For the rest of this blog post I’ll assume that the following variables are defined and that you intend to use the functions of the **fitbitViz** package from the R Console (or RStudio IDE):

#..................

# parameter setting #..................

USER\_ID = 'My user-id' # Specify here your 'user-id'

token = "My token" # Specify here your 'token'

WEEK = 11 # for this use case pick the 11th week of the year 2021

num\_character\_error = 135 # print that many character in case of an error

weeks\_2021 = fitbitViz:::split\_year\_in\_weeks(year = 2021) # split a year in weeks

# Start the week at monday

date\_start = lubridate::floor\_date(lubridate::ymd(weeks\_2021[WEEK]), unit = 'weeks') + 1

# Add 6 days to the 'date\_start' variable to come to a 7-days plot date\_end = date\_start + 6

sleep\_time\_begins = "00H 40M 0S" sleep\_time\_ends = "08H 00M 0S"

VERBOSE = FALSE # disable verbosity

The previous code snippet uses one week of my personal *Fitbit* data (the *11th week of 2021*) to plot my

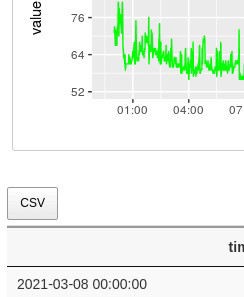
**heart rate time series heart rate heatmap**

**heart rate variability during sleep time sleep time series**

**GPS data of outdoor activities 3-dimensional map of activities**

The pre-processed data of all these functions are also available to download by clicking on the

**CSV** buttons,



# heart rate time series

The **heart\_rate\_time\_series()** function takes the **user-id**, **token**, the **start-** and **end-dates**, the **start-** and **end-time**, the **detail level** (1 minute) and returns the **heart rate time series**. Each output plot (of the *multiplot*) includes in the **x-axis** the **time** and in the **y-axis** the **heart rate value**. The highest heart rate value (peak) of the day is highlighted using a vertical and horizontal **blue** line,

#.......................

# heart rate time series #.......................

heart\_dat = fitbitViz::heart\_rate\_time\_series(user\_id = USER\_ID,

token = token, date\_start =

as.character(date\_start), as.character(date\_end),

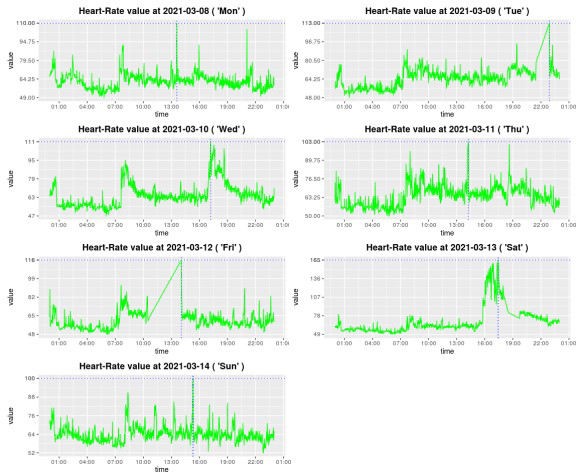
num\_character\_error) heart\_dat$plt

date\_end =

time\_start = '00:00',

time\_end = '23:59', detail\_level = '1min', ggplot\_intraday = TRUE, ggplot\_ncol = 2,

ggplot\_nrow = 4, verbose = VERBOSE, show\_nchar\_case\_error =



# heart rate heatmap

The **heart rate heatmap** shows the **min**, **median** and **max** heart rate Levels in the **y-axis** for each day of the specified week (**x-axis**). As the legend shows, the displayed values range from 40 to 220 and higher values appear in *purple* or *orange* color,

#............................

# heart rate intraday heatmap

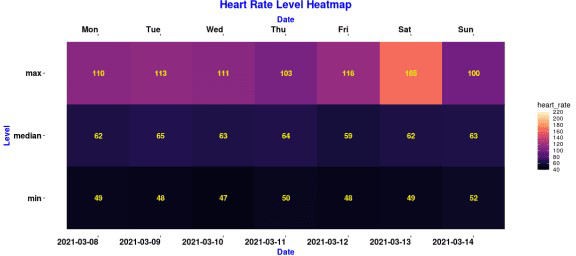
#............................

heart\_intra = heart\_dat$heart\_rate\_intraday

hrt\_heat = fitbitViz::heart\_rate\_heatmap(heart\_rate\_intraday\_data = heart\_intra,

angle\_x\_axis = 0)

hrt\_heat



# heart rate variability during sleep time

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

#.......................

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

hrt\_rt\_var = fitbitViz::heart\_rate\_variability\_sleep\_time(heart\_ rate\_data = heart\_dat,

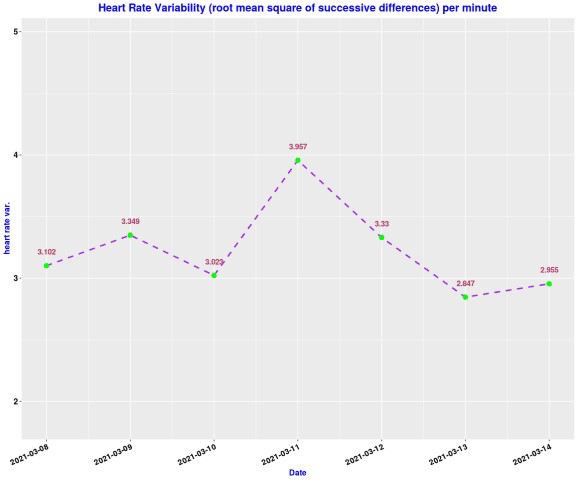
sleep\_time\_begins, sleep\_time\_ends,

= TRUE,

sleep\_begin = sleep\_end = ggplot\_hr\_var angle\_x\_axis

= 25)

hrt\_rt\_var$hr\_var\_plot



# sleep time series

The **sleep time series** visualization is similar to the *Fitbit Mobile* Visualization and in the **x-axis** shows the specified by the user **sleep time interval** whereas in the **y-axis** shows the **sleep Levels** (*wake*, *rem*, *light*, *deep*). Lower levels like *deep sleep* appear in dark blue whereas higher levels like *wake* appear in light blue,

#.......................

# sleep data time series #.......................

sleep\_ts = fitbitViz::sleep\_time\_series(user\_id = USER\_ID,

token = token, date\_start =

as.character(date\_start),

date\_end =

as.character(date\_end), 'ggsci::blue\_material',

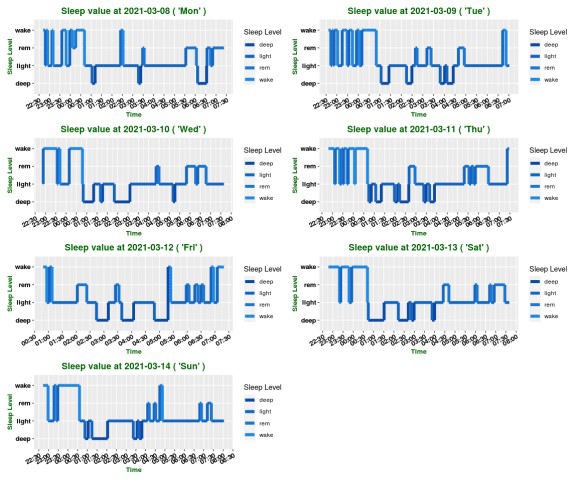
num\_character\_error,

ggplot\_color\_palette = ggplot\_ncol = 2,

ggplot\_nrow = 4, show\_nchar\_case\_error =

verbose = VERBOSE)

sleep\_ts$plt\_lev\_segments



# GPS data of outdoor activities

To make use of the *GPS data* from the Fitbit Application we have first to extract the **log-id** for a time interval after a specified *Date*,

#...................

# extract the log-id (required for the GPS data) #...................

log\_id = fitbitViz::extract\_LOG\_ID(user\_id = USER\_ID,

token = token, after\_Date =

as.character(date\_start),

# log\_id

limit = 10, sort = 'asc',

verbose = VERBOSE)

Once we have the *log-id* we can define the *time zone* of the route to receive all GPS data,

#....................................................

# return the gps-ctx data.table for the output log-id #....................................................

res\_tcx = fitbitViz::GPS\_TCX\_data(log\_id = log\_id,

user\_id = USER\_ID, token = token,

time\_zone = 'Europe/Athens', verbose = VERBOSE)

# res\_tcx

The following *Leaflet (Point Coordinates) Map* shows my outdoor activity during the *11th week of 2021* (the legend shows the elevation of the route),

#................................

# Create the Leaflet / LeafGL Map #................................

res\_lft = fitbitViz::leafGL\_point\_coords(dat\_gps\_tcx = res\_tcx,

color\_points\_column =

'AltitudeMeters', leaflet::providers$Esri.WorldImagery, rstudioapi::viewer,

provider = option\_viewer = CRS = 4326)

res\_lft



# 3-dimensional plots of activities

Another option of this package is to plot a route in 3-dimensional space. For this purpose we’ll use the rayshader package, which internally uses rgl (*OpenGL*). First, we have to extend the boundaries of our route for approximately *1.000 thousand meters* (adjust this value depending on your area of interest),

#...................................................

# compute the sf-object buffer and the raster-extend (1000 meters buffer)

#...................................................

sf\_rst\_ext = fitbitViz::extend\_AOI\_buffer(dat\_gps\_tcx = res\_tcx,

buffer\_in\_meters = 1000,

CRS = 4326,

verbose = VERBOSE)

# sf\_rst\_ext

Then for the extended area we will download Copernicus Digital Elevation Model (DEM) data.

The *Copernicus elevation data* come either in **30** or in **90** meter resolution. We will pick the *30* meter resolution product for this route. The **CopernicusDEM** is an R package, make sure that you have installed and configured the **awscli** Operating System Requirement if you intend to download and reproduce the next 3-dimensional map using the elevation data.

#..................................................................

# Download the Copernicus DEM 30m elevation data

# there is also the option to download the DEM 90m elevation data # which is of lower resolution but the image size is smaller which # means faster download

#..................................................................

dem\_dir = tempdir() # dem\_dir

dem30 = CopernicusDEM::aoi\_geom\_save\_tif\_matches(sf\_or\_file = sf\_rst\_ext$sfc\_obj,

dem\_dir,

parallel::detectCores(),

dir\_save\_tifs =

resolution = 30,

crs\_value = 4326, threads =

verbose = VERBOSE)

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 = CopernicusDEM::create\_VRT\_from\_dir(dir\_tifs = dem\_dir,

output\_path\_VRT =

file\_out,

}

if (length(TIF) == 1) {

verbose = VERBOSE)

#..................................................

# if I have a single .tif file keep the first index #..................................................

file\_out = TIF[1]

}

#.......................................

# crop the elevation DEM based on the

# coordinates extent of the GPS-CTX data #.......................................

raysh\_rst = fitbitViz::crop\_DEM(tif\_or\_vrt\_dem\_file = file\_out,

sf\_buffer\_obj = sf\_rst\_ext$sfc\_obj, CRS = 4326,

digits = 6, verbose = VERBOSE)

# sp::plot(raysh\_rst)

The GPS route that I use is an *ascending & descending* route therefore we can convert the GPS (TCX) data to a spatial *LINESTRING* by using the maximum altitude as a *split point* of the route to visualize the ascending route in *blue* and the descending in *red* (there is also the alternative to specify the split point based on time using the **time\_split\_asc\_desc** parameter),

linestring\_dat = fitbitViz::gps\_lat\_lon\_to\_LINESTRING(dat\_gps\_tcx = res\_tcx,

CRS = 4326,

time\_split\_asc\_desc = NULL, VERBOSE

verbose =

then we create the *‘elevation\_sample\_points’ data.table parameter* for the *3-dim* plot based on the *min.*, *middle* and *max.* altitude of the previously computed *‘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]

and finally we visualize the *3-dimensional Rayshader Map*,

snapshot\_rayshader\_path = file.path(tempdir(), 'rayshader\_img.png')

rgl::open3d(useNULL = TRUE) # this removes the second rgl-popup-window

fitbitViz::rayshader\_3d\_DEM(rst\_buf = raysh\_rst,

rst\_ext = sf\_rst\_ext$raster\_obj\_extent, rst\_bbx = sf\_rst\_ext$buffer\_bbox, linestring\_ASC\_DESC = linestring\_dat, elevation\_sample\_points = dat\_3m,

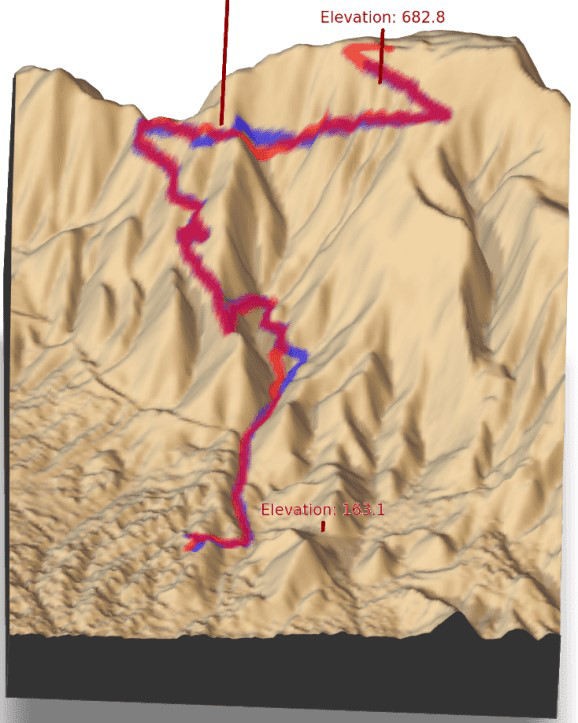
zoom = 0.3,

windowsize = c(1000, 800))

rgl::rgl.snapshot(snapshot\_rayshader\_path)

rgl::par3d(mouseMode = "trackball") # options: c("trackball", "polar", "zoom", "selecting")

rgl::rglwidget()



In the output map we observe

the *3 specified elevation vertical lines* (including their *altitude values* in meters) in *blue* color the *ascending* route

in *red* color the *descending* route

The attached map here is a screenshot.