Package 'streamsampler'

May 9, 2026

Title Characterize and Subsample Stream Data

Version 0.1.0

Description Characterize daily stream discharge and water quality data and subsample water quality data. Provide dates, discharge, and water quality measurements and 'streamsampler' can find gaps, get summary statistics, and subsample according to common stream sampling protocols. Stream sampling protocols are described in Lee et al. (2016) <doi:10.1016/j.jhydrol.2016.08.059> and Lee et al. (2019) <doi:10.3133/sir20195084>.

License CC0

URL https://github.com/Kyle-Hurley/streamsampler

BugReports https://github.com/Kyle-Hurley/streamsampler/issues

Encoding UTF-8

RoxygenNote 7.3.3

Suggests dataRetrieval (>= 2.7.16), knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 4.1)

LazyData true

Imports slider (>= 0.3.1), stats

VignetteBuilder knitr

Copyright This software is in the public domain because it contains materials that originally came from the United States Geological Survey, an agency of the United States Department of Interior. For more information, see the official USGS copyright policy at https://www.usgs.gov/information-policies-and-instructions/copyrights-and-credits

NeedsCompilation no

Author Kyle Hurley [aut, cre, cph] (ORCID: <https://orcid.org/0009-0002-6469-3769>), Jeff Chanat [aut] (ORCID: <https://orcid.org/0000-0002-3629-7307>)

Maintainer Kyle Hurley <kp.hurley87@gmail.com>

Repository CRAN

Date/Publication 2025-09-30 08:30:09 UTC

Contents

`eval_dates` *Evaluate completeness of dates*

Description

Evaluate the completeness of a sequence of dates compared to a hypothetically complete record of dates. `eval_dates()` will aggregate the dates by different time periods such as day, week, month, quarter, or year. Intended to be used on a water quality record.

Usage

```
eval_dates(dates, rec_start, rec_end, by = "day")
```

Arguments

`dates`	A vector of dates to be evaluated. Must be of class 'Date'.
`rec_start`	The start date of the recording period. Must be of class 'Date'.
`rec_end`	The end date of the recording period. Must be of class 'Date'.
`by`	A character string specifying the time period for aggregation. One of "day", "week", "month", "quarter", or "year". Default is "day".

Value

A data.frame with one row and two columns:

Name	Type	Description
pct_complete	numeric	Percent coverage of the `dates`
n_miss	integer	Number of missing time periods

Examples

```
dates <- seq.Date(as.Date("2020-01-01"), as.Date("2020-12-31"), by = "day")
rec_start <- as.Date("2020-01-01")
rec_end <- as.Date("2020-12-31")

# Evaluate by day
eval_dates(dates, rec_start, rec_end, by = "day")

# Evaluate by week
eval_dates(dates, rec_start, rec_end, by = "week")

# Evaluate by month
eval_dates(dates, rec_start, rec_end, by = "month")

# Evaluate by quarter
eval_dates(dates, rec_start, rec_end, by = "quarter")

# Evaluate by year
eval_dates(dates, rec_start, rec_end, by = "year")

# Example with missing dates
dates_with_na <- dates
dates_with_na[c(10, 20, 30)] <- NA
eval_dates(dates_with_na, rec_start, rec_end, by = "day")
```

---

eval_sign *Evaluate the sign of data*

Description

Evaluate a numeric vector for the proportions of positive (> 0), negative (<= 0), and NA values. Intended to be used on a water quality record.

Usage

```
eval_sign(values)
```

Arguments

`values`	A numeric vector to be evaluated.

Value

A data frame with one row and seven columns:

Name	Type	Description
n_pos	integer	number of non-negative values in the data
pos_pct	numeric	percentage of non-negative values in the data
n_neg	integer	number of negative values in the data
neg_pct	numeric	percentage of negative values in the data
n_na	integer	number of NA values in the data
na_pct	numeric	percentage of NA values in the data
tot_pct	numeric	sum of `pos_pct`, `neg_pct`, and `na_pct`

Examples

```
data <- c(-1, -2, 0, 1, 2, 3, -3, -4, 4)
eval_sign(data)

data_all_positive <- c(1, 2, 3, 4, 5)
eval_sign(data_all_positive)

data_all_negative <- c(-1, -2, -3, -4, -5)
eval_sign(data_all_negative)

data_mixed <- c(-1, 0, 1, 2, -2, -3, NA)
eval_sign(data_mixed)
```

---

find_gaps *Identify gaps in a sequence of dates*

Description

Identify gaps in a sequence of dates and return a data frame with the number of missing days, start and end dates, and the starting location of the gap in the vector. Intended to be used on a (near) daily water quality record.

Usage

```
find_gaps(dates)
```

Arguments

`dates`	A vector of dates to be evaluated. Must be 'Date' class.

Value

A data frame with the following columns:

Name	Type	Description
n_days	integer	The number of days in the gap
start	date	The start date of the gap
end	date	The end date of the gap
location	integer	The location index of the gap in the original dates vector

Examples

```
dates <- as.Date(c("2020-01-01", "2020-01-03", "2020-01-04", "2020-01-10", "2020-01-15"))
find_gaps(dates)

dates_no_gaps <- seq.Date(as.Date("2020-01-01"), as.Date("2020-01-05"), by = "day")
find_gaps(dates_no_gaps)

dates_with_na <- as.Date(c("2020-01-01", "2020-01-03", NA, "2020-01-10"))
find_gaps(dates_with_na)
```

---

local_x_index *Find indexed local minima/maxima*

Description

Identify local minima or maxima of a vector across a sliding date window. Intended to be used on a (near) daily water quality record.

Usage

```
local_max_index(
  dates,
  values,
  look_behind = 2,
  look_ahead = 2,
  look_units = "days"
)

local_min_index(
  dates,
  values,
  look_behind = 2,
  look_ahead = 2,
  look_units = "days"
)
```

Arguments

dates	A vector of dates of 'Date' class. There are 2 restrictions: • The vector must be in ascending order; duplicates are allowed • The vector cannot have missing values (i.e., no NA)
values	A vector of numeric values. The values must be in correspondence with dates, meaning the *i*th element in values must correspond to the *i*th date in dates.
look_behind, look_ahead	The number of look_units before and after the center date to include in the sliding window to determine local maxima or minima.
look_units	One of "days", "weeks", or "months". The units to give look_ahead and look_behind.

Details

The size of the moving window is adjusted to be shorter at the ends of a record. For example, at the start of a record only the first element of the record plus the values included in the look_ahead are evaluated.

Any object that can be added or subtracted from the dates with + and - can be used for look_behind and look_ahead. By default, these are both 2 days. This creates a 5-day sliding window where the 3rd day is the element evaluated.

If an element in values is NA, then FALSE is returned. If a group of elements within a window is all NA, then FALSE is returned.

Value

A logical vector with the same length as dates and values.

Examples

```
# Works as expected
date_vec <- seq.Date(
  from = as.Date("2020-01-01"),
  to = as.Date("2020-01-06"),
  by = "day"
)
num_vec <- c(10, 11, 50, 9, 8, 100)

results <- local_max_index(
  dates = date_vec,
  values = num_vec
)
print(data.frame(
  "date" = date_vec,
  "value" = num_vec,
  "local_max" = results
))

# Different look_ahead/behind
date_vec <- seq(
```

```
  from = as.Date("2020-01-01"),
  to = as.Date("2020-02-01"),
  by = "day"
)
set.seed(123)
num_vec <- sample(30:300, length(date_vec), replace = TRUE)

results <- local_max_index(
  dates = date_vec,
  values = num_vec,
  look_behind = 1,
  look_ahead = ,
  look_units = "days"
)
print(data.frame(
  "date" = date_vec,
  "value" = num_vec,
  "local_max" = results
))
```

`qw_stats` *Calculate summary statistics*

Description

Evaluate the completeness of a record compared to a hypothetical "ideal" record and simple statistics. Intended to be used on a (near) daily water quality record.

Usage

```
qw_stats(dates, values, rec_start, rec_end, by = "day")
```

Arguments

`dates`	A vector of dates to be evaluated. Must be of class `Date`.
`values`	A numeric vector to be evaluated. Must be of class `numeric`.
`rec_start`	The start date of the recording period. Must be of class `Date`.
`rec_end`	The end date of the recording period. Must be of class `Date`.
`by`	The time interval to use for the evaluation. One of "day", "week", "month", "quarter", or "year". Default is "day".

Value

A data frame with 1 row and 22 columns:

Name	Type	Description
rec_startDate	Date	Start date of ideal record

rec_endDate	Date	End date of ideal record
n_miss_dates	integer	Number of missing time periods (as specified by by) within the recording period.
pct_complete_dates	numeric	Percentage of the time period (as specified by by) covered by the dates
value_startDate	Date	Date of the first values in the record
value_endDate	Date	Date of the last values in the record
n_value	integer	Number of non-NA values
n_pos	integer	Number of non-negative values in the data
pos_pct	numeric	Percentage of non-negative values in the data
n_neg	integer	Number of negative values in the data
neg_pct	numeric	Percentage of negative values in the data
n_na	integer	Number of NA values in the data
na_pct	numeric	Percentage of NA values in the data
tot_pct	numeric	sum of pos_pct, neg_pct, and na_pct
Min.	numeric	Minimum of values
Q1	numeric	25th percentile of values
Median	numeric	Median (50th percentile) of values
Mean	numeric	Mean of values
Q3	numeric	75th percentile of values
Max.	numeric	Maximum of values
std	numeric	Standard deviation of values
vari	numeric	Variance of values

Examples

```
data <- data.frame(
 date_col = as.Date(c("2020-01-01", "2020-01-02", "2020-01-03", "2020-01-04", "2020-01-05")),
  var_col = c(1.2, 2.3, 3.1, NA, 4.5)
)
rec_start <- as.Date("2020-01-01")
rec_end <- as.Date("2020-01-05")
qw_stats(data$date_col, data$var_col, rec_start, rec_end, by = "day")

data_no_missing <- data.frame(
 date_col = as.Date(c("2020-01-01", "2020-01-02", "2020-01-03", "2020-01-04", "2020-01-05")),
  var_col = c(1.2, 2.3, 3.1, 4.5, 5.6)
)
qw_stats(data_no_missing$date_col, data_no_missing$var_col, rec_start, rec_end, by = "day")

data_weekly <- data.frame(
 date_col = as.Date(c("2020-01-01", "2020-01-08", "2020-01-15", "2020-01-22", "2020-01-29")),
  var_col = c(1.2, 2.3, 3.1, 4.5, 5.6)
)
qw_stats(data_weekly$date_col, data_weekly$var_col, rec_start, rec_end, by = "week")
```

---

rollmean_date *Rolling mean of values indexed by date*

Description

Calculate the rolling mean of a set of numbers indexed by date.

Usage

```
rollmean_date(
  dates,
  values,
  look_behind = 2,
  look_ahead = 0,
  look_units = "days"
)
```

Arguments

dates	A vector of dates of 'Date' class.
values	Numeric values. The values must be in correspondence with dates, meaning the *i*th element in values must correspond to the *i*th date in dates.
look_behind, look_ahead	The number of look_units before and after the center date to include in the sliding window.
look_units	One of "days", "weeks", or "months". The units to give look_ahead and look_behind.

Details

The amount of time to include in the rolling mean is defined by look_behind and look_ahead. All values within the look-behind, the index, and the look-ahead will be used to calculate the average. If look_behind is 5 days and look_ahead is 0 days, then all values within the 5 days before the current index (i.e., the 6th day) will be included - it is a 6-day moving average. If look_behind is 5 days and look_ahead is 5 days, then it is a 11-day moving average centered on the 6th day.

The mean is calculated without NA values. If one value in the window is NA, then the rolling mean is that of all values within the window which are not NA.

Value

A numeric vector with length equal to dates and values.

Examples

```
date_vec <- seq.Date(
  from = as.Date("2020-01-01"),
  to = as.Date("2020-01-05"),
  by = "day"
)
num_vec <- c(2, 3, 4, 20, 10)
results <- rollmean_date(
  dates = date_vec,
  values = num_vec,
  look_behind = 2
```

```
 )
data.frame(
  "date" = date_vec,
  "values" = num_vec,
  "roll_mean" = results
)

# Missing data and 10-day moving window
date_vec <- seq.Date(
  from = as.Date("2020-01-01"),
  to = as.Date("2020-03-31"),
  by = "day"
)
set.seed(123)
num_vec <- sample(1:30, length(date_vec), replace = TRUE)
set.seed(123)
r <- sample(1:length(date_vec), 20)
results <- rollmean_date(
  dates = date_vec[-r],
  values = num_vec[-r],
  look_behind = 5,
  look_ahead = 4
 )
head(
  data.frame(
    "date" = date_vec[-r],
    "values" = num_vec[-r],
    "roll_mean" = results
  ),
  10
)
```

`streamdat` *Stream Discharge and Specific Conductivity*

Description

Approximately 15 years of daily average discharge and specific conductivity of the Brandywine Creek at Wilmington, DE - stream gage number 01481500.

Usage

```
streamdat
```

Format

A data frame with 5,844 rows and 3 columns:

date Date of measurement

q Discharge in cubic feet per second

sc Specific conductivity in microsiemens per centimeter at 25 degrees Celsius

Source

https://waterdata.usgs.gov/monitoring-location/01481500/#parameterCode=00060&period=P7D&showMedian=false

---

`subsample` *Convert a daily record to discrete*

Description

Sample a daily or near-daily data set to one containing infrequent but periodic records based on a random sampling protocol. Intended to be used on a (near) daily water quality record.

Usage

```
subsample(
  dates,
  values,
  thresh_ref,
  season_start = 10,
  n_seasons = 4,
  half_win = 2,
  threshold = 0.8,
  n_samples = 1,
  freq = "month",
  n_et_samples = 8,
  season_weights = rep(1, n_seasons),
  target = "none",
  look_behind = 2,
  look_ahead = 2,
  look_units = "days",
  seed = 123
)
```

Arguments

`dates`	A vector of dates of 'Date' class.
`values`	Numeric values. The values must be in correspondence with `dates`, meaning the *i*th element in `values` must correspond to the *i*th date in `dates`.
`thresh_ref`	Numeric values to calculate a threshold. See 'Details'. The values must be in correspondence with `dates`, meaning the `i`th element in `thresh_ref` must correspond to the `i`th date in `dates`.

season_start	The starting month of the first season, specified as an integer from 1 to 12. Default is 10 (October).
n_seasons	The number of seasons in a year. Must be a factor of 12. Default is 4.
half_win	The half width of the window of years to group thresh_ref by. See 'Details'.
threshold	The quantile of thresh_ref above which values is sampled n_et_samples times per year.
n_samples	Integer of the number of below-threshold samples to be selected from values at a frequency defined by freq.
freq	Character of the frequency at which below-threshold samples are selected. May be "week", "month", "quarter", or "year". See 'Details'.
n_et_samples	Integer of the number of yearly exceeds-threshold samples to be selected from values. See 'Details'.
season_weights	A vector of integers of the weights to assign to seasons for random sampling of exceeds-threshold values. Based on the rank of the seasonal average thresh_ref (from highest to lowest). Must have length equal to the number of seasons (n_seasons). See 'Details'.
target	One of "none" or "peaks". See 'Details'.
look_behind, look_ahead	When target is "peaks", the number of look_units before and after the center date to include in the sliding window to determine local maxima.
look_units	One of "days", "weeks", or "months". The units to give look_ahead and look_behind.
seed	An integer which determines the state for the random number generator. Ensures random sampling is reproducible.

Details

values are randomly selected based on a provided sampling protocol using dates as an index and thresh_ref as a classifier. Elements in values equal to or less than their seasonal threshold are randomly sampled according the protocol set by n_samples and freq. n_et_samples elements in values greater than the threshold are randomly sampled for each year in values. This results in n_samples of below-threshold values for each unique freq and n_et_samples of exceeds-threshold values for each unique year.

Elements in values and thresh_ref must correspond with their respective values in dates.

subsample() is psuedo-random across time in that values are selected randomly in rolling chunks of time determined by freq. If, for example, freq is "week" and n_samples is 1, then the result will be 1 randomly selected below-threshold, non-NA value for each week. However, the selected values could be very close in time (e.g., Saturday and Sunday).

Thresholds are calculated based on groupings of seasonally adjusted years, accounting for seasons split across years. For example, if n_seasons = 4 and season_start = 12, then season 1 includes December of e.g. 2020, January 2021, and February 2021. The year is considered to begin in December and is designated by the year in which it ends (i.e., the seasonally adjusted year); 2021 in this example. If half_win is 2, the default, then a total of 5 years is used to calculate the threshold. For example, when half_win is 2, the threshold for season 1 of 2021 is the quantile defined by threshold of all season 1 values in 2019, 2020, 2021, 2022, and 2023.

The selection of exceeds-threshold values is always across an entire year with no guarantee of timing between selected values. Setting `threshold` to values near 1 would result in a smaller sample pool since there would conceivably be fewer values above 0.9 than 0.8 - thus increasing the likelihood of selected exceeds-threshold values being "far" apart in time.

Both `n_samples` and `n_et_samples` are adjusted lower when the number of unique `dates` in the defined `freq` is less than the number of unique dates in a complete `freq`. This adjustment is calculated by multiplying the number of unique dates in the given `freq` and the number of `*_samples`, dividing the number of dates in the complete `freq`, and then rounding to the nearest whole number. For example, when `n_samples` is 2 and `freq` is "week" but only 1 unique sample date exists for a given week, then `n_samples` is adjusted to 1 ((1 * 2) / 7) –> 1).

`season_weights` influences the random sampling of exceeds-threshold values by weighting the values according to the rank of the seasonal average of `thresh_ref` for the respective adjusted year. For example, if `n_seasons` is 2 and `season_weights` is c(2, 1) then each season with the highest seasonal average of `thresh_ref` values is given a weight of 2 and each season with the lowest is given a weight of 1 - making the exceeds-threshold `values` occurring in the highest ranking seasons more likely to be selected than if the weight was 1. See the details for the `prob` argument in `sample()` for more information.

When `target` is "none", the random selection of exceeds-threshold values is influenced only by `season_weights`. When "peaks", the weights are doubled for values corresponding to local maxima, exceeds-threshold `thresh_ref` values.

Value

A data.frame with the following columns:

Name	Type	Description
date	Date	Date
adj_year	integer	Adjusted year
season	integer	Season number '1':n_seasons
value	numeric	Input `values`
thresh_ref	numeric	Input `thresh_ref` values
threshold	numeric	Seasonal `threshold` quantile of `thresh_ref`
is_peak	logical	TRUE when `thresh_ref` value is local maximum. Only when `target` is "peaks". TRUE/FALSE
selection_type	character	Type of randomly selected value "not_selected" (a record not sampled), "below_threshold" (samp
weight	integer	Weight assigned to the value
ys_rank	integer	Unique year-season rank of the seasonal average `thresh_ref`

Examples

```
# Randomly sample using defaults
df <- subsample(
  dates = streamdat$date, values = streamdat$sc, thresh_ref = streamdat$q,
)
# Plotting function
create_plot <- function(df, log_xfrm = "x", xlab, ylab) {

  plot(
    df$thresh_ref[df$selection_type == "not_selected"],
```

```
    df$value[df$selection_type == "not_selected"],
    pch = 21, col = "gray",
    xlab = xlab, ylab = ylab,
    main = "Subsampled Daily Data",
    log = log_xfrm
  )
  points(
    df$thresh_ref[df$selection_type != "not_selected"],
    df$value[df$selection_type != "not_selected"],
    pch = 16, cex = 1.5,
    col = c(
      "below_threshold" = "#42047E",
      "exceeds_threshold" = "#07A49E"
    )[df$selection_type[df$selection_type != "not_selected"]]
  )
  legend(
    "topright",
    legend = c("Not Sampled", "Below Threshold", "Exceeds Threshold"),
    col = c("gray", "#42047E", "#07A49E"),
    pch = c(21, 16, 16),
    bty = "n"
  )
}
# Plot
create_plot(df, "x", "Discharge (CFS)", "Specific Conductivity (uS/cm)")


df <- subsample(
  dates = streamdat$date, values = streamdat$sc, thresh_ref = streamdat$q,
  target = "peaks",
  season_weights = c(2, 1, 1, 1) # default is four seasons
)
create_plot(df, "x", "Discharge (CFS)", "Specific Conductivity (uS/cm)")

df <- subsample(
  dates = streamdat$date, values = streamdat$sc, thresh_ref = streamdat$sc,
  target = "peaks",
  n_samples = 1, freq = "week",
  n_et_samples = 12,
  half_win = 3
)
df <- merge(streamdat, df, by.x = c("date", "sc"), by.y = c("date", "value"))
df <- df[, !colnames(df) %in% c("thresh_ref")]
colnames(df)[c(2, 3)] <- c("value", "thresh_ref")
create_plot(df, "x", "Discharge (CFS)", "Specific Conductivity (uS/cm)")
```

`subsample_routine` *Convert a daily record to periodic discrete*

Description

#' Subsample a daily or near-daily data set to one containing infrequent but regularly recurring records based on a specified frequency. Intended to be used on a (near) daily water quality record.

Usage

```
subsample_routine(dates, values, day = 15, freq = "month")
```

Arguments

dates	A vector of dates of 'Date' class.
values	Numeric values. The values must be in correspondence with dates, meaning the *i*th element in values must correspond to the *i*th date in dates.
day	An integer specifying the day of the specified frequency. See 'Details'.
freq	A character string indicating the frequency of selection. Must be one of "day", "week", or "month". See 'Details'.

Value

A data.frame with the following columns:

Name	Type	Description
date	Date	Date
value	numeric	Input values
selection_type	character	Type of randomly selected value "not_selected" (an observation not selected), "routine" (selected

Examples

```
create_plot <- function(df, log_xfrm = "x", xlab, ylab, subtitle) {
  plot(
    df$q[df$selection_type == "not_selected"],
    df$value[df$selection_type == "not_selected"],
    pch = 21, col = "gray",
    xlab = xlab, ylab = ylab,
    main = paste0("Subsampled Daily Data\n", subtitle),
    log = log_xfrm
  )
  points(
    df$q[df$selection_type != "not_selected"],
    df$value[df$selection_type != "not_selected"],
    pch = 16, cex = 1.5,
    col = c(
      "routine" = "#42047E"
    )[df$selection_type[df$selection_type != "not_selected"]]
  )
  legend(
    "topright",
    legend = c("Not Selected", "Routine"),
    col = c("gray", "#42047E"),
```

```
    pch = c(21, 16),
    bty = "n"
  )
}

# 15th of each month
sroutine <- subsample_routine(
  dates = streamdat$date, values = streamdat$sc,
  day = 15, freq = "month"
)
df <- merge(streamdat[, -3], sroutine)
create_plot(
  df, "x", "Discharge (CFS)",
  "Specific Conductivity (uS/cm)", "Subsampled on 15th of each month"
)

# Every Wednesday
sroutine <- subsample_routine(
  dates = streamdat$date, values = streamdat$sc,
  day = 4, freq = "week"
)
df <- merge(streamdat[, -3], sroutine)
create_plot(
  df, "x", "Discharge (CFS)",
  "Specific Conductivity (uS/cm)", "Subsampled on every Wednesday"
)

# Every 60th day
sroutine <- subsample_routine(
  dates = streamdat$date, values = streamdat$sc,
  day = 60, freq = "day"
)
df <- merge(streamdat[, -3], sroutine)
create_plot(
  df, "x", "Discharge (CFS)",
  "Specific Conductivity (uS/cm)", "Subsampled every 60th day"
)
```

`summarize_seasons` *Seasonal and monthly value ranking*

Description

Summarize data by calculating average values for each month and season, and the year-season rank. Allows for flexible season definitions by specifying a season start month and the number of seasons. Intended to be used on a (near) daily water quality record.

Usage

```
summarize_seasons(dates, values, season_start = 10, n_seasons = 4)
```

Arguments

dates	A vector of dates of 'Date' class.
values	Numeric values. The values must be in correspondence with dates, meaning the *i*th element in values must correspond to the *i*th date in dates.
season_start	The starting month of the first season, specified as an integer from 1 to 12. Default is 10 (October).
n_seasons	The number of seasons in a year. Must be a factor of 12. Default is 4.

Details

The start of a season, season_start, may be any integer from 1 to 12, indicating the month which is the start of the first season. For example season_start = 1 makes the first season start in January while season_start = 10 makes the season start in October.

The seasonal average accounts for seasons split across years. For example, if n_seasons = 4 and season_start = 12, then season 1 includes December of e.g. 2020, January 2021, and February 2021. The year is considered to begin in December and is designated by the year in which it ends (i.e., the seasonally adjusted year). Thus, the seasonal average for season 1 of 2021 would be calculated from December 2020 to February 2021.

Value

A list with two data frames:

Name	Type	Description
year	numeric	Year
month	numeric	Month
avg_value	numeric	Mean of the values for that year and month

monthly

Name	Type	Description
adj_year	numeric	Adjusted year based on season_start
season	integer	Season number; 1 being the first season of the year
avg_value	numeric	Mean of the values for that 'adj_year' and 'season'
ys_rank	numeric	Rank of the 'avg_value' for that 'adj_year' and 'season'; 1 being the highest 'avg_value'

seasonal

Examples

```
# Four seasons starting in October
date_vec <- seq.Date(
  from = as.Date("2020-05-03"),
  to = as.Date("2023-10-17"),
  by = "day"
```

```
)
set.seed(123)
num_vec <- sample(30:3000, length(date_vec), replace = TRUE)
# Four seasons starting in October
results <- summarize_seasons(
  dates = date_vec,
  values = num_vec,
  season_start = 10,
  n_seasons = 4
)
print(head(results$monthly))
print(head(results$seasonal))
# Three seasons starting in January
results <- summarize_seasons(
  dates = date_vec,
  values = num_vec,
  season_start = 1,
  n_seasons = 3
)
print(head(results$monthly))
print(head(results$seasonal))
```

`thresholds` *Find thresholds for each season*

Description

Calculate the threshold for each unique season in a window of years. Values exceeding these thresholds would be considered "high" based on the specified quantile. Intended to be used on a (near) daily water quality record.

Usage

```
thresholds(
  dates,
  values,
  season_start = 10,
  n_seasons = 4,
  half_win = 2,
  threshold = 0.8
)
```

Arguments

`dates`	A vector of dates of 'Date' class.
`values`	Numeric values. The values must be in correspondence with `dates`, meaning the *i*th element in `values` must correspond to the *i*th date in `dates`.

season_start	The starting month of the first season, specified as an integer from 1 to 12. Default is 10 (October).
n_seasons	The number of seasons in a year. Must be a factor of 12. Default is 4.
half_win	The half width of the window of years to group thresh_ref by. See 'Details'.
threshold	The quantile of thresh_ref above which values is sampled n_et_samples times per year.

Details

Thresholds are calculated based on groupings of seasonally adjusted years. If half_win is 2, the default, then a total of 5 years is used to calculate the threshold. Years are adjusted such that the year starts on the first of the month of the first season, as determined by season_start, and end one year later, and is designated by the calendar year in which it ends. For example, when half_win is 2, the threshold for season 1 of 2021 is the quantile defined by threshold of all season 1 values in 2019, 2020, 2021, 2022, and 2023.

Value

A data frame with three columns:

Name	Type	Description
season	integer	Season of the threshold
threshold	numeric	Seasonal threshold
center_year	integer	Middle year of the window

Examples

```
date_vec <- seq.Date(
  from = as.Date("2020-05-03"),
  to = as.Date("2023-10-17"),
  by = "day"
)
set.seed(123)
q_vec <- stats::runif(length(date_vec), min = -50, max = 150)
df <- data.frame("date" = date_vec, "q" = q_vec)

results <- thresholds(
  dates = date_vec,
  values = q_vec
)
print(head(results))

# Define seasons differently
results <- thresholds(
  dates = date_vec,
  values = q_vec,
  season_start = 1,
  n_seasons = 3,
  half_win = 2,
```

```
  threshold = 0.9
)
print(head(results))
```

Index