Package 'tidyweather'

November 5, 2026

Title Analysis the Weather Data for Agriculture

Version 0.3.4

Description Functions are collected to analyse weather data for agriculture purposes including to read weather records in multiple formats, calculate extreme climate index. Demonstration data are included the SILO daily climate data (licensed under CC BY 4.0, <https://www.longpaddock.qld.gov.au/silo/>).

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports stringr, tibble, dplyr, optree

Suggests testthat (>= 3.0.0), ggplot2

Config/testthat/edition 3

URL https://tidyweather.bangyou.me/, https://github.com/byzheng/tidyweather

BugReports https://github.com/byzheng/tidyweather/issues

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Bangou Zheng [aut, cre]

Maintainer Bangou Zheng <zheng.bangyou@gmail.com>

Repository CRAN

Date/Publication 2026-11-05 06:00:11 UTC

Contents

check_weather 2
day_length 3
interpolate_3hr 4
interpolate_hourly_sin_pp_adjusted 4

interpolation_function 6
last_frost_day 6
number_frost_day 7
read_weather 8
spherical_distance 9
summarise_weather 9
sun_rise_set 10
thermal_time 11
ttest_ts 12
wang_engel_tt 13
weather_options 14
write_weather 14

Index **16**

`check_weather` *Check weather records for data quality issues*

Description

This function validates weather records for:

- Continuous weather data (no gaps in dates)
- No missing values in key columns (mint, maxt, radn, rain)
- No extreme values (e.g., less than -100 or above 100 for temperature, less than 0 for radiation and rain)
- Latitude and longitude columns exist and contain a single non-NA value for all records

Usage

```
check_weather(
  data,
  key_cols = c("mint", "maxt", "radn", "rain"),
  temp_range = c(-100, 100),
  radn_range = c(0, 50),
  rain_range = c(0, 500),
  stop_on_error = FALSE
)
```

Arguments

`data`	A data.frame or tibble containing weather records with at minimum a date column, latitude, longitude, and key weather variables (mint, maxt, radn, rain).
`key_cols`	A character vector of column names to check for missing values and extreme values. Default is c("mint", "maxt", "radn", "rain").
`temp_range`	A numeric vector of length 2 specifying the acceptable range for temperature values (mint, maxt). Default is c(-100, 100).

radn_range	A numeric vector of length 2 specifying the acceptable range for radiation values. Default is c(0, 50).
rain_range	A numeric vector of length 2 specifying the acceptable range for rainfall values. Default is c(0, 500).
stop_on_error	Logical. If TRUE, the function will stop with an error when issues are found. If FALSE, it will return a list of issues. Default is FALSE.

Value

If stop_on_error is FALSE, returns a list with the following components:

is_valid	Logical indicating if all checks passed
date_gaps	Data frame of date gaps found, or NULL if none
missing_values	Data frame summarizing missing values, or NULL if none
extreme_values	Data frame of rows with extreme values, or NULL if none

If stop_on_error is TRUE and issues are found, the function stops with an error message.

Examples

```
file <- system.file("extdata/ppd_72150.met", package = "tidyweather")
records <- read_weather(file)
result <- check_weather(records)
if (result$is_valid) {
  print("Weather data passed all quality checks")
} else {
  print(result)
}
```

---

day_length *The time elapsed in hours between the specified sun angle from 90 degree in am and pm. +ve above the horizon, -ve below the horizon.*

Description

The time elapsed in hours between the specified sun angle from 90 degree in am and pm. +ve above the horizon, -ve below the horizon.

Usage

```
day_length(doy, latitude = NULL, angle = -6)
```

Arguments

doy	day of year number
latitude	latitude of site (deg)
angle	angle to measure time between, such as twilight (deg). angular distance between 90 deg and end of twilight - altitude of sun. +ve up, -ve down.

Value

day length in hours

`interpolate_3hr` *Interpolate 3-Hourly Temperature Values using sine curve.*

Description

Interpolates temperature values at 3-hourly intervals from daily minimum and maximum temperatures using a sine curve.

Usage

```
interpolate_3hr(mint, maxt)
```

Arguments

`mint`	A numeric vector of daily minimum temperatures.
`maxt`	A numeric vector of daily maximum temperatures.

Value

A numeric matrix of interpolated 3-hourly temperature values. Rows correspond to input minimum and maximum temperatures and columns correspond to the eight interpolated 3-hourly intervals.

Examples

```
mint <- c(0, 10)
maxt <- c(30, 40)
interpolate_3hr(mint = mint, maxt = maxt)
```

`interpolate_hourly_sin_pp_adjusted` *Interpolate Hourly Temperature Values using APSIM HourlySinPpAdjusted.*

Description

Interpolates hourly temperatures from daily minimum and maximum temperatures using the APSIM HourlySinPpAdjusted method. Daytime temperatures are sinusoidal and nighttime temperatures follow an exponential cooling curve. Day length, sunrise and sunset are calculated from `latitude` and `date`.

Usage

```
interpolate_hourly_sin_pp_adjusted(
  mint,
  maxt,
  date,
  latitude = NULL,
  hourly = FALSE
)
```

Arguments

mint	A numeric vector of daily minimum temperatures.
maxt	A numeric vector of daily maximum temperatures.
date	A Date vector with the same length as mint, used to determine the day of year for day length calculations.
latitude	Latitude of the site (deg) as a single numeric value.
hourly	A logical value indicating whether to return hourly temperatures. If FALSE, daily mean temperatures are returned.

Value

A numeric vector of daily mean temperatures, with one value per input day if hourly = FALSE. If hourly = TRUE, a data frame is returned with columns date, mint, maxt, hour, and temp.

Examples

```
mint <- c(10, 11, 12)
maxt <- c(30, 31, 32)
date <- as.Date(c("2020-01-01", "2020-01-02", "2020-01-03"))
interpolate_hourly_sin_pp_adjusted(mint, maxt, date = date, latitude = -27.5)

hourly <- interpolate_hourly_sin_pp_adjusted(
    mint,
    maxt,
    date = date,
    latitude = -27.5,
    hourly = TRUE
)
library(ggplot2)
hourly |>
    ggplot(aes(x = timestamp, y = temp)) +
    geom_line() +
    geom_point(aes(x = timestamp, y = mint), color = "blue") +
    geom_point(aes(x = timestamp, y = maxt), color = "red") +
    theme_bw() +
    labs(
        x = "Date Time",
        y = "Temperature (°C)",
        title = "Hourly Temperature Interpolation"
```

```
)
```

`interpolation_function` *Return a y value from a linear interpolation function*

Description

Return a y value from a linear interpolation function

Usage

```
interpolation_function(x, y, values, split = "\\s+")
```

Arguments

x	x
y	y
values	values
split	split

Value

The interpolated values

`last_frost_day` *Calculate the last frost day*

Description

This function calculates the last frost day from a numeric vector of daily minimum temperatures using tidyverse principles.

Usage

```
last_frost_day(
  .data,
  threshold = weather_options$get("extreme.frost_threshold"),
  hemisphere = "south",
  require_full_year = weather_options$get("require_full_year")
)
```

Arguments

.data	A data frame or tibble containing daily minimum temperatures in a column named "mint".
threshold	The stress temperature threshold for frost (default: 0)
hemisphere	Hemisphere indicator: "south" or "north" (default: "south"). If latitude information is available in the data, it will be used to determine the hemisphere.
require_full_year	Logical. If TRUE, requires exactly 365 or 366 days (default: TRUE)

Value

An data.frame or tibble representing the day of year for the last frost, or NA if no frost occurs

Examples

```
file <- system.file("extdata/ppd_72150.met", package = "tidyweather")
records <- read_weather(file)
records |>
    dplyr::group_by(year) |>
    last_frost_day(require_full_year = FALSE)
```

---

number_frost_day *Calculate the number of frost days*

Description

This function calculates the number of frost days from a numeric vector of daily minimum temperatures using tidyverse principles.

Usage

```
number_frost_day(
  .data,
  threshold = weather_options$get("extreme.frost_threshold"),
  require_full_year = weather_options$get("require_full_year")
)
```

Arguments

.data	A data frame or tibble containing daily minimum temperatures in a column named "mint".
threshold	The stress temperature threshold for frost (default: 0)
require_full_year	Logical. If TRUE, requires exactly 365 or 366 days (default: TRUE)

Value

An data.frame or tibble representing the number of frost days, or 0 if no frost occurs

Examples

```
file <- system.file("extdata/ppd_72150.met", package = "tidyweather")
records <- read_weather(file)
records |>
    dplyr::group_by(year) |>
    number_frost_day(require_full_year = FALSE)
```

`read_weather` *Read weather records from a file list and/or a folder list*

Description

Read weather records from a file list and/or a folder list

Usage

```
read_weather(file, format = "APSIM", ...)
```

Arguments

`file`	A character string to specify weather filename.
`format`	A character string to specify the format of weather file.
`...`	Other arguments

Value

A data.frame which contains all weather data.

Examples

```
file <- system.file("extdata/ppd_72150.met", package = "tidyweather")
records <- read_weather(file)
head(records)
```

spherical_distance *Calculate the sphere distance between two points on the Earth*

Description

Calculate the sphere distance between two points on the Earth

Usage

```
spherical_distance(lat1, lon1, lat2, lon2)
```

Arguments

lat1	Latitude of the first point in degrees. Numeric scalar (-90 to 90)
lon1	Longitude of the first point in degrees. Numeric scalar (-180 to 180)
lat2	Latitudes of the second point in degrees. Numeric vector (-90 to 90)
lon2	Longitudes of the second point in degrees. Numeric vector (-180 to 180)

Value

Distance in km

Examples

```
spherical_distance(34.05, -118.25, 40.7128, -74.0060) # Distance between Los Angeles and New York
```

summarise_weather *Summarise Weather Extremes and Key Indicators*

Description

This function calculates summary metrics for weather data, including the number of frost days and the last frost day, grouped by one or more grouping variables. The function uses package-wide options for thresholds and year completeness.

Usage

```
summarise_weather(.data)
```

Arguments

.data	A tibble or data frame containing daily weather data. Must include at least a mint column for daily minimum temperatures. A day column is recommended if require_full_year = TRUE.

Details

The function retrieves thresholds and settings from the global tidyweather options via `weather_options$get()`. The default frost threshold is `weather_options$get("extreme.frost_threshold")` and `require_full_year` is `weather_options$get("require_full_year")`. These can be changed using `weather_options$set()`.

This function is designed to work with grouped tibbles (e.g., after `dplyr::group_by()`), applying the summary per group.

Value

A tibble with one row per group, containing the following columns:

number_frost_days Number of days where minimum temperature is below the frost threshold.

last_frost_day The day of year of the last frost (or `NA` if none).

Examples

```
library(dplyr)

# Example weather data (daily minimum temperatures)
weather_data <- read_weather(system.file("extdata/ppd_72150.met", package = "tidyweather"))

# Summarise without grouping
weather_options$set("require_full_year" = FALSE)
summarise_weather(weather_data)

# Summarise by group (e.g., year)
weather_data_grouped <- weather_data %>% group_by(year)
summarise_weather(weather_data_grouped)
```

---

sun_rise_set *Sunrise and sunset times*

Description

Calculate the time of sunrise or sunset (or of the sun reaching a given angle below the horizon, such as twilight) for a vector of dates at a site, using the low-precision algorithm from the Almanac for Computers (accurate to about a minute or two).

Usage

```
sun_rise_set(
  date,
  latitude,
  longitude,
  timezone = "UTC",
  angle = 6,
  flag = c("rise", "set")
)
```

Arguments

date	dates of class Date.
latitude	latitude of site (deg), positive north, in [-90, 90].
longitude	longitude of site (deg), positive east, in [-180, 180].
timezone	Olson time zone name of the returned times, such as "UTC" (default) or "Australia/Brisbane". See OlsonNames().
angle	angle of the sun below the horizon (deg) at which the time is calculated. Use 0 for the geometric horizon, 0.833 for the conventional sunrise and sunset (refraction and solar disc), and 6, 12 or 18 for civil, nautical and astronomical twilight. Positive values are below the horizon. Default is 6.
flag	"rise" for the morning event or "set" for the evening event.

Value

A POSIXct vector in timezone, on the local calendar day of date. An error is raised if the sun does not reach angle on a date (for example polar day or night).

Examples

```
date <- as.Date("2021-06-21")
sun_rise_set(date, latitude = -27.5, longitude = 153, angle = 0)
sun_rise_set(date, latitude = -27.5, longitude = 153, timezone = "Australia/Brisbane",
             angle = 0, flag = "rise")
sun_rise_set(date, latitude = -27.5, longitude = 153, timezone = "Australia/Brisbane",
             angle = 0, flag = "set")
```

---

thermal_time *Calculate thermal time using cardinal temperatures*

Description

Calculate thermal time using cardinal temperatures

Usage

```
thermal_time(mint, maxt, x_temp, y_temp, method = NULL, ...)
```

Arguments

mint	The minimum temperature
maxt	The maximum temperature
x_temp	The cardinal temperatures
y_temp	The effective thermal time

method	The method to calculate thermal time. The default method is (maxt + mint) / 2 - base. The three hour temperature methods will be usesd if method = '3hr'. The APSIM HourlySinPpAdjusted method can be selected with method = 'HourlySinPpAdjusted'.
...	Additional arguments passed to the method-specific functions. Required arguments for method = 'HourlySinPpAdjusted' are, latitude and date.

Value

The thermal time.

Examples

```
mint <- c(0, 10)
maxt <- c(30, 40)
x_temp <- c(0, 20, 35)
y_temp <- c(0, 20, 0)
thermal_time(mint, maxt, x_temp, y_temp)
thermal_time(mint, maxt, x_temp, y_temp, method = '3hr')
date <- as.Date(c("2020-01-01", "2020-01-02"))
thermal_time(mint, maxt, x_temp, y_temp, method = 'HourlySinPpAdjusted',
             latitude = -27.5, date = date)
```

---

ttest_ts *Significantly t-test with auto-correlation for time serial data*

Description

Method is presented by Santer et al. 2000

Usage

```
ttest_ts(y, slope = NULL)
```

Arguments

y	A vector of time serial data
slope	Whether export slope

Value

p values of t-test

wang_engel_tt *Wang-Engel temperature response function*

Description

Calculate the shape of the Wang and Engel beta temperature response function defined by three cardinal temperatures.

Usage

```
wang_engel_tt(min_temp, opt_temp, max_temp, beta = 1, n = 101)
```

Arguments

min_temp	The minimum cardinal temperature.
opt_temp	The optimum cardinal temperature.
max_temp	The maximum cardinal temperature.
beta	The exponent applied to the relative response (0-1). Default is 1.
n	The number of evenly spaced temperatures between min_temp and max_temp at which the response is evaluated. Default is 101.

Value

A data frame with columns x and y for the temperature and response, respectively.

References

Wang E, Engel T (1998). Simulation of phenological development of wheat crops. *Agricultural Systems* 58, 1-24.

Examples

```
library(ggplot2)
tt <- wang_engel_tt(0, 25, 35)
tt |>
    ggplot() +
    geom_line(aes(x = x, y = y)) +
    labs(x = "Temperature", y = "Response") +
    geom_vline(xintercept = c(0, 25, 35), linetype = "dashed", color = "red") +
    geom_hline(yintercept = c(0, 25), linetype = "dashed", color = "blue") +
    theme_bw()
# Change the beta and number of points
tt <- wang_engel_tt(0, 25, 35, beta = 0.5, n = 36)
```

weather_options *tidyweather options*

Description

An options manager for configuring tidyweather parameters. This object provides methods to get and set weather-related parameters.

Usage

```
weather_options
```

Available Options

extreme.frost_threshold Frost threshold for extreme weather events. Default: 0

require_full_year Whether to require a full year of data for certain calculations. Default: TRUE

Methods

get(key) Retrieve the value of an option by its key (e.g., "extreme.frost_threshold")

set(key, value) Set the value of an option by its key

reset() Reset all options to their default values

Examples

```
# Get default frost threshold
weather_options$get("extreme.frost_threshold")

# Set custom values
weather_options$set("extreme.frost_threshold" = -2)

# Reset to defaults
weather_options$reset()
```

write_weather *Write weather data to file*

Description

Exports weather records to a file in the specified format. Currently supports APSIM format for agricultural modeling applications.

Usage

```
write_weather(records, file, format = "APSIM", overwrite = FALSE)
```

Arguments

records	A data frame containing weather data with columns for date, temperature, precipitation, and other meteorological variables
file	Character string specifying the output file path
format	Character string specifying the output format. Currently supports "APSIM" (default)
overwrite	Logical indicating whether to overwrite existing files. Default is FALSE

Value

Invisibly returns the file path of the written file

Examples

```
# Read sample weather data from package
file <- system.file("extdata/ppd_72150.met", package = "tidyweather")
records <- read_weather(file)

# Write to temporary file
temp_file <- tempfile(fileext = ".met")

# Write to APSIM format
write_weather(records, temp_file, format = "APSIM")

# Overwrite existing file
write_weather(records, temp_file, format = "APSIM", overwrite = TRUE)
```

Index

check_weather, 2

day_length, 3

interpolate_3hr, 4
interpolate_hourly_sin_pp_adjusted, 4
interpolation_function, 6

last_frost_day, 6

number_frost_day, 7

OlsonNames(), *11*

read_weather, 8

spherical_distance, 9
summarise_weather, 9
sun_rise_set, 10

thermal_time, 11
ttest_ts, 12

wang_engel_tt, 13
weather_options, 14
write_weather, 14