  north = NULL,  
  south = NULL,  
  east = NULL,  
  west = NULL,  
  solar_load = FALSE,  
  resolution = 0.25,  
  use_cache = TRUE,
```

```
    verbose = FALSE
  )
```

Arguments

request_id	Unique identifier for the data request
year	Year (for example, 2023)
month	Month (1-12)
json_file	Path to the GeoJSON file that defines the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
solar_load	If TRUE, include solar radiation (default: FALSE)
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	If TRUE, the function uses cached data (default: TRUE)
verbose	If TRUE, the function prints progress messages (default: FALSE)

Value

A data.frame with one row per grid point for the requested month: year, month, latitude, longitude, monthly means of the daily heat stress variables, and the categorical columns wbgt_risk, heat_index_risk and utci_stress.

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
heat_may <- GetERA5MonthlyHeatIndexData(
  request_id = "india_may2023",
  year = 2023,
  month = 5,
  json_file = region
)
```

GetPackageInfo	<i>Get package version information</i>
----------------	--

Description

Get package version information

Usage

```
GetPackageInfo(...)

get_package_info()
```

Arguments

... Arguments for the main function (for use by aliases).

Value

The `utils::sessionInfo()` object, invisibly. Called for its side effect of reporting the R version, platform and versions of key dependencies as messages.

Examples

```
get_package_info()
```

HadEX3Bbox

Download HadEX3 data on climate extremes for a bounding box

Description

Downloads HadEX3 data on ETCCDI climate extremes and filters the data to a geographic bounding box. The data is for 1901-2018 at a resolution of 1.25 x 1.875 degrees.

Usage

```
HadEX3Bbox(...)
```

```
hadex3_bbox(
  index,
  start_year,
  end_year,
  north,
  south,
  east,
  west,
  frequency = "annual",
  baseline = "61-90",
  output_dir = tempdir(),
  use_cache = TRUE
)
```

Arguments

...	Arguments that the aliases send to the primary function.
index	ETCCDI index name (e.g. "TXx", "Rx1day"). See list_hadex3_indices() .
start_year	Start year (1901-2018).
end_year	End year (1901-2018).
north	North boundary (latitude).
south	South boundary (latitude).

east	East boundary (longitude).
west	West boundary (longitude).
frequency	Temporal frequency: "annual" or "monthly" (default: "annual").
baseline	Reference period for the indices that use percentiles: "61-90" (default) or "81-10". This argument applies only to the annual frequency.
output_dir	Directory for temporary files (default: tempdir()).
use_cache	If TRUE, use the data in the cache (default: TRUE).

Value

A data.frame with the columns: year (and month for monthly data), latitude, longitude, value, index.

Examples

```
# Annual maximum of the daily maximum temperature for Europe, 1981-2010
data <- hadex3_bbox(
  index = "Tx",
  start_year = 1981, end_year = 2010,
  north = 71, south = 36, east = 40, west = -10
)

# Monthly Rx1day for India
data <- hadex3_bbox(
  index = "Rx1day",
  start_year = 2000, end_year = 2018,
  north = 35, south = 8, east = 97, west = 68,
  frequency = "monthly"
)
```

HadEX3Geojson

Download HadEX3 data on climate extremes for a GeoJSON region

Description

Downloads HadEX3 data on ETCCDI climate extremes and filters the data to a geographic region. A GeoJSON polygon defines the region.

Usage

```
HadEX3Geojson(...)
```

```
hadex3_geojson(
  index,
  start_year,
  end_year,
  geojson_file,
```

```
frequency = "annual",
baseline = "61-90",
output_dir = tempdir(),
use_cache = TRUE
)
```

Arguments

- ... Arguments that the aliases send to the primary function.
- index ETCCDI index name (e.g. "TXx", "Rx1day"). See [list_hadex3_indices\(\)](#).
- start_year Start year (1901-2018).
- end_year End year (1901-2018).
- geojson_file Path to a GeoJSON file that defines the region.
- frequency Temporal frequency: "annual" or "monthly" (default: "annual").
- baseline Reference period for the indices that use percentiles: "61-90" (default) or "81-10". This argument applies only to the annual frequency.
- output_dir Directory for temporary files (default: tempdir()).
- use_cache If TRUE, use the data in the cache (default: TRUE).

Value

A data.frame with the columns: year (and month for monthly data), latitude, longitude, value, index.

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
data <- hadex3_geojson(
  index = "TXx",
  start_year = 1981, end_year = 2010,
  geojson_file = region
)
```

heat_index	<i>Calculate heat index (apparent temperature)</i>
------------	--

Description

US National Weather Service heat index: Steadman’s simple formula, replaced by the Rothfus (1990) regression with its low- and high-humidity adjustments once the simple value averaged with temperature reaches 80F.

Usage

```
heat_index(temp_c, rh)
```

Arguments

temp_c	Air temperature in Celsius
rh	Relative humidity as a percentage (0-100)

Value

Heat index in Celsius

References

Rothfus, L.P. (1990). The Heat Index Equation. NWS Southern Region Technical Attachment, SR/SSD 90-23. https://www.weather.gov/media/ffc/ta_htindx.PDF

Examples

```
heat_index(35, 70)
```

```
heat_index_from_dewpoint
```

Calculate heat index from dewpoint

Description

Calculate heat index from dewpoint

Usage

```
heat_index_from_dewpoint(temp_c, dewpoint_c)
```

Arguments

temp_c	Air temperature in Celsius
dewpoint_c	Dewpoint temperature in Celsius

Value

Heat index in Celsius

Examples

```
heat_index_from_dewpoint(35, 25)
```

heat_index_risk_category	<i>Heat index risk categories</i>
--------------------------	-----------------------------------

Description

This function puts heat index values into heat stress risk categories. It uses the NWS categories.

Usage

```
heat_index_risk_category(hi)
```

Arguments

hi Heat index value in Celsius

Value

Factor with the risk category

Examples

```
heat_index_risk_category(c(27, 33, 40, 46, 55))
```

heatwave_days	<i>Heatwave days</i>
---------------	----------------------

Description

Flags the days of a daily maximum temperature series that fall in a heatwave, under one of three definitions:

- "imd": India Meteorological Department. A day qualifies when Tmax is at least 40 C (plains), 37 C (coastal) or 30 C (hilly) and departs at least 4.5 C from the calendar-day normal, or when Tmax is at least 45 C. Severe: departure above 6.4 C with Tmax at least 40 C (plains, coastal) or 30 C (hilly), or Tmax of at least 47 C. A heatwave needs min_duration (2) consecutive qualifying days. IMD also requires 2 stations in a meteorological sub-division; each series here is judged on its own.
- "percentile": Tmax above the calendar-day percentile of the baseline years (5-day window) for at least min_duration (3) days, as in Rohini et al. (2016) for India.
- "ehf": days covered by a three-day period with positive `excess_heat_factor()`; severe when EHF reaches EHF85.

Usage

```

heatwave_days(
  date,
  tmax,
  tmin = NULL,
  method = c("imd", "percentile", "ehf"),
  baseline = c(1991, 2020),
  terrain = c("plains", "coastal", "hilly"),
  percentile = 0.9,
  min_duration = NULL,
  window = 2L
)

```

Arguments

date	Dates of a strictly increasing daily series for one location.
tmax	Daily maximum temperature (degrees Celsius).
tmin	Daily minimum temperature; required for method = "ehf".
method	"imd", "percentile" or "ehf".
baseline	Climatology years, c(first, last), contained in date.
terrain	For "imd": "plains", "coastal" or "hilly"; see imd_terrain() to classify a location.
percentile	Threshold quantile for "percentile".
min_duration	Minimum run of qualifying days; defaults to 2 for "imd" and 3 for "percentile" and "ehf".
window	Half-width in days of the calendar-day window.

Details

The calendar-day normal is the baseline mean over a $2 * \text{window} + 1$ day window. Percentiles use [stats::quantile\(\)](#) type 8 without the in-base bootstrap of the ETCCDI indices.

Value

A data.frame with date, day (in a heatwave), severe, spell_start and spell_length (on the first day of each heatwave).

References

- India Meteorological Department. Heat wave and warm night definitions. <https://mausam.imd.gov.in/pdfs/heatcolduser/Definition.pdf>
- Rohini, P., Rajeevan, M. & Srivastava, A.K. (2016). On the Variability and Increasing Trends of Heat Waves over India. Scientific Reports, 6, 26153. doi:10.1038/srep26153

Examples

```
d <- seq(as.Date("2001-01-01"), as.Date("2003-12-31"), by = "day")
doy <- as.POSIXlt(d)$yday
tmax <- 33 + 8 * sin(2 * pi * (doy - 80) / 365)
tmax[d >= as.Date("2003-05-10") & d <= as.Date("2003-05-13")] <- 46
hw <- heatwave_days(d, tmax, baseline = c(2001, 2002))
subset(hw, day)
```

humidex	<i>Calculate humidex (Canadian heat index)</i>
---------	--

Description

Calculate humidex (Canadian heat index)

Usage

```
humidex(temp_c, dewpoint_c)
```

Arguments

temp_c	Air temperature in Celsius
dewpoint_c	Dewpoint temperature in Celsius

Value

Humidex in Celsius

References

Masterson, J. & Richardson, F.A. (1979). Humidex, A Method of Quantifying Human Discomfort Due to Excessive Heat and Humidity. Environment Canada, Atmospheric Environment Service.

Environment and Climate Change Canada. Glossary: Humidex. https://climate.weather.gc.ca/glossary_e.html

Examples

```
humidex(30, 22)
```

imd_temperature_by_region	<i>Download IMD temperature aggregated for each GeoJSON feature (region x time)</i>
---------------------------	---

Description

`imd_temperature_geojson()` returns the grid points in the GeoJSON boundary. This function is different. It returns one row for each feature (for example, each state) and each time period. Each row contains the mean temperature of the region.

Usage

```
imd_temperature_by_region(  
  request_id,  
  start_year,  
  end_year,  
  geojson_file,  
  feature_col = NULL,  
  var_type = "tmean",  
  frequency = "monthly",  
  use_cache = TRUE  
)
```

Arguments

request_id	Unique identifier for the request.
start_year	Start year (≥ 1951).
end_year	End year (≤ 2024).
geojson_file	Path to a GeoJSON file. Each feature in the file is one region.
feature_col	Name of the GeoJSON property that sets the groups (for example, "state_name"). If NULL, the function uses the first column that is not the geometry column.
var_type	"tmax", "tmin", or "tmean" (mean of tmax and tmin). The default is "tmean".
frequency	"monthly" (default), "yearly", or "daily".
use_cache	If TRUE, use the cached IMD downloads when they are available (default TRUE).

Value

A data.frame with the columns region, year, month (only for monthly data), and temperature.

Examples

```
temp <- tryCatch(
  imd_temperature_by_region(
    request_id = "demo_region",
    start_year = 2023, end_year = 2023,
    geojson_file = system.file("extdata", "region.geojson", package = "varunayan"),
    var_type = "tmax", frequency = "monthly", use_cache = FALSE
  ),
  error = function(e) message("IMD server unavailable: ", conditionMessage(e))
)
```

imd_terrain

IMD heatwave terrain class of a location

Description

Classifies locations as "hilly", "coastal" or "plains" for the IMD heatwave definition in [heatwave_days\(\)](#). IMD sets the Tmax floor by terrain but does not define it. A location is hilly when it lies in Jammu and Kashmir, Ladakh, Himachal Pradesh or Uttarakhand, whose sub-divisions IMD treats as hilly down to the Jammu plains, or when its [india_elevation\(\)](#) reaches hill_elevation, the 1200 m boundary of the mid-hill regions by default. Otherwise it is coastal when land and open sea both lie within about 9 km.

Usage

```
imd_terrain(latitude, longitude, hill_elevation = 1200, cell_size = NULL)
```

Arguments

latitude, longitude	Coordinates in decimal degrees.
hill_elevation	Elevation (m) at or above which a location is hilly.
cell_size	Grid spacing in degrees when the coordinates are cell centres, such as 1 for the IMD temperature grid; NULL for points.

Details

For gridded data set cell_size to classify whole cells from their Indian land: a cell is hilly when most of it lies in those states or its median elevation reaches hill_elevation, and coastal when any of it is.

Value

A character vector of "hilly", "coastal" or "plains", NA outside the grid or over sea.

References

NOAA National Centers for Environmental Information (2022). ETOPO 2022 15 Arc-Second Global Relief Model. doi:10.25921/fd45gt74

Ministry of Panchayati Raj. Local Government Directory district boundaries, via <https://bharatviz.org>.

See Also

`heatwave_days()`, `exposure_indicators()`

Examples

```
# Shimla, Jammu, Delhi, Chennai
imd_terrain(c(31.10, 32.73, 28.61, 13.08), c(77.17, 74.86, 77.21, 80.27))
# IMD 1 degree cells holding Jammu and Rajkot
imd_terrain(c(32.5, 22.5), c(74.5, 70.5), cell_size = 1)
```

IMDExposureBbox

IMD heat exposure indicators for a bounding box

Description

Downloads IMD gridded daily maximum and minimum temperature (1 degree) for the requested years and the baseline, then computes `exposure_indicators()` for each grid cell.

Usage

```
IMDExposureBbox(...)

imd_exposure_bbox(
  start_year,
  end_year,
  north,
  south,
  east,
  west,
  baseline = c(1991, 2020),
  ...,
  output_dir = tempdir(),
  use_cache = TRUE
)
```

Arguments

... Further arguments to `exposure_indicators()`, such as `months`, `period`, `heatwave_method`, `terrain` or `hill_elevation`; `terrain` is classified per 1 degree cell.

`start_year`, `end_year` Years to report (1951-2025).

`north` Latitude of the north boundary.

`south` Latitude of the south boundary.

`east` Longitude of the east boundary.

`west` Longitude of the west boundary.

`baseline` Climatology years, `c(first, last)`, for normals and percentiles.

`output_dir` Directory for the output files (default: `tempdir()`).

`use_cache` If TRUE, use the cached data when it is available (default: TRUE).

Value

The data.frame of `exposure_indicators()` for `start_year` to `end_year`, one row per grid cell (latitude, longitude) and period.

Examples

```
jaipur <- imd_exposure_bbox(2022, 2024,
  north = 28, south = 26,
  east = 77, west = 75, months = 3:6
)
```

IMDRainfallBbox

Download and process IMD rainfall data for a bounding box

Description

Downloads IMD gridded rainfall data and keeps only the data in the specified bounding box.

Usage

```
IMDRainfallBbox(...)
```

```
imd_rainfall_bbox(
  request_id,
  start_year,
  end_year,
  north,
  south,
  east,
  west,
```

```

    resolution = 0.25,
    frequency = "daily",
    save_raw = FALSE,
    output_dir = tempdir(),
    use_cache = TRUE
  )

```

Arguments

<code>...</code>	Arguments that the aliases send to the primary function.
<code>request_id</code>	Unique identifier for the request.
<code>start_year</code>	Start year.
<code>end_year</code>	End year.
<code>north</code>	Latitude of the north boundary.
<code>south</code>	Latitude of the south boundary.
<code>east</code>	Longitude of the east boundary.
<code>west</code>	Longitude of the west boundary.
<code>resolution</code>	Spatial resolution: 0.25 or 1.0 degrees (default: 0.25).
<code>frequency</code>	Temporal frequency: "daily", "monthly", "yearly" (default: "daily").
<code>save_raw</code>	If TRUE, keep the raw downloaded files (default: FALSE).
<code>output_dir</code>	Directory for the output files (default: tempdir()).
<code>use_cache</code>	If TRUE, use the cached data when it is available (default: TRUE).

Value

A data.frame with columns date, latitude, longitude and rainfall (mm), restricted to the requested region and aggregated to frequency.

Examples

```

data <- imd_rainfall_bbox(
  request_id = "maharashtra_rain",
  start_year = 2023,
  end_year = 2024,
  north = 22, south = 16,
  east = 80, west = 73,
  resolution = 0.25,
  frequency = "monthly"
)

```

IMDRainfallGeojson	<i>Download and process IMD rainfall data for a GeoJSON region</i>
--------------------	--

Description

Download and process IMD rainfall data for a GeoJSON region

Usage

```
IMDRainfallGeojson(...)  
  
imd_rainfall_geojson(  
  request_id,  
  start_year,  
  end_year,  
  geojson_file,  
  resolution = 0.25,  
  frequency = "daily",  
  save_raw = FALSE,  
  output_dir = tempdir(),  
  use_cache = TRUE  
)
```

Arguments

...	Arguments that the aliases send to the primary function.
request_id	Unique identifier for the request.
start_year	Start year (1901-2025).
end_year	End year (1901-2025).
geojson_file	Path to the GeoJSON file.
resolution	Spatial resolution: 0.25 or 1.0 degrees (default: 0.25).
frequency	Temporal frequency (default: "daily").
save_raw	If TRUE, keep the raw files (default: FALSE).
output_dir	Directory for the output files (default: tempdir()).
use_cache	If TRUE, use the cached data when it is available (default: TRUE).

Value

A data.frame with columns date, latitude, longitude and rainfall (mm), restricted to the requested region and aggregated to frequency.

Examples

```

region <- system.file("extdata", "region.geojson", package = "varunayan")
data <- imd_rainfall_geojson(
  request_id = "mh_rain", start_year = 2023, end_year = 2023,
  geojson_file = region, frequency = "monthly"
)

```

IMDRainfallPoint	<i>Download and process IMD rainfall for a single point (nearest grid cell)</i>
------------------	---

Description

Extracts the daily rainfall series for the IMD grid cell that is nearest to a (lat, lon) point. For each year, the function reads only that one cell from the disk. Thus, this function is the method with the lowest cost to get a long series for one location.

Usage

```

IMDRainfallPoint(...)

imd_rainfall_point(
  request_id,
  start_year,
  end_year,
  latitude,
  longitude,
  resolution = 0.25,
  frequency = "daily",
  save_raw = FALSE,
  output_dir = tempdir(),
  use_cache = TRUE
)

```

Arguments

...	Arguments that the aliases send to the primary function.
request_id	Unique identifier for the request.
start_year	Start year (1901-2024).
end_year	End year (1901-2024).
latitude	Latitude of the point.
longitude	Longitude of the point.
resolution	Spatial resolution: 0.25 or 1.0 degrees (default: 0.25).
frequency	Temporal frequency: "daily", "monthly", "yearly" (default: "daily").

save_raw If TRUE, keep the raw downloaded files (default: FALSE).
output_dir Directory for the output files (default: tempdir()).
use_cache If TRUE, use the cached data when it is available (default: TRUE).

Value

data.frame with the rainfall series of the nearest cell.

Examples

```
powai <- tryCatch(  
  imd_rainfall_point("powai", 2023, 2023, 19.1277, 72.9047,  
    resolution = 1, use_cache = FALSE  
  ),  
  error = function(e) message("IMD server unavailable: ", conditionMessage(e))  
)
```

IMDStations	<i>IMD weather stations with coordinates</i>
-------------	--

Description

Gets the India Meteorological Department (IMD) city-weather station list. As an option, it also gets the latitude and longitude of each station. The legacy IMD endpoints give only the station names. The responsive IMD site gives a JSON record for each station (api/fetchCity_static.php). This record contains lat and lon. The function adds these values to the station list. You can use the returned coordinates directly as input to [era5ify_point\(\)](#). You can also use them to make maps.

Usage

```
IMDStations(...)  
  
imd_stations(  
  with_coordinates = TRUE,  
  cores = if (.Platform$OS.type == "windows") 1L else 2L  
)
```

Arguments

... Not used. This argument is necessary for compatibility with the aliases.
with_coordinates Logical. If TRUE (default), the function gets latitude and longitude for each station (one request for each station). If FALSE, the function returns only station_id and name.
cores Integer. The number of parallel workers that get the coordinates (the function sends this value to [parallel::mclapply\(\)](#)). The default is 2 (1 on Windows, where forking is not available).

Value

A data.frame with the columns station_id and name. If with_coordinates = TRUE, the data frame also has the columns latitude and longitude. The value is NA if IMD has no coordinate for a station.

Examples

```
stations <- tryCatch(
  imd_stations(with_coordinates = FALSE),
  error = function(e) message("IMD server unavailable: ", conditionMessage(e))
)
```

IMDTemperatureBbox	<i>Download and process IMD temperature data for a bounding box</i>
--------------------	---

Description

Download and process IMD temperature data for a bounding box

Usage

```
IMDTemperatureBbox(...)

imd_temperature_bbox(
  request_id,
  start_year,
  end_year,
  north,
  south,
  east,
  west,
  var_type = "tmax",
  frequency = "daily",
  save_raw = FALSE,
  output_dir = tempdir(),
  use_cache = TRUE
)
```

Arguments

...	Arguments that the aliases send to the primary function.
request_id	Unique identifier for the request.
start_year	Start year (1951-2025).
end_year	End year (1951-2025).
north	Latitude of the north boundary.

south	Latitude of the south boundary.
east	Longitude of the east boundary.
west	Longitude of the west boundary.
var_type	Temperature type: "tmax" or "tmin".
frequency	Temporal frequency (default: "daily").
save_raw	If TRUE, keep the raw files (default: FALSE).
output_dir	Directory for the output files (default: tempdir()).
use_cache	If TRUE, use the cached data when it is available (default: TRUE).

Value

A data.frame with columns date, latitude, longitude and temperature (degrees Celsius), restricted to the requested region and aggregated to frequency.

Examples

```
data <- imd_temperature_bbox(
  request_id = "delhi_tmax",
  start_year = 2023,
  end_year = 2024,
  north = 29, south = 28,
  east = 78, west = 76,
  var_type = "tmax"
)
```

IMDTemperatureGeojson *Download and process IMD temperature data for a GeoJSON region*

Description

Download and process IMD temperature data for a GeoJSON region

Usage

```
IMDTemperatureGeojson(...)

imd_temperature_geojson(
  request_id,
  start_year,
  end_year,
  geojson_file,
  var_type = "tmax",
  frequency = "daily",
  save_raw = FALSE,
  output_dir = tempdir(),
  use_cache = TRUE
)
```

Arguments

...	Arguments that the aliases send to the primary function.
request_id	Unique identifier for the request.
start_year	Start year (1951-2025).
end_year	End year (1951-2025).
geojson_file	Path to the GeoJSON file.
var_type	Temperature type: "tmax" or "tmin".
frequency	Temporal frequency (default: "daily").
save_raw	If TRUE, keep the raw files (default: FALSE).
output_dir	Directory for the output files (default: tempdir()).
use_cache	If TRUE, use the cached data when it is available (default: TRUE).

Value

A data.frame with columns date, latitude, longitude and temperature (degrees Celsius), restricted to the requested region and aggregated to frequency.

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
data <- imd_temperature_geojson(
  request_id = "mh_tmax", start_year = 2023, end_year = 2023,
  geojson_file = region, var_type = "tmax", frequency = "monthly"
)
```

india_elevation	<i>Elevation of locations in India</i>
-----------------	--

Description

Looks up surface elevation from a 5 arc-minute (about 9 km) grid bundled with the package, covering 6-38 N and 66-98 E. Each value is the median land elevation of the 5 arc-minute cell holding the location, built by data-raw/imd_terrain.R from ETOPO 2022 at 60 arc-seconds.

Usage

```
india_elevation(latitude, longitude)
```

Arguments

latitude, longitude	Coordinates in decimal degrees.
---------------------	---------------------------------

Value

Elevation in metres, NA over open sea or outside the grid.

References

NOAA National Centers for Environmental Information (2022). ETOPO 2022 15 Arc-Second Global Relief Model. doi:[10.25921/fd45gt74](https://doi.org/10.25921/fd45gt74)

See Also

[imd_terrain\(\)](#)

Examples

```
# Shimla, Delhi, Leh
india_elevation(c(31.10, 28.61, 34.16), c(77.17, 77.21, 77.58))
```

is_max_var

Find if a variable uses maximum aggregation

Description

Find if a variable uses maximum aggregation

Usage

```
is_max_var(var_name)
```

Arguments

var_name Variable name as a character string

Value

Logical. TRUE if the variable uses maximum aggregation

Examples

```
is_max_var("2m_temperature")
```

is_min_var	<i>Find if a variable uses minimum aggregation</i>
------------	--

Description

Find if a variable uses minimum aggregation

Usage

```
is_min_var(var_name)
```

Arguments

var_name	Variable name as a character string
----------	-------------------------------------

Value

Logical. TRUE if the variable uses minimum aggregation

Examples

```
is_min_var("2m_temperature")
```

is_rate_var	<i>Find if a variable is a rate variable</i>
-------------	--

Description

Find if a variable is a rate variable

Usage

```
is_rate_var(var_name)
```

Arguments

var_name	Variable name as a character string
----------	-------------------------------------

Value

Logical. TRUE if the variable is a rate

Examples

```
is_rate_var("2m_temperature")
```

is_sum_var	<i>Find if a variable uses sum aggregation</i>
------------	--

Description

Find if a variable uses sum aggregation

Usage

```
is_sum_var(var_name)
```

Arguments

var_name	Variable name as a character string
----------	-------------------------------------

Value

Logical. TRUE if the variable uses sum aggregation

Examples

```
is_sum_var("total_precipitation")
```

is_valid_geojson	<i>Find if data is valid GeoJSON</i>
------------------	--------------------------------------

Description

Find if data is valid GeoJSON

Usage

```
is_valid_geojson(data)
```

Arguments

data	List or data structure to examine
------	-----------------------------------

Value

Logical. TRUE if the data is valid GeoJSON

list_cache	<i>List cached files</i>
------------	--------------------------

Description

List cached files

Usage

```
list_cache(source = NULL)
```

Arguments

source Filter by source: "era5", "imd", "cru_ts", "hadex3", or NULL for all

Value

A data.frame with columns file, source, size_mb and modified, one row per cached file; zero rows if the cache is empty.

Examples

```
list_cache()
```

list_imd_datasets	<i>List available IMD datasets</i>
-------------------	------------------------------------

Description

List available IMD datasets

Usage

```
list_imd_datasets()
```

Value

data.frame with the available IMD datasets and their properties.

Examples

```
list_imd_datasets()
```

ListAvailableVariables

List all available variables for a dataset type

Description

List all available variables for a dataset type

Usage

```
ListAvailableVariables(...)
```

```
list_available_variables(dataset_type = "all")
```

Arguments

... Arguments that the aliases send to the primary function.
dataset_type Character string that gives the dataset type

Value

Character vector of variable names

Examples

```
head(list_available_variables("single"))
```

ListCRUTSVariables

Variable data dictionary for CRU TS

Description

Gives the common names, units, and ERA5 equivalents of the CRU TS variable codes.

Usage

```
ListCRUTSVariables(...)
```

```
list_cru_ts_variables(...)
```

Arguments

... Arguments that the aliases send to the primary function.

Value

A data.frame with the columns: variable, name, description, unit, era5_equivalent, hadex3_equivalent.

Examples

```
list_cru_ts_variables()
```

ListHadEX3Indices	<i>List the available HadEX3 ETCCDI indices</i>
-------------------	---

Description

Returns a data.frame that describes all the ETCCDI indices of climate extremes in the HadEX3 dataset. The data.frame shows the temporal availability and the physical quantity of each index.

Usage

```
ListHadEX3Indices(...)
```

```
list_hadex3_indices(frequency = "all")
```

Arguments

...	Arguments that the aliases send to the primary function.
frequency	Filter by frequency: "annual", "monthly", or "all" (default).

Value

A data.frame with the columns: index, category, description, unit, annual, monthly.

Examples

```
list_hadex3_indices()
list_hadex3_indices(frequency = "monthly")
```

max_vars	<i>Variables for maximum aggregation</i>
----------	--

Description

Variables for maximum aggregation

Usage

```
max_vars
```

Value

A character vector of variable names.

Examples

```
head(max_vars)
```

```
mean_radiant_temperature
```

Estimate mean radiant temperature

Description

This function estimates the mean radiant temperature (Tmrt) from meteorological variables. If full radiation data is not available, the function uses a simplified method.

Usage

```
mean_radiant_temperature(temp_c, solar_radiation = NULL, wind_speed = NULL)
```

Arguments

temp_c	Air temperature in Celsius
solar_radiation	Solar radiation in W/m2 (optional)
wind_speed	Wind speed in m/s (optional)

Value

Mean radiant temperature in Celsius

Examples

```
mean_radiant_temperature(30, solar_radiation = 800, wind_speed = 2)
```

```
min_vars
```

Variables for minimum aggregation

Description

Variables for minimum aggregation

Usage

```
min_vars
```

Value

A character vector of variable names.

Examples

```
head(min_vars)
```

ndbi	<i>Normalized Difference Built-up Index</i>
------	---

Description

NDBI = (SWIR - NIR) / (SWIR + NIR) from surface reflectance, using the 1.55-1.75 um shortwave infrared band, e.g. Landsat 8-9 bands 6 and 5, Sentinel-2 bands 11 and 8, or Landsat 5 TM bands 5 and 4 as in Zha et al. (2003). Positive values indicate built-up or bare surfaces. Works on numeric vectors and on terra::SpatRaster layers.

Usage

```
ndbi(swir, nir)
```

Arguments

swir	Shortwave infrared reflectance.
nir	Near-infrared reflectance.

Value

NDBI in the range -1 to 1, the same shape as the inputs.

References

Zha, Y., Gao, J. & Ni, S. (2003). Use of normalized difference built-up index in automatically mapping urban areas from TM imagery. International Journal of Remote Sensing, 24(3), 583-594.
[doi:10.1080/01431160304987](https://doi.org/10.1080/01431160304987)

See Also

[ndvi\(\)](#)

Examples

```
ndbi(swir = 0.30, nir = 0.22)
```

ndvi	<i>Normalized Difference Vegetation Index</i>
------	---

Description

NDVI = (NIR - Red) / (NIR + Red) from surface reflectance, e.g. Landsat 8-9 bands 5 and 4, Sentinel-2 bands 8 and 4, or MODIS bands 2 and 1. Works on numeric vectors and on `terra::SpatRaster` layers.

Usage

```
ndvi(nir, red)
```

Arguments

nir	Near-infrared reflectance.
red	Red reflectance.

Value

NDVI in the range -1 to 1, the same shape as the inputs.

References

Tucker, C.J. (1979). Red and photographic infrared linear combinations for monitoring vegetation. *Remote Sensing of Environment*, 8(2), 127-150. doi:[10.1016/00344257\(79\)900130](https://doi.org/10.1016/00344257(79)900130)

See Also

[ndbi\(\)](#)

Examples

```
ndvi(nir = 0.45, red = 0.08)
```

parse_date_flexible	<i>Parse a date string in different formats</i>
---------------------	---

Description

Parse a date string in different formats

Usage

```
parse_date_flexible(date_string)
```

Arguments

date_string Character string in one of different date formats

Value

Date object

PopulateSampleCache *Fill the cache with sample IMD data*

Description

Copies sample IMD data files to the cache for offline tests.

Usage

```
PopulateSampleCache(...)
```

```
populate_sample_cache(force = FALSE)
```

Arguments

... Arguments for the main function (for use by aliases).

force If TRUE, overwrite the cached files that exist (default: FALSE)

Value

Invisibly, a named list of paths to the cached sample files, or NULL if no sample directory exists.

Examples

```
populate_sample_cache()
```

print_timing_summary *Print a summary of the time data*

Description

Print a summary of the time data

Usage

```
print_timing_summary()
```

Value

The timing data frame from `get_timings()`, invisibly (NULL if empty).

Examples

```
print_timing_summary()
```

process_imd_files	<i>Process multiple IMD files</i>
-------------------	-----------------------------------

Description

Reads and combines multiple IMD data files.

Usage

```
process_imd_files(  
  file_paths,  
  var_type,  
  resolution = NULL,  
  years = NULL,  
  bbox = NULL  
)
```

Arguments

file_paths	Character vector of file paths.
var_type	Variable type: "rain", "tmax", or "tmin".
resolution	Resolution (for rainfall): 0.25 or 1.0.
years	Integer vector. The year of each file.
bbox	Optional named list or vector (north, south, east, west). The function sends it to the rainfall reader, which crops the grid at read time. If NULL, the function reads the full grid (default).

Value

A data.frame with columns date, latitude, longitude and the value column (rainfall or temperature), combining all files.

Examples

```
files <- download_imd_temperature(2023, 2024, var_type = "tmax")  
data <- process_imd_files(files, "tmax", years = c(2023, 2024))
```

`process_netcdf_to_dataframe`*Extract data from a NetCDF file into a data.frame*

Description

This function uses ncdf4 to read the file quickly. If ncdf4 fails, it uses stars.

Usage

```
process_netcdf_to_dataframe(netcdf_file)
```

Arguments

`netcdf_file` Path to the NetCDF file, as a character string

Value

data.frame with processed climate data

`rate_vars`*Rate variables to average*

Description

Rate variables to average

Usage

```
rate_vars
```

Value

A character vector of variable names.

Examples

```
head(rate_vars)
```

read_cdsapirc	<i>Read CDS API credentials from ~/.cdsapirc file</i>
---------------	---

Description

Read CDS API credentials from ~/.cdsapirc file

Usage

```
read_cdsapirc(file_path = "~/cdsapirc")
```

Arguments

file_path	Path to .cdsapirc file (default: "~/cdsapirc")
-----------	--

Value

A named list of the fields in the file (typically url and key), or NULL if the file does not exist or cannot be parsed.

Examples

```
f <- tempfile()
writeLines(c("url: https://cds.climate.copernicus.eu/api", "key: abc-123"), f)
read_cdsapirc(f)
```

read_imd_rainfall	<i>Read IMD NetCDF rainfall file</i>
-------------------	--------------------------------------

Description

Reads IMD NetCDF rainfall data and converts the data to a data.frame.

Usage

```
read_imd_rainfall(file_path, resolution, year, bbox = NULL)
```

Arguments

file_path	Path to the IMD NetCDF file (.nc).
resolution	Spatial resolution: 0.25 or 1.0.
year	Year of the data.
bbox	Optional named list or vector (north, south, east, west). If you set it, the function reads only that spatial slab from the disk (fast path). If NULL, the function reads the full grid.

Value

data.frame with columns: date, latitude, longitude, rainfall

Examples

```
f <- download_imd_rainfall(2023, 2023, resolution = 0.25)
rain_data <- read_imd_rainfall(f, 0.25, 2023)
```

read_imd_temperature	<i>Read IMD binary temperature file</i>
----------------------	---

Description

Reads IMD binary temperature data (maximum or minimum) and converts the data to a data.frame.

Usage

```
read_imd_temperature(file_path, var_type, year)
```

Arguments

file_path	Path to the IMD binary file (.grd).
var_type	Variable type: "tmax" or "tmin".
year	Year of the data.

Value

data.frame with columns: date, latitude, longitude, temperature

Examples

```
f <- download_imd_temperature(2023, 2023, var_type = "tmax")
tmax_data <- read_imd_temperature(f, "tmax", 2023)
```

relative_humidity_from_dewpoint
<i>Calculate relative humidity from temperature and dewpoint</i>

Description

This function uses the Magnus-Tetens approximation to calculate the relative humidity from the air temperature and the dewpoint temperature.

Usage

relative_humidity_from_dewpoint(temp_c, dewpoint_c)

Arguments

- temp_c Air temperature in Celsius
- dewpoint_c Dewpoint temperature in Celsius

Value

Relative humidity as a percentage (0-100)

Examples

relative_humidity_from_dewpoint(25, 18)

SearchVariable	<i>Search for ERA5 variables by keyword</i>
----------------	---

Description

Search for ERA5 variables by keyword

Usage

SearchVariable(...)

search_variable(keyword, dataset_type = "all", exact_match = FALSE)

Arguments

- ... Arguments that the aliases send to the primary function.
- keyword Character string to find in variable names and descriptions
- dataset_type Character string that gives the dataset type ("single", "pressure", or "all")
- exact_match Logical. If TRUE, use an exact match

Value

A data.frame with columns variable_name, description, units, dataset_type and category, one row per matching variable.

Examples

```
# Search for temperature variables
temp_vars <- search_variable("temperature")

# Search only in pressure level data
pressure_temp <- search_variable("temperature", dataset_type = "pressure")
```

SetupCdsCredentials *Configure CDS credentials for ERA5 data access*

Description

This function configures the Copernicus Climate Data Store (CDS) credentials for ERA5 downloads. The function examines the credentials in this order:

1. The key argument that you give directly
2. CDS_API_KEY environment variable
3. ~/.cdsapirc file

Usage

```
SetupCdsCredentials(...)

setup_cds_credentials(key = NULL)
```

Arguments

...	Arguments passed to the main function (used by aliases).
key	Your CDS API key (UUID format). If NULL, the function tries to read the key from the CDS_API_KEY environment variable or the ~/.cdsapirc file.

Value

Invisibly, TRUE once the key has been registered with 'ecmwf'. Errors if no key can be found.

Examples

```
setup_cds_credentials()
```

ShowCacheInfo	<i>Show cache contents and statistics</i>
---------------	---

Description

Show cache contents and statistics

Usage

```
ShowCacheInfo(...)
```

```
show_cache_info()
```

Arguments

... Arguments for the main function (for use by aliases).

Value

Invisibly, a data.frame with columns name, size_mb and mtime describing cached files, or NULL if the cache is empty. Called mainly for its side effect of printing a summary as messages.

Examples

```
show_cache_info()
```

sum_vars	<i>Variables to sum over time</i>
----------	-----------------------------------

Description

Accumulation variables (precipitation, radiation). Temporal aggregation uses the sum for these variables.

Usage

```
sum_vars
```

Value

A character vector of variable names.

Examples

```
head(sum_vars)
```

time_operation	<i>Measure the time of an operation and record the duration</i>
----------------	---

Description

Measure the time of an operation and record the duration

Usage

```
time_operation(expr, label, verbose = TRUE)
```

Arguments

expr	Expression to measure
label	Label for the operation
verbose	If TRUE, print the time information (default: TRUE)

Value

The value of expr. The elapsed time is recorded and can be retrieved with [get_timings\(\)](#).

Examples

```
x <- time_operation(sum(1:10), "sum")
get_timings()
```

utci	<i>Calculate UTCI (Universal Thermal Climate Index)</i>
------	---

Description

This function uses the polynomial approximation of Bröde et al. (2012) to calculate UTCI. The approximation uses a multi-node model of human thermoregulation.

Usage

```
utci(temp_c, wind_speed, dewpoint_c, tmrt = NULL)
```

Arguments

temp_c	Air temperature in Celsius (-50 to 50)
wind_speed	Wind speed at 10m height in m/s (0.5 to 17)
dewpoint_c	Dewpoint temperature in Celsius (to calculate vapor pressure)
tmrt	Mean radiant temperature in Celsius (optional, the default is temp_c)

Value

UTCI in Celsius

References

Bröde, P., et al. (2012). Deriving the operational procedure for the Universal Thermal Climate Index (UTCI). International Journal of Biometeorology, 56(3), 481-494. doi:[10.1007/s0048401104541](https://doi.org/10.1007/s0048401104541)

Examples

```
utci(30, wind_speed = 2, dewpoint_c = 20)
utci(35, wind_speed = 1, dewpoint_c = 25, tmrt = 45)
```

utci_category	<i>UTCI thermal stress categories</i>
---------------	---------------------------------------

Description

This function puts UTCI values into thermal stress categories.

Usage

```
utci_category(utci_val)
```

Arguments

utci_val	UTCI value in Celsius
----------	-----------------------

Value

Factor with the thermal stress category

Examples

```
utci_category(c(-20, 0, 20, 30, 40))
```

utci_sparse	<i>Calculate UTCI with a sparse regression approximation</i>
-------------	--

Description

This function uses the sparse regression approximation of Roman et al. (2025) to calculate UTCI. This approximation uses Legendre polynomials. It is more accurate than the original polynomial of Bröde et al. (2012). It has fewer terms and better numerical stability.

Usage

```
utci_sparse(temp_c, tmrt, wind_speed, rh)
```

Arguments

temp_c	Air temperature in Celsius (-50 to 50)
tmrt	Mean radiant temperature in Celsius
wind_speed	Wind speed at 10m height in m/s (0.5 to 30.3)
rh	Relative humidity as a percentage (5 to 100)

Value

UTCI in Celsius

References

Roman, S., Todorovski, L., Dzeroski, S., & Skok, G. (2025). Approximating the universal thermal climate index using sparse regression with orthogonal polynomials. [doi:10.48550/arXiv.2508.11307](https://doi.org/10.48550/arXiv.2508.11307)

Roman, S. (2025). Code for Approximating the universal thermal climate index (UTCI) using sparse regression with orthogonal polynomials. Zenodo. [doi:10.5281/zenodo.17465548](https://doi.org/10.5281/zenodo.17465548)

Examples

```
utci_sparse(30, tmrt = 40, wind_speed = 2, rh = 60)
utci_sparse(35, tmrt = 35, wind_speed = 1, rh = 80)
```

validate_bbox	<i>Make sure that bounding box coordinates are valid</i>
---------------	--

Description

Make sure that bounding box coordinates are valid

Usage

```
validate_bbox(north, south, east, west)
```

Arguments

north	Northern boundary
south	Southern boundary
east	Eastern boundary
west	Western boundary

Value

Logical. TRUE if the bounding box is valid

validate_coordinates	<i>Make sure that geographic coordinates are valid</i>
----------------------	--

Description

Make sure that geographic coordinates are valid

Usage

```
validate_coordinates(lat, lon)
```

Arguments

lat	Latitude value
lon	Longitude value

Value

Logical. TRUE if the coordinates are valid

`validate_era5_variables`*Validate ERA5 variable names*

Description

Validate ERA5 variable names

Usage

```
validate_era5_variables(variables, dataset_type)
```

Arguments

<code>variables</code>	Character vector of variable names to validate
<code>dataset_type</code>	Character string: "single" or "pressure"

Value

Logical vector that shows which variables are valid

`vapor_pressure_from_dewpoint`*Calculate vapor pressure from dewpoint temperature*

Description

Calculate vapor pressure from dewpoint temperature

Usage

```
vapor_pressure_from_dewpoint(dewpoint_c)
```

Arguments

<code>dewpoint_c</code>	Dewpoint temperature in Celsius
-------------------------	---------------------------------

Value

Vapor pressure in hPa (hectopascals)

Examples

```
vapor_pressure_from_dewpoint(20)
```

varunayan-aliases *PascalCase Aliases*

Description

PascalCase aliases for all exported functions. The aliases keep backward compatibility.

Usage

```
ERA5ifyGeojson(...)

era5ify_geojson(
  request_id,
  variables,
  start_date,
  end_date,
  json_file,
  dataset_type = "single",
  pressure_levels = NULL,
  frequency = "hourly",
  resolution = 0.25,
  save_raw = FALSE,
  output_dir = tempdir(),
  use_cache = TRUE,
  convert_units = TRUE,
  verbose = FALSE,
  level = c("union", "feature")
)
```

Arguments

...	Arguments passed to the main function (used by aliases).
request_id	Unique identifier for the data request
variables	List of ERA5 variables to download
start_date	Start date in "YYYY-MM-DD" format
end_date	End date in "YYYY-MM-DD" format
json_file	Path to the GeoJSON file that defines the region
dataset_type	Type of dataset ("single" or "pressure")
pressure_levels	Pressure levels for pressure dataset
frequency	Temporal frequency ("hourly", "daily", "monthly")
resolution	Spatial resolution in degrees (default: 0.25)
save_raw	If TRUE, save the raw downloaded files
output_dir	Directory for the saved files (default: tempdir())

use_cache	If TRUE, use cached data when it is available (default: TRUE)
convert_units	If TRUE, convert units to user-friendly formats (default: TRUE). When TRUE, the function returns temperature variables in Celsius (converted from Kelvin). It returns precipitation variables in mm (converted from meters).
verbose	If TRUE, print detailed progress messages
level	"union" (default) returns one row for each grid cell inside the union of all the features. "feature" returns one row for each feature (for example, each district), with the mean of the cells inside it. Such rows have a feature_id column (the row number of the feature) and the feature properties in place of latitude and longitude. A feature smaller than a grid cell gets the nearest cell.

Value

data.frame with processed climate data. When convert_units = TRUE:

- Temperature variables (e.g., 2m_temperature): Celsius
- Precipitation variables (e.g., total_precipitation): mm
- Other variables: original ERA5 units

Examples

```
region <- system.file("extdata", "region.geojson", package = "varunayan")
setup_cds_credentials(key = "your-cds-api-key")

data <- era5ify_geojson(
  request_id = "my_request",
  variables = c("2m_temperature", "total_precipitation"),
  start_date = "2023-01-01",
  end_date = "2023-01-31",
  json_file = region
)
# Temperature is in Celsius. Precipitation is in mm.
```

visualize_grid_overlap

Show the overlap of grid points and polygons

Description

Show the overlap of grid points and polygons

Usage

```
visualize_grid_overlap(
  grid_data,
  polygons,
  show_centroids = TRUE,
  grid_color = "#E74C3C",
  polygon_fill = NA,
  polygon_color = "#2C3E50",
  centroid_color = "#3498DB",
  title = "Grid-Polygon Overlap"
)
```

Arguments

<code>grid_data</code>	Data frame with longitude and latitude columns, or the path to a NetCDF or CSV file
<code>polygons</code>	sf object, or the path to a GeoJSON file
<code>show_centroids</code>	Logical. If TRUE, show the polygon centroids
<code>grid_color</code>	Color for grid points
<code>polygon_fill</code>	Fill color for polygons
<code>polygon_color</code>	Border color for polygons
<code>centroid_color</code>	Color for centroids
<code>title</code>	Plot title

Value

ggplot object

Examples

```
grid <- expand.grid(longitude = seq(72.5, 77.5, by = 1), latitude = seq(18.5, 21.5, by = 1))
grid$temperature <- seq_len(nrow(grid))
cells <- sf::st_make_grid(sf::st_as_sfc(sf::st_bbox(
  c(xmin = 72, ymin = 18, xmax = 78, ymax = 22),
  crs = 4326
)), n = c(2, 1))
polys <- sf::st_sf(region = c("west", "east"), geometry = cells)
visualize_grid_overlap(grid, polys)
```

warm_nights

*Warm nights***Description**

Flags warm nights in a daily minimum temperature series:

- "imd": India Meteorological Department. Tmax of at least 40 C and Tmin departing at least 4.5 C from the calendar-day normal; very warm (flagged severe) above 6.4 C.
- "percentile": Tmin above the calendar-day percentile of the baseline years (ETCCDI TN90p with the default 0.9).
- "absolute": Tmin above threshold (ETCCDI tropical nights, TR, with the default 20 C).

Usage

```
warm_nights(
  date,
  tmin,
  tmax = NULL,
  method = c("imd", "percentile", "absolute"),
  baseline = c(1991, 2020),
  percentile = 0.9,
  threshold = 20,
  min_duration = 1L,
  window = 2L
)
```

Arguments

date	Dates of a strictly increasing daily series for one location.
tmin	Daily minimum temperature (degrees Celsius).
tmax	Daily maximum temperature; required for method = "imd".
method	"imd", "percentile" or "absolute".
baseline	Climatology years, c(first, last), contained in date.
percentile	Threshold quantile for "percentile".
threshold	Tmin threshold for "absolute" (degrees Celsius).
min_duration	Minimum run of warm nights counted as a spell.
window	Half-width in days of the calendar-day window.

Value

A data.frame with date, day (a warm night within a spell), severe, spell_start and spell_length.

References

India Meteorological Department. Heat wave and warm night definitions. <https://mausam.imd.gov.in/pdfs/heatcolduser/Definition.pdf>

Zhang, X., Alexander, L., Hegerl, G.C., Jones, P., Klein Tank, A., Peterson, T.C., Trewin, B. & Zwiers, F.W. (2011). Indices for monitoring changes in extremes based on daily temperature and precipitation data. WIREs Climate Change, 2(6), 851-870. doi:10.1002/wcc.147

Examples

```
d <- seq(as.Date("2001-01-01"), as.Date("2001-12-31"), by = "day")
tmin <- 22 + 6 * sin(2 * pi * (as.POSIXlt(d)$yday - 80) / 365)
wn <- warm_nights(d, tmin, method = "absolute", threshold = 27)
sum(wn$day)
```

wbgt	<i>Wet Bulb Globe Temperature</i>
------	-----------------------------------

Description

Computes WBGT by one of two methods:

- "simple": the Australian Bureau of Meteorology closed form $0.567 * T + 0.393 * e + 3.94$ from temperature and dewpoint only (`wbgt_simple()`). Assumes moderate sun and light wind; suited to daily data.
- "liljegren": the physical outdoor model of `wbgt_liljegren()`, which needs wind, solar radiation, time and location; suited to hourly data.

Usage

```
wbgt(temp_c, dewpoint_c, method = c("simple", "liljegren"), ...)
```

Arguments

temp_c	Air temperature (degrees Celsius).
dewpoint_c	Dewpoint temperature (degrees Celsius).
method	"simple" or "liljegren".
...	For method = "liljegren", arguments passed to <code>wbgt_liljegren()</code> : wind_speed, solar, datetime, lat, lon and optionally pressure, wind_height, urban, delta_t.

Value

A numeric vector of WBGT (degrees Celsius).

Examples

```
wbgt(38, 24)
wbgt(38, 24,
  method = "liljegren", wind_speed = 3, solar = 800,
  datetime = as.POSIXct("2024-05-20 08:30:00", tz = "UTC"),
  lat = 26.9, lon = 75.8
)
```

wbgt_from_dewpoint	<i>Calculate WBGT from dewpoint</i>
--------------------	-------------------------------------

Description

This function is an alias for `wbgt_simple()`. It calculates the BoM closed-form WBGT from temperature and dewpoint.

Usage

```
wbgt_from_dewpoint(temp_c, dewpoint_c)
```

Arguments

temp_c	Air temperature in Celsius
dewpoint_c	Dewpoint temperature in Celsius

Value

WBGT in Celsius

Examples

```
wbgt_from_dewpoint(35, 25)
```

wbgt_liljegren	<i>Outdoor Wet Bulb Globe Temperature (Liljegren method)</i>
----------------	--

Description

Physically based outdoor WBGT from standard meteorological variables, following Liljegren et al. (2008). Globe and natural wet bulb temperatures are solved iteratively from energy balances that include direct and diffuse solar radiation, so this is the sun-exposed WBGT of ISO 7243. Inputs should be instantaneous or hourly values; for ERA5, solar is the hourly accumulated surface solar radiation downwards divided by 3600 and datetime the middle of the accumulation hour.

Usage

```
wbgt_liljegren(
  temp_c,
  rh,
  wind_speed,
  solar,
  datetime,
  lat,
  lon,
  pressure = 1013.25,
  wind_height = 10,
  urban = FALSE,
  delta_t = 0,
  components = FALSE
)
```

Arguments

temp_c	Air temperature (degrees Celsius).
rh	Relative humidity (percent).
wind_speed	Wind speed (m/s) measured at wind_height.
solar	Global horizontal solar irradiance (W/m2).
datetime	POSIXct time of each observation; converted to UTC.
lat, lon	Latitude and longitude (decimal degrees).
pressure	Surface pressure (hPa).
wind_height	Height of the wind measurement (m). Values other than 2 are scaled to 2 m with a stability-dependent power law.
urban	Use the urban (TRUE) or rural (FALSE) power-law exponents.
delta_t	Vertical temperature difference, upper minus lower level (degrees Celsius); sets the night-time stability class only.
components	If TRUE, return globe, natural wet bulb and psychrometric wet bulb temperatures as well.

Details

This product includes software produced by UChicago Argonne, LLC under Contract No. DE-AC02-06CH11357 with the Department of Energy.

Value

A numeric vector of WBGT (degrees Celsius), or with components = TRUE a data.frame with columns wbgt, globe, natural_wet_bulb, wet_bulb and wind_2m. NA where the iteration does not converge.

References

Liljegren, J.C., Carhart, R.A., Lawday, P., Tschopp, S. & Sharp, R. (2008). Modeling the Wet Bulb Globe Temperature Using Standard Meteorological Measurements. *Journal of Occupational and Environmental Hygiene*, 5(10), 645-655. doi:[10.1080/15459620802310770](https://doi.org/10.1080/15459620802310770)

See Also

[wbgt\(\)](#) to choose between this and the closed-form estimate.

Examples

```
t <- as.POSIXct("2024-05-20 08:30:00", tz = "UTC") # 14:00 IST
wbgt_liljegren(42,
  rh = 25, wind_speed = 3, solar = 850,
  datetime = t, lat = 26.9, lon = 75.8
)
wbgt_liljegren(42, 25, 3, 850, t, 26.9, 75.8, components = TRUE)
```

wbgt_risk_category	<i>Heat stress risk categories</i>
--------------------	------------------------------------

Description

Classify heat stress risk using the US National Weather Service WBGT flag thresholds of 78, 82, 85 and 88 F (25.6, 27.8, 29.4 and 31.1 C).

Usage

```
wbgt_risk_category(wbgt)
```

Arguments

wbgt	WBGT value in Celsius
------	-----------------------

Value

Factor with the risk category

Examples

```
wbgt_risk_category(c(25, 28, 31, 34))
```

wbgt_shade*Calculate shade/indoor Wet Bulb Globe Temperature (ISO 7243)*

Description

This function uses the indoor formula $0.7T_w + 0.3T_a$. To get a closed-form estimate from only temperature and humidity, use `wbgt_simple()`.

Usage

```
wbgt_shade(temp_c, wet_bulb_c)
```

Arguments

<code>temp_c</code>	Air temperature in Celsius
<code>wet_bulb_c</code>	Wet bulb temperature in Celsius

Value

WBGT in Celsius

Examples

```
wbgt_shade(35, 27)
```

wbgt_simple*Calculate WBGT (Wet Bulb Globe Temperature)*

Description

Australian Bureau of Meteorology approximation $WBGT = 0.567 * T + 0.393 * e + 3.94$, with e the vapour pressure in hPa. It assumes moderately high solar radiation and light wind, so it overestimates WBGT in cloudy or windy conditions and at night. Use daily maximum temperature for afternoon conditions. For the physical outdoor model see [wbgt_liljegren\(\)](#).

Usage

```
wbgt_simple(temp_c, dewpoint_c)
```

Arguments

<code>temp_c</code>	Air temperature in Celsius
<code>dewpoint_c</code>	Dewpoint temperature in Celsius

Value

WBGT in Celsius

References

Australian Bureau of Meteorology. Thermal Comfort observations: About the approximation to the WBGT. https://www.bom.gov.au/info/thermal_stress/ (formula from American College of Sports Medicine (1984), Prevention of thermal injuries during distance running: position stand, Medical Journal of Australia, 141, 876).

Carter, A.W., Zaitchik, B.F., Gohlke, J.M., Wang, S., & Richardson, M.B. (2020). Methods for Estimating Wet Bulb Globe Temperature From Remote and Low-Cost Data: A Comparative Study in Central Alabama. GeoHealth, 4(5), e2019GH000231. doi:10.1029/2019GH000231

Examples

```
wbgt_simple(35, 25)
```

```
wet_bulb_from_dewpoint
```

Calculate wet bulb temperature from dewpoint

Description

Calculate wet bulb temperature from dewpoint

Usage

```
wet_bulb_from_dewpoint(temp_c, dewpoint_c)
```

Arguments

temp_c	Air temperature in Celsius
dewpoint_c	Dewpoint temperature in Celsius

Value

Wet bulb temperature in Celsius

Examples

```
wet_bulb_from_dewpoint(30, 22)
```

wet_bulb_temperature *Calculate wet bulb temperature*

Description

This function uses the Stull (2011) empirical formula to calculate the wet bulb temperature. The formula is applicable to temperatures between -20C and 50C and to RH between 5% and 99%.

Usage

```
wet_bulb_temperature(temp_c, rh)
```

Arguments

temp_c	Air temperature in Celsius
rh	Relative humidity as a percentage (0-100)

Value

Wet bulb temperature in Celsius

References

Stull, R. (2011). Wet-Bulb Temperature from Relative Humidity and Air Temperature. Journal of Applied Meteorology and Climatology, 50(11), 2267-2269. doi:[10.1175/JAMCD110143.1](https://doi.org/10.1175/JAMCD110143.1)

Examples

```
wet_bulb_temperature(30, 60)
```

Index

aggregate_by_frequency, 5
aggregate_imd_by_frequency, 5
aggregate_to_polygons, 6

cache_stats, 7
calc_heat_indices, 8
categorize_variables, 8
check_grid_coverage, 9
check_system_readiness
 (CheckSystemReadiness), 9
CheckSystemReadiness, 9
cleanup_temp_files, 10
clear_cache (ClearCache), 11
clear_timings, 11
ClearCache, 11
combine_netcdf_files, 12
compare_grids, 12
convert_to_geojson, 13
create_geojson_from_bbox, 14
create_temp_geojson_file, 14
create_temporal_chunks, 15
cru_ts_bbox (CRUTSBbox), 15
cru_ts_geojson (CRUTSGeojson), 17
CRUTSBbox, 15
CRUTSGeojson, 17

describe_variables (DescribeVariables),
 18
DescribeVariables, 18
download_era5_pressure, 19
download_era5_single, 20
download_imd_rainfall, 21
download_imd_temperature, 22

ecmwfr::wf_set_key(), 36
ensure_output_dir, 23
era5ify_bbox (ERA5ifyBbox), 23
era5ify_geojson (varunayan-aliases), 94
era5ify_geojson(), 40, 41
era5ify_point (ERA5ifyPoint), 25

era5ify_point(), 68
ERA5ifyBbox, 23
ERA5ifyGeojson (varunayan-aliases), 94
ERA5ifyPoint, 25
excess_heat_factor, 27
excess_heat_factor(), 58
exclude_cols, 28
exposure_indicators, 28
exposure_indicators(), 63, 64
extract_point_data, 30

filter_dataframe_by_geojson, 31
filter_imd_by_bbox, 31
filter_imd_by_geojson, 32
filter_netcdf_by_bbox, 33
filter_netcdf_by_geojson, 33
format_date_range, 34
format_pressure_levels, 34

get_available_datasets, 35
get_available_pressure_levels
 (GetAvailablePressureLevels),
 39
get_bbox_from_sf, 35
get_cache_dir (GetCacheDir), 40
get_cds_credentials, 36
get_era5_country_temperature
 (GetERA5CountryTemperature), 40
get_era5_daily_climate
 (GetERA5DailyClimate), 42
get_era5_daily_heat_data
 (GetERA5DailyHeatIndexData), 43
get_era5_daily_heat_index_data
 (GetERA5DailyHeatIndexData), 43
get_era5_daily_humidity
 (GetERA5DailyHumidity), 46
get_era5_daily_temperature
 (GetERA5DailyTemperature), 47
get_era5_daily_wet_bulb
 (GetERA5DailyWetBulb), 48

- `get_era5_daily_wind` (`GetERA5DailyWind`), [50](#)
- `get_era5_monthly_heat_data` (`GetERA5MonthlyHeatIndexData`), [51](#)
- `get_era5_monthly_heat_index_data` (`GetERA5MonthlyHeatIndexData`), [51](#)
- `get_imd_grid_specs`, [36](#)
- `get_package_info` (`GetPackageInfo`), [53](#)
- `get_pressure_level_variable_names`, [37](#)
- `get_pressure_level_variables`, [37](#)
- `get_single_level_variable_names`, [37](#)
- `get_single_level_variables`, [38](#)
- `get_time_values`, [38](#)
- `get_timings`, [39](#)
- `get_timings()`, [82](#), [89](#)
- `GetAvailablePressureLevels`, [39](#)
- `GetCacheDir`, [40](#)
- `GetERA5CountryTemperature`, [40](#)
- `GetERA5DailyClimate`, [42](#)
- `GetERA5DailyHeatData` (`GetERA5DailyHeatIndexData`), [43](#)
- `GetERA5DailyHeatIndexData`, [43](#)
- `GetERA5DailyHumidity`, [46](#)
- `GetERA5DailyTemperature`, [47](#)
- `GetERA5DailyWetBulb`, [48](#)
- `GetERA5DailyWind`, [50](#)
- `GetERA5MonthlyHeatData` (`GetERA5MonthlyHeatIndexData`), [51](#)
- `GetERA5MonthlyHeatIndexData`, [51](#)
- `GetPackageInfo`, [53](#)
-
- `hadex3_bbox` (`HadEX3Bbox`), [54](#)
- `hadex3_geojson` (`HadEX3Geojson`), [55](#)
- `HadEX3Bbox`, [54](#)
- `HadEX3Geojson`, [55](#)
- `heat_index`, [56](#)
- `heat_index_from_dewpoint`, [57](#)
- `heat_index_risk_category`, [58](#)
- `heatwave_days`, [58](#)
- `heatwave_days()`, [29](#), [62](#), [63](#)
- `humidex`, [60](#)
-
- `imd_exposure_bbox` (`IMDEXposureBbox`), [63](#)
- `imd_exposure_bbox()`, [30](#)
- `imd_rainfall_bbox` (`IMDRainfallBbox`), [64](#)
-
- `imd_rainfall_geojson` (`IMDRainfallGeojson`), [66](#)
- `imd_rainfall_point` (`IMDRainfallPoint`), [67](#)
- `imd_stations` (`IMDStations`), [68](#)
- `imd_temperature_bbox` (`IMDTemperatureBbox`), [69](#)
- `imd_temperature_by_region`, [61](#)
- `imd_temperature_geojson` (`IMDTemperatureGeojson`), [70](#)
- `imd_temperature_geojson()`, [61](#)
- `imd_terrain`, [62](#)
- `imd_terrain()`, [29](#), [59](#), [72](#)
- `IMDEXposureBbox`, [63](#)
- `IMDRainfallBbox`, [64](#)
- `IMDRainfallGeojson`, [66](#)
- `IMDRainfallPoint`, [67](#)
- `IMDStations`, [68](#)
- `IMDTemperatureBbox`, [69](#)
- `IMDTemperatureGeojson`, [70](#)
- `india_elevation`, [71](#)
- `india_elevation()`, [62](#)
- `is_max_var`, [72](#)
- `is_min_var`, [73](#)
- `is_rate_var`, [73](#)
- `is_sum_var`, [74](#)
- `is_valid_geojson`, [74](#)
-
- `list_available_variables` (`ListAvailableVariables`), [76](#)
- `list_cache`, [75](#)
- `list_cru_ts_variables` (`ListCRUTSVariables`), [76](#)
- `list_cru_ts_variables()`, [16](#), [17](#)
- `list_hadex3_indices` (`ListHadEX3Indices`), [77](#)
- `list_hadex3_indices()`, [54](#), [56](#)
- `list_imd_datasets`, [75](#)
- `ListAvailableVariables`, [76](#)
- `ListCRUTSVariables`, [76](#)
- `ListHadEX3Indices`, [77](#)
-
- `max_vars`, [77](#)
- `mean_radiant_temperature`, [78](#)
- `min_vars`, [78](#)
-
- `ndbi`, [79](#)
- `ndbi()`, [30](#), [80](#)
- `ndvi`, [80](#)

ndvi(), [30](#), [79](#)

parallel::mclapply(), [68](#)

parse_date_flexible, [80](#)

populate_sample_cache
 (PopulateSampleCache), [81](#)

PopulateSampleCache, [81](#)

print_timing_summary, [81](#)

process_imd_files, [82](#)

process_netcdf_to_dataframe, [83](#)

rate_vars, [83](#)

read_cdsapirc, [84](#)

read_imd_rainfall, [84](#)

read_imd_temperature, [85](#)

relative_humidity_from_dewpoint, [86](#)

search_variable (SearchVariable), [86](#)

SearchVariable, [86](#)

setup_cds_credentials
 (SetupCdsCredentials), [87](#)

SetupCdsCredentials, [87](#)

show_cache_info (ShowCacheInfo), [88](#)

ShowCacheInfo, [88](#)

stats::quantile(), [59](#)

sum_vars, [88](#)

time_operation, [89](#)

utci, [89](#)

utci_category, [90](#)

utci_sparse, [91](#)

utils::sessionInfo(), [54](#)

validate_bbox, [92](#)

validate_coordinates, [92](#)

validate_era5_variables, [93](#)

vapor_pressure_from_dewpoint, [93](#)

varunayan-aliases, [94](#)

visualize_grid_overlap, [95](#)

warm_nights, [97](#)

warm_nights(), [29](#)

wbgt, [98](#)

wbgt(), [101](#)

wbgt_from_dewpoint, [99](#)

wbgt_liljegren, [99](#)

wbgt_liljegren(), [29](#), [98](#), [102](#)

wbgt_risk_category, [101](#)

wbgt_shade, [102](#)

wbgt_simple, [102](#)

wbgt_simple(), [29](#), [98](#)

wet_bulb_from_dewpoint, [103](#)

wet_bulb_temperature, [104](#)