

Package: varunayan (via r-universe)

October 12, 2026

Type Package

Title Download and Process ERA5, IMD, HadEX3 and CRU TS Climate 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 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Roxigen 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 2026-10-12 16:29:41 UTC

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

RemoteRef HEAD

RemoteSha 4a294632424defd72ad8e0834c49ab1833e5f10b

Contents

aggregate_by_frequency	5
aggregate_imd_by_frequency	5
aggregate_to_polygons	6
cache_stats	7
calc_heat_indices	7
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
cru_ts	15
CRUTSBbox	16
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
hadex3	54
HadEX3Bbox	54
HadEX3Geojson	56
heat_index	57
heat_index_from_dewpoint	58
heat_index_risk_category	58
heat_stress	59
heatwave_days	59
humidex	61
imd_temperature_by_region	61
imd_terrain	63
IMDExposureBbox	64
IMDRainfallBbox	65
IMDRainfallGeojson	66

IMDRainfallPoint	67
IMDStations	68
IMDTemperatureBbox	69
IMDTemperatureGeojson	71
india_elevation	72
is_max_var	73
is_min_var	73
is_rate_var	74
is_sum_var	74
is_valid_geojson	75
list_cache	75
list_imd_datasets	76
ListAvailableVariables	76
ListCRUTSVariables	77
ListHadEX3Indices	77
max_vars	78
mean_radiant_temperature	78
min_vars	79
ndbi	79
ndvi	80
parse_date_flexible	81
PopulateSampleCache	81
print_timing_summary	82
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
variable_lists	94
varunayan-aliases	94
varunayan-utils	95
visualize_grid_overlap	96
warm_nights	97
wbgt	98

<i>aggregate_by_frequency</i>	5
wbgt_from_dewpoint	99
wbgt_liljegren	99
wbgt_risk_category	101
wbgt_shade	102
wbgt_simple	102
wet_bulb_from_dewpoint	103
wet_bulb_temperature	104
Index	105

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

Description

Aggregate data by temporal frequency

Usage

`aggregate_by_frequency(data, frequency, verbose = FALSE)`

Arguments

- | | |
|------------------------|---|
| <code>data</code> | data.frame with datetime column |
| <code>frequency</code> | Character string: "daily", "monthly", "yearly" |
| <code>verbose</code> | Logical indicating whether to print progress messages |

Value

data.frame with temporally aggregated data

<i>aggregate_imd_by_frequency</i>
<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

Aggregate gridded data to polygon regions

Usage

```
aggregate_to_polygons(
  data,
  polygons,
  value_col,
  method = c("nearest_centroid", "point_in_polygon"),
  polygon_id_cols = NULL,
  agg_fun = mean
)
```

Arguments

data	Data frame with longitude, latitude, and value columns
polygons	sf object with polygon geometries
value_col	Name of the column containing values to aggregate
method	Aggregation method: "point_in_polygon" or "nearest_centroid"
polygon_id_cols	Character vector of column names to use as polygon identifiers
agg_fun	Aggregation function (default: mean)

Value

A tibble with one row per polygon: the polygon_id_cols columns and value, the mean of value_col over the grid points assigned to that polygon.

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

Calculate 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>Categorize variables by aggregation method</i>
----------------------	---

Description

Categorize variables by aggregation method

Usage

```
categorize_variables(var_names)
```

Arguments

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

Value

Named list with variables categorized by aggregation method

Examples

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

check_grid_coverage	<i>Check grid coverage statistics for polygons</i>
---------------------	--

Description

Check 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 to identify 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>Check system readiness for varunayan operations</i>
----------------------	--

Description

Check system readiness for varunayan operations

Usage

```
CheckSystemReadiness(...)

check_system_readiness(verbose = TRUE)
```

Arguments

... Arguments passed to the main function (used by aliases).
verbose Whether to 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>Clean up temporary files matching pattern</i>
--------------------	--

Description

Clean up temporary files matching pattern

Usage

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

Arguments

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

Value

Number of files removed

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

Description

Clear all timing 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 passed to the main function (used by aliases). |
| source | Clear only "era5", "imd", "cru_ts", "hadex3", or NULL for all |
| older_than | Remove files older than this many 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 multiple NetCDF files into single data.frame</i>
----------------------	---

Description

Combine multiple NetCDF files into single data.frame

Usage

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

Arguments

netcdf_files	Character vector of NetCDF file paths
verbose	Logical indicating whether to print progress messages

Value

data.frame with combined data from all files

compare_grids	<i>Compare multiple grids</i>
---------------	-------------------------------

Description

Compare multiple grids

Usage

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

Arguments

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

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

Value

GeoJSON formatted data

`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 representing 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 unique filename

Value

Path to temporary GeoJSON file

```
create_temporal_chunks
```

Apply temporal chunking for large date ranges

Description

Chunk size defaults to `compute_optimal_chunk_size(frequency)`, which scales with the CDS field limit (hourly: 24 fields/day, daily: 1/day, monthly: 12/year) so each chunk fills one CDS job.

Usage

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

Arguments

<code>start_date</code>	Start date
<code>end_date</code>	End date
<code>frequency</code>	Temporal frequency ("hourly", "daily", "monthly")
<code>n_variables</code>	Number of variables in the request (fields = vars x steps)
<code>max_days_per_chunk</code>	Maximum days per chunk. NA (default) -> derive from frequency via <code>compute_optimal_chunk_size</code> .

Value

List of date pairs for chunked processing

```
cru_ts
```

CRU TS Gridded Climate Dataset

Description

Functions to download and process the University of East Anglia Climatic Research Unit Time-Series (CRU TS) gridded monthly climate dataset. Covers global land areas (excl. Antarctica) at 0.5 degree resolution, 1901-2022.

Usage

```
.CRU_TS_VERSION
```

References

Harris et al. (2020) Version 4 of the CRU TS monthly high-resolution gridded multivariate climate dataset. Sci. Data 7:109. doi:[10.1038/s4159702004533](https://doi.org/10.1038/s4159702004533)

CRUTSBbox

Download CRU TS climate data for a bounding box

Description

Downloads and filters CRU TS v4.07 monthly gridded climate data to a geographic bounding box. Data covers global land areas at 0.5 degree resolution, 1901-2022.

Usage

```
CRUTSBbox(...)

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

Arguments

...	Arguments passed to the main function (used by aliases).
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	Northern latitude boundary.
south	Southern latitude boundary.
east	Eastern longitude boundary.
west	Western longitude boundary.
output_dir	Directory for temporary files (default: tempdir()).
use_cache	Whether to use cached data (default: TRUE).

Value

data.frame with 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 and filters CRU TS v4.07 monthly gridded climate data to a geographic region defined by a GeoJSON polygon.

Usage

```
CRUTSGeojson(...)

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

Arguments

...	Arguments passed to the main function (used by aliases).
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 defining the region.
output_dir	Directory for temporary files (default: tempdir()).
use_cache	Whether to use cached data (default: TRUE).

Value

data.frame with 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

Describe ERA5 variables in detail

Description

Describe ERA5 variables in detail

Usage

```
DescribeVariables(...)
```

```
describe_variables(variables, dataset_type = "all")
```

Arguments

...	Arguments passed to the main function (used by aliases).
variables	Character vector of variable names to describe
dataset_type	Character string indicating 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 using ecmwfr

Description

Download ERA5 pressure-level data using 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

variables	Character vector of variable names
pressure_levels	Character vector of pressure levels (e.g., c("1000", "850", "500"))
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
use_cache	Use cached data if available (default: TRUE)
verbose	Logical indicating whether to print progress messages

Value

Character string path to downloaded file

download_era5_single *Download ERA5 single-level data using ecmwfr*

Description

Download ERA5 single-level data using 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	Use cached data if available (default: TRUE)
verbose	Logical indicating whether to 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 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 to save downloaded files.
<code>use_cache</code>	Whether to use cached data if available (default: TRUE).

Value

Character vector of downloaded file paths

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 (max or min) from IMD. Temperature data is at 1.0 degree resolution only.

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 to save downloaded files.
<code>use_cache</code>	Whether to use cached data if available (default: TRUE).

Value

Character vector of downloaded file paths

Examples

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

ensure_output_dir	Create output directory if it doesn't exist
-------------------	---

Description

Create output directory if it doesn't exist

Usage

```
ensure_output_dir(output_dir)
```

Arguments

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

Value

Character string path to created/existing directory

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>	Whether to save raw downloaded files.
<code>output_dir</code>	Directory to save files (default: <code>tempdir()</code>).
<code>use_cache</code>	Whether to use cached data if available (default: <code>TRUE</code>).
<code>convert_units</code>	Whether to convert units to user-friendly formats (default: <code>TRUE</code>). When <code>TRUE</code> , temperature variables are returned in Celsius (converted from Kelvin) and precipitation variables are returned in mm (converted from meters).
<code>verbose</code>	Whether to 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").

<code>pressure_levels</code>	Pressure levels for pressure dataset.
<code>frequency</code>	Temporal frequency ("hourly", "daily", "monthly").
<code>save_raw</code>	Whether to save raw downloaded files.
<code>output_dir</code>	Directory to save files (default: <code>tempdir()</code>).
<code>use_cache</code>	Whether to use cached data if available (default: <code>TRUE</code>).
<code>convert_units</code>	Whether to convert units to user-friendly formats (default: <code>TRUE</code>). When <code>TRUE</code> , temperature variables are returned in Celsius (converted from Kelvin) and precipitation variables are returned in mm (converted from meters).
<code>verbose</code>	Whether to print detailed progress messages.
<code>use_timeseries_api</code>	Whether to use the CDS time-series endpoint, which serves a point directly and is much faster than subsetting a gridded download (default: <code>NA</code> , meaning use it for hourly single-level requests, the only ones it supports). Set <code>FALSE</code> to force the gridded download. The gridded download is used automatically if the endpoint is unavailable.

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

These are coordinate and metadata columns that should not be aggregated.

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 specific point location (nearest neighbor)</i>
--------------------	--

Description

Extract data for specific point location (nearest neighbor)

Usage

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

Arguments

netcdf_file	Character string path to NetCDF file
target_lat	Target latitude
target_lon	Target longitude

Value

data.frame with data for nearest point

filter_dataframe_by_geojson
Filter data.frame by GeoJSON polygon

Description

Tests only unique (lat, lon) coordinates against the polygon, then joins back to full dataset via 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	Northern latitude boundary
south	Southern latitude boundary
east	Eastern longitude boundary
west	Western longitude 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 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	Character string path to NetCDF file
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	Character string path to NetCDF file
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 character format required by CDS API</i>
------------------------	--

Description

Convert pressure levels to character format required by 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 multiple 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` Suppress informational messages

Value

TRUE if credentials are available, FALSE otherwise.

Examples

```
get_cds_credentials()
```

```
get_imd_grid_specs
```

Get IMD grid specifications

Description

Returns grid specifications for different IMD datasets.

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 pressure-level variables dataset

Description

Get pressure-level variables dataset

Usage

```
get_pressure_level_variables()
```

Value

List containing 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	<i>Get single-level variables dataset</i>
----------------------------	---

Description

Get single-level variables dataset

Usage

```
get_single_level_variables()
```

Value

List containing single-level variable categories

get_time_values	<i>Get time values based on frequency</i>
-----------------	---

Description

Get time values based on frequency

Usage

```
get_time_values(frequency)
```

Arguments

frequency	Character string indicating temporal frequency
-----------	--

Value

Character vector of time strings

get_timings	<i>Get all recorded timings</i>
-------------	---------------------------------

Description

Get all recorded timings

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 standard pressure levels available in ERA5</i>
----------------------------	---

Description

Get standard pressure levels available in ERA5

Usage

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

```
get_available_pressure_levels()
```

Arguments

... Arguments passed to the main function (used by aliases).

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 passed to the main function (used by aliases).
<code>create</code>	Create directory if it doesn't exist (default: TRUE)

Value

Character string with cache directory path

Examples

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

GetERA5CountryTemperature	<i>Get ERA5 country-level temperature data</i>
---------------------------	--

Description

Download and aggregate ERA5 temperature data for a country. Resolves the country boundary, downloads monthly data via [era5ify_geojson\(\)](#), and returns area-weighted (cosine-latitude) country-level averages by year and month.

Usage

```
GetERA5CountryTemperature(...)

get_era5_country_temperature(
  country,
  start_date,
  end_date,
```

```

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

```

Arguments

...	Additional arguments passed to <code>era5ify_geojson()</code> .
country	One of: a country name string (resolved via <code>maps::map()</code>), a path to a GeoJSON file, or an <code>sf</code> object. Common aliases like "USA" and "UK" are handled automatically.
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. Auto-generated from the country name if not provided.
resolution	Spatial resolution in degrees (default: 0.25).
verbose	Whether to print detailed progress messages.

Value

A data.frame with columns year, month, and one column per requested variable (e.g., temperature_mean, temperature_max, temperature_min). 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 USA
get_era5_country_temperature("USA", "2023-01-01", "2023-12-31",
  variables = c("mean", "max")
)

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

```

GetERA5DailyClimate *Download daily ERA5 climate data*

Description

Download 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 GeoJSON file defining the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	Whether to use cached data (default: TRUE)
verbose	Whether to print 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

Download ERA5 temperature, dewpoint, and wind data for a region and calculate 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 GeoJSON file defining the region (for geojson method)
north	Northern latitude boundary (for bbox method)
south	Southern latitude boundary (for bbox method)
east	Eastern longitude boundary (for bbox method)
west	Western longitude boundary (for bbox method)
solar_load	Whether to include solar radiation in calculations (default: FALSE). When TRUE, downloads solar radiation data and uses it for WBGT and UTCI.
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	Whether to use cached data if available (default: TRUE)
verbose	Whether to print 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. When solar_load=TRUE, also includes 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

Downloads temperature, dewpoint, and calculates 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 GeoJSON file defining the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
resolution	Spatial resolution in degrees (default: 0.25)

use_cache	Whether to use cached data (default: TRUE)
verbose	Whether to print 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

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 GeoJSON file defining the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	Whether to use cached data (default: TRUE)
verbose	Whether to print 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	<i>Download daily ERA5 wet bulb temperature data</i>
---------------------	--

Description

Downloads temperature, dewpoint, and calculates RH and 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 GeoJSON file defining 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>	Whether to use cached data (default: TRUE)
<code>verbose</code>	Whether to print 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

Downloads 10m wind components and calculates 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

<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 GeoJSON file defining 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>	Whether to use cached data (default: TRUE)
<code>verbose</code>	Whether to print 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

Download ERA5 heat stress indices aggregated to a given 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 (e.g., 2023)
month	Month (1-12)
json_file	Path to GeoJSON file defining the region
north, south, east, west	Bounding box coordinates (alternative to json_file)
solar_load	Whether to include solar radiation (default: FALSE)
resolution	Spatial resolution in degrees (default: 0.25)
use_cache	Whether to use cached data (default: TRUE)
verbose	Whether to print 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 passed to the main function (used 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()
```

hadex3	<i>HadEX3 Climate Extremes Dataset</i>
--------	--

Description

Functions to download and process the Met Office HadEX3 global climate extremes dataset based on ETCCDI indices.

Usage

```
.HADEX3_ANNUAL_INDICES
```

References

Dunn et al. (2020) Development of an updated global land in-situ-based dataset of temperature and precipitation extremes: HadEX3. J. Geophys. Res. Atmos. doi:[10.1029/2019JD032263](https://doi.org/10.1029/2019JD032263)

HadEX3Bbox	<i>Download HadEX3 climate extremes data for a bounding box</i>
------------	---

Description

Downloads and filters HadEX3 ETCCDI climate extremes data to a geographic bounding box. Data covers 1901-2018 at 1.25 x 1.875 degree resolution.

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

<code>...</code>	Arguments passed to the main function (used by aliases).
<code>index</code>	ETCCDI index name (e.g. "TXx", "Rx1day"). See list_hadex3_indices() .
<code>start_year</code>	Start year (1901-2018).
<code>end_year</code>	End year (1901-2018).
<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>frequency</code>	Temporal frequency: "annual" or "monthly" (default: "annual").
<code>baseline</code>	Reference period for percentile-based indices: "61-90" (default) or "81-10". Only applies to annual frequency.
<code>output_dir</code>	Directory for temporary files (default: <code>tempdir()</code>).
<code>use_cache</code>	Whether to use cached data (default: <code>TRUE</code>).

Value

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

Examples

```
# Annual maximum daily maximum temperature for Europe 1981-2010
data <- hadex3_bbox(
  index = "TXx",
  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 climate extremes data for a GeoJSON region

Description

Downloads and filters HadEX3 ETCCDI climate extremes data to a geographic region defined by a GeoJSON polygon.

Usage

```
HadEX3Geojson(...)

hadex3_geojson(
  index,
  start_year,
  end_year,
  geojson_file,
  frequency = "annual",
  baseline = "61-90",
  output_dir = tempdir(),
  use_cache = TRUE
)
```

Arguments

<code>...</code>	Arguments passed to the main function (used by aliases).
<code>index</code>	ETCCDI index name (e.g. "TXx", "Rx1day"). See list_hadex3_indices() .
<code>start_year</code>	Start year (1901-2018).
<code>end_year</code>	End year (1901-2018).
<code>geojson_file</code>	Path to a GeoJSON file defining the region.
<code>frequency</code>	Temporal frequency: "annual" or "monthly" (default: "annual").
<code>baseline</code>	Reference period for percentile-based indices: "61-90" (default) or "81-10". Only applies to annual frequency.
<code>output_dir</code>	Directory for temporary files (default: <code>tempdir()</code>).
<code>use_cache</code>	Whether to use cached data (default: <code>TRUE</code>).

Value

data.frame with columns: year (+ month for monthly), 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 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

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

Value

Heat index in Celsius

Examples

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

`heat_index_risk_category`*Heat index risk categories*

Description

Classify heat stress risk based on heat index using NWS categories.

Usage

```
heat_index_risk_category(hi)
```

Arguments

<code>hi</code>	Heat index value in Celsius
-----------------	-----------------------------

Value

A factor of risk categories, the same length as the input.

Examples

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

heat_stress	<i>heat stress index calculations</i>
-------------	---------------------------------------

Description

heat stress index calculations

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 <code>imd_terrain()</code> 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 per GeoJSON feature (region x time)</i>
---------------------------	--

Description

Unlike `imd_temperature_geojson()`, which returns grid points clipped to the GeoJSON boundary, this returns one row per feature (e.g. per state) per time period, with the region-mean temperature.

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

<code>request_id</code>	Unique identifier for the request.
<code>start_year</code>	Start year (≥ 1951).
<code>end_year</code>	End year (≤ 2024).
<code>geojson_file</code>	Path to a GeoJSON whose features are the regions.
<code>feature_col</code>	Name of the GeoJSON property to group by (e.g. "state_name"). If NULL, the first non-geometry column is used.
<code>var_type</code>	"tmax", "tmin", or "tmean" (mean of tmax & tmin). Default "tmean".
<code>frequency</code>	"monthly" (default), "yearly", or "daily".
<code>use_cache</code>	Use cached IMD downloads if available (default TRUE).

Value

A data.frame: region, year, month (if monthly), temperature.

Examples

```
## Not run:
temp <- imd_temperature_by_region(
  request_id = "india_states",
  start_year = 2009, end_year = 2024,
  geojson_file = "india_states.geojson",
  feature_col = "state_name", var_type = "tmean", frequency = "monthly"
)

## End(Not run)
```

imd_terrain	<i>IMD heatwave terrain class of a location</i>
-------------	---

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](https://doi.org/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

<code>...</code>	Further arguments to <code>exposure_indicators()</code> , such as <code>months</code> , <code>period</code> , <code>heatwave_method</code> , <code>terrain</code> or <code>hill_elevation</code> ; <code>terrain</code> is classified per 1 degree cell.
<code>start_year</code> , <code>end_year</code>	Years to report (1951-2025).
<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>baseline</code>	Climatology years, <code>c(first, last)</code> , for normals and percentiles.
<code>output_dir</code>	Directory to save files (default: <code>tempdir()</code>).
<code>use_cache</code>	Whether to use cached data if available (default: <code>TRUE</code>).

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

Description

Downloads IMD gridded rainfall data and filters it to a 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 passed to the main function (used by aliases).
<code>request_id</code>	Unique identifier for the request.
<code>start_year</code>	Start year.
<code>end_year</code>	End year.
<code>north</code>	Northern latitude boundary.
<code>south</code>	Southern latitude boundary.

east	Eastern longitude boundary.
west	Western longitude boundary.
resolution	Spatial resolution: 0.25 or 1.0 degrees (default: 0.25).
frequency	Temporal frequency: "daily", "monthly", "yearly" (default: "daily").
save_raw	Whether to save raw downloaded files (default: FALSE).
output_dir	Directory to save files (default: tempdir()).
use_cache	Whether to use cached data if 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

<code>...</code>	Arguments passed to the main function (used by aliases).
<code>request_id</code>	Unique identifier for the request.
<code>start_year</code>	Start year (1901-2025).
<code>end_year</code>	End year (1901-2025).
<code>geojson_file</code>	Path to GeoJSON file.
<code>resolution</code>	Spatial resolution: 0.25 or 1.0 degrees (default: 0.25).
<code>frequency</code>	Temporal frequency (default: "daily").
<code>save_raw</code>	Whether to save raw files (default: FALSE).
<code>output_dir</code>	Directory to save files (default: <code>tempdir()</code>).
<code>use_cache</code>	Whether to use cached data if 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 nearest to a (lat, lon). Reads only that one cell off disk per year, so it is the cheapest way to get a long series for a 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 passed to the main function (used by aliases).
request_id	Unique identifier for the request.
start_year	Start year (1901-2024).
end_year	End year (1901-2024).
latitude	Point latitude.
longitude	Point longitude.
resolution	Spatial resolution: 0.25 or 1.0 degrees (default: 0.25).
frequency	Temporal frequency: "daily", "monthly", "yearly" (default: "daily").
save_raw	Whether to save raw downloaded files (default: FALSE).
output_dir	Directory to save files (default: tempdir()).
use_cache	Whether to use cached data if available (default: TRUE).

Value

data.frame with the nearest cell’s rainfall series.

Examples

```
## Not run:
powai <- imd_rainfall_point("powai", 1901, 2024, 19.1277, 72.9047)

## End(Not run)
```

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

Description

Retrieves the India Meteorological Department (IMD) city-weather station list and, optionally, each station’s latitude/longitude. IMD’s legacy endpoints expose only station names; its responsive site serves a per-station JSON record ([api/fetchCity_static.php](#)) that includes lat/lon, which this function pairs with the station list. The returned coordinates can be fed directly to [era5ify_point\(\)](#) or used to build maps.

Usage

```
IMDStations(...)

imd_stations(with_coordinates = TRUE, cores = 4)
```

Arguments

...	Ignored (present for alias compatibility).
with_coordinates	Logical; if TRUE (default) fetch latitude and longitude for every station (one request per station). If FALSE, return only station_id and name.
cores	Integer; number of parallel workers used to fetch coordinates (passed to <code>parallel::mclapply()</code>). Defaults to 4; set to 1 on Windows.

Value

A data.frame with columns station_id, name, and — when with_coordinates = TRUE — latitude, longitude (NA where IMD has no coordinate for a station).

Examples

```
## Not run:
stations <- imd_stations()
shimla <- stations[grepl("Shimla", stations$name), ]

## End(Not run)
```

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 passed to the main function (used by aliases).
request_id	Unique identifier for the request.
start_year	Start year (1951-2025).
end_year	End year (1951-2025).
north	Northern latitude boundary.
south	Southern latitude boundary.
east	Eastern longitude boundary.
west	Western longitude boundary.
var_type	Temperature type: "tmax" or "tmin".
frequency	Temporal frequency (default: "daily").
save_raw	Whether to save raw files (default: FALSE).
output_dir	Directory to save files (default: tempdir()).
use_cache	Whether to use cached data if 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 passed to the main function (used by aliases).
request_id	Unique identifier.
start_year	Start year (1951-2025).
end_year	End year (1951-2025).
geojson_file	Path to GeoJSON file.
var_type	Temperature type: "tmax" or "tmin".
frequency	Temporal frequency (default: "daily").
save_raw	Whether to save raw files (default: FALSE).
output_dir	Directory for output (default: tempdir()).
use_cache	Whether to use cached data if 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	<i>Check if a variable should use maximum</i>
------------	---

Description

Check if a variable should use maximum

Usage

```
is_max_var(var_name)
```

Arguments

var_name	Character string variable name
----------	--------------------------------

Value

Logical indicating if variable should use max

Examples

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

is_min_var	<i>Check if a variable should use minimum</i>
------------	---

Description

Check if a variable should use minimum

Usage

```
is_min_var(var_name)
```

Arguments

var_name	Character string variable name
----------	--------------------------------

Value

Logical indicating if variable should use min

Examples

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

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

Description

Check if a variable is a rate variable

Usage

```
is_rate_var(var_name)
```

Arguments

var_name	Character string variable name
----------	--------------------------------

Value

Logical indicating if variable is a rate

Examples

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

is_sum_var	<i>Check if a variable should be summed</i>
------------	---

Description

Check if a variable should be summed

Usage

```
is_sum_var(var_name)
```

Arguments

var_name	Character string variable name
----------	--------------------------------

Value

Logical indicating if variable should be summed

Examples

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

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

Description

Check if data is valid GeoJSON

Usage

```
is_valid_geojson(data)
```

Arguments

data	List or data structure to check
------	---------------------------------

Value

Logical indicating if 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()
```

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

Description

List available IMD datasets

Usage

```
list_imd_datasets()
```

Value

data.frame with available IMD datasets and their characteristics

Examples

```
list_imd_datasets()
```

<code>ListAvailableVariables</code>

<i>List all available variables for a dataset type</i>
--

Description

List all available variables for a dataset type

Usage

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

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

Arguments

<code>...</code>	Arguments passed to the main function (used by aliases).
<code>dataset_type</code>	Character string indicating dataset type

Value

Character vector of variable names

Examples

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

ListCRUTSVariables	<i>Variable data dictionary for CRU TS</i>
--------------------	--

Description

Maps CRU TS variable codes to common names, units, and ERA5 equivalents.

Usage

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

Arguments

... Arguments passed to the main function (used by aliases).

Value

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

Examples

```
list_cru_ts_variables()
```

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

Description

Returns a data.frame describing all ETCCDI climate extremes indices available in the HadEX3 dataset, including their temporal availability and physical interpretation.

Usage

```
ListHadEX3Indices(...)  
  
list_hadex3_indices(frequency = "all")
```

Arguments

... Arguments passed to the main function (used by aliases).
frequency Filter by frequency: "annual", "monthly", or "all" (default).

Value

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

Examples

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

max_vars	<i>Variables that should use maximum aggregation</i>
----------	--

Description

Variables that should use maximum aggregation

Usage

```
max_vars
```

Value

A character vector of variable names.

Examples

```
head(max_vars)
```

mean_radiant_temperature	<i>Estimate mean radiant temperature</i>
--------------------------	--

Description

Estimates mean radiant temperature (Tmrt) from meteorological variables. Uses simplified approach when full radiation data unavailable.

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	<i>Variables that should use minimum aggregation</i>
----------	--

Description

Variables that should use 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

Normalized Difference Vegetation Index

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 date string flexibly</i>
---------------------	-----------------------------------

Description

Parse date string flexibly

Usage

```
parse_date_flexible(date_string)
```

Arguments

date_string	Character string in various date formats
-------------	--

Value

Date object

PopulateSampleCache	<i>Populate cache with sample IMD data</i>
---------------------	--

Description

Copies sample IMD data files to cache for offline testing.

Usage

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

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

Arguments

...	Arguments passed to the main function (used by aliases).
force	Overwrite existing cached files (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 timing summary

Description

Print timing summary

Usage

```
print_timing_summary()
```

Value

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

Examples

```
print_timing_summary()
```

```
process_imd_files
```

Process multiple IMD files

Description

Reads and combines multiple IMD data files.

Usage

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

Arguments

<code>file_paths</code>	Character vector of file paths.
<code>var_type</code>	Variable type: "rain", "tmax", or "tmin".
<code>resolution</code>	Resolution (for rainfall): 0.25 or 1.0.
<code>years</code>	Integer vector. Years corresponding to each file.
<code>bbox</code>	Optional named list/vector (north, south, east, west) passed to the rainfall reader to crop at read time; NULL 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	<i>Process NetCDF file and extract data as data.frame</i>
-----------------------------	---

Description

Uses ncdf4 directly for fast reading, with stars as fallback.

Usage

```
process_netcdf_to_dataframe(netcdf_file)
```

Arguments

netcdf_file Character string path to NetCDF file

Value

data.frame with processed climate data

rate_vars	<i>Rate variables that should be averaged</i>
-----------	---

Description

Rate variables that should be averaged

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 to data.frame.

Usage

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

Arguments

file_path	Path to IMD NetCDF file (.nc).
resolution	Spatial resolution: 0.25 or 1.0.
year	Year of the data.
bbox	Optional named list/vector (north, south, east, west). When set, only that spatial slab is read off disk (fast path); NULL 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 (max or min) and converts to data.frame.

Usage

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

Arguments

file_path	Path to 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

Calculate relative humidity from temperature and dewpoint

Description

Uses the Magnus-Tetens approximation to calculate relative humidity from air temperature and 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 percentage (0-100)

Examples

```
relative_humidity_from_dewpoint(25, 18)
```

SearchVariable

Search for ERA5 variables by keyword

Description

Search for ERA5 variables by keyword

Usage

```
SearchVariable(...)
```

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

Arguments

...	Arguments passed to the main function (used by aliases).
keyword	Character string to search for in variable names and descriptions
dataset_type	Character string indicating dataset type ("single", "pressure", or "all")
exact_match	Logical indicating whether to use exact matching

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

SetupCdsCredentials *Set up CDS credentials for ERA5 data access*

Description

Sets up Copernicus Climate Data Store (CDS) credentials for ERA5 downloads. Credentials are checked in the following order:

1. Directly provided key argument
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, attempts to read from environment variable CDS_API_KEY or ~/.cdsapirc file.

Value

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

Examples

```
## Not run:
setup_cds_credentials(key = "your-cds-api-key-here")
Sys.setenv(CDS_API_KEY = "your-cds-api-key-here")
setup_cds_credentials()

## End(Not run)
```

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

Description

Show cache contents and statistics

Usage

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

```
show_cache_info()
```

Arguments

... Arguments passed to the main function (used 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 that should be summed over time</i>
----------	--

Description

Accumulation variables (precipitation, radiation) that use sum for temporal aggregation.

Usage

```
sum_vars
```

Value

A character vector of variable names.

Examples

```
head(sum_vars)
```

time_operation	<i>Time an operation and log the duration</i>
----------------	---

Description

Time an operation and log the duration

Usage

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

Arguments

expr	Expression to time
label	Label for the operation
verbose	Whether to print timing info (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

Calculates UTCI using the polynomial approximation from Bröde et al. (2012), based on 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, defaults to 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

Classify thermal stress based on UTCI values.

Usage

```
utci_category(utci_val)
```

Arguments

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

Value

A factor of thermal stress categories, the same length as the input.

Examples

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

utci_sparse	<i>Calculate UTCI using sparse regression approximation</i>
-------------	---

Description

Calculates UTCI using the sparse regression approximation with Legendre polynomials from Roman et al. (2025). This is a more accurate approximation than the original Bröde et al. (2012) polynomial, with 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 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>Validate bounding box coordinates</i>
---------------	--

Description

Validate bounding box coordinates

Usage

validate_bbox(north, south, east, west)

Arguments

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

Value

Logical indicating if bounding box is valid

validate_coordinates	<i>Validate geographic coordinates</i>
----------------------	--

Description

Validate geographic coordinates

Usage

validate_coordinates(lat, lon)

Arguments

lat	Latitude value
lon	Longitude value

Value

Logical indicating if coordinates are valid

`validate_era5_variables`*Validate ERA5 variable names*

Description

Validate ERA5 variable names

Usage

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

Arguments

`variables` Character vector of variable names to validate

`dataset_type` Character string: "single" or "pressure"

Value

Logical vector indicating 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

`dewpoint_c` Dewpoint temperature in Celsius

Value

Vapor pressure in hPa (hectopascals)

Examples

```
vapor_pressure_from_dewpoint(20)
```

variable_lists	<i>ERA5 variable aggregation rules</i>
----------------	--

Description

ERA5 variable aggregation rules

varunayan-aliases	<i>PascalCase Aliases</i>
-------------------	---------------------------

Description

PascalCase aliases for all exported functions for 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
)
```

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 GeoJSON file defining 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	Whether to save raw downloaded files
output_dir	Directory to save files (default: tempdir())
use_cache	Whether to use cached data if available (default: TRUE)
convert_units	Whether to convert units to user-friendly formats (default: TRUE). When TRUE, temperature variables are returned in Celsius (converted from Kelvin) and precipitation variables are returned in mm (converted from meters).
verbose	Whether to 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

```
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
```

Description

Utility functions for Varunayan R package

visualize_grid_overlap

Visualize grid-polygon overlap

Description

Visualize grid-polygon overlap

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

grid_data	Data frame with longitude and latitude columns, or path to NetCDF/CSV
polygons	sf object or path to GeoJSON file
show_centroids	Logical, whether to show polygon centroids
grid_color	Color for grid points
polygon_fill	Fill color for polygons
polygon_color	Border color for polygons
centroid_color	Color for centroids
title	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	Calculate WBGT from dewpoint
--------------------	------------------------------

Description

Alias for `wbgt_simple()` (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	Outdoor Wet Bulb Globe Temperature (Liljegren method)
----------------	---

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

A factor of risk categories, the same length as the input.

Examples

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

wbgt_shade	<i>Calculate shade/indoor Wet Bulb Globe Temperature (ISO 7243)</i>
------------	---

Description

Uses the indoor formula $0.7T_w + 0.3T_a$. For a closed-form estimate from temperature and humidity alone, use `wbgt_simple()`.

Usage

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

Arguments

- temp_c Air temperature in Celsius
- wet_bulb_c Wet bulb temperature in Celsius

Value

WBGT in Celsius

Examples

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

wbgt_simple	<i>Calculate WBGT (Wet Bulb Globe Temperature)</i>
-------------	--

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

- temp_c Air temperature in Celsius
- dewpoint_c 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	<i>Calculate wet bulb temperature</i>
----------------------	---------------------------------------

Description

Calculates wet bulb temperature using the Stull (2011) empirical formula. Valid for temperatures between -20C and 50C and RH between 5% and 99%.

Usage

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

Arguments

temp_c	Air temperature in Celsius
rh	Relative humidity as 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

.CRU_TS_VERSION (cru_ts), 15
.HADEX3_ANNUAL_INDICES (hadex3), 54

aggregate_by_frequency, 5
aggregate_imd_by_frequency, 5
aggregate_to_polygons, 6

cache_stats, 7
calc_heat_indices, 7
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, 15
cru_ts_bbox (CRUTSBbox), 16
cru_ts_geojson (CRUTSGeojson), 17
CRUTSBbox, 16
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(), 59
exclude_cols, 28
exposure_indicators, 28
exposure_indicators(), 63–65
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, 54
- hadex3_bbox (HadEX3Bbox), 54
- hadex3_geojson (HadEX3Geojson), 56
- HadEX3Bbox, 54
- HadEX3Geojson, 56
- heat_index, 57
- heat_index_from_dewpoint, 58
- heat_index_risk_category, 58
- heat_stress, 59
- heatwave_days, 59
- heatwave_days(), 29, 63

- humidex, 61

- imd_exposure_bbox (IMDExposureBbox), 64
- imd_exposure_bbox(), 30
- imd_rainfall_bbox (IMDRainfallBbox), 65
- 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), 71
- imd_temperature_geojson(), 61
- imd_terrain, 63
- imd_terrain(), 29, 60, 72
- IMDExposureBbox, 64
- IMDRainfallBbox, 65
- IMDRainfallGeojson, 66
- IMDRainfallPoint, 67
- IMDStations, 68
- IMDTemperatureBbox, 69
- IMDTemperatureGeojson, 71
- india_elevation, 72
- india_elevation(), 63
- is_max_var, 73
- is_min_var, 73
- is_rate_var, 74
- is_sum_var, 74
- is_valid_geojson, 75

- list_available_variables
(ListAvailableVariables), 76
- list_cache, 75
- list_cru_ts_variables
(ListCRUTSVVariables), 77
- list_cru_ts_variables(), 16, 17
- list_hadex3_indices
(ListHadEX3Indices), 77
- list_hadex3_indices(), 55, 56
- list_imd_datasets, 76
- ListAvailableVariables, 76
- ListCRUTSVVariables, 77
- ListHadEX3Indices, 77

- max_vars, 78
- mean_radiant_temperature, 78

min_vars, 79

ndbi, 79

ndbi(), 30, 80

ndvi, 80

ndvi(), 30, 80

parallel::mclapply(), 69

parse_date_flexible, 81

populate_sample_cache
 (PopulateSampleCache), 81

PopulateSampleCache, 81

print_timing_summary, 82

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(), 60

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

variable_lists, 94

varunayan-aliases, 94

varunayan-utils, 95

visualize_grid_overlap, 96

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