So, [Strava’s heatmap](https://labs.strava.com/heatmap/#7.00/-120.90000/38.36000/hot/all) made [quite a stir](http://www.wired.co.uk/article/strava-military-bases-area-51-map-afghanistan-gchq-military). I decided to give it a try myself. I wanted to create some kind of “personal heatmap” of my runs, using Strava’s API. Also, combining the data with [Leaflet](http://leafletjs.com/) maps allows us to make use of the beautiful map tiles supported by Leaflet and to zoom and move the maps around – with the runs on it, of course.

So, let’s get started. First, you will need an access token for Strava’s API. As soon as you have the token, you have access to your own data.

Now, let’s load some packages and define functions for getting and handling the data. For the get.activities() function, I adapted R code that generates charts frpm Starva API:

|  |
| --- |
| library(rjson) |
|  | library(httr) |
|  | library(ggplot2) |
|  | library(scales) |
|  |  |
|  | if (interactive()) { |
|  | token <- readline(prompt="Enter Strava access token: ") |
|  | activities <- GET("https://www.strava.com/", path = "api/v3/activities", |
|  | query = list(access\_token = token, per\_page = 200)) |
|  | activities <- content(activities, "text") |
|  | activities <- fromJSON(activities) |
|  | activities <- lapply(activities, function(x) { |
|  | x[sapply(x, is.null)] <- NA |
|  | unlist(x) |
|  | }) |
|  | df <- data.frame(do.call("rbind", activities)) |
|  |  |
|  | df$distance <- as.numeric(as.character(df$distance)) / 1000 # m -> km |
|  | df$moving\_time <- as.numeric(as.character(df$moving\_time)) / 60 # s -> m |
|  | df$start\_date <- as.Date(df$start\_date) # yyyy-mm-dd |
|  | df$average\_speed <- as.numeric(as.character(df$average\_speed)) \* 3.6 # m/s -> km/h |
|  |  |
|  | ggplot(df, aes(x=distance, y=moving\_time)) + |
|  | geom\_point(size=1, aes(colour = type)) + |
|  | geom\_smooth(method=lm, aes(colour = type)) + |
|  | xlab("distance per activity (kilometers)") + |
|  | ylab("moving time (minutes)") |
|  | ggsave("/tmp/moving-time.png") |
|  |  |
|  | ggplot(df, aes(x=start\_date, y=average\_speed)) + |
|  | geom\_point(size=1, aes(colour = type)) + |
|  | scale\_x\_date(labels = date\_format("%Y-%m-%d")) + |
|  | xlab("date") + |
|  | ylab("average speed (kilometers per hour)") |
|  | ggsave("/tmp/speed.png") |
|  |  |
|  | } else |
|  | writeLines("Non interactive session, giving up") |

library(httr)  
library(rjson)  
library(OpenStreetMap)  
library(leaflet)  
library(scales)  
library(dplyr)  
  
token <- “”  
  
get.coord.df.from.stream <- function (stream.obj) {  
  data.frame(lat = sapply(stream.obj[[1]]$data, USE.NAMES = F, FUN = function (x) x[[1]]),  
             lon = sapply(stream.obj[[1]]$data, USE.NAMES = F, FUN = function (x) x[[2]]))  
}  
  
get.stream.from.activity <- function (act.id, token) {  
  stream <- GET(“https://www.strava.com/”,  
                path = paste0(“api/v3/activities/”, act.id, “/streams/latlng”),  
                query = list(access\_token = token))  
  content(stream)  
}  
  
get.activities <- function (token) {  
  activities <- GET(“https://www.strava.com/”, path = “api/v3/activities”,  
                    query = list(access\_token = token, per\_page = 200))  
  activities <- content(activities, “text”)  
  activities <- fromJSON(activities)  
  activities <- lapply(activities, function(x) {  
    x[sapply(x, is.null)] <- NA  
    unlist(x)  
  })  
  data.frame(do.call(“rbind”, activities))  
}  
  
get.multiple.streams <- function (act.ids, token) {  
  res.list <- list()  
  for (act.id.i in 1:length(act.ids)) {  
    if (act.id.i %% 5 == 0) cat(“Actitivy no.”, act.id.i, “of”, length(act.ids), “\n”)  
    stream <- get.stream.from.activity(act.ids[act.id.i], token)  
    coord.df <- get.coord.df.from.stream(stream)  
    res.list[[length(res.list) + 1]] <- list(act.id = act.ids[act.id.i],  
                                             coords = coord.df)  
  }  
  res.list  
}  
  
We have all the functions we need to get and parse the APIs output available now. Let’s apply them. The logic is: First, we get all activities. This dataframe has a column called ‘id’ which we can use to get all the raw data for all activities (called ‘streams’ in the Strava API). The function get.coord.df.from.stream() creates a dataframe with lat/lon coordinates for one stream.

activities <- get.activities(token)  
stream.list <- get.multiple.streams(activities$id, token)

We might want to get the boundaries of the cumulated set of all streams. We can use these boundaries as a bounding box for plotting the data. This means that all activities are going to be in the plotted map section.

all.lats <- unlist(sapply(stream.list, USE.NAMES = F, FUN = function (x) {  
  x$coords$lat  
}))  
all.lons <- unlist(sapply(stream.list, USE.NAMES = F, FUN = function (x) {  
  x$coords$lon  
}))  
lats.range <- range(all.lats)  
lons.range <- range(all.lons)

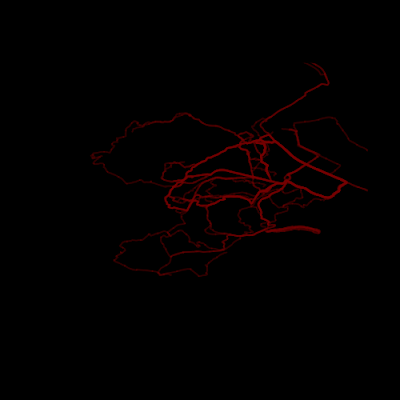
Alternatively, you can set your own bounding box. These are the boundaries for Stuttgart, Germany. One suggestion: to find your own boundaries, you can plot your first map and use the locator() function in RStudio, this is a very convenient way of getting coordinates by clicking.

lons.range <- c(9.156572, 9.237580)

lats.range <- c(48.74085, 48.82079)

We start by plotting the tracks in red against a black background. You can play around with the alpha value and the lwd parameter to change the appearance. By plotting several tracks over each other, thicker lines represent routes I took more often.

# Setting up the plot  
par(bg = “black”)  
plot(x = lons.range, y = lats.range, type = “n”, bty = “n”, xlab = “”, ylab = “”, xaxt = “n”, yaxt = “n”)  
  
# Plotting tracks one by one  
for (el in stream.list) {  
  lines(el$coords$lon, el$coords$lat,  
        col = alpha(“darkred”, .4), lwd = 2)  
}

[](https://i1.wp.com/4.bp.blogspot.com/-8JrmCBE97ug/WnFrJEYKLsI/AAAAAAAADFU/giHiu1fMDR4IgIeYDusvFklqdxTAlbo4QCLcBGAs/s1600/map_black_red.png?ssl=1)

A black-and-red plot of my runs through Stuttgart.

Now, this already looks quite nice and we see some kind of network through the city. But the city itself is missing. We need to get some map below the tracks. I am using the OpenStreetMap package for this. Note that getting the map tiles from the servers might take a long time and might fail in some cases. Loading time (and the resolution of the final map) will depend heavily on the ‘zoom’ parameter.

map <- openmap(c(max(lats.range), min(lons.range)),

               c(min(lats.range), max(lons.range)), type = “maptoolkit-topo”, zoom = 14)

transmap <- openproj(map, projection = “+proj=longlat”)

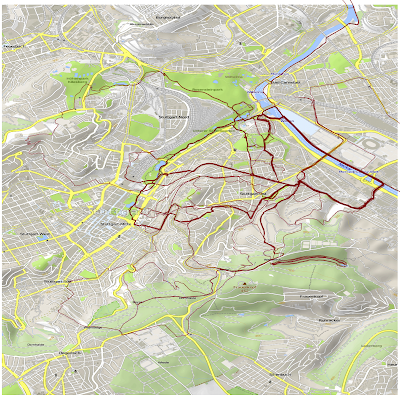
plot(transmap, raster = T)

for (el in stream.list) {

  lines(el$coords$lon, el$coords$lat,

        col = alpha(“darkred”, .5), lwd = 3)

}

[](https://i0.wp.com/4.bp.blogspot.com/-okjGYS74nM8/WnF1RinZRjI/AAAAAAAADFk/-rlKo_ZLjMUDFMPaw11whs7oj2Bi33nZgCLcBGAs/s1600/map1.png?ssl=1)

My runs through Stuttgart on an OpenStreetMap, click to enlarge

I also like a simple satellite map. I am using the ‘bing’ map type for this.

map <- openmap(c(max(lats.range), min(lons.range)),

               c(min(lats.range), max(lons.range)), type = “bing”, zoom = 15)

transmap <- openproj(map, projection = “+proj=longlat”)

plot(transmap, raster = T)

for (el in stream.list) {

  lines(el$coords$lon, el$coords$lat,

        col = alpha(“yellow”, 1/3), lwd = 3)

}

[](https://i1.wp.com/3.bp.blogspot.com/-aobYGZHz83Q/WnF1eBX6IEI/AAAAAAAADFo/aopGRFggLmsHD5a4otlpAxZ0mWZ4gwP_gCLcBGAs/s1600/map2-ld.png?ssl=1)

My runs through Stuttgart on a Bing satellite map, click to enlarge

Now, for the final step. These static maps already look quite nice and with sufficient resolution (as in the HD case with the satellite map), we can also zoom the map without loosing too much quality. But a more dynamical map would also be nice. Let’s use the wonderful leaflet package for this. I am using some pipe notation from the dplyr package to adapt the map and get the tracks onto it.

Library(Leaflet)

map <- leaflet() %>%

  addProviderTiles(“CartoDB.Positron”,

                   options = providerTileOptions(noWrap = T)) %>%

  fitBounds(lng1 = min(lons.range), lat1 = max(lats.range), lng2 <- max(lons.range), lat2 = min(lats.range))

for (el in stream.list) {

  map <- addPolylines(map, lng = el$coords$lon, lat = el$coords$lat,

                      color = “red”, opacity = 1/3, weight = 2)

}

With the function saveWidget(), we can save the resulting. You can move the map to Basel or Zürich to find some more tracks I ran there.

With the leaflet functions, we could even associate each track with a little mouseover text (like total distance or the date). I did not include this here because quite a few tracks have been plotted over each other and mouseover texts might just confuse us here.

Have fun running and plotting.