

Package: aelab (via r-universe)

October 21, 2026

Type Package

Title Data Processing for Aquatic Ecology

Version 1.1.6

Maintainer Zhao-Jun Yong <nuannuan0425@gmail.com>

Description Facilitate the analysis of data related to aquatic ecology, specifically the establishment of carbon budget. Currently, the package allows the below analysis. (i) the calculation of greenhouse gas flux based on data obtained from trace gas analyzer using the method described in Lin et al. (2024). (ii) the calculation of Dissolved Oxygen (DO) metabolism based on data obtained from dissolved oxygen data logger using the method described in Staehr et al. (2010). Yong et al. (2024) <[doi:10.5194/bg-21-5247-2024](https://doi.org/10.5194/bg-21-5247-2024)>. Staehr et al. (2010) <[doi:10.4319/lom.2010.8.0628](https://doi.org/10.4319/lom.2010.8.0628)>.

Imports tibble, lubridate, stats, dplyr (>= 1.1.0), openxlsx, readxl, ggplot2, readr, tidyr, stringr, purrr, rlang, grDevices, multcompView, FSA, rcompanion, rnaturlaearth, sf, ggspatial, flextable, officer

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0), rnaturlaearthdata, suncalc

Config/testthat/edition 3

Depends R (>= 4.1.0)

VignetteBuilder knitr

LazyData true

Config/pak/sysreqs libabsl-dev libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev libgdal-dev gdal-bin libgeos-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev zlib1g-dev

Repository <https://zhao-jun-yong.r-universe.dev>

Date/Publication 2026-10-21 10:25:17 UTC

RemoteUrl <https://github.com/zhao-jun-yong/aelab>

RemoteRef HEAD

RemoteSha e86054317636fa34ff9fe234dc4d445bd3a0c32d

Contents

add_sun_times	3
aelab_palettes	4
analyzer_precision	4
aov_test	5
calc_chla_trichromatic	6
calculate_do	6
calculate_ghg_flux	7
calculate_MDF	8
calculate_regression	9
calculate_total_co2e	13
combine_hobo	14
combine_weather	14
combine_weather_mh	15
combine_weather_month	16
convert_ghg_unit	17
convert_time	18
descriptive_statistic	19
df_trans	20
filter_complete_days	20
find_outlier	21
gantt_table	22
gantt_text	23
hobo	24
ks_test	24
make_reference	25
n2o	26
normality_test_aov	27
normality_test_t	27
plot_bar	28
plot_box	29
plot_ghg_flux	30
plot_hobo	32
plot_line	32
plot_map_taiwan	33
plot_point	34
process_hobo	35
process_info	36
process_weather	37
process_weather_mh	37

<i>add_sun_times</i>	3
process_weather_month	38
scale_colour_aelab_c	39
scale_colour_aelab_d	40
scale_fill_aelab_c	40
scale_fill_aelab_d	41
sig_labels	42
tidy_ghg_analyzer	43
Index	44

<code>add_sun_times</code>	<i>add_sun_times</i>
----------------------------	----------------------

Description

Attach per-day sunrise and sunset times to a data frame of readings, computed from a site's coordinates with the **suncalc** package. Replaces the manual lookup of sunrise/sunset for each deployment date; the resulting **sunrise**/**sunset** columns are consumed by [calculate_do](#).

Usage

```
add_sun_times(df, lat, lon, time_col = "date_time", tz = "Asia/Taipei")
```

Arguments

- `df` A data frame containing a POSIXct time column.
- `lat, lon` Site latitude and longitude in decimal degrees (length 1).
- `time_col` Name of the POSIXct time column. Default "date_time".
- `tz` Time zone for the returned sun times. Default "Asia/Taipei".

Value

`df` with two added POSIXct columns, **sunrise** and **sunset**, matched to each row's calendar date.

Examples

```
## Not run:
df <- add_sun_times(df, lat = 22.3900, lon = 120.5777)

## End(Not run)
```

<code>aelab_palettes</code>	<i>aelab_palettes</i>
-----------------------------	-----------------------

Description

Retrieve a named aelab colour palette as a character vector.

Usage

```
aelab_palettes(name, n, type = c("discrete", "continuous"))
```

Arguments

name	Name of the palette (string).
n	Number of colours to return. Defaults to the full palette length.
type	"discrete" (default) or "continuous". For "continuous", colours are interpolated to length n.

Details

Available palette names: "rainbow", "two", "control", "control2", "control3", "period", "ghg".

Value

A character vector of hex colour codes with class "palette".

Examples

```
aelab_palettes("rainbow", 5)
aelab_palettes("ghg", type = "continuous", n = 20)
```

<code>analyzer_precision</code>	<i>Instrument precision for supported trace gas analyzers</i>
---------------------------------	---

Description

A lookup table of 1-second (1) precision values used to compute `kappamax` in [calculate_regression](#) when `fit_type = "auto"`. Add rows for additional analyzer models as needed.

Format

A data frame with 5 rows and 4 variables:

model	Analyzer model name (character).
gas	Gas measured: "CH4", "CO2", or "N2O" (character).
precision	Instrument precision (1 , 1 s averaging) in unit (numeric).
unit	Unit the analyzer reports for that gas, so precision matches the data from tidy_ghg_analyzer ("ppb" for LI-COR CH4 and N2O, otherwise "ppm") (character).

Source

LI-7810 and LI-7820 specifications pages (LI-COR Biosciences; CH4 0.60 ppb at 2 ppm, CO2 3.5 ppm at 400 ppm, N2O 0.40 ppb at 330 ppb; checked 2026-10-21); LGR (Los Gatos Research) UGGA product specifications.

Examples

```
analyzer_precision
analyzer_precision[analyzer_precision$gas == "N20", "precision"]
```

aov_test	<i>aov_test</i>
----------	-----------------

Description

Perform one-way ANOVA followed by Tukey HSD post-hoc test with compact letter display.

Usage

```
aov_test(df, variable_name, group)
```

Arguments

df A data frame.

variable_name Name of the response variable column (string).

group Name of the grouping column (string).

Value

A named list with elements `anova_summary`, `tukey_results`, and `compact_letters`.

Examples

```
df <- data.frame(
  grp = rep(c("A","B","C"), each = 5),
  val = c(1,2,1,2,1, 3,4,3,4,3, 5,6,5,6,5)
)
aov_test(df, "val", "grp")
```

<code>calc_chla_trichromatic</code>	<i>calc_chla_trichromatic</i>
-------------------------------------	-------------------------------

Description

Calculate chlorophyll-a concentration from trichromatic spectrophotometric absorbance readings using the Jeffrey & Humphrey (1975) equations.

Usage

```
calc_chla_trichromatic(wl_630, wl_647, wl_664, wl_750)
```

Arguments

<code>wl_630</code>	Absorbance at 630 nm.
<code>wl_647</code>	Absorbance at 647 nm.
<code>wl_664</code>	Absorbance at 664 nm.
<code>wl_750</code>	Absorbance at 750 nm (turbidity blank).

Details

Absorbance values should be measured in a 1 cm path-length cuvette. The 750 nm reading is used as a turbidity blank correction. Formula: $11.85 \times E_{664} - 1.54 \times E_{647} - 0.08 \times E_{630}$ where $E_{\lambda} = A_{\lambda} - A_{750}$.

Value

Chlorophyll-a concentration in $\mu\text{g L}^{-1}$ (assuming a 1 cm path length and standard extraction volume).

Examples

```
calc_chla_trichromatic(wl_630 = 0.05, wl_647 = 0.08, wl_664 = 0.20, wl_750 = 0.01)
```

<code>calculate_do</code>	<i>calculate_do</i>
---------------------------	---------------------

Description

Calculate the Net Ecosystem Production, Gross Primary Production and Ecosystem respiration based on the change in dissolved oxygen concentration.

Usage

```
calculate_do(df)
```

Arguments

df Merged dataframe produced by process_hobo(), process_weather() and process_info() functions.

Value

A dataframe.

Examples

```
data(hobo)
calculate_do(hobo)
```

calculate_ghg_flux	<i>calculate_ghg_flux</i>
--------------------	---------------------------

Description

Calculate the greenhouse gas (GHG) flux based on input parameters from a data frame.

Usage

```
calculate_ghg_flux(  
  data,  
  slope = "slope",  
  area = "area",  
  volume = "volume",  
  temp = "temp"  
)
```

Arguments

data	A data frame containing relevant data with columns for slope, area, volume, and temperature.
slope	Name of the column in 'data' that contains the slope values of the GHG concentration change (in ppm/s).
area	Name of the column in 'data' that contains the values of the area of the chamber (in square meter).
volume	Name of the column in 'data' that contains values of the volume of the chamber (in litre).
temp	Name of the column in 'data' that contains values of the temperature of the gas (in Celsius).

Value

A list containing the calculated flux and its unit.

Examples

```
data <- data.frame(
  slope = c(1.2, 1.5, 1.1),
  area = c(100, 150, 120),
  volume = c(10, 15, 12),
  temp = c(25, 30, 22)
)
results <- calculate_ghg_flux(data)
print(results)
```

calculate_MDF

calculate_MDF

Description

Calculate the Minimum Detectable Flux (MDF) for a static chamber GHG measurement system.

Usage

```
calculate_MDF(
  precision_ppm,
  closure_time_s,
  data_point_n,
  chamber_volume_m3,
  temperature_C,
  chamber_area_m2,
  pressure_pa = 101325,
  ideal_constant = 8.314,
  ghg = "co2"
)
```

Arguments

<code>precision_ppm</code>	Precision of the gas analyser (ppm).
<code>closure_time_s</code>	Closure time of the measurement (seconds).
<code>data_point_n</code>	Number of data points recorded during the closure period.
<code>chamber_volume_m3</code>	Internal volume of the chamber (m ³).
<code>temperature_C</code>	Air temperature at the measurement location (°C).
<code>chamber_area_m2</code>	Base area of the chamber (m ²).
<code>pressure_pa</code>	Atmospheric pressure (Pa). Default 101325.
<code>ideal_constant</code>	Ideal gas constant (J mol ⁻¹ K ⁻¹). Default 8.314.
<code>ghg</code>	Greenhouse gas type: "co2", "ch4", or "n2o". Default "co2".

Value

A named list with MDF (numeric, $\mu\text{g m}^{-2} \text{ h}^{-1}$) and **unit** (string).

Examples

```
calculate_MDF(
  precision_ppm      = 1,
  closure_time_s     = 300,
  data_point_n       = 300,
  chamber_volume_m3  = 0.0064,
  temperature_C       = 25,
  chamber_area_m2    = 0.07
)
```

```
calculate_regression  calculate_regression
```

Description

Calculate the slope of GHG concentration change over time using linear or exponential regression.

Usage

```
calculate_regression(
  data,
  reference_df,
  ghg,
  reference_time,
  site,
  analyzer_code,
  duration_minutes = 7,
  num_rows = 300,
  fit_type = "linear",
  t_zero = 0,
  window_type = "sliding",
  start_offset_s = 0,
  r2_threshold = 0.8,
  cor_threshold = 0.5,
  instrument_precision = NULL,
  kappamax = NULL,
  g_factor_threshold = NULL,
  min_deadband = 30,
  max_deadband = 60,
  deadband_step = 10,
  min_window_size = 60,
  cutoff_step = 10,
  n_boot = 200
)
```

Arguments

data	Data from the GHG analyzer that has been processed and time-converted.
reference_df	A data frame containing measurement reference times, site names, and an analyzer column.
ghg	Column name in data containing GHG concentration values (e.g., "CH4", "N2O").
reference_time	Column name in reference_df containing the measurement start times (POSIXct).
site	Column name in reference_df containing site identifiers.
analyzer_code	String pattern used to filter results by the analyzer column of reference_df .
duration_minutes	The duration of the measurement window in minutes. Default 7. When window_type = "fixed" and reference_df contains a duration_minutes column, the per-row value (if not NA) overrides this global parameter for that specific measurement. This allows different window lengths for individual chamber closures.
num_rows	The number of rows used per sliding regression window. Only used when window_type = "sliding". Default 300.
fit_type	Model used to fit the concentration curve. One of "linear" (default), "quadratic" (second-order polynomial; slope is the initial rate at t=0), "exp_tz" (exponential with linear drift and user-specified t_zero), "exp_zhao18" (exponential with linear drift and estimated t0), or "auto" (automatic model selection: tries linear -> quadratic -> exp_tz in order, accepting a more complex model only when its AICc is lower (Kutzbach et al. 2007), a quadratic only when it levels off rather than accelerates (Levy et al. 2011), and exp_tz only when its curvature parameter does not exceed kappamax). When fit_type = "auto" and instrument_precision = NULL, precision is looked up from analyzer_precision by gas name. The model chosen for each measurement is recorded in the fit_used output column.
t_zero	Reference time in seconds from window start for the "exp_tz" slope calculation. Default 0.
window_type	"sliding" (default) selects the num_rows -length sub-window with the best R^2 within the measurement period. "fixed" fits a single window starting at start_offset_s seconds after the reference time and running for duration_minutes . Use "fixed" to ensure all gases are regressed over the same time window. "pareto" independently searches deadband (start skip) and cutoff (window end) times, trading off fit quality against propagated slope uncertainty via a Pareto front - adapted from the window-selection idea in Kobayashi et al. (2026), with two deliberate differences from that method: the existing kappamax gate is always applied (no candidate can be a numerically degenerate exponential fit), and a candidate pool that is empty or whose selected window fails the existing zero/discard checks is reported as such rather than returning a confident-looking number (see flag).

Experimental, slope_uncertainty_lo/slope_uncertainty_hi not yet validated for reporting. The winner's-curse correction applied to this interval (see `.pareto_winners_curse_correct()`'s source comment) is calibrated only against idealized synthetic data (i.i.d. Gaussian noise around an exactly linear trend) - not yet checked against real chamber closures, autocorrelated noise, or non-linear true trends. Do not cite these intervals as validated in a manuscript or report until that broader validation is done; `window_type = "fixed"/"sliding"` are unaffected by this.

<code>start_offset_s</code>	Seconds to skip from the reference time before the regression window starts. Only used when <code>window_type = "fixed"</code> . Default 0. When <code>reference_df</code> contains a <code>start_offset_s</code> column, the per-row value (if not NA) overrides this global parameter for that specific measurement.
<code>r2_threshold</code>	Minimum R-squared for a measurement to pass quality check. Default 0.8.
<code>cor_threshold</code>	Minimum absolute Pearson correlation between concentration and elapsed time. Measurements below this have no detectable trend and are flagged "zero". Default 0.5.
<code>instrument_precision</code>	Optional. Analyser precision (1) in the same concentration unit as the <code>ghg</code> column (ppb for LI-COR CH ₄ /N ₂ O, ppm for CO ₂). Used in two ways: (1) minimum-detectable-slope check - <code>below_mdf</code> is TRUE when the absolute slope falls below <code>instrument_precision / window_duration_s</code> ; the flag is not changed, because low fluxes with a good fit are kept (Gaudard et al. 2025; Maier et al. 2022); (2) kappamax for <code>fit_type = "auto"</code> - following Hueppi et al. (2018), $\kappa_{max} = \text{slope}_{lm} /\text{precision}$, so that stronger fluxes permit more curvature and near-detection-limit measurements are kept linear.
<code>kappamax</code>	Optional. Maximum acceptable exponential curvature parameter b (s ⁻¹). Only applies when <code>fit_type</code> is "exp_tz" or "exp_zhao18". When $b > \kappa_{max}$, the fit implies an implausibly fast equilibration and the slope is replaced by the linear equivalent. Corresponds to the <i>kappamax</i> quality criterion in Hueppi et al. (2018).
<code>g_factor_threshold</code>	Optional. Maximum acceptable ratio of the chosen model slope to the linear baseline slope (g-factor; Gaudard et al. 2025). Only computed when <code>fit_type = "auto"</code> and the selected model is not linear. Values far from 1 indicate that the nonlinear model diverges strongly from the linear fit, which may signal noise artefacts or poor convergence. When this threshold is exceeded, the flag is overridden to "discard". Default NULL (disabled). goFlux uses 2 as its default (<code>g.limit</code> ; Rheault et al. 2024), but there it is two-sided ($ g > 2$ or < 0.5) and switches to the linear flux rather than discarding; no published study justifies a value.
<code>min_deadband, max_deadband, deadband_step</code>	Only used when <code>window_type = "pareto"</code> . Define the candidate deadband (start-skip) grid in seconds, relative to the reference time: <code>seq(min_deadband,</code>

`max_deadband - deadband_step`, `by = deadband_step`). Defaults 30, 60, 10. `min_deadband` defaults to 30 (rather than 0) to match this package's own existing `start_offset_s = 30` convention used in `window_type = "fixed"` batch processing across this lab's campaigns; still fully overridable, e.g. set to 0 to let the search consider no dead-band at all.

`min_window_size`, `cutoff_step`
Only used when `window_type = "pareto"`. `min_window_size` (default 60) is the minimum candidate window length in seconds (`cutoff - deadband`); `cutoff_step` (default 10) is the step size of the candidate cutoff-time grid, which runs from `min_deadband + min_window_size` to the window's own maximum elapsed time.

`n_boot`
Only used when `window_type = "pareto"`. Number of residual-bootstrap refits used to estimate each candidate window's slope uncertainty (16th/84th percentile of the bootstrapped slopes). Default 200. Each refit resamples (with replacement) the candidate's own degrees-of-freedom-corrected residuals (`resid * sqrt(n / (n - p))`), `p` = number of fitted parameters for the model actually used), adds them to the candidate's fitted values to build a synthetic concentration series, and refits the same model family via `calculate_regression`'s internal fitting routine - no MCMC and no new dependency.

Value

A tibble with columns: `start_time`, `end_time`, `slope` (ppm s⁻¹), `r_square`, `correlation`, `b_param` (exponential curvature; NA for linear/quadratic), `fit_used` (model actually used: "linear", "quadratic", or "exp_tz"; always matches `fit_type` unless `fit_type = "auto"`), `g_factor` (ratio of chosen slope to linear baseline slope; NA when `fit_used = "linear"` or for explicit non-auto fits), `flag` ("ok", "discard", "zero", or "no_data"), `below_mdf` (TRUE when the slope is below the minimum detectable slope; NA without `instrument_precision`), `reference_time`, `site`, `analyzer`. When `window_type = "pareto"`, five additional columns are included: `deadband/cutoff` (the selected candidate window, seconds from the reference time; NA for other window types), `slope_uncertainty_lo/ slope_uncertainty_hi` (16th/84th percentile of the residual-bootstrap slope distribution for the selected window), and `n_candidates` (number of deadband/cutoff candidates that produced a usable fit; 0 when the measurement was flagged "discard" because no candidate survived).

References

- Hutchinson, G. L., & Mosier, A. R. (1981). Improved soil cover method for field measurement of nitrous oxide fluxes. *Soil Science Society of America Journal*, 45(2), 311-316. doi:10.2136/sssaj1981.03615995004500020017x
- Hüppi, R., Felber, R., Krauss, M., Six, J., Leifeld, J., & Fuß, R. (2018). Restricting the nonlinearity parameter in soil greenhouse gas flux calculations for more reliable flux estimates. *PLOS ONE*, 13(7), e0200876. doi:10.1371/journal.pone.0200876
- Gaudard, J., et al. (2025). fluxible: An R package to process ecosystem gas fluxes from closed-loop chambers in an automated and reproducible way. *Methods in Ecology and Evolution*. doi:10.1111/2041210X.70161

Kobayashi, A. N. A., Roques, C., Hunkeler, D., & Brunner, P. (2026). Uncertainty quantification in soil gas fluxes: Reducing the user's arbitrariness. *European Journal of Soil Science*, 77, e70415. doi:10.1111/ejss.70415

Zrnic, T., & Fithian, W. (2025). Locally simultaneous inference. *Annals of Statistics*. The `window_type = "pareto" slope_uncertainty_lo/slope_uncertainty_hi` interval applies a correction inspired by (not an implementation of) this paper's Section 4 "effective Bonferroni" idea to the winner's-curse bias in the selected candidate's bootstrap interval, sized by Pareto-front membership with an empirically-calibrated (not analytically derived) correlation discount - see `.pareto_winners_curse_correct()`'s source comment for calibration caveats not yet resolved for real field data.

Examples

```
data(n2o)
n2o_converted <- convert_time(n2o)
ref <- data.frame(
  date_time = n2o_converted$real_datetime[1],
  site = "S1",
  analyzer = "1074"
)
calculate_regression(n2o_converted, ref, "N2O",
  reference_time = "date_time", site = "site",
  analyzer_code = "1074")
```

calculate_total_co2e calculate_total_co2e

Description

Convert individual GHG fluxes ($\text{mg m}^{-2} \text{ h}^{-1}$) to a total CO_2 -equivalent flux ($\text{g m}^{-2} \text{ d}^{-1}$) using IPCC AR6 100-year GWPs ($\text{CO}_2 = 1$, $\text{CH}_4 = 27$, $\text{N}_2\text{O} = 273$).

Usage

```
calculate_total_co2e(co2 = 0, ch4 = 0, n2o = 0)
```

Arguments

co2	CO_2 flux in $\text{mg m}^{-2} \text{ h}^{-1}$. Default 0.
ch4	CH_4 flux in $\text{mg m}^{-2} \text{ h}^{-1}$. Default 0.
n2o	N_2O flux in $\text{mg m}^{-2} \text{ h}^{-1}$. Default 0.

Value

Total CO_2e flux as a numeric scalar ($\text{g m}^{-2} \text{ d}^{-1}$), printed with a diagnostic message.

Examples

```
calculate_total_co2e(co2 = 4.02, ch4 = 0.001, n2o = 0.003)
```

combine_hobo	<i>combine_hobo</i>
--------------	---------------------

Description

Tidy multiple data retrieved from HOBO U26 Dissolved Oxygen Data Logger.

Usage

```
combine_hobo(file_path, file_prefix = "no.")
```

Arguments

<code>file_path</code>	Directory of the folder containing the files.
<code>file_prefix</code>	The prefix before the code for the data logger, defaults to "no."

Value

A dataframe.

Examples

```
hobo_data_path <- system.file("extdata", package = "aelab")
df <- combine_hobo(hobo_data_path, file_prefix = "ex_ho")
```

combine_weather	<i>combine_weather</i>
-----------------	------------------------

Description

Tidy multiple daily weather data downloaded from weather station in Taiwan.

Usage

```
combine_weather(file_path, start_date, end_date, zone)
```

Arguments

<code>file_path</code>	Directory of folder containing the files (including the character in the file name that precedes the date).
<code>start_date</code>	Date of the daily weather data in yyyy-mm-dd format.
<code>end_date</code>	Date of the daily weather data in yyyy-mm-dd format.
<code>zone</code>	Code for the region of the weather station.

Value

A dataframe.

Examples

```
weather_data_path <- system.file("extdata", package = "aelab")
modified_data_path <- paste0(weather_data_path, "/ex_")
df <- combine_weather(modified_data_path,
  start_date = "2024-01-01",
  end_date = "2024-01-02", "site_A"
)
```

combine_weather_mh	<i>combine_weather_mh</i>
--------------------	---------------------------

Description

Batch-import and combine multiple CWA "MH" (Multifield Hourly) weather reports. Hourly CODiS downloads are capped at a one-month range, so a date span typically yields several `LotsDataReports.txt` files; this reads them all and row-binds the result.

Usage

```
combine_weather_mh(
  file_paths,
  station_id = NULL,
  expand_30min = TRUE,
  tz = "Asia/Taipei"
)
```

Arguments

<code>file_paths</code>	Either a character vector of MH file paths, or a single directory path (all <code>.txt</code> files within are read, recursively).
<code>station_id</code>	Optional character vector of station codes to keep. Passed to process_weather_mh .
<code>expand_30min</code>	Passed to process_weather_mh . Default <code>TRUE</code> .
<code>tz</code>	Time zone. Default <code>"Asia/Taipei"</code> .

Value

A combined data frame, de-duplicated and ordered by station and time.

Examples

```
## Not run:
df <- combine_weather_mh("data/raw/sw_mono_do/weather_mh",
  station_id = c("COR660", "COR850")
)

## End(Not run)
```

```
combine_weather_month
      combine_weather_month
```

Description

Batch-import monthly weather CSV files from a Taiwan Central Weather Administration station for a consecutive range of months.

Usage

```
combine_weather_month(file_path, start_month, end_month, year = 2024, zone)
```

Arguments

<code>file_path</code>	Path prefix (directory + filename prefix before the date portion, e.g. "data/weather/").
<code>start_month</code>	First month to import (1-9; two-digit months not yet supported).
<code>end_month</code>	Last month to import.
<code>year</code>	Four-digit year. Default 2024.
<code>zone</code>	Character label for the weather station / region.

Details

File names are expected to follow the pattern `<file_path><year>-0<month>.csv` (e.g. 2024-01.csv).

Value

A combined data frame produced by [process_weather_month](#).

Examples

```
## Not run:
df <- combine_weather_month("data/weather/",
  start_month = 1,
  end_month = 6, year = 2024, zone = "site_A"
)

## End(Not run)
```

convert_ghg_unit	<i>convert_ghg_unit</i>
------------------	-------------------------

Description

Convert a greenhouse gas (GHG) flux value (or a character string containing one or more numeric values, e.g. "0.002 +/- 0.003") between flux units. Defaults to micrograms per square meter per hour.

Usage

```
convert_ghg_unit(
  input,
  ghg,
  mass = "µg",
  area = "m2",
  time = "hr",
  to_mass = "µg",
  to_area = "m2",
  to_time = "hr",
  digits = 2,
  ratio = FALSE
)
```

Arguments

input	A numeric vector or a character string containing one or more numbers. Numeric vectors are handled element-wise, making the function compatible with <code>dplyr::mutate()</code> .
ghg	The molecular formula of the greenhouse gas: "co2", "ch4", or "n2o".
mass	Mass unit of the input flux. One of "mmol", "mg", "g", "ug" (micrograms), "nmol", "Mg", "umol" (micromoles), "mol". Default "ug".
area	Area unit of the input flux. One of "ha", "m2". Default "m2".
time	Time unit of the input flux. One of "yr", "day", "hr", "sec", "min". Default "hr".
to_mass	Target mass unit. Same options as mass . Default "ug" (micrograms). Existing calls without this argument are unaffected.
to_area	Target area unit. One of "ha", "m2". Default "m2".
to_time	Target time unit. Same options as time . Default "hr".
digits	Number of decimal places to round to. Default 2.
ratio	Logical. If TRUE , apply an elemental-ratio correction (C-basis for CH ₄ , N-basis for N ₂ O). Default FALSE .

Details

Numeric values embedded in a string (e.g. mean +/- SD notation) are each converted individually and the surrounding text is preserved. Commas are treated as decimal separators.

Value

For numeric input: a numeric vector of converted values. For character input: a character vector with numbers converted in-place (e.g. "0.002 +/- 0.003" becomes "0.03 +/- 0.05"). Returns NA for missing input.

Examples

```
convert_ghg_unit(97, ghg = "ch4", mass = "mg", area = "m2", time = "hr")
```

<code>convert_time</code>	<i>convert_time</i>
---------------------------	---------------------

Description

Convert the time of the LI-COR Trace Gas Analyzer to match the time in real life.

Usage

```
convert_time(data, day = 0, hr = 0, min = 0, sec = 0)
```

Arguments

<code>data</code>	Data from the LI-COR Trace Gas Analyzer that had been processed by <code>tidy_licor()</code> .
<code>day</code>	Day(s) to add or subtract.
<code>hr</code>	Hour(s) to add or subtract.
<code>min</code>	Minute(s) to add or subtract.
<code>sec</code>	Second(s) to add or subtract.

Value

The input data with a new column in POSIXct format converted based on the input value.

Examples

```
data(n2o)
converted_n2o <- convert_time(n2o, min = -10, sec = 5)
```

```
descriptive_statistic
      descriptive_statistic
```

Description

Compute grouped mean $\pm$ SD and min-max summary statistics for one or more numeric variables.

Usage

```
descriptive_statistic(data, vars, groups, digits = 2)
```

Arguments

data	A data frame.
vars	<[‘tidy-select’][dplyr::dplyr_tidy_select]> Columns to summarise. All selected columns must be numeric.
groups	<[‘tidy-select’][dplyr::dplyr_tidy_select]> Grouping columns. Their original classes (e.g. factor, Date) are preserved.
digits	Number of decimal places to round to. Must be a single non-negative integer. Default is 2.

Value

A tibble with one row per group and two summary columns per variable (‘<var>_mean_sd’ and ‘<var>_min_max’). A group with no non-missing values returns ‘NA’ for both columns instead of ‘NaN’/‘Inf’. A group with exactly one non-missing value reports ‘NA’ for the SD rather than ‘0’.

Examples

```
df <- data.frame(
  group = c("A", "A", "B", "B", "C"),
  value = c(1.1, 2.3, 3.5, 4.7, NA)
)
descriptive_statistic(df, vars = value, groups = group)
```

<code>df_trans</code>	<i>df_trans</i>
-----------------------	-----------------

Description

Apply a reverse square-root or reverse log transformation to a numeric column and append the result as a new column.

Usage

```
df_trans(df, variable_name, transformation)
```

Arguments

`df` A data frame.
`variable_name` Name of the column to transform (string).
`transformation` Transformation type: "sqrt" or "log".

Value

The input data frame with an additional column named `<variable_name>_sqrt` or `<variable_name>_log`.

Examples

```
df <- data.frame(val = c(1, 4, 9, 16))
df_trans(df, "val", "sqrt")
```

<code>filter_complete_days</code>	<i>filter_complete_days</i>
-----------------------------------	-----------------------------

Description

Keep only the calendar days that have (near-)complete diel coverage, per logger. A day qualifies when the number of distinct time-of-day slots present reaches `min_slots`. This replaces the manual inspection of which deployment days run a full 00:00-24:00, so daily DO metabolism is computed only from complete days.

Usage

```
filter_complete_days(
  df,
  group_cols = "no_hobo",
  time_col = "date_time",
  min_slots = 44L,
  tz = "Asia/Taipei",
  summary_only = FALSE
)
```

Arguments

<code>df</code>	A data frame of HOBO readings, e.g. from <code>process_hobo</code> or <code>combine_hobo</code> , containing a POSIXct time column.
<code>group_cols</code>	Character vector of grouping columns identifying one logger deployment, default <code>"no_hobo"</code> .
<code>time_col</code>	Name of the POSIXct time column. Default <code>"date_time"</code> .
<code>min_slots</code>	Minimum number of distinct 30-minute slots a day must have to count as complete. Default 44 (of 48; allows minor gaps).
<code>tz</code>	Time zone used to derive the calendar date. Default <code>"Asia/Taipei"</code> .
<code>summary_only</code>	If TRUE, return a per-day summary (<code>group_cols</code> , <code>date</code> , <code>n_slots</code> , <code>complete</code>) instead of the filtered readings. Useful for inspecting coverage. Default FALSE.

Value

Either the input rows restricted to complete days (default), or a coverage summary when `summary_only = TRUE`.

Examples

```
## Not run:
hobo <- combine_hobo("data/raw/sw_mono_do/csv/FL_T1")
filter_complete_days(hobo, summary_only = TRUE) # inspect coverage
clean <- filter_complete_days(hobo) # keep complete days

## End(Not run)
```

<code>find_outlier</code>	<i>find_outlier</i>
---------------------------	---------------------

Description

Identify outliers in a numeric column using the IQR method (values outside $1.5 \times \text{IQR}$ from $Q1/Q3$).

Usage

```
find_outlier(df, var, other_var = NULL)
```

Arguments

<code>df</code>	A data frame.
<code>var</code>	Name of the column to check for outliers (string).
<code>other_var</code>	Character vector of additional column names to return alongside the outlier values, or NULL.

Value

A tibble with columns `row_index`, `outlier_value`, and any requested `other_var` columns.

Examples

```
df <- data.frame(val = c(1, 2, 2, 3, 100), id = 1:5)
find_outlier(df, "val", "id")
```

<code>gantt_table</code>	<i>Gantt-style coloured progress table</i>
--------------------------	--

Description

Creates a coloured `flextable` where columns are sampling campaigns and rows are measurement categories by system. Works for both a sampling schedule (cells = dates) and a processing-progress table (cells = sample counts).

Usage

```
gantt_table(
  df,
  colors,
  system_labels = NULL,
  done_cols = NULL,
  na_color = "#BFBFBF",
  notes = NULL
)
```

Arguments

<code>df</code>	Data frame with columns <code>category</code> , <code>system</code> , then one column per campaign. Campaign column names become header labels.
<code>colors</code>	Named character vector mapping system identifiers to hex fill colors, e.g. <code>c(tilapia = "#D8AEFF", prawn = "#F7CAAC")</code> .
<code>system_labels</code>	Named character vector of short display labels for the system column (Col 2). Defaults to <code>names(colors)</code> .
<code>done_cols</code>	Character vector of campaign column names that are completed. Completed cells with values get system fill; others with values get white fill (planned). If <code>NULL</code> (default), all values get fill.
<code>na_color</code>	Hex fill for NA / empty cells. Default grey (" <code>#BFBFBF</code> ") suits sampling schedules. Use " <code>white</code> " for processing-progress tables where NA means "not yet processed".
<code>notes</code>	Optional character string added as a small-font footer.

Value

A `flextable` object.

gantt_text*Plain-text Gantt for terminal and Neovim*

Description

Prints a Unicode box-drawing table to the console. Same data contract as [gantt_table](#).

Run from the project root with: `Rscript -e "source('R/gantt_config.R'); aelab::gantt_text(gantt_sched`

Usage

```
gantt_text(  
  df,  
  colors = NULL,  
  system_labels = NULL,  
  done_cols = NULL,  
  na_color = "#BFBFBF",  
  notes = NULL  
)
```

Arguments

df	Data frame with columns category , system , then one column per campaign. Campaign column names become header labels.
colors	Named character vector mapping system identifiers to hex fill colors, e.g. <code>c(tilapia = "#D8AEFF", prawn = "#F7CAAC")</code> .
system_labels	Named character vector of short display labels for the system column (Col 2). Defaults to <code>names(colors)</code> .
done_cols	Character vector of campaign column names that are completed. Completed cells with values get system fill; others with values get white fill (planned). If <code>NULL</code> (default), all values get fill.
na_color	Hex fill for NA / empty cells. Default grey (" <code>#BFBFBF</code> ") suits sampling schedules. Use " <code>white</code> " for processing-progress tables where NA means "not yet processed".
notes	Optional character string added as a small-font footer.

Value

Invisibly returns `df`.

hobo	<i>Processed data from Onset HOBO Dissolved Oxygen Data Logger. A dataset containing 336 dissolved oxygen concentrations changed over time.</i>
------	---

Description

Processed data from Onset HOBO Dissolved Oxygen Data Logger. A dataset containing 336 dissolved oxygen concentrations changed over time.

Format

A data.frame with 336 rows and 13 variables:

- date_time: Date and time in POSIXct format.
- pressure_hpa: Atmospheric pressure (hpa).
- wind_ms: Wind speed (m/s).
- do: Dissolved oxygen concentrations (mg/L)
- temp: Water temperature (Celsius)
- depth: Water depth (m).
- salinity: Salinity (ppt).
- start_date_time: Start date and time of the deployment.
- end_date_time: End date and time of the deployment.
- sunrise: Sunrise time during that day.
- sunset: Sunset time during that day.
- no_hobo: Name for the data logger .
- site: Name for the site.

Source

own data.

ks_test	<i>ks_test</i>
---------	----------------

Description

Perform Kruskal-Wallis test followed by Dunn post-hoc test (Bonferroni correction) with compact letter display.

Usage

```
ks_test(df, variable_name, group)
```

Arguments

df A data frame.

variable_name Name of the response variable column (string).

group Name of the grouping column (string).

Value

A named list with elements `ks_results`, `dunn_results`, `mean_summary`, and `compact_letters`.

Examples

```
df <- data.frame(
  grp = rep(c("A","B","C"), each = 5),
  val = c(1,2,1,2,1, 3,4,3,4,3, 5,6,5,6,5)
)
ks_test(df, "val", "grp")
```

<code>make_reference</code>	<i>make_reference</i>
-----------------------------	-----------------------

Description

Build a `reference_df` for [calculate_regression](#) from a compact inline string of field notes. Each line is one closure measurement with whitespace-separated columns (any number of spaces or tabs): `time site analyzer [dur=N] [off=N]`.

The first three columns are positional (required). Optional per-row window overrides use `key=value` syntax in any order: `dur=` sets `duration_minutes`, `off=` sets `start_offset_s`. Rows without an override simply omit the token - no NA placeholder needed.

Usage

```
make_reference(date, measurements, output = NULL, tz = "Asia/Taipei")
```

Arguments

date Sampling date applied to all rows. A single "YYYY-MM-DD" string or `Date`. For multi-day campaigns, supply a character/Date vector with one value per row.

measurements A multiline character string, one row per closure: `time site analyzer [dur=N] [off=N]`. `time` accepts "HH:MM" or "HH:MM:SS".

output Optional path for the output `.xlsx` file. If NULL (default), the tibble is returned.

tz Timezone for `date_time`. Default "Asia/Taipei".

Value

A tibble with columns `site`, `date_time`, `analyzer`, and optionally `duration_minutes` and/or `start_offset_s` (only present when at least one row specifies the override). When output is given, the tibble is written to Excel and returned invisibly.

Examples

```
make_reference(
  date = "2026-05-22",
  measurements = "
    10:30:15 JN1-1 1337_1074
    10:42:00 JN1-2 1337_1074
    10:54:30 JN2-1 1337_1074
  "
)

# per-row window overrides - no NA placeholders needed
make_reference(
  date = "2026-05-22",
  measurements = "
    10:30:15 TT3-1 1450 dur=3.5
    10:42:00 TT3-2 1450 dur=2.0
    10:54:30 TT3-3 1450 off=45
    11:10:00 TT1-1 1450
  "
)
```

n2o

Processed data from N2O LI-COR Trace Gas Analyzer. A dataset containing 567 N2O concentrations changed over time.

Description

Processed data from N2O LI-COR Trace Gas Analyzer. A dataset containing 567 N2O concentrations changed over time.

Format

A data.frame with 567 rows and 4 variables:

- DATE: Date in character format.
- TIME: Time in character format.
- N2O: Concentrations of nitrous oxide (N2O), in ppb.
- date_time: Date and time in POSIXct format.

Source

own data.

normality_test_aov	<i>normality_test_aov</i>
--------------------	---------------------------

Description

Test normality of ANOVA model residuals using Shapiro-Wilk on raw, square-root, and log10 transforms (one-way or two-way).

Usage

```
normality_test_aov(df, variable_name, group_1, group_2 = NULL)
```

Arguments

<code>df</code>	A data frame.
<code>variable_name</code>	Name of the response variable column (string).
<code>group_1</code>	Name of the first grouping column (string).
<code>group_2</code>	Name of the second grouping column (string), or <code>NULL</code> for a one-way model.

Value

A tibble with Shapiro-Wilk p-values for each transformation.

Examples

```
df <- data.frame(  
  grp = c("A", "A", "B", "B"),  
  val = c(1.1, 1.4, 3.2, 3.8)  
)  
normality_test_aov(df, "val", "grp")
```

normality_test_t	<i>normality_test_t</i>
------------------	-------------------------

Description

Test normality of a variable within two groups using Shapiro-Wilk on raw, square-root, and log10 transforms (for t-test context).

Usage

```
normality_test_t(df, variable_name, group, group_1, group_2)
```

Arguments

<code>df</code>	A data frame.
<code>variable_name</code>	Name of the numeric variable column (string).
<code>group</code>	<[‘data-masking’][dplyr::dplyr_data_masking]> The grouping column.
<code>group_1</code>	Value identifying the first group.
<code>group_2</code>	Value identifying the second group.

Value

A tibble with Shapiro-Wilk p-values for each group $\times$ transformation combination.

Examples

```
df <- data.frame(
  grp = c("A", "A", "A", "B", "B", "B"),
  val = c(1.1, 2.0, 1.5, 4.2, 3.8, 4.5)
)
normality_test_t(df, "val", grp, "A", "B")
```

plot_bar

plot_bar

Description

Create a bar plot using the aelab theme.

Usage

```
plot_bar(
  df,
  x,
  y,
  z = NULL,
  base_size = 25,
  line_width = 1,
  text_color = "black",
  facet = FALSE,
  facet_x = NULL,
  facet_y = NULL,
  style = "bw",
  position = "dodge",
  stat = "identity"
)
```

Arguments

df	A data frame.
x	<[‘data-masking’][ggplot2::aes]> Column mapped to the x-axis.
y	<[‘data-masking’][ggplot2::aes]> Column mapped to the y-axis.
z	<[‘data-masking’][ggplot2::aes]> Optional column mapped to fill colour.
base_size	Base font size. Default 25.
line_width	Bar outline width. Default 1.
text_color	Text colour. Default "black".
facet	Logical; add facet grid? Default FALSE.
facet_x	Column name (string) for the horizontal facet dimension.
facet_y	Column name (string) for the vertical facet dimension.
style	Theme style. Default "bw".
position	Bar position: "dodge" or "stack". Default "dodge".
stat	Stat type: "identity" or "count". Default "identity".

Value

A ggplot object.

Examples

```
## Not run:
df <- data.frame(x = c("A","B","A","B"), g = c("X","X","Y","Y"), y = c(1,2,3,4))
plot_bar(df, x, y, g)

## End(Not run)
```

plot_box

plot_box

Description

Create a box plot with mean overlay using the aelab theme.

Usage

```
plot_box(
  df,
  x,
  y,
  z = NULL,
  base_size = 25,
  line_width = 0.5,
  outlier_size = 1.5,
```

```

    text_color = "black",
    facet = FALSE,
    facet_x = NULL,
    facet_y = NULL,
    style = "bw"
  )

```

Arguments

<code>df</code>	A data frame.
<code>x</code>	<[‘data-masking’][ggplot2::aes]> Column mapped to the x-axis.
<code>y</code>	<[‘data-masking’][ggplot2::aes]> Column mapped to the y-axis.
<code>z</code>	<[‘data-masking’][ggplot2::aes]> Optional column mapped to fill colour.
<code>base_size</code>	Base font size. Default 25.
<code>line_width</code>	Box outline width. Default 0.5.
<code>outlier_size</code>	Outlier point size. Default 1.5.
<code>text_color</code>	Text colour. Default "black".
<code>facet</code>	Logical; add facet grid? Default FALSE.
<code>facet_x</code>	Column name (string) for the horizontal facet dimension.
<code>facet_y</code>	Column name (string) for the vertical facet dimension.
<code>style</code>	Theme style. Default "bw".

Value

A ggplot object.

Examples

```

## Not run:
df <- data.frame(x = rep(c("A","B"), each = 5), y = c(1:5, 3:7))
plot_box(df, x, y)

## End(Not run)

```

plot_ghg_flux

plot_ghg_flux

Description

Plot GHG concentration over time for each measurement window, overlaid with the fitted regression line and colored by quality flag. Use this for visual QC of [calculate_regression](#) output.

Usage

```
plot_ghg_flux(
  data,
  regression_result,
  ghg,
  fit_type = "linear",
  t_zero = 0,
  flag_colors = c(ok = "#2ca25f", discard = "#fc8d59", zero = "#74c7ef", no_data =
    "#aaaaaa")
)
```

Arguments

<code>data</code>	Analyzer data processed by <code>tidy_ghg_analyzer()</code> and <code>convert_time()</code> .
<code>regression_result</code>	Output tibble from <code>calculate_regression()</code> .
<code>ghg</code>	Column name in <code>data</code> for the gas concentration (e.g. "CH4", "N2O").
<code>fit_type</code>	Model used to draw the fitted line. Should match the <code>fit_type</code> used in <code>calculate_regression()</code> . One of "linear" (default), "exp_tz", "exp_zhao18".
<code>t_zero</code>	Reference time in seconds for "exp_tz" slope. Default 0.
<code>flag_colors</code>	Named character vector mapping flag values to colors.

Value

A `ggplot` object with one facet per site. Each panel shows raw concentration points and the fitted line, with strip text showing slope, R^2 , and flag.

Examples

```
data(n2o)
n2o_converted <- convert_time(n2o)
ref <- data.frame(
  date_time = n2o_converted$real_datetime[1],
  site = "S1", analyzer = "1074"
)
result <- calculate_regression(n2o_converted, ref, "N2O",
  reference_time = "date_time", site = "site",
  analyzer_code = "1074")
plot_ghg_flux(n2o_converted, result, "N2O")
```

plot_hobo	<i>plot_hobo</i>
-----------	------------------

Description

Plot the dissolved oxygen concentration over time series grouped by different data loggers to observe the variations.

Usage

```
plot_hobo(df)
```

Arguments

df Dataframe produced by process_hobo() function.

Value

A plot generated by ggplot2.

Examples

```
data(hobo)
plot_hobo(hobo)
```

plot_line	<i>plot_line</i>
-----------	------------------

Description

Create a line plot using the aelab theme.

Usage

```
plot_line(  
  df,  
  x,  
  y,  
  z = NULL,  
  base_size = 25,  
  line_width = 3,  
  text_color = "black",  
  facet = FALSE,  
  facet_x = NULL,  
  facet_y = NULL,  
  style = "bw"  
)
```

Arguments

<code>df</code>	A data frame.
<code>x</code>	<[‘data-masking’][ggplot2::aes]> Column mapped to the x-axis.
<code>y</code>	<[‘data-masking’][ggplot2::aes]> Column mapped to the y-axis.
<code>z</code>	<[‘data-masking’][ggplot2::aes]> Optional column mapped to colour and group.
<code>base_size</code>	Base font size. Default 25.
<code>line_width</code>	Line width. Default 3.
<code>text_color</code>	Text colour. Default "black".
<code>facet</code>	Logical; add facet grid? Default FALSE.
<code>facet_x</code>	Column name (string) for the horizontal facet dimension.
<code>facet_y</code>	Column name (string) for the vertical facet dimension.
<code>style</code>	Theme style. Default "bw".

Value

A ggplot object.

Examples

```
## Not run:
df <- data.frame(x = 1:6, y = c(1,3,2,5,4,6), g = rep(c("A","B"), 3))
plot_line(df, x, y, g)

## End(Not run)
```

<code>plot_map_taiwan</code>	<i>plot_map_taiwan</i>
------------------------------	------------------------

Description

Plot sampling sites on a map of Taiwan with a north arrow and scale bar.

Usage

```
plot_map_taiwan(
  long,
  lat,
  names,
  color = "darkgrey",
  textsize = 5,
  basesize = 16,
  shape_type = 22
)
```

Arguments

<code>long</code>	Numeric vector of longitudes.
<code>lat</code>	Numeric vector of latitudes.
<code>names</code>	Character vector of site labels (same length as <code>long</code>).
<code>color</code>	Fill colour for site markers. Default "darkgrey".
<code>textsize</code>	Size for annotation and point labels. Default 5.
<code>basesize</code>	Base font size for the map theme. Default 16.
<code>shape_type</code>	ggplot2 point shape number. Default 22 (filled square).

Value

A ggplot object (also printed to the active device).

Examples

```
## Not run:
plot_map_taiwan(
  long = c(120.2, 121.5),
  lat  = c(22.9, 24.1),
  names = c("Site A", "Site B")
)

## End(Not run)
```

plot_point

plot_point

Description

Create a scatter plot using the aelab theme.

Usage

```
plot_point(
  df,
  x,
  y,
  z = NULL,
  base_size = 25,
  point_size = 3,
  stroke_size = 1,
  text_color = "black",
  facet = FALSE,
  facet_x = NULL,
  facet_y = NULL,
  style = "bw"
)
```

Arguments

df	A data frame.
x	<[‘data-masking’][ggplot2::aes]> Column mapped to the x-axis.
y	<[‘data-masking’][ggplot2::aes]> Column mapped to the y-axis.
z	<[‘data-masking’][ggplot2::aes]> Optional column mapped to fill colour.
base_size	Base font size passed to the ggplot2 theme. Default 25.
point_size	Point size. Default 3.
stroke_size	Point stroke width. Default 1.
text_color	Text colour. Default "black".
facet	Logical; add facet grid? Default FALSE.
facet_x	Column name (string) for the horizontal facet dimension.
facet_y	Column name (string) for the vertical facet dimension.
style	Theme style. One of "bw", "minimal", "classic", "graycolor", "light". Default "bw".

Value

A ggplot object.

Examples

```
## Not run:
df <- data.frame(x = 1:5, y = c(2,4,1,5,3), g = c("A","A","B","B","A"))
plot_point(df, x, y, g)

## End(Not run)
```

process_hobo

process_hobo

Description

Tidy data exported from a HOBO data logger. Supports the HOBO U26 Dissolved Oxygen Data Logger (**type** = "do") and the HOBO Pro v2 Temperature/RH Logger (**type** = "temp"). Handles both Chinese (AM/PM Chinese-locale headers) and English (AM/PM or 24-hour) HOBOware locale exports, with 2- or 4-digit years. For **type** = "do", readings with a negative (sensor-error) dissolved-oxygen or temperature value are dropped before aggregation.

Usage

```
process_hobo(file_path, no_hobo, type = c("do", "temp"))
```

Arguments

- `file_path` Path to the CSV file exported from HOBOWare.
- `no_hobo` The code for the data logger.
- `type` Logger type: "do" (default) for HOBO U26 DO logger, or "temp" for HOBO Pro v2 temperature/RH logger.

Value

A dataframe. For `type = "do"`: columns `date_time`, `do`, `temp`, `no_hobo`, aggregated to 30-minute intervals. For `type = "temp"`: columns `date_time`, `air_temp`, `rh`, `no_hobo`, one row per logger reading.

Examples

```
hobo_data_path <- system.file("extdata", "ex_hobo.csv", package = "aelab")
df <- process_hobo(hobo_data_path, "code_for_logger")
```

<code>process_info</code>	<i>process_info</i>
---------------------------	---------------------

Description

Import and process the necessary information, including the sunrise and sunset times of the day, the date and time range of the deployment, and the code for the data logger.

Usage

```
process_info(file_path)
```

Arguments

- `file_path` Directory of file.

Value

A dataframe.

Examples

```
info_data_path <- system.file("extdata", "info.xlsx", package = "aelab")
df <- process_info(info_data_path)
```

process_weather	<i>convert_time</i>
-----------------	---------------------

Description

Tidy the daily weather data downloaded from weather station in Taiwan.

Usage

```
process_weather(file_path, date, zone)
```

Arguments

file_path	Directory of file.
date	Date of the daily weather data in yyyy-mm-dd format.
zone	Code for the region of the weather station.

Value

A dataframe.

Examples

```
weather_data_path <- system.file("extdata", "ex_weather.csv", package = "aelab")
df <- process_weather(weather_data_path, "2024-01-01", "site_A")
```

process_weather_mh	<i>process_weather_mh</i>
--------------------	---------------------------

Description

Import and tidy an hourly weather report in the CWA "MH" (Multifield Hourly) text format, as bulk-downloaded from the CODiS portal (`LotsDataReports.txt`). The file is fixed-/space-delimited with a '#' column-title row listing element codes (e.g. PS01, WD01) and '*' comment rows. One file may contain multiple stations and a full month of hourly data.

Usage

```
process_weather_mh(
  file_path,
  station_id = NULL,
  expand_30min = TRUE,
  tz = "Asia/Taipei"
)
```

Arguments

<code>file_path</code>	Path to the MH-format .txt file.
<code>station_id</code>	Optional character vector of station codes (e.g. "COR660") to keep. Default NULL keeps all stations.
<code>expand_30min</code>	If TRUE (default), each hourly value is expanded to the two 30-minute marks (H-1):30 and H:00, matching <code>process_weather</code> so it merges with 30-minute HOBO data. If FALSE, hourly rows are returned unchanged.
<code>tz</code>	Time zone of the observations. Default "Asia/Taipei".

Details

Element codes are renamed as: PS01 -> `pressure_hpa`, WD01 -> `wind_ms`, TX01 -> `air_temp`, RH01 -> `humidity`, WD02 -> `wind_dir`, PP01 -> `rain_mm`. Only the codes present in the file are returned. CWA missing-value codes (large negatives, e.g. -9991) are set to NA. The timestamp `yyyymmddhh` uses hours 01-24, where hour 24 denotes 00:00 of the following day.

Value

A data frame with columns `stno`, `date_time`, and the renamed weather elements present in the file.

Examples

```
## Not run:
df <- process_weather_mh("LotsDataReports.txt", station_id = "COR660")

## End(Not run)
```

<code>process_weather_month</code>
<i>process_weather_month</i>

Description

Import and tidy a monthly weather CSV file downloaded from a Taiwan Central Weather Administration station. Column selection is done via regex so minor header changes are handled gracefully.

Usage

```
process_weather_month(file_path, month, year = 2024, zone)
```

Arguments

<code>file_path</code>	Path to the monthly CSV file.
<code>month</code>	Month number (1-12) covered by the file.
<code>year</code>	Four-digit year. Default 2024.
<code>zone</code>	Character label for the weather station / region.

Value

A data frame with columns `day`, `pressure_hpa`, `temp`, `humidity_percent`, `wind_ms`, `rain_mm`, `daylight_hr`, `radiation`, `date`, and `zone`.

Examples

```
## Not run:
df <- process_weather_month("path/to/2024-01.csv",
  month = 1, year = 2024,
  zone = "site_A"
)

## End(Not run)
```

`scale_colour_aelab_c` *scale_colour_aelab_c*

Description

Continuous ggplot2 colour scale using an aelab palette.

Usage

```
scale_colour_aelab_c(name, direction = 1)

scale_color_aelab_c(name, direction = 1)
```

Arguments

<code>name</code>	Palette name passed to aelab_palettes .
<code>direction</code>	1 (default) for normal order; -1 to reverse.

Value

A ggplot2 scale.

Examples

```
## Not run:
ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg, colour = mpg)) +
  ggplot2::geom_point() + scale_colour_aelab_c("rainbow")

## End(Not run)
```

`scale_colour_aelab_d` *scale_colour_aelab_d*

Description

Discrete ggplot2 colour scale using an aelab palette.

Usage

```
scale_colour_aelab_d(name, direction = 1)
```

```
scale_color_aelab_d(name, direction = 1)
```

Arguments

<code>name</code>	Palette name passed to aelab_palettes .
<code>direction</code>	1 (default) for normal order; -1 to reverse.

Value

A ggplot2 scale.

Examples

```
## Not run:  
ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg, colour = factor(cyl))) +  
  ggplot2::geom_point() + scale_colour_aelab_d("rainbow")  
  
## End(Not run)
```

`scale_fill_aelab_c` *scale_fill_aelab_c*

Description

Continuous ggplot2 fill scale using an aelab palette.

Usage

```
scale_fill_aelab_c(name, direction = 1)
```

Arguments

<code>name</code>	Palette name passed to aelab_palettes .
<code>direction</code>	1 (default) for normal order; -1 to reverse.

Value

A `ggplot2` scale.

Examples

```
## Not run:
ggplot2::ggplot(mtcars, ggplot2::aes(factor(cyl), fill = mpg)) +
  ggplot2::geom_col() + scale_fill_aelab_c("ghg")

## End(Not run)
```

scale_fill_aelab_d	<i>scale_fill_aelab_d</i>
--------------------	---------------------------

Description

Discrete `ggplot2` fill scale using an aelab palette.

Usage

```
scale_fill_aelab_d(name, direction = 1)
```

Arguments

<code>name</code>	Palette name passed to aelab_palettes .
<code>direction</code>	1 (default) for normal order; -1 to reverse.

Value

A `ggplot2` scale.

Examples

```
## Not run:
ggplot2::ggplot(mtcars, ggplot2::aes(factor(cyl), fill = factor(cyl))) +
  ggplot2::geom_bar() + scale_fill_aelab_d("control")

## End(Not run)
```

sig_labels

sig_labels

Description

Run `ks_test` separately for each level of a faceting variable and return compact letter display (CLD) annotations ready for `geom_text`. Only facets where the Kruskal-Wallis p-value is below `alpha` are included in the output.

Usage

```
sig_labels(
  stat_data,
  variable,
  group,
  by = "year",
  plot_data = NULL,
  alpha = 0.05
)
```

Arguments

<code>stat_data</code>	Data frame of raw observations used for the statistical test. Should already be filtered to the relevant subset (e.g. a single water type).
<code>variable</code>	Name of the response variable column (string).
<code>group</code>	Name of the grouping column (string).
<code>by</code>	Name of the faceting column whose levels are iterated over (string, default "year").
<code>plot_data</code>	Data frame used to compute label y-positions via <code>max(variable)</code> per group. Defaults to <code>stat_data</code> . Pass the aggregated/boxplot data here when it differs from the raw test data.
<code>alpha</code>	Significance threshold; facets with KW p-value $\geq$ <code>alpha</code> are dropped (default 0.05).

Value

A data frame with columns `<group>`, `Letter`, `MonoLetter`, `y_pos`, and `<by>`. Returns an empty data frame when no facet reaches significance.

Examples

```
set.seed(1)
df <- data.frame(
  year = rep(c("2023", "2024"), each = 20),
  grp = rep(c("A", "B", "C", "D"), 10),
  val = c(rnorm(20, mean = rep(c(1,3,2,6), 5)), rnorm(20))
)
sig_labels(df, "val", "grp", by = "year")
```

tidy_ghg_analyzer	<i>tidy_ghg_analyzer</i>
-------------------	--------------------------

Description

Tidy the data downloaded from GHG Analyzer, with optional timestamp correction.

Usage

```
tidy_ghg_analyzer(  
  file_path,  
  gas,  
  analyzer = "licor",  
  day = 0,  
  hr = 0,  
  min = 0,  
  sec = 0  
)
```

Arguments

file_path	Directory of file.
gas	Choose between CO2/CH4 or N2O LI-COR Trace Gas Analyzer, which is "ch4" and "n2o", respectively.
analyzer	The brand of the analyzer which the data was downloaded from.
day	Day offset to correct the analyzer timestamp.
hr	Hour offset to correct the analyzer timestamp.
min	Minute offset to correct the analyzer timestamp.
sec	Second offset to correct the analyzer timestamp.

Value

Return the loaded XLSX file after tidying. If any time offset is non-zero, a **real_datetime** column is added via [convert_time](#).

Examples

```
ghg_data_path <- system.file("extdata", "ch4.xlsx", package = "aelab")  
tidy_ghg_analyzer(ghg_data_path, "ch4")  
tidy_ghg_analyzer(ghg_data_path, "ch4", sec = -30)
```

Index

add_sun_times, 3
aelab_palettes, 4, 39–41
analyzer_precision, 4, 10
aov_test, 5

calc_chla_trichromatic, 6
calculate_do, 3, 6
calculate_ghg_flux, 7
calculate_MDF, 8
calculate_regression, 4, 9, 12, 25, 30
calculate_total_co2e, 13
combine_hobo, 14, 21
combine_weather, 14
combine_weather_mh, 15
combine_weather_month, 16
convert_ghg_unit, 17
convert_time, 18, 43

descriptive_statistic, 19
df_trans, 20

filter_complete_days, 20
find_outlier, 21

ganttable, 22, 23
ganttext, 23

hobo, 24

ks_test, 24, 42

make_reference, 25

n2o, 26
normality_test_aov, 27
normality_test_t, 27

plot_bar, 28
plot_box, 29
plot_ghg_flux, 30
plot_hobo, 32

plot_line, 32
plot_map_taiwan, 33
plot_point, 34
process_hobo, 21, 35
process_info, 36
process_weather, 37, 38
process_weather_mh, 15, 37
process_weather_month, 16, 38

scale_color_aelab_c
 (scale_colour_aelab_c), 39
scale_color_aelab_d
 (scale_colour_aelab_d), 40
scale_colour_aelab_c, 39
scale_colour_aelab_d, 40
scale_fill_aelab_c, 40
scale_fill_aelab_d, 41
sig_labels, 42

tidy_ghg_analyzer, 4, 43