Package 'fitbitViz'

May 8, 2026

Type Package

Title 'Fitbit' Visualizations

Version 1.0.8

Date 2026-04-27

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

URL https://github.com/mlampros/fitbitViz,
https://mlampros.github.io/fitbitViz/

Description Visualization of pre-downloaded 'Fitbit' personal health data using 'ggplot2' Visualizations, 'Leaflet' and 3-dimensional 'Rayshader' Maps. The 3-dimensional 'Rayshader' Map requires the installation of the 'CopernicusDEM' R package which includes the 30- and 90-meter elevation data.

License GPL-3

Encoding UTF-8

Depends R(>= 3.5)

Imports glue, ggplot2, lubridate, patchwork, data.table, stats, viridis, scales, ggthemes, paletteer, XML, hms, leaflet, sf, rstudioapi, grDevices, leafgl, raster (>= 3.6-3), terra, magrittr, utils, lifecycle, reshape2

Suggests CopernicusDEM, testthat (>= 3.0.0), knitr, rmarkdown, DT, rgl, rayshader, magick

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Lampros Mouselimis [aut, cre] (ORCID: <https://orcid.org/0000-0002-8024-1546>)

Repository CRAN

Date/Publication 2026-04-27 08:50:03 UTC

Contents

`crop_DEM` *Function to crop the AOI from the downloaded DEM .tif file*

Description

Function to crop the AOI from the downloaded DEM .tif file

Usage

```
crop_DEM(tif_or_vrt_dem_file, sf_buffer_obj, verbose = FALSE)
```

Arguments

`tif_or_vrt_dem_file`	a valid path to the elevation .tif or .vrt file
`sf_buffer_obj`	a simple features ('sf') object that will be used to crop the input elevation raster file ('tif_or_vrt_dem_file' parameter)
`verbose`	a boolean. If TRUE then information will be printed out in the console

Value

an object of class SpatRaster

Examples

```
## Not run:

require(fitbitViz)

# export the TCX file from the Fitbit website or app for the desired activity

res_tcx <- GPS_TCX_data(
```

```
  tcx_file = "/path/to/activity.tcx",
  time_zone = "Europe/Athens",
  verbose = TRUE
)
str(res_tcx)

# ................................................
# then compute the sf-object buffer and raster-extend
# ................................................

sf_rst_ext <- extend_AOI_buffer(
  dat_gps_tcx = res_tcx,
  buffer_in_meters = 1000,
  CRS = 4326,
  verbose = TRUE
)
sf_rst_ext

# ............................................................
# Download the Copernicus DEM 30m elevation data because it has
# a better resolution, it takes a bit longer to download because
# the .tif file size is bigger
# ............................................................

dem_dir <- tempdir()
# dem_dir

dem30 <- CopernicusDEM::aoi_geom_save_tif_matches(
  sf_or_file = sf_rst_ext$sfc_obj,
  dir_save_tifs = dem_dir,
  resolution = 30,
  crs_value = 4326,
  threads = parallel::detectCores(),
  verbose = TRUE
)

TIF <- list.files(dem_dir, pattern = ".tif", full.names = T)
# TIF

if (length(TIF) > 1) {
  # ................................................
  # create a .VRT file if I have more than 1 .tif files
  # ................................................

  file_out <- file.path(dem_dir, "VRT_mosaic_FILE.vrt")

  vrt_dem30 <- create_VRT_from_dir(
    dir_tifs = dem_dir,
    output_path_VRT = file_out,
    verbose = TRUE
  )
}
```

```
if (length(TIF) == 1) {
  # ..............................................
  # if I have a single .tif file keep the first index
  # ..............................................

  file_out <- TIF[1]
}

raysh_rst <- crop_DEM(
  tif_or_vrt_dem_file = file_out,
  sf_buffer_obj = sf_rst_ext$sfc_obj,
  verbose = TRUE
)

terra::plot(raysh_rst)

## End(Not run)
```

`extend_AOI_buffer` *Extract the sf-object and raster extent based on a buffer (in meters)*

Description

Extract the sf-object and raster extent based on a buffer (in meters)

Usage

```
extend_AOI_buffer(
  dat_gps_tcx,
  buffer_in_meters = 1000,
  CRS = 4326,
  verbose = FALSE
)
```

Arguments

`dat_gps_tcx`	this parameter corresponds to the output data.table of the 'GPS_TCX_data()' function
`buffer_in_meters`	an integer value specifying the buffer in meters. The bounding box of the input coordinates (longitudes, latitudes) will be extended by that many meters. The default value is 1000 meters.
`CRS`	an integer specifying the Coordinates Reference System. The recommended value for this data is 4326 (which is also the default value)
`verbose`	a boolean. If TRUE then information will be printed out in the console

Details

To create the buffer in meters using the 'sf' package I had to transform to another projection - by default I've used 7801 - as suggested in the following stackoverflow thread, https://stackoverflow.com/a/54754935/8302386

Value

an object of class list

Examples

```
## Not run:

require(fitbitViz)

# export the TCX file from the Fitbit website or app for the desired activity

res_tcx <- GPS_TCX_data(
  tcx_file = "/path/to/activity.tcx",
  time_zone = "Europe/Athens",
  verbose = TRUE
)
str(res_tcx)

# ................................................
# then compute the sf-object buffer and raster-extend
# ................................................

sf_rst_ext <- extend_AOI_buffer(
  dat_gps_tcx = res_tcx,
  buffer_in_meters = 1000,
  CRS = 4326,
  verbose = TRUE
)
sf_rst_ext

## End(Not run)
```

---

fitbit_data_type_by_date

Process pre-downloaded Fitbit data for Blood Oxygen Saturation, Heart Rate Variability, Breathing Rate, Temperature and Cardio Fitness Score (or VO2 Max) by Date

---

Description

Process pre-downloaded Fitbit data for Blood Oxygen Saturation, Heart Rate Variability, Breathing Rate, Temperature and Cardio Fitness Score (or VO2 Max) by Date

Usage

```
fitbit_data_type_by_date(data, type = "spo2", plot = FALSE)
```

Arguments

data	a list object containing the pre-downloaded Fitbit data for the specified type. This is the parsed JSON response from the Fitbit Web API endpoint for that type and date. See the 'references' section for the expected structure of each data type.
type	a character string specifying the fitbit data type. One of 'spo2', 'hrv', 'br', 'temp', 'cardioscore'. See the 'details' and 'references' sections for more information
plot	a boolean. If TRUE then the minutes data will be plotted. This parameter is applicable only to the 'spo2' and 'hrv' types because they return minute data (see the details section for more information). The remaining types ('br', 'temp', 'cardioscore') return daily data.

Details

'spo2' (*Blood Oxygen Saturation*) This endpoint returns the SpO2 intraday data for a single date. SpO2 applies specifically to a user's "main sleep", which is the longest single period of time asleep on a given date. Spo2 values are calculated on a 5-minute exponentially-moving average

'hrv' (*Heart Rate Variability*) This endpoint returns the Heart Rate Variability (HRV) intraday data for a single date. HRV data applies specifically to a user's "main sleep", which is the longest single period of time asleep on a given date. It measures the HRV rate at various times and returns Root Mean Square of Successive Differences (rmssd), Low Frequency (LF), High Frequency (HF), and Coverage data for a given measurement. Rmssd measures short-term variability in your heart rate while asleep. LF and HF capture the power in interbeat interval fluctuations within either high frequency or low frequency bands. Finally, coverage refers to data completeness in terms of the number of interbeat intervals

'br' (*Breathing Rate*) This endpoint returns intraday breathing rate data for a specified date. It measures the average breathing rate throughout the day and categories your breathing rate by sleep stage. Sleep stages vary between light sleep, deep sleep, REM sleep, and full sleep

'temp' (*Temperature*) This endpoint returns the Temperature (Skin) data for a single date. It only returns a value for dates on which the Fitbit device was able to record Temperature (skin) data. Temperature (Skin) data applies specifically to a user's "main sleep", which is the longest single period of time asleep on a given date

'cardioscore' (*Cardio Fitness Score or VO2 Max*) The Cardio Fitness Score (also known as VO2 Max) endpoints are used for querying the maximum or optimum rate at which the user's heart, lungs, and muscles can effectively use oxygen during exercise

If the 'type' parameter is one of 'spo2' or 'hrv' and the 'plot' parameter is set to TRUE then the results will appear as a line plot. In case of 'hrv' a multiplot with the following variables will be displayed:

'rmssd' *The Root Mean Square of Successive Differences (RMSSD) between heart beats. It measures short-term variability in the user's heart rate in milliseconds (ms)*

'coverage' *Data completeness in terms of the number of interbeat intervals*

'hf' *The power in interbeat interval fluctuations within the high frequency band (0.15 Hz - 0.4 Hz)*

'lf' *The power in interbeat interval fluctuations within the low frequency band (0.04 Hz - 0.15 Hz)*

Value

a data.frame

References

https://dev.fitbit.com/build/reference/web-api/intraday/get-spo2-intraday-by-date/

https://dev.fitbit.com/build/reference/web-api/intraday/get-hrv-intraday-by-date/

https://dev.fitbit.com/build/reference/web-api/intraday/get-br-intraday-by-date/

https://dev.fitbit.com/build/reference/web-api/temperature/get-temperature-skin-summary-by-date

https://dev.fitbit.com/build/reference/web-api/cardio-fitness-score/get-vo2max-summary-by-date/

Examples

```
## Not run:

require(fitbitViz)

# 'spo2_data' is the parsed JSON list from the Fitbit SpO2 intraday endpoint for a single date
# (i.e. the result of jsonlite::fromJSON() on the API response for that date)

res_type <- fitbit_data_type_by_date(
  data = spo2_data,
  type = "spo2",
  plot = TRUE
)
res_type

## End(Not run)
```

---

gps_lat_lon_to_LINESTRING

Convert the GPS, TCX data to a LINESTRING

Description

Convert the GPS, TCX data to a LINESTRING

Usage

```
gps_lat_lon_to_LINESTRING(
  dat_gps_tcx,
  CRS = 4326,
  verbose = FALSE,
  time_split_asc_desc = NULL
)
```

Arguments

dat_gps_tcx	this parameter corresponds to the output data.table of the 'GPS_TCX_data()' function
CRS	an integer specifying the Coordinates Reference System. The recommended value for this data is 4326 (which is also the default value)
verbose	a boolean. If TRUE then information will be printed out in the console
time_split_asc_desc	if NULL then the maximum altitude coordinates point will be used as a split point of the route, otherwise the user can give a lubridate 'hours-minutes-seconds' object such as: lubridate::hms('17:05:00')

Details

Separate the Ascending and Descending coordinate points into 2 groups and give a different color to the Ascending and Descending routes

Value

an object of class list

Examples

```
## Not run:

require(fitbitViz)

# export the TCX file from the Fitbit website or app for the desired activity

res_tcx <- GPS_TCX_data(
  tcx_file = "/path/to/activity.tcx",
  time_zone = "Europe/Athens",
  verbose = TRUE
)
str(res_tcx)


# ..............................................................
# By using using the maximum altitude as a split point of the route
# ..............................................................

linestring_dat_init <- gps_lat_lon_to_LINESTRING(
```

```
  dat_gps_tcx = res_tcx,
  CRS = 4326,
  time_split_asc_desc = NULL,
  verbose = TRUE
)

# ..........................................................
# By using a customized split of the route (ascending, descending)
# ..........................................................

linestring_dat_lubr <- gps_lat_lon_to_LINESTRING(
  dat_gps_tcx = res_tcx,
  CRS = 4326,
  time_split_asc_desc = lubridate::hms("17:05:00"),
  verbose = TRUE
)

## End(Not run)
```

GPS_TCX_data *The GPS-TCX data as a formated data.table*

Description

The GPS-TCX data as a formated data.table

Usage

```
GPS_TCX_data(tcx_file, time_zone = "Europe/Athens", verbose = FALSE)
```

Arguments

tcx_file	a character string specifying the path to a TCX file exported from the Fitbit app or website. TCX (Training Center XML) files can be exported from the Fitbit website under account settings or from the Fitbit app by sharing a specific activity.
time_zone	a character string specifying the time zone parameter ('tz') as is defined in the 'lubridate::ymd_hms()' function
verbose	a boolean. If TRUE then information will be printed out in the console

Value

either NULL or an object of class data.table

Examples

```
## Not run:

require(fitbitViz)

# export the TCX file from the Fitbit website or app for the desired activity
# then pass the file path to GPS_TCX_data()

res_tcx <- GPS_TCX_data(
  tcx_file = "/path/to/activity.tcx",
  time_zone = "Europe/Athens",
  verbose = TRUE
)
str(res_tcx)

## End(Not run)
```

---

heart_rate_heatmap *Heart Rate Intraday Heatmap (by extracting the 'min.', 'median' and 'max.' values of the day)*

Description

Heart Rate Intraday Heatmap (by extracting the 'min.', 'median' and 'max.' values of the day)

Usage

```
heart_rate_heatmap(heart_rate_intraday_data, angle_x_axis = 0)
```

Arguments

heart_rate_intraday_data	a list object specifying the intraday heart rate data (this is one of the sublists returned from the 'heart_rate_time_series' function)
angle_x_axis	an integer specifying the angle of the x-axis labels. The default values is 0 (it can take for instance values such as 45, 90 etc.)

Value

a plot object of class ggplot2

Examples

```
## Not run:

require(fitbitViz)

# ........................................
# first compute the heart rate intraday data
```

```
# ......................................

USER_ID <- "99xxxx"
token <- "my_long_web_api_token"

heart_dat <- heart_rate_time_series(
  user_id = USER_ID,
  token = token,
  date_start = "2021-03-09",
  date_end = "2021-03-16",
  time_start = "00:00",
  time_end = "23:59",
  detail_level = "1min",
  ggplot_intraday = TRUE,
  verbose = TRUE,
  show_nchar_case_error = 135
)

# ......................................
# use the heart-rate-intraday data as input
# to the 'heart_rate_heatmap' function
# ......................................

hrt_heat <- heart_rate_heatmap(
  heart_rate_intraday_data = heart_dat$heart_rate_intraday,
  angle_x_axis = 0
)
hrt_heat

## End(Not run)
```

`heart_rate_time_series` *heart rate activity time series*

Description

heart rate activity time series

Usage

```
heart_rate_time_series(
  heart_rate_intraday_list,
  heart_rate = NULL,
  detail_level = "1min",
  ggplot_intraday = FALSE,
  ggplot_ncol = NULL,
  ggplot_nrow = NULL,
  verbose = FALSE
)
```

Arguments

heart_rate_intraday_list	a named list of data.tables with the intraday heart rate data. Each element is a data.table with columns `time` (character, format "YYYY-MM-DD HH:MM:SS") and `value` (numeric heart rate). Names must be dates in "YYYY-MM-DD" format. This is the pre-downloaded intraday heart rate data.
heart_rate	either NULL or a data.frame with the daily heart rate summary data
detail_level	a character string specifying the detail level of the heart rate time series. It can be either '1min' or '1sec', for 1-minute and 1-second intervals
ggplot_intraday	a boolean. If TRUE then the ggplot of the heart rate time series will be returned too
ggplot_ncol	either NULL or an integer specifying the number of columns of the output ggplot
ggplot_nrow	either NULL or an integer specifying the number of rows of the output ggplot
verbose	a boolean. If TRUE then information will be printed out in the console

Value

an object of class list

Examples

```
## Not run:

require(fitbitViz)

# load pre-downloaded intraday heart rate data (one data.table per day, named by date)
# heart_rate_intraday_list is a named list where each element has columns 'time' and 'value'

heart_dat <- heart_rate_time_series(
  heart_rate_intraday_list = heart_rate_intraday_list,
  heart_rate = heart_rate_daily,
  detail_level = "1min",
  ggplot_intraday = TRUE,
  verbose = TRUE
)
heart_dat$plt
heart_dat$heart_rate
heart_dat$heart_rate_intraday

## End(Not run)
```

heart_rate_variability_sleep_time *Heart Rate Variability during Sleep Time (the root mean square of successive differences)*

Description

‘r lifecycle::badge("deprecated")‘

This function was deprecated, so please use the 'fitbit_data_type_by_date()' function instead with the 'type' parameter set to 'hrv' (Heart Rate Variability). See the documentation and the example section of the 'fitbit_data_type_by_date()' function for more details.

Usage

```
heart_rate_variability_sleep_time(
  heart_rate_data,
  sleep_begin = "00H 40M 0S",
  sleep_end = "08H 00M 0S",
  ggplot_hr_var = TRUE,
  angle_x_axis = 45
)
```

Arguments

heart_rate_data	a list object. This is the output of the 'heart_rate_time_series()' function
sleep_begin	a character string specifying the begin of the sleep time. For instance, the time "00H 40M 0S" where the input order is 'hours-minutes-seconds' and the format corresponds to the 'lubridate::hms()' function
sleep_end	a character string specifying the end of the sleep time. For instance, the time "08H 00M 0S" where the input order is 'hours-minutes-seconds' and the format corresponds to the 'lubridate::hms()' function
ggplot_hr_var	a boolean. If TRUE then the ggplot of the heart rate variability will be returned
angle_x_axis	an integer specifying the angle of the x-axis labels. The default values is 45 (it can take for instance values such as 0, 90 etc.)

Details

I use the '1min' rather than the '1sec' interval because it is consistent (it shows the 1-minute differences), whereas in case of '1sec' the difference between observations varies between 1 second and less than 60 seconds

This function calculates the root mean square of successive differences (RMSSD) and a higher heart rate variability is linked with better health

Based on the Fitbit application information weblink and the Wikipedia article (https://en.wikipedia.org/wiki/Heart_rate_varial the heart rate variability is computed normally in ms (milliseconds)

Value

an object of class list

Examples

```
## Not run:

require(fitbitViz)

# ........................................
# first compute the heart rate intraday data
# ........................................

USER_ID <- "99xxxx"
token <- "my_long_web_api_token"

heart_dat <- heart_rate_time_series(
  user_id = USER_ID,
  token = token,
  date_start = "2021-03-09",
  date_end = "2021-03-16",
  time_start = "00:00",
  time_end = "23:59",
  detail_level = "1min",
  ggplot_intraday = TRUE,
  verbose = TRUE,
  show_nchar_case_error = 135
)

# ......................
# heart rate variability
# ......................

hrt_rt_var <- heart_rate_variability_sleep_time(
  heart_rate_data = heart_dat,
  sleep_begin = "00H 40M 0S",
  sleep_end = "08H 00M 0S",
  ggplot_hr_var = TRUE,
  angle_x_axis = 25
)

hrt_rt_var

## End(Not run)
```

---

leafGL_point_coords *Create a Leafet map (including information pop-ups)*

Description

Create a Leafet map (including information pop-ups)

Usage

```
leafGL_point_coords(
  dat_gps_tcx,
  color_points_column = "AltitudeMeters",
  provider = leaflet::providers$Esri.WorldImagery,
  option_viewer = rstudioapi::viewer,
  CRS = 4326
)
```

Arguments

dat_gps_tcx	this parameter corresponds to the output data.table of the 'GPS_TCX_data()' function
color_points_column	a character string specifying the column of the output data.table ('GPS_TCX_data()' function) that is used in the map-markers. The default value is 'AltitudeMeters' but it can be any column of type numeric
provider	either a character string specifying a leaflet provider (such as 'Esri.WorldImagery') or a direct call to the leaflet provider list (such as leaflet::providers$Esri.WorldImagery). The default value is leaflet::providers$Esri.WorldImagery
option_viewer	either NULL or rstudioapi::viewer. If NULL then the output map will be shown in the web browser
CRS	an integer specifying the Coordinates Reference System. The recommended value for this data is 4326 (which is also the default value)

Value

a leaflet map of class 'leaflet'

Examples

```
## Not run:

require(fitbitViz)

# export the TCX file from the Fitbit website or app for the desired activity

res_tcx <- GPS_TCX_data(
  tcx_file = "/path/to/activity.tcx",
  time_zone = "Europe/Athens",
  verbose = TRUE
)
str(res_tcx)


# ......................
# then visualize the data
# ......................
```

```
res_lft <- leafGL_point_coords(
  dat_gps_tcx = res_tcx,
  color_points_column = "AltitudeMeters",
  provider = leaflet::providers$Esri.WorldImagery,
  option_viewer = rstudioapi::viewer,
  CRS = 4326
)
res_lft

## End(Not run)
```

`rayshader_3d_DEM` *Rayshader 3-dimensional using the Copernicus DEM elevation data*

Description

Rayshader 3-dimensional using the Copernicus DEM elevation data

Usage

```
rayshader_3d_DEM(
  rst_buf,
  rst_ext,
  linestring_ASC_DESC = NULL,
  elevation_sample_points = NULL,
  zoom = 0.5,
  windowsize = c(1600, 1000),
  add_shadow_rescale_original = FALSE,
  verbose = FALSE
)
```

Arguments

`rst_buf`	this parameter corresponds to the 'sfc_obj' object of the 'extend_AOI_buffer()' function
`rst_ext`	this parameter corresponds to the 'raster_obj_extent' object of the 'extend_AOI_buffer()' function
`linestring_ASC_DESC`	If NULL then this parameter will be ignored. Otherwise, it can be an 'sf' object or a named list of length 2 (that corresponds to the output of the 'gps_lat_lon_to_LINESTRING()' function)
`elevation_sample_points`	if NULL then this parameter will be ignored. Otherwise, it corresponds to a data.table with column names 'latitude', 'longitude' and 'AltitudeMeters'. For instance, it can consist of 3 or 4 rows that will be displayed as vertical lines in the 3-dimensionsal map to visualize sample locations of the route (the latitudes and longitudes must exist in the output data.table of the 'GPS_TCX_data()' function)

zoom	a float number. Lower values increase the 3-dimensional DEM output. The default value is 0.5
windowsize	a numeric vector specifying the window dimensions (x,y) of the output 3-dimensional map. The default vector is c(1600, 1000)
add_shadow_rescale_original	a boolean. If TRUE, then 'hillshade' will be scaled to match the dimensions of 'shadowmap'. See also the 'rayshader::add_shadow()' function for more information.
verbose	a boolean. If TRUE then information will be printed out in the console

Value

it doesn't return an object but it displays a 3-dimensional 'rayshader' object

References

https://www.tylermw.com/a-step-by-step-guide-to-making-3d-maps-with-satellite-imagery-in-r/

Examples

```
## Not run:

require(fitbitViz)

# export the TCX file from the Fitbit website or app for the desired activity

res_tcx <- GPS_TCX_data(
  tcx_file = "/path/to/activity.tcx",
  time_zone = "Europe/Athens",
  verbose = TRUE
)
str(res_tcx)

# ................................................
# then compute the sf-object buffer and raster-extend
# ................................................

sf_rst_ext <- extend_AOI_buffer(
  dat_gps_tcx = res_tcx,
  buffer_in_meters = 1000,
  CRS = 4326,
  verbose = TRUE
)
sf_rst_ext

# ..........................................................
# Download the Copernicus DEM 30m elevation data because it has
# a better resolution, it takes a bit longer to download because
# the .tif file size is bigger
# ..........................................................
```

```
dem_dir <- tempdir()
# dem_dir

dem30 <- CopernicusDEM::aoi_geom_save_tif_matches(
  sf_or_file = sf_rst_ext$sfc_obj,
  dir_save_tifs = dem_dir,
  resolution = 30,
  crs_value = 4326,
  threads = parallel::detectCores(),
  verbose = TRUE
)

TIF <- list.files(dem_dir, pattern = ".tif", full.names = T)
# TIF

if (length(TIF) > 1) {
  # ................................................
  # create a .VRT file if I have more than 1 .tif files
  # ................................................

  file_out <- file.path(dem_dir, "VRT_mosaic_FILE.vrt")

  vrt_dem30 <- create_VRT_from_dir(
    dir_tifs = dem_dir,
    output_path_VRT = file_out,
    verbose = TRUE
  )
}

if (length(TIF) == 1) {
  # ................................................
  # if I have a single .tif file keep the first index
  # ................................................

  file_out <- TIF[1]
}

raysh_rst <- crop_DEM(
  tif_or_vrt_dem_file = file_out,
  sf_buffer_obj = sf_rst_ext$sfc_obj,
  verbose = TRUE
)

# terra::plot(raysh_rst)


# ..............................................................
# create the 'elevation_sample_points' data.table parameter based
# on the min., middle  and max. altitude of the 'res_tcx' data
# ..............................................................

idx_3m <- c(
  which.min(res_tcx$AltitudeMeters),
```

```
  as.integer(length(res_tcx$AltitudeMeters) / 2),
  which.max(res_tcx$AltitudeMeters)
)

cols_3m <- c("latitude", "longitude", "AltitudeMeters")
dat_3m <- res_tcx[idx_3m, ..cols_3m]
# dat_3m

# ..............................................................
# Split the route in 2 parts based on the maximum altitude value
# ..............................................................

linestring_dat <- gps_lat_lon_to_LINESTRING(
  dat_gps_tcx = res_tcx,
  CRS = 4326,
  time_split_asc_desc = NULL,
  verbose = TRUE
)

# ..................................................
# Conversion of the 'SpatRaster' to a raster object
# because the 'rayshader' package accepts only rasters
# ..................................................

rst_obj <- raster::raster(raysh_rst)
raster::projection(rst_obj) <- terra::crs(raysh_rst, proj = TRUE)


# ..................................
# open the 3-dimensional rayshader map
# ..................................

ray_out <- rayshader_3d_DEM(
  rst_buf = rst_obj,
  rst_ext = sf_rst_ext$raster_obj_extent,
  linestring_ASC_DESC = linestring_dat,
  elevation_sample_points = dat_3m,
  zoom = 0.5,
  windowsize = c(1600, 1000),
  add_shadow_rescale_original = FALSE,
  verbose = TRUE
)

## End(Not run)
```

`sleep_single_day` *Sleep Data of single day*

Description

Sleep Data of single day

Usage

```
sleep_single_day(
  sleep_data,
  date = "2021-03-09",
  ggplot_color_palette = "ggsci::blue_material",
  verbose = FALSE
)
```

Arguments

sleep_data	a list object containing the pre-downloaded sleep data for the specified date. This is the parsed JSON response from the Fitbit sleep endpoint (e.g. `jsonlite::fromJSON()` on the API response for `/1.2/user/[user-id]/sleep/date/[date]/[date].json`). The list must have a `$sleep` element.
date	a character string specifying the Date for which the sleep data should be returned. For instance, the date '2021-12-31' where the input order is 'year-month-day'
ggplot_color_palette	a character string specifying the color palette to be used. For a full list of palettes used in the ggplot see: https://pmassicotte.github.io/paletteer_gallery/ The following color-palettes were tested and work well: "rcartocolor::Purp", "rcartocolor::Teal"
verbose	a boolean. If TRUE then information will be printed out in the console

Value

an object of class list

Examples

```
## Not run:

require(fitbitViz)

# 'sleep_data_day' is the parsed JSON list for a single date from the Fitbit sleep endpoint

lst_out <- sleep_single_day(
  sleep_data = sleep_data_day,
  date = "2021-03-09",
  ggplot_color_palette = "ggsci::blue_material",
  verbose = TRUE
)
str(lst_out)

## End(Not run)
```

sleep_time_series *Sleep Data Time Series*

Description

Sleep Data Time Series

Usage

```
sleep_time_series(
  sleep_data_list,
  ggplot_color_palette = "ggsci::blue_material",
  ggplot_ncol = NULL,
  ggplot_nrow = NULL,
  verbose = FALSE
)
```

Arguments

sleep_data_list	a named list of pre-downloaded sleep data, one element per day. Names must be dates in "YYYY-MM-DD" format. Each element is the parsed JSON response from the Fitbit sleep endpoint for that date (the same structure as the `sleep_data` parameter of `sleep_single_day()`).
ggplot_color_palette	a character string specifying the color palette to be used. For a full list of palettes used in the ggplot see: https://pmassicotte.github.io/paletteer_gallery/ The following color-palettes were tested and work well: "rcartocolor::Purp", "rcartocolor::Teal"
ggplot_ncol	either NULL or an integer specifying the number of columns of the output ggplot
ggplot_nrow	either NULL or an integer specifying the number of rows of the output ggplot
verbose	a boolean. If TRUE then information will be printed out in the console

Value

an object of class list

Examples

```
## Not run:

require(fitbitViz)

# 'sleep_data_list' is a named list (names = dates "YYYY-MM-DD")
# of parsed Fitbit sleep endpoint responses

sleep_ts <- sleep_time_series(
```

```
  sleep_data_list = sleep_data_list,
  ggplot_color_palette = "ggsci::blue_material",
  verbose = TRUE
)

sleep_ts$plt_lev_segments
sleep_ts$plt_lev_heatmap
sleep_ts$heatmap_data


# ........................................
# (option to) save the ggplot to a .png file
# ........................................

png_file <- tempfile(fileext = ".png")

ggplot2::ggsave(
  filename = png_file,
  plot = sleep_ts$plt_lev_segments,
  device = "png",
  scale = 1,
  width = 35,
  height = 25,
  limitsize = TRUE
)

## End(Not run)
```

Index