Package 'mpathsenser'

June 1, 2022

Title Process and Analyse Data from m-Path Sense

Version 1.0.3

Description Overcomes one of the major challenges in mobile (passive) sensing, namely being able to pre-process the raw data that comes from a mobile sensing app, specifically ``m-Path Sense'' <https://m-path.io>. The main task of 'mpathsenser' is therefore to read ``m-Path Sense'' JSON files into a database and provide several convenience functions to aid in data processing.

Depends R (>= 4.0.0)

License GPL (>= 3)

URL https://gitlab.kuleuven.be/ppw-okpiv/researchers/u0134047/mpathsenser/

BugReports https://gitlab.kuleuven.be/ppw-okpiv/researchers/u0134047/mpathsenser/-/issues

Encoding UTF-8

RoxygenNote 7.2.0

Imports DBI, dbplyr, dplyr, furrr, future, jsonlite, lubridate, magrittr, purrr, rjson, RSQLite, stats, tibble, tidyr

Suggests curl, dbx, ggplot2, httr, lme4, lmerTest, progressr, rlang, rvest, sodium, testthat (>= 3.0.0), vroom

Config/testthat/edition 3

Config/Needs/website tidyverse/tidytemplate

NeedsCompilation no

Author Koen Niemeijer [aut, cre] (<https://orcid.org/0000-0002-0816-534X>)

Maintainer Koen Niemeijer <koen.niemeijer@kuleuven.be>

Repository CRAN

Date/Publication 2022-06-01 09:50:02 UTC

R topics documented:

app_category *Find the category of an app on the Google Play Store*

Description

This function scrapes the Google Play Store by using name as the search term. From there it selects the first result in the list and its corresponding category and package name.

Usage

```
app_category(name, num = 1, rate_limit = 5)
```

Arguments

name	The name of the app to search for.
num	Which result should be selected in the list of search results. Defaults to one.
rate_limit	The time interval to keep between queries, in seconds. If the rate limit is too low, the Google Play Store may reject further requests or even ban your entirely.

Value

A list containing the following fields:

package	the package name that was selected from the Google Play search
genre	the corresponding genre of this package

Warning

Do not abuse this function or you will be banned by the Google Play Store. The minimum delay between requests seems to be around 5 seconds, but this is untested. Also make sure not to do batch lookups, as many subsequent requests will get you blocked as well.

Examples

```
app_category('whatsapp')

# Example of a generic app name where we can't find a specific app
app_category('weather') # Weather forecast channel

# Get OnePlus weather
app_category('net.oneplus.weather')
```

ccopy *Copy mpathsenser zip files to a new location*

Description

Copy zip files from a source destination to an origin destination where they do not yet exist. That is, it only updates the origin folder from the source folder.

Usage

```
ccopy(from, to = getwd(), recursive = TRUE)
```

Arguments

from	A path to copy files from.
to	A path to copy files to.
recursive	Should files from subdirectories be copied?

Value

A message indicating how many files were copied.

Examples

```
## Not run:
ccopy('K:/data/myproject/', '~/myproject')

## End(Not run)
```

---

close_db *Close a database connection*

Description

This is a convenience function that is simply a wrapper around dbDisconnect.

Usage

```
close_db(db)
```

Arguments

db	A database connection to an m-Path Sense database.

Value

close_db returns invisibly regardless of whether the database is active, valid, or even exists.

See Also

open_db for opening an mpathsenser database.

copy_db *Copy (a subset of) a database to another database*

Description

Copy (a subset of) a database to another database

Usage

```
copy_db(
  from_db,
  to_db = NULL,
  sensor = "All",
  path = getwd(),
  db_name = "sense.db"
)
```

Arguments

from_db	A mpathsenser database connection from where the data will be transferred.
to_db	A mpathsenser database connection where the data will be transferred to. If no new_db is specified, a path (and possibly a db_name) must be specified for create_db to create a new database.
sensor	A character vector containing one or multiple sensors. See sensors for a list of available sensors. Use "All" for all available sensors.
path	The path to the database. Use NULL to use the full path name in db_name.
db_name	The name of the database.

Value

No return value, called for side effects.

coverage *Create a coverage chart of the sampling rate*

Description

Only applicable to non-reactive sensors with 'continuous' sampling

Usage

```
coverage(
  db,
  participant_id,
  sensor = "All",
  frequency = mpathsenser::freq,
  relative = TRUE,
  offset = "None",
  start_date = NULL,
  end_date = NULL,
  plot = TRUE
)
```

Arguments

db	A valid database connection. Schema must be that as it is created by open_db.
participant_id	A character string of *one* participant ID.
sensor	A character vector containing one or multiple sensors. See sensors for a list of available sensors. Use 'All' for all available sensors.
frequency	A named numeric vector with sensors as names and the number of expected samples per hour
relative	Show absolute number of measurements or relative to the expected number? Logical value.
offset	Currently not used.
start_date	A date (or convertible to a date using as.Date) indicating the earliest date to show. Leave empty for all data. Must be used with end_date.
end_date	A date (or convertible to a date using as.Date) indicating the latest date to show.Leave empty for all data. Must be used with start_date.
plot	Whether to return a ggplot or its underlying data.

Value

A ggplot of the coverage results if plot is TRUE or a tibble containg the hour, type of measure (i.e. sensor), and (relative) coverage.

Examples

```
## Not run:
fix_json()
unzip()
freq <- c(
  Accelerometer = 720, # Once per 5 seconds. Can have multiple measurements.
  AirQuality = 1,
  AppUsage = 2, # Once every 30 minutes
  Bluetooth = 60, # Once per minute. Can have multiple measurements.
  Gyroscope = 720, # Once per 5 seconds. Can have multiple measurements.
  Light = 360, # Once per 10 seconds
```

```
  Location = 60, # Once per 60 seconds
  Memory = 60, # Once per minute
  Noise = 120,
  Pedometer = 1,
  Weather = 1,
  Wifi = 60 # once per minute
)
coverage(
  db = db,
  participant_id = '12345',
  sensor = c('Accelerometer', 'Gyroscope'),
  frequency = mpathsenser::freq,
  start_date = '2021-01-01',
  end_date = '2021-05-01'
)

## End(Not run)
```

`create_db` *Create a new mpathsenser database*

Description

Create a new mpathsenser database

Usage

```
create_db(path = getwd(), db_name = "sense.db", overwrite = FALSE)
```

Arguments

`path`	The path to the database.
`db_name`	The name of the database.
`overwrite`	In case a database with `db_name` already exists, indicate whether it should be overwritten or not. Otherwise, this option is ignored.

Value

A database connection using prepared database schemas.

decrypt_gps *Decrypt GPS data from a curve25519 public key*

Description

By default, the latitude and longitude of the GPS data collected by m-Path Sense will be encrypted using an asymmetric curve25519 key to provide extra protection for these highly sensitive data. This function takes the entire location data set and decrypts its longitude and latitude columns using the provided `key`.

Usage

```
decrypt_gps(data, key)
```

Arguments

`data`	A (lazy) tibble containing the GPS data
`key`	A curve25519 public key

Value

A tibble containing the non-lazy, decrypted GPS data

device_info *Get the device info for one or more participants*

Description

Get the device info for one or more participants

Usage

```
device_info(db, participant_id = NULL)
```

Arguments

`db`	A database connection to an m-Path Sense database.
`participant_id`	A character string identifying a single participant. Use `get_participants` to retrieve all participants from the database. Leave empty to get data for all participants.

Value

A tibble containing device info for each participant

first_date *Extract the date of the first entry*

Description

A helper function for extracting the first date of entry of (of one or all participant) of one sensor. Note that this function is specific to the first date of a sensor. After all, it wouldn't make sense to extract the first date for a participant of the accelerometer, while the first device measurement occurred a day later.

Usage

```
first_date(db, sensor, participant_id = NULL)
```

Arguments

db	A database connection to an m-Path Sense database.
sensor	The name of a sensor. See sensors for a list of available sensors.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.

Value

A string in the format 'YYYY-mm-dd' of the first entry date.

Examples

```
## Not run:
db <- open_db()
first_date(db, 'Accelerometer', '12345')

## End(Not run)
```

fix_jsons *Fix the end of JSON files*

Description

When copying data directly coming from m-Path Sense, JSON files are sometimes corrupted due to the app not properly closing them. This function attempts to fix the most common problems associated with improper file closure by m-Path Sense.

Usage

```
fix_jsons(path = getwd(), files = NULL, recursive = TRUE, parallel = FALSE)
```

Arguments

path	The path name of the JSON files.
files	Alternatively, a character list of the input files
recursive	Should the listing recurse into directories?
parallel	A logical value whether you want to check in parallel. Useful for a lot of files.

Value

A message indicating how many files were fixed.

Progress

You can be updated of the progress by this function by using the progress package. See progressr's vignette on how to subscribe to these updates.

Examples

```
## Not run:
future::plan(future::multisession)
files <- test_jsons()
fix_jsons(files = files)

## End(Not run)
```

---

freq *Measurement frequencies per sensor*

Description

A numeric vector containing (an example) of example measurement frequencies per sensor. Such input is needed for coverage.

Usage

```
freq
```

Format

An object of class numeric of length 11.

Value

This vector contains the following information:

Sensor	Frequency (per hour)	Full text
Accelerometer	720	Once per 5 seconds. Can have multiple instances.
AirQuality	1	Once per hour.
AppUsage	2	Once every 30 minutes. Can have multiple instances.
Bluetooth	12	Once every 5 minutes. Can have multiple instances.
Gyroscope	720	Once per 5 seconds. Can have multiple instances.
Light	360	Once per 10 seconds.
Location	60	Once every 60 seconds.
Memory	60	Once per minute
Noise	120	Once every 30 seconds. Microhone cannot be used in the background in Android 11
Weather	1	Once per hour.
Wifi	60	Once per minute.

`geocode_rev` *Reverse geocoding with latitude and longitude*

Description

This functions allows you to extract information about a place based on the latitude and longitude from the OpenStreetMaps nominatim API.

Usage

```
geocode_rev(lat, lon, zoom = 18, email = "", rate_limit = 1)
```

Arguments

`lat`	The latitude of the location (in degrees)
`lon`	The longitude of the location (in degrees)
`zoom`	The desired zoom level from 1-18. The lowest level, 18, is building level.
`email`	If you are making large numbers of request please include an appropriate email address to identify your requests. See Nominatim's Usage Policy for more details.
`rate_limit`	The time interval to keep between queries, in seconds. If the rate limit is too low, the OpenStreetMaps may reject further requests or even ban your entirely.

Value

A list of information about the location. See Nominatim's documentation for more details.

Warning

Do not abuse this function or you will be banned by OpenStreetMap. The maximum number of requests is around 1 per second. Also make sure not to do batch lookups, as many subsequent requests will get you blocked as well.

Examples

```
# Frankfurt Airport
geocode_rev(50.037936, 8.5599631)
```

`get_activity` *Get a summary of physical activity (recognition)*

Description

Get a summary of physical activity (recognition)

Usage

```
get_activity(
  db,
  participant_id = NULL,
  confidence = 70,
  direction = "forward",
  start_date = NULL,
  end_date = NULL,
  by = c("Total", "Day", "Hour")
)
```

Arguments

db	A database connection to an m-Path Sense database.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.
confidence	The minimum confidence (0-100) that should be assigned to an observation by Activity Recognition.
direction	The directionality of the duration calculation, i.e. $t_{t-1} - t$ or $t - t_{t+1}$.
start_date	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
end_date	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.
by	Either 'Total', 'Hour', or 'Day' indicating how to summarise the results.

Value

A tibble containing a column 'activity' and a column 'duration' for the hourly activity duration.

`get_app_usage` *Get app usage per hour*

Description

This function extracts app usage per hour for either one or multiple participants. If multiple days are selected, the app usage time is averaged.

Usage

```
get_app_usage(
  db,
  participant_id = NULL,
  start_date = NULL,
  end_date = NULL,
  by = c("Total", "Day", "Hour")
)
```

Arguments

`db`	A database connection to an m-Path Sense database.
`participant_id`	A character string identifying a single participant. Use `get_participants` to retrieve all participants from the database. Leave empty to get data for all participants.
`start_date`	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
`end_date`	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.
`by`	Either 'Total', 'Hour', or 'Day' indicating how to summarise the results.

Value

A data frame containing a column 'app' and a column 'usage' for the hourly app usage.

get_data *Generic helper function from extracting data from an m-Path Sense database*

Description

This is a generic function to help extract data from an m-Path sense database. For some sensors that require a bit more pre-processing, such as app usage and screen time, more specialised functions are available (e.g. `get_app_usage` and `screen_duration`).

Usage

```
get_data(db, sensor, participant_id = NULL, start_date = NULL, end_date = NULL)
```

Arguments

db	A database connection to an m-Path Sense database.
sensor	The name of a sensor. See sensors for a list of available sensors.
participant_id	A character string identifying a single participant. Use `get_participants` to retrieve all participants from the database. Leave empty to get data for all participants.
start_date	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
end_date	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.

Value

A lazy `tbl` containing the requested data.

Examples

```
## Not run:
# Open a database
db <- open_db()

# Retrieve some data
get_data(db, 'Accelerometer', '12345')

# Or within a specific window
get_data(db, 'Accelerometer', '12345', '2021-01-01', '2021-01-05')

## End(Not run)
```

get_installed_apps *Get installed apps*

Description

Extract installed apps for one or all participants. Contrarily to other get_* functions in this package, start and end dates are not used since installed apps are assumed to be fixed throughout the study.

Usage

```
get_installed_apps(db, participant_id = NULL)
```

Arguments

db	A database connection to a mpathsenser database.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.

Value

A tibble containing app names.

get_nrows *Get the number of rows sensors in a mpathsenser database*

Description

Get the number of rows sensors in a mpathsenser database

Usage

```
get_nrows(
  db,
  sensor = "All",
  participant_id = NULL,
  start_date = NULL,
  end_date = NULL
)
```

Arguments

db	db A database connection, as created by create_db.
sensor	A character vector of one or multiple vectors. Use "All" for all sensors. See sensors for a list of all available sensors.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.
start_date	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
end_date	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.

Value

A named vector containing the number of rows for each sensor.

---

get_participants *Get all participants*

Description

Get all participants

Usage

```
get_participants(db, lazy = FALSE)
```

Arguments

db	db A database connection, as created by create_db.
lazy	Whether to evaluate lazily using dbplyr.

Value

A data frame containing all participant_id and study_id.

get_processed_files *Get all processed files from a database*

Description

Get all processed files from a database

Usage

```
get_processed_files(db)
```

Arguments

db	A database connection, as created by create_db.

Value

A data frame containing the `file_name`, `participant_id`, and `study_id` of the processed files.

get_studies *Get all studies*

Description

Get all studies

Usage

```
get_studies(db, lazy = FALSE)
```

Arguments

db	db A database connection, as created by create_db.
lazy	Whether to evaluate lazily using dbplyr.

Value

A data frame containing all studies.

haversine *Calculate the Great-Circle Distance between two points in kilometers*

Description

Calculate the great-circle distance between two points using the Haversine function.

Usage

```
haversine(lat1, lon1, lat2, lon2, r = 6371)
```

Arguments

lat1	The latitude of point 1 in degrees.
lon1	The longitude of point 1 in degrees.
lat2	The latitude of point 2 in degrees.
lon2	The longitude of point 2 in degrees.
r	The average earth radius.

Value

A numeric value of the distance between point 1 and 2 in kilometers.

Examples

```
fra <- c(50.03333, 8.570556) # Frankfurt Airport
ord <- c(41.97861, -87.90472) # Chicago O'Hare International Airport
haversine(fra[1], fra[2], ord[1], ord[2]) # 6971.059 km
```

identify_gaps *Identify gaps in mpathsenser mobile sensing data*

Description

Oftentimes in mobile sensing, gaps appear in the data as a result of the participant accidentally closing the app or the operating system killing the app to save power. This can lead to issues later on during data analysis when it becomes unclear whether there are no measurements because no events occurred or because the app quit in that period. For example, if no screen on/off event occur in a 6-hour period, it can either mean the participant did not turn on their phone in that period or that the app simply quit and potential events were missed. In the latter case, the 6-hour missing period has to be compensated by either removing this interval altogether or by subtracting the gap from the interval itself (see examples).

Usage

```
identify_gaps(
  db,
  participant_id = NULL,
  min_gap = 60,
  sensor = "Accelerometer"
)
```

Arguments

db	A database connection to an m-Path Sense database.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.
min_gap	The minimum time (in seconds) passed between two subsequent measurements for it to be considered a gap..
sensor	The name of a sensor. See sensors for a list of available sensors.

Details

While any sensor can be used for identifying gaps, it is best to choose a sensor with a very high, near-continuous sample rate such as the accelerometer or gyroscope. This function then creates time between two subsequent measurements and returns the period in which this time was larger than min_gap.

Note that the from and to columns in the output are character vectors in UTC time.

Value

A tibble containing the time period of the gaps. The strucute of this tibble is as follows:

participant_id	the participant_id of where the gap occurred
from	the time of the last measurement before the gap
to	the time of the first measurement after the gap
gap	the time passed between from and to, in seconds

Examples

```
## Not run:
# Find the gaps for a participant and convert to datetime
gaps <- identify_gaps(db, '12345', min_gap = 60) %>%
  mutate(across(c(to, from), ymd_hms)) %>%
  mutate(across(c(to, from), with_tz, 'Europe/Brussels'))

# Get some sensor data and calculate a statistic, e.g. the time spent walking
# You can also do this with larger intervals, e.g. the time spent walking per hour
walking_time <- get_data(db, 'Activity', '12345') %>%
  collect() %>%
  mutate(datetime = ymd_hms(paste(date, time))) %>%
  mutate(datetime = with_tz(datetime, 'Europe/Brussels')) %>%
```

```
  arrange(datetime) %>%
  mutate(prev_time = lag(datetime)) %>%
  mutate(duration = datetime - prev_time) %>%
  filter(type == 'WALKING')

# Find out if a gap occurs in the time intervals
walking_time %>%
  rowwise() %>%
  mutate(gap = any(gaps$from >= prev_time & gaps$to <= datetime))

## End(Not run)
```

`import` *Import mpathsenser files into a database (mpathsenser data scheme)*

Description

Import JSON files from m-Path Sense into a structured database. This function is the bread and butter of this package, as it creates (or rather fills) the database that (almost) all the other functions use.

Usage

```
import(
  path = getwd(),
  db = NULL,
  dbname = "sense.db",
  overwrite_db = TRUE,
  sensors = NULL,
  batch_size = 24,
  backend = "RSQLite",
  recursive = TRUE,
  parallel = FALSE
)
```

Arguments

`path`	The path to the file directory
`db`	Valid database connection.
`dbname`	If no database is provided, a new database dbname is created.
`overwrite_db`	If a database with the same `dbname` already exists, should it be overwritten?
`sensors`	Select one or multiple sensors as in `sensors`. Leave NULL to extract all sensor data.
`batch_size`	The number of files that are to be processed in a single batch.
`backend`	Name of the database backend that is used. Currently, only RSQLite is supported.

recursive	Should the listing recurse into directories?
parallel	A value that indicates whether to do reading in and processing in parallel. If this argument is a number, this indicates the number of workers that will be used.

Details

import is highly customisable in the sense that you can specify which sensors to import (even though there may be more in the files) and it also allows batching for a speedier writing process. If parallel is TRUE, it is recommended to batch_size be a scalar multiple of the number of CPUs the parallel cluster can use. If a single JSON file in the batch causes and error, the batch is terminated (but not the function) and it is up to the user to fix the file. This means that if batch_size is large, many files will not be processed. Set batch_size to 1 for sequential file processing (i.e. one-by-one).

Currently, only SQLite is supported as a backend. Due to its concurrency restriction, the parallel option is disabled. To get an indication of the progress so far, set one of the handlers using the progressr package, e.g. progressr::handlers('progress').

Value

A message indicating how many files were imported. Imported database can be reopened using open_db.

Progress

You can be updated of the progress by this function by using the progress package. See progressr's vignette on how to subscribe to these updates.

---

index_db *Create indexes for a mpathsenser database*

Description

Create indexes for a mpathsenser database

Usage

```
index_db(db)
```

Arguments

db	A database connection to an m-Path Sense database.

Value

No return value, called for side effects.

last_date *Extract the date of the last entry*

Description

A helper function for extracting the last date of entry of (of one or all participant) of one sensor. Note that this function is specific to the last date of a sensor. After all, it wouldn't make sense to extract the last date for a participant of the device info, while the last accelerometer measurement occurred a day later.

Usage

```
last_date(db, sensor, participant_id = NULL)
```

Arguments

db	A database connection to an m-Path Sense database.
sensor	The name of a sensor. See sensors for a list of available sensors.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.

Value

A string in the format 'YYYY-mm-dd' of the last entry date.

Examples

```
## Not run:
db <- open_db()
first_date(db, 'Accelerometer', '12345')

## End(Not run)
```

link *Match y to the time scale of x*

Description

Function for linking mobile sensing and ESM data

Usage

```
link(x, y, by = NULL, offset)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). Both x and y must have a column called time.
by	If NULL, the default, *_join() will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying by explicitly. To join by different variables on x and y, use a named vector. For example, by = c('a' = 'b') will match x$a to y$b To join by multiple variables, use a vector with length > 1. For example, by = c('a', 'b') will match x$a to y$a and x$b to y$b. Use a named vector to match different variables in x and y. For example, by = c('a' = 'b', 'c' = 'd') will match x$a to y$b and x$c to y$d. To perform a cross-join, generating all combinations of x and y, use by = character().
offset	The time window in which y is to be matched to x. Must be convertible to a period by as.period.

Details

assumption: both x and y have column 'time' containing DateTimeClasses

Value

A tibble with the data of x with a new column data with the matched data of y according to offset.

---

link2 *Link two sensors OR one sensor and an external data frame*

Description

This function is specific to mpathsenser databases. It is a wrapper around link but extracts data in the database for you.

Usage

```
link2(
  db,
  sensor_one,
  sensor_two = NULL,
  offset,
  participant_id = NULL,
  start_date = NULL,
  end_date = NULL,
  external = NULL,
  reverse = FALSE,
  ignore_large = FALSE
)
```

Arguments

db	A database connection to an m-Path Sense database.
sensor_one	The name of a primary sensor. See sensors for a list of available sensors.
sensor_two	The name of a secondary sensor. See sensors for a list of available sensors. Cannot be used together with external.
offset	The time window in which y is to be matched to x. Must be convertible to a period by as.period.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.
start_date	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
end_date	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.
external	Optionally, specify an external data frame. Cannot be used at the same time as a second sensor. This data frame must have a column called time.
reverse	Switch sensor_one with either sensor_two or external? Particularly useful in combination with external.
ignore_large	Safety override to prevent long wait times. Set to TRUE to do this function on lots of data.

Value

A tibble with the data of sensor_one with a new column data with the matched data of either sensor_two or external according to offset. The other way around when reverse = TRUE.

See Also

link

---

moving_average *Moving average for values in an mpathsenser database*

Description

Moving average for values in an mpathsenser database

Usage

```
moving_average(
  db,
  sensor,
  participant_id,
  ...,
  n,
  start_date = NULL,
  end_date = NULL
)
```

Arguments

db	A database connection to an m-Path Sense database.
sensor	The name of a sensor. See sensors for a list of available sensors.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database.
...	Unquoted names of columns of the sensor table to average over.
n	The number of observations to average over.
start_date	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
end_date	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.

Value

A tibble with the same columns as the input, modified to be a moving average.

Examples

```
## Not run:
get_moving_average(db, 'Light', '12345', mean_lux, max_lux, n = 5)

## End(Not run)
```

`mpathsenser` *mpathsenser: Process and Analyse Data from m-Path Sense*

Description

Overcomes one of the major challenges in mobile (passive) sensing, namely being able to preprocess the raw data that comes from a mobile sensing app, specifically "m-Path Sense" https://m-path.io. The main task of 'mpathsenser' is therefore to read "m-Path Sense" JSON files into a database and provide several convenience functions to aid in data processing.

Author(s)

Maintainer: Koen Niemeijer <koen.niemeijer@kuleuven.be> (ORCID)

Other contributors:

- KU Leuven [copyright holder, funder]

`n_screen_on` *Get number of times screen turned on*

Description

Get number of times screen turned on

Usage

```
n_screen_on(
  db,
  participant_id,
  start_date = NULL,
  end_date = NULL,
  by = c("Total", "Hour", "Day")
)
```

Arguments

`db`	A database connection to an m-Path Sense database.
`participant_id`	A character string identifying a single participant. Use `get_participants` to retrieve all participants from the database. Leave empty to get data for all participants.
`start_date`	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
`end_date`	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.
`by`	Either 'Total', 'Hour', or 'Day' indicating how to summarise the results. Defaults to total.

Value

In case grouping is by the total amount, returns a single numeric value. For date and hour grouping returns a tibble with columns 'date' or 'hour' and the number of screen on's 'n'.

`n_screen_unlocks` *Get number of screen unlocks*

Description

Get number of screen unlocks

Usage

```
n_screen_unlocks(
  db,
  participant_id,
  start_date = NULL,
  end_date = NULL,
  by = c("Total", "Hour", "Day")
)
```

Arguments

`db`	A database connection to an m-Path Sense database.
`participant_id`	A character string identifying a single participant. Use `get_participants` to retrieve all participants from the database. Leave empty to get data for all participants.
`start_date`	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
`end_date`	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.
`by`	Either 'Total', 'Hour', or 'Day' indicating how to summarise the results. Defaults to total.

Value

In case grouping is by the total amount, returns a single numeric value. For date and hour grouping returns a tibble with columns 'date' or 'hour' and the number of screen unlocks 'n'.

open_db *Open an mpathsenser database.*

Description

Open an mpathsenser database.

Usage

```
open_db(path = getwd(), db_name = "sense.db")
```

Arguments

path	The path to the database. Use NULL to use the full path name in db_name.
db_name	The name of the database.

Value

A connection to an mpathsenser database.

See Also

close_db for closing a database; copy_db for copying (part of) a database; index_db for indexing a database; get_data for extracting data from a database.

screen_duration *Screen duration by hour or day*

Description

Calculate the screen duration time where the screen was *unlocked* (i.e. not just on).

Usage

```
screen_duration(
  db,
  participant_id,
  start_date = NULL,
  end_date = NULL,
  by = c("Hour", "Day")
)
```

Arguments

db	A database connection to an m-Path Sense database.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.
start_date	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
end_date	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.
by	Either 'Hour' or 'Day' indicating how to summarise the results. Leave empty to get raw screen duration per measurement.

Value

A tibble with either 'hour' and 'duration' columns or 'date' and 'duration' columns depending on the by argument. Alternatively, if no by is specified, a remote tibble is returned with the date, time, and duration since the previous measurement.

---

sensors *Available Sensors*

Description

A list containing all available sensors in this package you can work with. This variable was created so it is easier to use in your own functions, e.g. to loop over sensors.

Usage

```
sensors
```

Format

An object of class character of length 25.

Value

A character vector containing all sensor names supported by mpathsenser.

Examples

```
sensors
```

step_count *Get step count*

Description

Extracts the number of steps per hour as sensed by the underlying operating system.

Usage

```
step_count(db, participant_id = NULL, start_date = NULL, end_date = NULL)
```

Arguments

db	A database connection to an m-Path Sense database.
participant_id	A character string identifying a single participant. Use get_participants to retrieve all participants from the database. Leave empty to get data for all participants.
start_date	Optional search window specifying date where to begin search. Must be convertible to date using as.Date. Use first_date to find the date of the first entry for a participant.
end_date	Optional search window specifying date where to end search. Must be convertible to date using as.Date. Use last_date to find the date of the last entry for a participant.

Value

A tibble with the 'date', 'hour', and the number of 'steps'.

test_jsons *Test JSON files for being in the correct format.*

Description

Test JSON files for being in the correct format.

Usage

```
test_jsons(
  path = getwd(),
  files = NULL,
  db = NULL,
  recursive = TRUE,
  parallel = FALSE
)
```

Arguments

path	The path name of the JSON files.
files	Alternatively, a character list of the input files.
db	A mpathsenser database connection (optional). If provided, will be used to check which files are already in the database and check only those JSON files which are not.
recursive	Should the listing recurse into directories?
parallel	A logical value whether you want to check in parallel. Useful when there are a lot of files. If you have already used plan, you can leave this parameter to FALSE.

Value

A message indicating whether there were any issues and a character vector of the file names that need to be fixed. If there were no issues, no result is returned.

Progress

You can be updated of the progress by this function by using the progress package. See progressr's vignette on how to subscribe to these updates.

---

unzip_data *Unzip m-Path Sense output*

Description

Similar to unzip, but makes it easier to unzip all files in a given path with one function call.

Usage

```
unzip_data(
  path = getwd(),
  overwrite = FALSE,
  recursive = TRUE,
  parallel = FALSE
)
```

Arguments

path	The path to the directory containing the zip files.
overwrite	Logical value whether you want to overwrite already existing zip files.
recursive	Logical value indicating whether to unzip files in subdirectories as well. These files will then be unzipped in their respective subdirectory.
parallel	A logical value whether you want to check in parallel. Useful when there are a lot of files. If you have already used future::plan, you can leave this parameter to FALSE.

Value

A message indicating how many files were unzipped.

Progress

You can be updated of the progress by this function by using the `progress` package. See `progressr`'s vignette on how to subscribe to these updates.

Index