# Package 'fitbitViz'

March 7, 2022

**Type** Package

**Title** 'Fitbit' Visualizations

**Version** 1.0.4

**Date** 2022-03-07

**Maintainer** Lampros Mouselimis <mouselimislampros@gmail.com>

**URL** https://github.com/mlampros/fitbitViz

**Description** Connection to the 'Fitbit' Web API <https://dev.fitbit.com/build/reference/web-api/> by including '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

**SystemRequirements** update: apt-get -y update (deb)

**Depends** R(>= 3.5)

**Imports** glue, httr, jsonlite, ggplot2, lubridate, patchwork, data.table, stats, viridis, scales, ggthemes, varian, paletteer, XML, hms, leaflet, sf, rstudioapi, grDevices, leafgl, raster, terra, magrittr, rayshader, utils

**Suggests** CopernicusDEM, testthat (>= 3.0.0), knitr, rmarkdown, DT, rgl, magick, rgdal (>= 1.5-23)

**RoxygenNote** 7.1.2

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**NeedsCompilation** no

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

**Repository** CRAN

**Date/Publication** 2022-03-07 15:10:02 UTC

# R topics documented:



---

`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)

#.........................
# first extract the log-id(s)
```

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

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

log_id = extract_LOG_ID(user_id = USER_ID,
                        token = token,
                        after_Date = '2021-03-13',
                        limit = 10,
                        sort = 'asc',
                        verbose = TRUE)
str(log_id)

#..............................
# then return the gps-ctx data.table
#..............................

res_tcx = GPS_TCX_data(log_id = log_id,
                       user_id = USER_ID,
                       token = token,
                       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)

#.........................
# first extract the log-id(s)
#.........................

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

log_id = extract_LOG_ID(user_id = USER_ID,
                        token = token,
                        after_Date = '2021-03-13',
                        limit = 10,
                        sort = 'asc',
                        verbose = TRUE)
str(log_id)

#...............................
# then return the gps-ctx data.table
#...............................

res_tcx = GPS_TCX_data(log_id = log_id,
                       user_id = USER_ID,
                       token = token,
                       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)
```

---

`extract_LOG_ID` *Extract the log-id (it's possible that I receive more than one id)*

---

## Description

Extract the log-id (it's possible that I receive more than one id)

## Usage

```
extract_LOG_ID(
  user_id,
  token,
  after_Date = "2021-03-13",
  limit = 10,
  sort = "asc",
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `user_id` | a character string specifying the encoded ID of the user. For instance '99xxxx' of the following URL 'https://www.fitbit.com/user/99xxxx' of the user's account corresponds to the 'user_id' |
| `token` | a character string specifying the secret token that a user receives when registers a new application in https://dev.fitbit.com/apps |
| `after_Date` | a character string specifying the Date after which the log-ids will be returned. For instance, the date '2021-12-31' where the input order is 'year-month-day' |
| `limit` | an integer specifying the total of log-id's to return. The default value is 10 |
| `sort` | a character string specifying the order ('asc', 'desc') based on which the output log-id's should be sorted |
| `verbose` | a boolean. If TRUE then information will be printed out in the console |

## Value

an integer specifying the log ID

## Examples

```
## Not run:

require(fitbitViz)

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

log_id = extract_LOG_ID(user_id = USER_ID,
                        token = token,
                        after_Date = '2021-03-13',
                        limit = 10,
                        sort = 'asc',
                        verbose = TRUE)
log_id


## 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)

#.........................
# first extract the log-id(s)
#.........................

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

log_id = extract_LOG_ID(user_id = USER_ID,
                        token = token,
                        after_Date = '2021-03-13',
                        limit = 10,
                        sort = 'asc',
                        verbose = TRUE)
str(log_id)

#..............................
# then return the gps-ctx data.table
#..............................

res_tcx = GPS_TCX_data(log_id = log_id,
                       user_id = USER_ID,
                       token = token,
                       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(
  log_id,
  user_id,
  token,
  time_zone = "Europe/Athens",
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| log_id | the returned log-id of the 'extract_LOG_ID()' function |
| user_id | a character string specifying the encoded ID of the user. For instance '99xxxx' of the following URL 'https://www.fitbit.com/user/99xxxx' of the user's account corresponds to the 'user_id' |
| token | a character string specifying the secret token that a user receives when registers a new application in https://dev.fitbit.com/apps |
| 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)

#.........................
# first extract the log-id(s)
#.........................

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

log_id = extract_LOG_ID(user_id = USER_ID,
                        token = token,
                        after_Date = '2021-03-13',
                        limit = 10,
                        sort = 'asc',
                        verbose = TRUE)
str(log_id)

#..............................
# then return the gps-ctx data.table
#..............................

res_tcx = GPS_TCX_data(log_id = log_id,
                       user_id = USER_ID,
                       token = token,
                       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(
  user_id,
  token,
  date_start,
  date_end,
  time_start = "00:00",
  time_end = "23:59",
  detail_level = "1min",
  ggplot_intraday = FALSE,
  ggplot_ncol = NULL,
  ggplot_nrow = NULL,
  verbose = FALSE,
  show_nchar_case_error = 135
)
```

## Arguments

| | |
|---|---|
| user_id | a character string specifying the encoded ID of the user. For instance '99xxxx' of the following URL 'https://www.fitbit.com/user/99xxxx' of the user's account corresponds to the 'user_id' |
| token | a character string specifying the secret token that a user receives when registers a new application in https://dev.fitbit.com/apps |
| date_start | a character string specifying a start Date. For instance, the date '2021-12-31' where the input order is 'year-month-day' |
| date_end | a character string specifying a end Date. For instance, the date '2021-12-31' where the input order is 'year-month-day' |
| time_start | a character string specifying the start time. For instance, the time '00:00' where the input order is 'hours-minutes' |
| time_end | a character string specifying the end time. For instance, the time '23:59' where the input order is 'hours-minutes' |
| 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 |
| show_nchar_case_error | an integer that specifies the number of characters that will be returned in case on an error. The default value is 135 characters. |

## Value

an object of class list

## Examples

```
## Not run:

require(fitbitViz)

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_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

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

## 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)

#.........................
# first extract the log-id(s)
#.........................

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

log_id = extract_LOG_ID(user_id = USER_ID,
                        token = token,
                        after_Date = '2021-03-13',
                        limit = 10,
                        sort = 'asc',
                        verbose = TRUE)
str(log_id)

#..............................
# then return the gps-ctx data.table
#..............................

res_tcx = GPS_TCX_data(log_id = log_id,
                       user_id = USER_ID,
                       token = token,
                       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)

#.........................
# first extract the log-id(s)
#.........................

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

log_id = extract_LOG_ID(user_id = USER_ID,
                        token = token,
                        after_Date = '2021-03-13',
                        limit = 10,
                        sort = 'asc',
                        verbose = TRUE)
str(log_id)

#..............................
# then return the gps-ctx data.table
#..............................

res_tcx = GPS_TCX_data(log_id = log_id,
                       user_id = USER_ID,
                       token = token,
```

```
                      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(
  user_id,
  token,
  date = "2021-03-09",
  ggplot_color_palette = "ggsci::blue_material",
  show_nchar_case_error = 135,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `user_id` | a character string specifying the encoded ID of the user. For instance '99xxxx' of the following URL 'https://www.fitbit.com/user/99xxxx' of the user's account corresponds to the 'user_id' |
| `token` | a character string specifying the secret token that a user receives when registers a new application in https://dev.fitbit.com/apps |
| `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" |
| `show_nchar_case_error` | an integer that specifies the number of characters that will be returned in case on an error. The default value is 135 characters. |
| `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)

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

lst_out = sleep_single_day(user_id = USER_ID,
                           token = token,
                           date = '2021-03-09',
                           ggplot_color_palette = 'ggsci::blue_material',
                           show_nchar_case_error = 135,
                           verbose = TRUE)
str(lst_out)


## End(Not run)
```

---

`sleep_time_series` *Sleep Data Time Series*

---

## Description

Sleep Data Time Series

## Usage

```
sleep_time_series(
  user_id,
  token,
  date_start,
  date_end,
  ggplot_color_palette = "ggsci::blue_material",
  ggplot_ncol = NULL,
  ggplot_nrow = NULL,
  show_nchar_case_error = 135,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| `user_id` | a character string specifying the encoded ID of the user. For instance '99xxxx' of the following URL 'https://www.fitbit.com/user/99xxxx' of the user's account corresponds to the 'user_id' |
| `token` | a character string specifying the secret token that a user receives when registers a new application in https://dev.fitbit.com/apps |

date_start — a character string specifying the start Date for which the sleep data should be returned. For instance, the date '2021-12-31' where the input order is 'year-month-day'

date_end — a character string specifying the end 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"

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

show_nchar_case_error — an integer that specifies the number of characters that will be returned in case on an error. The default value is 135 characters.

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)

#.....................................
# first compute the sleep time time series
#.....................................

USER_ID = '99xxxx'
token = 'my_long_web_api_token'

sleep_ts = sleep_time_series(user_id = USER_ID,
                             token = token,
                             date_start = '2021-03-09',
                             date_end = '2021-03-16',
                             ggplot_color_palette = 'ggsci::blue_material',
                             show_nchar_case_error = 135,
                             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