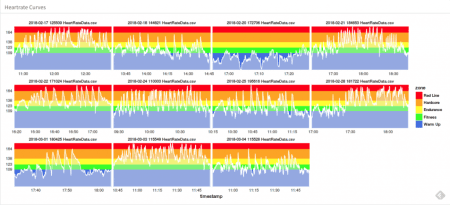
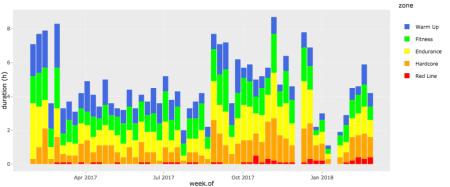
To remain consistent with the heart rate app I use, [MotiFIT](http://motifitapp.com/), It copied the charting style for individual workouts. The screenshot at right shows how the app displays heart rate. We find this view useful during weightlifting sessions because I can tell when I’ve rested sufficiently to start another set.



The dashboard has a date selector so you can use small multiples to compare several workouts at once. The last workout in the multiples corresponds to the screenshot from the app, and the similarity is apparent.

The weekly view shows the aggregate amount of time spent in each heart rate zone. You can see a big surge in the number of hours of exercise in September 2017, which is when I started playing in a tennis league. There are noticeable gaps in December and January, when I got sick and then suffered a couple of injuries. I’ve slowly added more hours back as I’ve rehabbed the injuries.



There are a couple of other views, but I’m still trying to decide how to display some of the concepts effectively. The code for the flexdashboard is below.

---

title: "Exercise Dashboard"

runtime: shiny

output:

flexdashboard::flex\_dashboard:

orientation: rows

vertical\_layout: fill

---

```{r setup, include=FALSE}

library(lubridate)

library(readr)

library(tidyverse)

library(plotly)

library(flexdashboard)

library(ggridges)

library(data.table)

library(scales)

ifelse(

dir.exists("/shiny-server/data/dir"),

datadir <- "shiny-server/data/dir",

datadir <- "/local/data/dir"

)

```

```{r load HR data}

out.file <- data.frame(timestamp = as.POSIXct(character()), bpm = integer(), stringsAsFactors = FALSE)

file.names <- dir(datadir, pattern = "^.\*HeartRateData.\*.csv") # You'll need to set datadir

library(doParallel)

registerDoParallel()

dir.traversal <- system.time({

# out.file <- foreach (file.name = tail(file.names, 10), .combine = "rbind") %dopar% {

out.file <- foreach (file.name = file.names, .combine = "rbind") %dopar% {

# for (file.name in file.names) {

# exported files are in at least 2 different encodings, so we're going to guess using the guess\_encoding function from the readr package

# We can't use readr's read\_csv or data.table's fread because some of the encodings are UTF-16LE, which makes this process s

encoding <- guess\_encoding(paste(datadir, "/", file.name, sep=""), n\_max = 1000)

file <- read.csv(

paste(datadir, "/", file.name, sep=""),

skip = 1,

sep = ",",

strip.white = TRUE,

stringsAsFactors = FALSE,

fileEncoding = toString(encoding[1,1])

)

# exported data has 2 or three columns, so if it's three, we're going to join the date and time fields

if (length(file) > 2) {

temp.date <- file[,1]

temp.time <- trimws(file[,2])

temp.bpm <- trimws(file[,3])

# date format is Wed Feb 8 2017

temp.datetime <- as.POSIXct(paste(temp.date, temp.time, sep=" "), format = "%a %b %d %Y %H:%M")

temp.df <- data.frame(temp.datetime, temp.bpm, stringsAsFactors = FALSE)

names(temp.df) <- c("timestamp", "bpm")

file <- temp.df

rm(temp.df)

} else {

names(file) <- c("timestamp", "bpm")

file$timestamp <- as.POSIXct(file$timestamp)

}

file$name <- file.name

#out.file <- rbind(out.file, file)

file

}

})

```

```{r add week number }

# Used to calculate week number below

start.date <- as.Date(as.character(min(out.file$timestamp)))

week.starts <- seq(from = start.date, to = as.Date(Sys.Date()), by = 7)

week.starts <- data.frame(

week = 0:(NROW(week.starts) - 1),

`week of` = as.Date(week.starts)

)

HR <- out.file %>%

mutate(

week = as.numeric(as.Date(timestamp, tz = Sys.timezone()) - start.date) %/% 7,

bpm = as.numeric(bpm)

) %>%

left\_join(week.starts, by = c("week" = "week")) %>%

filter(! is.na(week)) %>%

arrange(week)

HR$week.of <- factor(HR$week.of)

HR$`week.of` <- factor(HR$`week.of`, ordered = T, rev(unique(HR$`week.of`)))

# HR$week <- factor(HR$week, ordered = TRUE, levels = rev(unique(HR$week)))

```

```{r set HR zones}

cuts <- c(-Inf, 109, 123, 138, 164, Inf)

labs <- c("Warm Up", "Fitness", "Endurance", "Hardcore", "Red Line")

HR$zone <- cut(HR$bpm, breaks = cuts, labels = labs, include.lowest=TRUE, ordered\_result = TRUE)

fitness.rainbow <- c("royalblue", "royalblue", "green", "yellow", "orange", "red")

rects <- data.frame(ystart = cuts[1:5], yend = cuts[2:6], zone = factor(labs, levels = rev(labs), ordered = TRUE))

bpm.min <- reactive({

min(HR.filtered()$bpm, na.rm = T)

})

bpm.max <- reactive({

max(HR.filtered()$bpm, na.rm = T)

})

zone.breaks <- reactive({

c(

bpm.min(),

(bpm.min() + cuts[2]) / 2,

(cuts[2] + cuts[3]) / 2,

(cuts[3] + cuts[4]) / 2,

(cuts[4] + cuts[5]) / 2,

bpm.max()

)

})

```

```{r make HR reactive }

HR.filtered <- reactive({

HR %>%

filter(as.Date(timestamp, tz = Sys.timezone()) >= input$date[1]) %>%

filter(as.Date(timestamp, tz = Sys.timezone()) <= input$date[2])

})

```

```{r calc durations }

HR.duration <- reactive({

HR.filtered() %>%

mutate(date = as.Date(format(timestamp, "%Y-%m-%d"))) %>%

group\_by(name, date) %>% # name is filename so two files from same date don't skew calculations

summarize(start = min(timestamp, na.rm = T),

end = max(timestamp, na.rm = T),

`duration (m)` = round(as.numeric(difftime(end, start, units = "mins")), 1),

`average bpm` = round(mean(as.numeric(bpm)), 0),

sd = round(sd(as.numeric(bpm)), 1),

max.bpm = max(as.numeric(bpm), na.rm = T)) %>%

ungroup() %>%

mutate(start = format(start, "%H:%M:%S"),

end = format(end, "%H:%M:%S"))

})

HR.daily.duration <- reactive({

HR %>%

filter(as.Date(timestamp, tz = Sys.timezone()) >= input$dailyDate[1]) %>%

filter(as.Date(timestamp, tz = Sys.timezone()) <= input$dailyDate[2]) %>%

mutate(date = as.Date(format(timestamp, "%Y-%m-%d"))) %>%

group\_by(name, date) %>%

summarize(start = min(timestamp, na.rm = T),

end = max(timestamp, na.rm = T),

`duration (m)` = round(as.numeric(difftime(end, start, units = "mins")), 1),

`average bpm` = round(mean(as.numeric(bpm)), 0),

sd = round(sd(as.numeric(bpm)), 1),

max.bpm = max(as.numeric(bpm), na.rm = T)) %>%

ungroup() %>%

mutate(start = format(start, "%H:%M:%S"),

end = format(end, "%H:%M:%S"))

})

HR.zone.duration.weekly <- reactive({

HR.filtered() %>%

group\_by(week, zone) %>%

summarize(

`duration (s)` = n(),

`duration (m)` = round(n() / 60, 1),

`duration (h)` = round(n() / 3600, 1)

) %>%

ungroup() %>%

left\_join(week.starts, by = "week") %>%

filter(! is.na(week)) %>%

arrange(week)

})

HR.weekly.duration <- reactive({

HR.zone.duration.weekly() %>%

group\_by(week, `week.of`) %>%

summarize(

`duration (h)` = sum(`duration (h)`),

`exercise sessions` = n()

) %>%

ungroup() %>%

arrange(week)

})

HR.zone.duration.daily <- reactive({

HR %>%

mutate(date = as.Date(timestamp, tz = Sys.timezone())) %>%

filter(date >= input$dailyDate[1]) %>%

filter(date <= input$dailyDate[2]) %>%

group\_by(date, zone) %>%

summarize(

`duration (s)` = n(),

`duration (m)` = round(n() / 60, 1),

`duration (h)` = round(n() / 3600, 1)

) %>%

ungroup()

})

```

Weekly

=======================================================================

Inputs {.sidebar}

-----------------------------------------------------------------------

Enter a date range for the weekly charts:

```{r input date range for weekly chart }

dateRangeInput("date", "Date Range", start = as.Date(min(HR$timestamp, na.rm = T), tz = Sys.timezone()))

```

Column

-----------------------------------------------------------------------

### Weekly Exercise

```{r weekly barplot}

renderPlotly({

ggplot(data = HR.zone.duration.weekly(), aes(x = `week.of`, y = `duration (h)`, fill = zone)) +

geom\_bar(

stat = "identity",

position = "stack"

) +

scale\_fill\_manual(values = fitness.rainbow[2:6]) +

scale\_y\_continuous(breaks = seq(from = 0, to = 8, by = 2), minor\_breaks = seq(from = 0, to = 8, by = 1))

})

```

Column

-----------------------------------------------------------------------

### Heartrate Zones

```{r ridgeline plot}

renderPlot({

ggplot(

HR.filtered(),

aes(x = bpm, y = `week.of`, fill = ..x..)

) +

scale\_fill\_gradientn(

colors = fitness.rainbow,

breaks = zone.breaks()

) +

scale\_x\_continuous(breaks = function(x) {seq(from = 0, to = max(x), by = 10)}) +

geom\_density\_ridges\_gradient(na.rm = TRUE, col = "grey70", scale = 4) +

theme\_ridges(font\_size = 7) +

theme(

legend.position = "none"

)

})

```

### Weekly Durations

```{r total duration bar plot}

renderPlotly({

ggplot(data = HR.weekly.duration(), aes(x = `week.of`, y = `duration (h)`)) +

geom\_bar(

aes(

text = paste(

"week: ", week, "  
",

"# of sessions: ", `exercise sessions`,

sep = ""

)

),

stat = "identity"

) +

geom\_smooth(span = 0.35)

})

# renderTable({

# head(HR.weekly.duration())

# })

```

Daily {data-orientation=rows}

=======================================================================

Inputs {.sidebar}

-----------------------------------------------------------------------

Enter a date range for the daily charts:

```{r input date range for daily calcs }

dateRangeInput("dailyDate", "Date Range", start = as.Date(Sys.Date() %m+% days(-15)))

```

Column

-----------------------------------------------------------------------

### Daily Heartrate Zones

```{r individual exercise zones}

renderPlotly({

ggplot(data = HR.zone.duration.daily(), aes(x = date, y = `duration (m)`, fill = zone)) +

geom\_bar(

stat = "identity",

position = "dodge"

) +

scale\_fill\_manual(values = fitness.rainbow[2:6]) +

scale\_y\_continuous(breaks = seq(from = 0, to = max(HR.zone.duration.daily()$`duration (m)`, na.rm = TRUE), by = 15)) +

theme(

legend.position = "none"

)

})

```

### Daily Workouts, BPM vs. Duration

```{r bubble plot}

renderPlotly({

ggplot(HR.daily.duration(), aes(x = `duration (m)`, y = `average bpm`)) +

geom\_point(aes(color = `average bpm` + sd, size = -sd), alpha = 0.5) +

scale\_color\_gradientn(

colors = fitness.rainbow[2:6], # "royalblue" "green" "yellow" "orange" "red"

values = rescale(zone.breaks(), to = c(0, 1)), # 54 81.5 116 130.5 151 190

na.value = "black",

breaks = cuts, # -Inf 109 123 138 164 Inf

limits = c(min(zone.breaks()), max(zone.breaks()))

) +

scale\_x\_continuous(

limits = c(0, as.numeric(max(HR.daily.duration()$`duration (m)`))),

breaks = seq(from = 0, to = as.numeric(max(HR.daily.duration()$`duration (m)`)), by = 15)

) +

scale\_y\_continuous(

limits = c(min(zone.breaks()), max(zone.breaks())),

breaks = cuts[2:5],

minor\_breaks = NULL

) +

#geom\_smooth(span = 0.35) +

theme(

legend.position = "none"

)

})

# renderTable({

# head(HR.daily.duration())

# })

# renderText({

# rescale(

# HR.daily.duration()$max.bpm,

# from = c(min(zone.breaks()), max(zone.breaks())),

# to = c(0, 1)

# )

# })

```

Column

-----------------------------------------------------------------------

### Heartrate Curves

```{r individual exercise plots}

renderPlot({

temp <- HR.filtered() %>%

# temp <- HR %>%

mutate(date = as.Date(timestamp, tz = Sys.timezone())) %>%

filter(as.Date(timestamp, tz = Sys.timezone()) >= input$dailyDate[1]) %>%

filter(as.Date(timestamp, tz = Sys.timezone()) <= input$dailyDate[2])

ggplot() +

geom\_rect(data = rects, aes(ymin = ystart, ymax = yend, fill = zone), xmin=-Inf, xmax=Inf, inherit.aes = FALSE) +

geom\_ribbon(data = temp, aes(x = timestamp, ymax = bpm, ymin = min(temp$bpm)), color = "white", fill="grey90", alpha = .5) +

scale\_fill\_manual(values = rev(fitness.rainbow[2:6])) +

theme\_minimal() +

facet\_wrap(~ name, scales = "free\_x") +

scale\_x\_datetime(expand = c(0, 0)) +

scale\_y\_continuous(expand = c(0, 0), breaks = cuts[2:5]) +

theme(panel.grid = element\_blank(), panel.border = element\_blank())

})

```