Package 'walkboutr'

May 8, 2026

Title Generate Walk Bouts from GPS and Accelerometry Data

Version 0.6.0

Description Process GPS and accelerometry data to generate walk bouts. A walk bout is a period of activity with accelerometer movement matching the patterns of walking with corresponding GPS measurements that confirm travel. The inputs of the 'walkboutr' package are individual-level accelerometry and GPS data. The outputs of the model are walk bouts with corresponding times, duration, and summary statistics on the sample population, which collapse all personally identifying information. These bouts can be used to measure walking both as an outcome of a change to the built environment or as a predictor of health outcomes such as a cardioprotective behavior. Kang B, Moudon AV, Hurvitz PM, Saelens BE (2017) <doi:10.1016/j.trd.2017.09.026>.

License BSD_3_clause + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

URL https://github.com/rwalkbout/walkboutr, https://rwalkbout.github.io/walkboutr/

BugReports https://github.com/rwalkbout/walkboutr/issues

Config/testthat/edition 3

Imports data.table, dplyr, geosphere, ggforce, ggplot2, lubridate, lwgeom, magrittr, measurements, sf, sp, stats, tidyr

Depends R (>= 4.3.0)

Suggests knitr, rmarkdown, testthat, tinytest, utils

VignetteBuilder knitr

NeedsCompilation no

Author Lauren Blair Wilner [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-4439-3734>), Stephen J Mooney [aut]

Maintainer Lauren Blair Wilner <wilnerl@uw.edu>

Repository CRAN

Date/Publication 2024-01-30 00:10:02 UTC

Contents

add_date_and_format *Add date and format to activity counts*

Description

This function takes a data frame of activity counts and adds a column of time stamps in POSIXct format. The time stamps start at "2012-04-07 00:00:30" and increase by 30 seconds for each row of the data frame.

Usage

```
add_date_and_format(counts)
```

Arguments

counts	a data frame containing activity counts

Value

a data frame with time stamps added in POSIXct format

assign_epoch_start_time *Assign Epoch Start Time*

Description

Assign Epoch Start Time

Usage

```
assign_epoch_start_time(gps_data, epoch_length)
```

Arguments

gps_data	A data frame with GPS data including a column of timestamps and columns for latitude and longitude
epoch_length	The duration of an epoch in seconds

Details

Selects the closest 30 second increment to assign epoch start time and takes the GPS coordinates associated with the latest time if there are multiple GPS data points in a given 30 second increment. This function returns a data frame of GPS data with a column of epoch times.

Value

A data frame of GPS data with an additional column indicating epoch start time

collate_arguments *Collate Arguments This function collates user-provided arguments with pre-defined parameters and constants.*

Description

Collate Arguments This function collates user-provided arguments with pre-defined parameters and constants.

Usage

```
collate_arguments(..., collated_arguments = NULL)
```

Arguments

`...`	named arguments passed by the user
`collated_arguments`	NULL or previously collated arguments

Value

A list of all arguments, including both pre-defined parameters and constants and any user-provided arguments.

constants *List of Constants* `non_wearing_min_threshold_epochs` *Number of consecutive epochs with activity counts of 0 that constitute a non_wearing period.* `min_wearing_hours_per_day` *Minimum number of hours in a day an individual must wear an accelerometer for the day to be considered complete.* `min_gps_obs_within_bout` *Minimum number of GPS observations within a bout for that bout to be considered to have complete GPS data.* `min_gps_coverage_ratio` *Minimum ratio of data points with versus without GPS data for the bout to be considered to have complete GPS data.* `dwellbout_radii_quantile` *Threshold for outliering GPS data points - any data points above the 95th percentile are outliered.* `max_dwellbout_radii_ft` *Maximum radius, in feet, of a bounding circle that would be considered a dwell bout (rather than a potential walk bout).* `min_dwellbout_obs` *Minimum number of observations to consider something a potential dwell bout.* `max_walking_cpe` *Maxiumum CPE value before the accelerometer is considered to be picking up on an activity other than walking.* `min_walking_speed_km_h` *Minimum speed considered walking.* `max_walking_speed_km_h` *Maximum speed considered walking.*

Description

List of Constants non_wearing_min_threshold_epochs Number of consecutive epochs with activity counts of 0 that constitute a non_wearing period. min_wearing_hours_per_day Minimum number of hours in a day an individual must wear an accelerometer for the day to be considered complete. min_gps_obs_within_bout Minimum number of GPS observations within a bout for that bout to be considered to have complete GPS data. min_gps_coverage_ratio Minimum ratio of data points with versus without GPS data for the bout to be considered to have complete GPS data. dwellbout_radii_quantile Threshold for outliering GPS data points - any data points above the 95th percentile are outliered. max_dwellbout_radii_ft Maximum radius, in feet, of a bounding circle that would be considered a dwell bout (rather than a potential walk bout). min_dwellbout_obs Minimum number of observations to consider something a potential dwell bout. max_walking_cpe Maxiumum CPE value before the accelerometer is considered to be picking up on an activity other than walking. min_walking_speed_km_h Minimum speed considered walking. max_walking_speed_km_h Maximum speed considered walking.

Usage

```
constants
```

Format

An object of class list of length 10.

---

evaluate_gps_completeness

Evaluate GPS completeness for each walking bout

Description

This function evaluates the completeness of GPS data for each walking bout. For each bout, it checks if the number of valid GPS records (with speed, latitude, and longitude data) is greater than a specified threshold, and if the ratio of valid GPS records to total records is greater than a specified minimum. If both of these conditions are met, the function considers the GPS data for the bout to be complete. The function also calculates the median speed for each bout.

Usage

```
evaluate_gps_completeness(
  walk_bouts,
  min_gps_obs_within_bout,
  min_gps_coverage_ratio
)
```

Arguments

walk_bouts	A data frame containing information about walking bouts, including GPS data.
min_gps_obs_within_bout	The minimum number of GPS observations required for a bout to be considered to have complete GPS data.
min_gps_coverage_ratio	The minimum ratio of GPS observations with valid data to total GPS observations for a bout to be considered to have complete GPS data.

Value

A data frame containing information about the GPS completeness and median speed for each bout.

---

generate_bout_category *Generate bout categories*

Description

Given accelerometer bout data, this function generates bout categories, which includes dwell bouts, non-walk bouts that are either too slow, too fast, or too vigorous, and bouts with an unknown lack of GPS data.

Usage

```
generate_bout_category(
  walk_bouts,
  bout_radii,
  gps_completeness,
  max_dwellbout_radii_ft,
  max_walking_cpe,
  min_walking_speed_km_h,
  max_walking_speed_km_h
)
```

Arguments

walk_bouts	a data frame that contains bout information for walking bouts.
bout_radii	a data frame that contains bout radii information.
gps_completeness	a data frame that contains GPS data completeness information.
max_dwellbout_radii_ft	a numeric scalar that specifies the maximum radius, in feet, of a bounding circle that would be considered a dwell bout.

`max_walking_cpe`
: a numeric scalar that specifies the maximum activity counts per epoch value before the accelerometer is considered to be picking up on an activity other than walking.

`min_walking_speed_km_h`
: a numeric scalar that specifies the minimum speed considered walking.

`max_walking_speed_km_h`
: a numeric scalar that specifies the maximum speed considered walking.

Details

The function uses the bout information for walking bouts, bout radii information, and GPS data completeness information to generate the bout categories.

The function first generates dwell bouts by joining the bout radii information and GPS data completeness information on the bout column, and then filters out the rows that have bout values that are missing using the filter function. Then, it calculates the dwell bout values as TRUE if the complete_gps column is TRUE and the bout_radius column is less than max_dwellbout_radii_ft. The resulting data frame only contains the bout and dwell_bout columns. The function then joins the resulting data frame with the walking bout data frame using the bout column. Then, for the non-walk bouts, the function calculates whether they are too vigorous, too slow, or too fast. For the non-walk bouts that are too vigorous, the function calculates the mean activity_counts for each bout, and then sets the non_walk_too_vigorous value as TRUE if the mean activity_counts value is greater than max_walking_cpe. For the non-walk bouts that are too slow or too fast, the function calculates the median speed for each bout, and then sets the non_walk_slow or non_walk_fast value as TRUE if the median speed value is less than min_walking_speed_km_h or greater than max_walking_speed_km_h, respectively. Finally, the function generates a non_walk_incomplete_gps value as TRUE if the complete_gps value is FALSE for the bout. The resulting data frame contains the following columns: bout, dwell_bout (T/F), non_walk_too_vigorous (T/F), non_walk_slow (T/F), non_walk_fast (T/F), non_walk_incomplete_gps (T/F).

Value

a data frame with the following columns: bout, dwell_bout (T/F), non_walk_too_vigorous (T/F), non_walk_slow (T/F), non_walk_fast (T/F), non_walk_incomplete_gps (T/F)

---

generate_bout_plot *Generate Bout Plot*

Description

This function generates a plot of accelerometry counts and GPS radius for a specific bout.

Usage

```
generate_bout_plot(
  accelerometry_counts,
  gps_data,
  bout_number,
  leading_minutes = 8,
  trailing_minutes = 12,
  gps_target_size = 0.25,
  ...,
  collated_arguments = NULL
)
```

Arguments

accelerometry_counts	A data frame or tibble containing accelerometry counts.
gps_data	A data frame or tibble containing GPS data.
bout_number	The number of the bout to be plotted.
leading_minutes	number of minutes before a bout starts that we want to plot
trailing_minutes	number of minutes after a bout ends that we want to plot
gps_target_size	proportional size of circle plot
...	Additional arguments to be passed to the function
collated_arguments	A list of collated arguments

Value

A ggplot object representing the bout plot.

---

generate_bout_radius *Generate Bounding Circle Radius for Walking Bouts*

Description

This function generates a bounding circle radius for each walking bout identified in the input data. The bounding circle is defined as the smallest circle that fully contains all GPS locations observed during a walking bout.

Usage

```
generate_bout_radius(walk_bouts, dwellbout_radii_quantile)
```

Arguments

walk_bouts	A data frame containing GPS locations for each walking bout, with columns "longitude", "latitude", and "bout" (a unique identifier for each bout)
dwellbout_radii_quantile	A quantile (between 0 and 1) used to filter outlying GPS data points before generating the bounding circle. GPS points with a distance from the center greater than the radius of the circle that contains (1 - dwellbout_radii_quantile) of the GPS points are considered outliers and are excluded.

Value

A data frame containing the bout identifier and the radius of the bounding circle for each walking bout.

---

generate_gps_data *Generate a dataset with date-time, speed, and latitude and longitude of someone moving through space on a walk in Seattle*

Description

Generate a dataset with date-time, speed, and latitude and longitude of someone moving through space on a walk in Seattle

Usage

```
generate_gps_data(
  start_lat,
  start_long,
  start_time,
  n_epochs = 110,
  time_interval = 30,
  seed = 1234
)
```

Arguments

start_lat	The starting latitude of the walk.
start_long	The starting longitude of the walk.
start_time	The start time of a series of data
n_epochs	The number of epochs in the series
time_interval	The time interval between points in seconds.
seed	random seed

Value

A data frame with four columns: "timestamp", "lat", "lon", and "speed".

generate_walking_in_seattle_gps_data

Generate GPS data for a walking activity in Seattle, WA

Description

This function generates a data frame containing GPS data for a walking activity in Seattle, WA on April 7th, 2012. It calls the function generate_gps_data to create a series of GPS locations and speeds. The resulting data frame has columns for time, latitude, longitude, and speed.

Usage

```
generate_walking_in_seattle_gps_data(start_lat, start_long, start_time)
```

Arguments

start_lat	The starting latitude of the walk.
start_long	The starting longitude of the walk.
start_time	The start time of a series of data

Value

A data frame with columns time, latitude, longitude, speed

identify_bouts

Identify Bouts:

Description

Identify Bouts:

Usage

```
identify_bouts(
  accelerometry_counts,
  maximum_number_consec_inactive_epochs_in_bout,
  active_counts_per_epoch_min,
  minimum_bout_length
)
```

Arguments

`accelerometry_counts`
: A data frame containing accelerometry counts and times

`maximum_number_consec_inactive_epochs_in_bout`
: Maximum number of consecutive inactive epochs in a bout without ending the bout

`active_counts_per_epoch_min`
: Minimum accelerometer counts for an epoch to be considered active (vs. inactive)

`minimum_bout_length`
: Minimum number of epochs for a period of activity to be considered as a potential bout

Details

This function partitions the accelerometry data into bouts of activity and non-bouts by first identifying all epochs that are definitely not part of bouts. Then, it uses run length encoding to partition the data into potential bouts and non-bouts, and labels each potential bout as a bout or non-bout based on whether it meets the criteria for bout length and the number of consecutive inactive epochs allowed. Finally, the function adds a new column to the input data frame `accelerometry_counts` named `bout` that indicates whether each epoch is part of a bout (1) or not (0).

Value

A data frame with the same columns as the input data frame `accelerometry_counts`, but with a new column named `bout` that indicates whether each epoch is part of a bout (in which case it gets a bout number assigned) or not (0)

`identify_complete_days`

Identify complete wearing days This function identifies complete days based on accelerometry data by calculating the total number of epochs worn per day and comparing it to the minimum number of wearing epochs per day required to consider a day complete.

Description

Identify complete wearing days This function identifies complete days based on accelerometry data by calculating the total number of epochs worn per day and comparing it to the minimum number of wearing epochs per day required to consider a day complete.

Usage

```
identify_complete_days(
  accelerometry_counts,
  min_wearing_hours_per_day,
```

```
  epoch_length,
  local_time_zone
)
```

Arguments

accelerometry_counts	A data frame containing accelerometry counts and non-wearing epochs.
min_wearing_hours_per_day	Minimum number of hours of wearing time required for a day to be considered complete.
epoch_length	The duration of an epoch in seconds.
local_time_zone	The local time zone of the data. The data come in and are returned in UTC, but the local time zone is used to compute complete_days.

Value

A data frame containing accelerometer counts, non-wearing epochs, and a binary variable indicating if the day is complete or not.

---

identify_non_wearing_periods

Identify non-wearing periods: This function identifies non-wearing periods in accelerometry data based on a threshold of consecutive epochs with activity counts of 0.

Description

Identify non-wearing periods: This function identifies non-wearing periods in accelerometry data based on a threshold of consecutive epochs with activity counts of 0.

Usage

```
identify_non_wearing_periods(
  accelerometry_counts,
  non_wearing_min_threshold_epochs
)
```

Arguments

accelerometry_counts	a data frame containing columns for time (in POSIXct format) and activity_counts
non_wearing_min_threshold_epochs	an integer value indicating the minimum number of consecutive epochs with 0 activity counts that constitute a non-wearing period

Details

Identify periods where the accelerometer is not being worn based on the activity counts and a minimum threshold value.

Value

a data frame with the same columns as the input data frame `accelerometry_counts`, but with a new column named `non_wearing` that indicates whether the individual was wearing their accelerometer during a given period.

`identify_walk_bouts_in_gps_and_accelerometry_data` *Identify walking bouts in GPS and accelerometry data:*

Description

This function identifies walking bouts in GPS and accelerometry data. It processes the GPS data and accelerometry counts to create walk bouts.

Usage

```
identify_walk_bouts_in_gps_and_accelerometry_data(
  gps_data,
  accelerometry_counts,
  ...,
  collated_arguments = NULL
)
```

Arguments

`gps_data`	A data frame containing GPS data
`accelerometry_counts`	A data frame containing accelerometry counts
`...`	Additional arguments to be passed to the function
`collated_arguments`	A list of collated arguments

Value

A data frame containing identified walk bouts

make_active_period *Generate accelerometry datasets*

Description

This function generates a list of activity epochs with specified minimum active counts per epoch, minimum bout length, maximum number of consecutive inactive epochs in a bout, minimum non-wearing length, and minimum complete day length.

Usage

```
make_active_period(
  length = 1,
  is_bout = TRUE,
  non_wearing = FALSE,
  complete_day = FALSE
)
```

Arguments

length	Length of the active period
is_bout	Logical indicating if the active period is a bout
non_wearing	Logical indicating if the active period is a non-wearing period
complete_day	Logical indicating if the active period is a complete day

Value

A list of activity epochs

make_full_day_bout *Create activity counts for a full day bout*

Description

This function creates a data frame with activity counts for a full day bout. A full day bout is defined as an uninterrupted period of activity with a length of at least min_complete_day. The function calls the make_non_bout_window(), make_smallest_bout_window(), and make_smallest_complete_day_activity() functions to generate the activity counts for the non-bout window, smallest bout window, and smallest complete day activity, respectively.

Usage

```
make_full_day_bout()
```

Value

A data frame with activity counts for a full day bout

make_full_day_bout_without_metadata

Create activity counts for a full day bout without metadata

Description

This function creates a data frame with activity counts for a full day bout. A full day bout is defined as an uninterrupted period of activity with a length of at least `min_complete_day`. The function calls the `make_non_bout_window()`, `make_smallest_bout_window()`, and `make_smallest_complete_day_activity()` functions to generate the activity counts for the non-bout window, smallest bout window, and smallest complete day activity, respectively.

Usage

```
make_full_day_bout_without_metadata()
```

Value

A data frame with activity counts for a full day bout without metadata

make_full_walk_bout_df

Create a data frame of walking bouts with GPS data

Description

This function combines accelerometer and GPS data to create a data frame of walking bouts. It generates a full day of activity with bouts of minimum and non-bout periods, and GPS data for walking in Seattle. The accelerometer data is processed into bouts using the `process_accelerometry_counts_into_bouts` function. The GPS data is processed into epochs using the `process_gps_data_into_gps_epochs` function.

Usage

```
make_full_walk_bout_df()
```

Value

A data frame of walking bouts with GPS data

Examples

```
make_full_walk_bout_df()
```

make_inactive_period *Create an inactive period*

Description

This function creates an inactive period with a given length.

Usage

```
make_inactive_period(
  length = 1,
  is_bout = FALSE,
  non_wearing = FALSE,
  complete_day = FALSE
)
```

Arguments

length	The length of the inactive period.
is_bout	Logical value indicating whether this period is part of a bout of inactivity.
non_wearing	Logical value indicating whether this period is due to non-wearing of the accelerometer.
complete_day	Logical value indicating whether this period occurs during a complete day of wearing the accelerometer.

Value

A data frame with columns activity_counts, bout, non_wearing, and complete_day, where activity_counts is set to 0 for the entire length, and bout, non_wearing, and complete_day are set according to the input values.

make_non_bout_window *Create a non-bout window*

Description

This function creates a non-bout window, which is a period of inactivity that is not long enough to be considered as an inactive bout.

Usage

```
make_non_bout_window(maximum_number_consec_inactive_epochs_in_bout = 3)
```

Arguments

maximum_number_consec_inactive_epochs_in_bout
: maximum number of consecutive inactive epochs in a bout before it is terminated

Value

a data frame with columns "activity_counts", "bout", "non_wearing", "complete_day"

Examples

```
make_non_bout_window()
```

---

make_smallest_bout *Make the smallest bout dataset*

Description

Generates a dataset representing the smallest bout, consisting of a sequence of inactive periods followed by the smallest active period.

Usage

```
make_smallest_bout()
```

Value

A data frame containing the activity counts and bout information for the smallest bout.

Examples

```
make_smallest_bout()
```

---

make_smallest_bout_window *Create the smallest bout window*

Description

This function creates an active period of minimum length defined by the parameter `minimum_bout_length`.

Usage

```
make_smallest_bout_window(
  minimum_bout_length = 10,
  is_bout = TRUE,
  non_wearing = FALSE,
  complete_day = FALSE
)
```

Arguments

minimum_bout_length	is the minimum number of epochs for something to be considered a bout
is_bout	Logical indicating if the active period is a bout
non_wearing	Logical indicating if the active period is a non-wearing period
complete_day	Logical indicating if the active period is a complete day

Value

A data.frame with columns `activity_counts`, `bout`, `non_wearing`, and `complete_day` representing the smallest bout window.

---

make_smallest_bout_without_metadata

Create the smallest bout window without metadata

Description

This function creates the smallest bout window without the metadata columns. It calls the `make_smallest_bout` function and then removes the columns "non_wearing", "complete_day", and "bout" using `dplyr::select`.

Usage

```
make_smallest_bout_without_metadata()
```

Value

A data frame containing the smallest bout window without metadata.

Examples

```
make_smallest_bout_without_metadata()
```

`make_smallest_bout_with_largest_inactive_period` *Generate a sequence of accelerometer counts representing the smallest bout with the largest inactive period*

Description

This function generates a sequence of accelerometer counts representing the smallest bout with the largest inactive period. The length of the inactive period is determined by the value of `maximum_number_consec_inactive_e` variable.

Usage

```
make_smallest_bout_with_largest_inactive_period(
  maximum_number_consec_inactive_epochs_in_bout = 3
)
```

Arguments

`maximum_number_consec_inactive_epochs_in_bout`
: maximum number of consecutive inactive epochs in a bout before it is terminated

Value

A data frame with columns `activity_counts` and `time`, representing the accelerometer counts and the corresponding time stamps.

Examples

```
make_smallest_bout_with_largest_inactive_period()
```

`make_smallest_bout_with_smallest_non_wearing_period` *Generate the smallest bout with the smallest non-wearing period dataset*

Description

This function creates a dataset consisting of the smallest bout and the smallest non-wearing period. The bout length, non-wearing period length, and epoch length are defined in the global variables: minimum_bout_length, maximum_number_consec_inactive_epochs_in_bout, and min_non_wearing_length, respectively.

Usage

```
make_smallest_bout_with_smallest_non_wearing_period()
```

Value

A data frame with columns for activity counts and date-time stamps.

Examples

```
make_smallest_bout_with_smallest_non_wearing_period()
```

`make_smallest_complete_day_activity`
Generate an activity sequence for a complete day with minimal activity

---

Description

This function generates an activity sequence for a complete day with a minimal activity count.

Usage

```
make_smallest_complete_day_activity(min_complete_day = 8602)
```

Arguments

`min_complete_day`
minimum number of epochs for something to be a complete day

Value

An activity sequence data frame with minimum activity counts for a complete day.

Examples

```
make_smallest_complete_day_activity()
```

`make_smallest_nonwearing_window`
Create smallest non-wearing window

Description

Create an inactive period that represents the smallest non-wearing window. This function uses the `make_inactive_period()` function to create the non-wearing window.

Usage

```
make_smallest_nonwearing_window(min_non_wearing_length = 20 * 2)
```

Arguments

`min_non_wearing_length`
: minimum non_wearing time before a bout is terminated

Value

An inactive period data frame that represents the smallest non-wearing window.

Examples

```
make_smallest_nonwearing_window()
```

---

next_lat_long *Calculate next latitude and longitude based on current location, speed, direction, and time elapsed.*

Description

Given a current location (latitude and longitude), speed, direction (in radians), and time elapsed (in seconds), this function calculates the next latitude and longitude. The calculations are based on the assumption of a constant speed and direction during the elapsed time.

Usage

```
next_lat_long(latitude, longitude, speed, direction, dt)
```

Arguments

`latitude`	The current latitude in decimal degrees.
`longitude`	The current longitude in decimal degrees.
`speed`	The speed in kilometers per hour.
`direction`	The direction of movement in radians from due north (0 radians).
`dt`	The elapsed time in seconds.

Value

A numeric vector of length 2 containing the next latitude and longitude in decimal degrees.

`outlier_gps_points` *Outlier GPS data points This function identifies outlier GPS points for the bout radius calculation from a given set of latitude and longitude coordinates.*

Description

Outlier GPS data points This function identifies outlier GPS points for the bout radius calculation from a given set of latitude and longitude coordinates.

Usage

```
outlier_gps_points(lat_long, dwellbout_radii_quantile)
```

Arguments

`lat_long`	A data frame containing the latitude and longitude coordinates for the GPS points.
`dwellbout_radii_quantile`	The threshold for outliering GPS data points - any data points above the specified percentile are outliered.

Value

A data frame containing the latitude and longitude coordinates for the non-outlier GPS points.

`parameters` *Global parameters and constants*

Description

List of Parameters `epoch_length` The duration of an epoch in seconds. `active_counts_per_epoch_min` Minimum accelerometer counts for an epoch to be considered active (vs. inactive). `minimum_bout_length` Minimum number of epochs for a period of activity to be considered as a potential bout. `local_time_zone` Local time zone of the data - data come in and are returned in UTC, but local time zone is used to compute complete_days. `maximum_number_consec_inactive_epochs_in_bout` Number of consecutive epochs that can be labeled as inactive during a bout without ending the bout.

Usage

```
parameters
```

Format

An object of class `list` of length 5.

process_accelerometry_counts_into_bouts
Process Accelerometry Counts into Bouts

Description

This function processes accelerometry counts into bouts of activity and returns those bouts as well as flags for whether the individual was wearing their device and if the wearing day can be considered complete

Usage

```
process_accelerometry_counts_into_bouts(
  accelerometry_counts,
  ...,
  collated_arguments = NULL
)
```

Arguments

accelerometry_counts	A data frame with two columns: time and activity counts (CPE, counts per epoch)
...	Additional arguments to be passed to the function.
collated_arguments	An optional list of previously collated arguments.

Details

The input schema for the accelerometry data is `time` and `activity_counts`.

- `time` should be a column in date-time format, in the UTC time zone, with no null values.
- `activity_counts` should be a positive numeric column with no null values.

This function processes accelerometry counts into bouts of activity. The function first validates the input data in the first step. In the second step, the function identifies bouts of activity based on a specified minimum number of active counts per epoch, a maximum number of consecutive inactive epochs allowed within a bout, and a minimum bout length. In the third step, the function identifies non-wearing periods based on a specified threshold of consecutive epochs with 0 activity counts. In the fourth step, the function identifies complete days of wearing the accelerometer based on a specified minimum number of hours of wearing and the epoch length. The returned list includes information about each complete day, including the start and end times of each day, the duration of the day in seconds, the number of epochs, the total number of cpm for the day, and the bouts of activity within the day.

Value

A list of processed data frames containing identified walk bouts, non-wearing periods, and complete days, based on the provided accelerometry counts and processing parameters.

`process_bouts_and_gps_epochs_into_walkbouts`
Process bouts and GPS epochs into walk bouts

Description

This function processes bouts and GPS epochs into walk bouts. It uses a set of parameters and constants to determine whether an epoch is active or inactive, the minimum number of epochs for a period of activity to be considered as a potential bout, the local time zone of the data, and other relevant information. It takes in two data frames, "bouts" and "gps_epochs", and returns a processed data frame, "walk_bouts", with added columns "bout", "bout_radius", "bout_category", "complete_days", "non_wearing", and "speed".#'

Usage

```
process_bouts_and_gps_epochs_into_walkbouts(
  bouts,
  gps_epochs,
  ...,
  collated_arguments = NULL
)
```

Arguments

`bouts`	a data frame containing bout information
`gps_epochs`	a data frame containing GPS information
`...`	additional arguments to be passed on to other functions
`collated_arguments`	a list of arguments collated from other functions

Details

The function first collates the arguments passed to it with the collate_arguments() function. It then merges "gps_epochs" and "bouts" data frames by "time" column, and orders the resulting data frame by "time". Then, it generates the "bout_radius" using the generate_bout_radius() function, which calculates the radius of a bounding circle that would be considered a dwell bout. Next, the function evaluates the completeness of GPS data using the evaluate_gps_completeness() function, which determines the number of GPS observations within a bout and the ratio of data points with versus without GPS data. Finally, the function generates the "bout_category" using the generate_bout_category() function, which determines whether a bout is a walk bout or a dwell bout, and calculates the complete days, non-wearing periods, and speed. The function categorizes bouts into the following categories:

- dwell bout
- non-walk too vigorous
- non-walk too slow
- non-walk too fast
- unknown lack of gps

NOTE: If there are multiple GPS points associated with a given epoch interval, we use the latest possible GPS data point within that epoch. As such, median walking speed is calculated for only the latest available GPS data point in each epoch.

NOTE: The median speed is calculated using only the GPS data points that remain after GPS data processing. All GPS data points that are outliered for the calculation of a bout radius, are, however, included in the assessment of GPS completeness as they are outliers but are still present GPS data points.

NOTE: Outliered data points are excluded from the radius calculation but are included in subsequent functions that assess GPS completeness. They are also returned from these functions with the original data and all new variables.

Value

a processed data frame, "walk_bouts", with added columns "bout", "bout_radius", "bout_category", "complete_days", "non_wearing", and "speed"#'

`process_gps_data_into_gps_epochs`
Convert GPS data into GPS epochs

Description

The input schema for the accelerometry data is `time`, `latitude`, `longitude`, and `speed`.

- `time` should be a column in date-time format, in the UTC time zone, with no null values.
- `latitude` should be a numeric, non-null latitude coordinate between -90 and 90
- `longitude` should be a numeric, non-null longitude coordinate between -180 and 180
- `speed` should be a numeric, non-null value in kilometers per hour

Usage

```
process_gps_data_into_gps_epochs(gps_data, ..., collated_arguments = NULL)
```

Arguments

`gps_data`	A data frame containing GPS data. Must have columns "Latitude", "Longitude"
`...`	Additional arguments to be passed to the function.
`collated_arguments`	A named list of arguments, used to avoid naming conflicts when calling this function as part of a pipeline. Optional.

Details

This function processes GPS data into GPS epochs, with each epoch having a duration specified by `epoch_length`.

Value

A data frame with columns latitude, longitude, time, and speed, where time is now the nearest epoch start time

---

run_length_encode *Run Length Encoding:*

Description

A function that runs a normal run length encoding and adds some extra variables for use in calculations.

Usage

```
run_length_encode(x)
```

Arguments

`x`	a vector to run the function on

Value

a data.frame with columns for lengths, values, end, and begin

---

summarize_walk_bouts *Summarize walking bouts: This function summarizes walking bouts and calculates the median speed, complete day, non-wearing, bout start, and duration of each bout.*

Description

Summarize walking bouts: This function summarizes walking bouts and calculates the median speed, complete day, non-wearing, bout start, and duration of each bout.

Usage

```
summarize_walk_bouts(walk_bouts, ..., collated_arguments = NULL)
```

Arguments

`walk_bouts`	A data frame containing identified walk bouts
`...`	Additional arguments to be passed to the function
`collated_arguments`	A list of collated arguments

Value

A data frame summarizing identified walk bouts

`validate_accelerometry_data` *Validate accelerometry input data*

Description

The input schema for the accelerometry data is `time` and `activity_counts`.

- `time` should be a column in date-time format, in the UTC time zone, with no null values.
- `activity_counts` should be a positive numeric column with no null values.

Usage

```
validate_accelerometry_data(accelerometry_counts)
```

Arguments

`accelerometry_counts`	Raw accelerometry data with the expected schema.

Details

This function checks the schema of the accelerometry input data and raises an error if any schema constraints are violated.

The following schema validations are performed on the input data:

- The input data must contain two columns, named `time` and `activity_counts`.
- The `time` column must be in date-time format, in the UTC time zone, with no null values.
- The `activity_counts` column must be a positive numeric column with no null values.

Value

This function does not return anything. It throws an error if the accelerometry data fails any of the validation checks.

Examples

```
# Example usage:
data <- data.frame(
  time = seq(
    as.POSIXct("2021-01-01 00:00:00", tz = "UTC"),
    as.POSIXct("2021-01-01 23:59:59", tz = "UTC"),
    by = "5 mins"
  )) %>%
  dplyr::mutate(activity_counts = sample(0:100, length(time), replace = TRUE))
validate_accelerometry_data(data)
```

`validate_gps_data` *Validate GPS data*

Description

This function validates GPS data for required variables, correct variable class, and correct data range.

Usage

```
validate_gps_data(gps_data)
```

Arguments

`gps_data`	A data frame containing GPS data with the following variables: time, latitude, longitude, and speed.

Value

This function does not return anything. It throws an error if the GPS data fails any of the validation checks.

Index