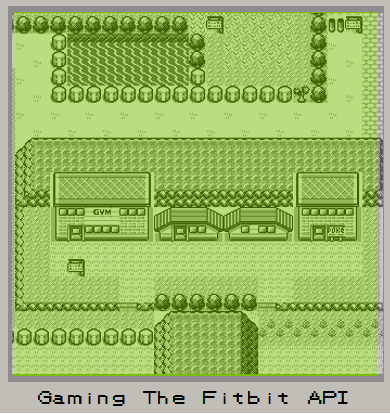
**“Anyone can look for fashion in a boutique or history in a museum. The creative explorer looks for history in a hardware store and fashion in an airport” — Robert Wieder**



As much as I like Medium for it’s ability to reach an audience I prefer the customization afforded by [github.io](http://github.io). **If you want to see a far-fancier version of this post please** [**visit this page instead**](https://moldach.github.io/fitbit-project/)**.**

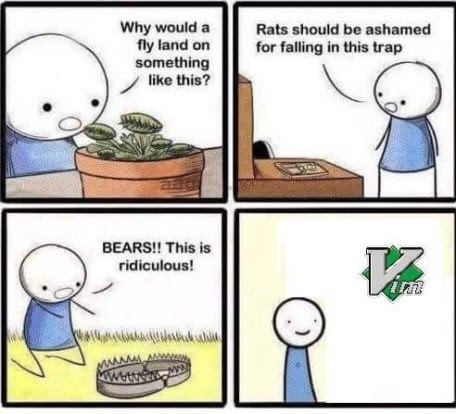
**Project Inspiration**

You may, or may not, have heard of the term gamification but chances are you’ve experienced it.

Gamification is the application of game-design elements, and game principles, in non-game contexts. The idea is, if you use elements of games, like linking rules and rewards into a feedback system, you can make (almost) any activity motivating and fun.

Gamification is the concept behind **eLearning**. In elementary school I remember all the students wanted to play [The Oregon Trail](https://classicreload.com/oregon-trail.html) in computer class. I also remember another game where you had to solve math problems before something hit the floor. Okay, maybe it wasn’t the most thrilling introduction to gamification but I remember it nonetheless.

At some point in my career, I got tired of using nano and decided to I wanted to try to learn Vim.

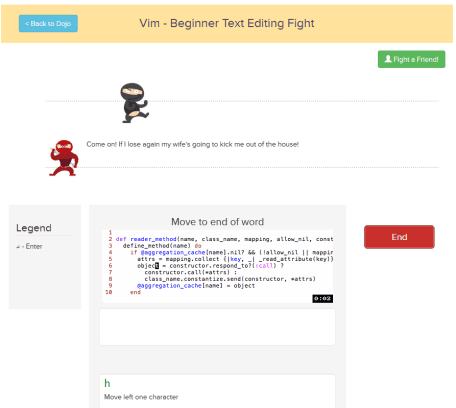


It was then that I discovered two very enjoyable examples of gamification:

* [Vim Adventures](https://vim-adventures.com/) is kind of like Zelda for Gameboy where you have to move through the environment and solve riddles – except with Vim commands! You can [watch it being played on Twitch here](https://www.youtube.com/watch?v=zGk_ipjk9uw).



* [shortcutFoo](https://www.shortcutfoo.com/) teaches you shortcuts for Vim, Emacs, Command Line, Regex *etc.* via **interval training**, which is essentially spaced repetition. This helps you memorize shortcuts more efficiently.



Today, I enjoy eLearning-gamification on platforms like [DuoLingo](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwiQm7ib677eAhUovlkKHRr-BygQFjAAegQIAhAC&url=https%3A%2F%2Fwww.duolingo.com%2F&usg=AOvVaw0GVHP2wYuntJMUTG1EDY5y), and [DataCamp](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwiQm7ib677eAhUovlkKHRr-BygQFjAAegQIAhAC&url=https%3A%2F%2Fwww.duolingo.com%2F&usg=AOvVaw0GVHP2wYuntJMUTG1EDY5y).

I’ve also recently started to participate in a Kaggle competition, “[PUBG Finish Placement Prediction](https://www.kaggle.com/c/pubg-finish-placement-prediction)”. Kaggle is a Google owned hangout for data science enthusiasts where they can use machine learning to solve predictive analytics problems for cash and clout. Similar to chess there are so-called **Kaggle Grandmasters**.

**The Quest**

Our laboratory studies perinatal influences on the biological embedding of early adversity of mental health outcomes. We combine genetic, epigenetic and epidemiological approaches to identify pregnant women who’s offspring may potentially be at risk for adverse mental health outcomes.

*My supervisor approached me with a challenge; how feasible would it be to access biometric data from 200 Fitbits?*

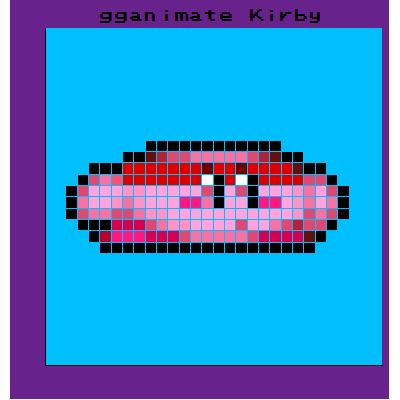
**So I bought myself a Fitbit Charge2 fitness tracker and hit the gym!**

At some point I think we both realized that this project was going to be a big undertaking. Perhaps R isn’t really intended to do large scale real-time data management from API services. It’s great for static files, or static endpoints, but if you’re working with multiple participants a dedicated solution like [Fitabase](https://www.fitabase.com/) may work the best – or so they claim.

Nonetheless, I wanted to try out a bunch of cool new things in R like making a personal website using blogdown, using gganimate with Rokemon, accessing the fitbit API with httr as well as adding a background image with some custom CSS/HTML. Is there possibly a better way to possibly gamify my leaRning curve – I think not.

**The following was my attempt at e-learning gamification for R.**

I used the [blogdown](https://bookdown.org/yihui/blogdown/) package to allow me to write blog posts as R Markdown documents, knitting everything to a nice neat static website that I can push online. It was a nice opportunity to learn a bit about pandoc, Hugo, CSS/HTML lurking beneath the server side code. I decided to go with the [Academic theme](https://themes.gohugo.io/academic/) for [Hugo](https://gohugo.io/), pull in as much data as I could from the Fitbit API, clean it up, and then perform some exploratory data analysis. In the process, I generated some cool animated sprites and use video game inspired visualizations.



**Setting up a Fitbit Developer Account**

Fitbit uses [**OAuth 2.0**](https://dev.fitbit.com/build/reference/web-api/oauth2/) **Access** Token for making HTTP request to the Fitbit API. You need to set up an account to use the API and include your token in R. Instead of reading the [FITBIT DEV HELP](https://dev.fitbit.com/build/reference/web-api/basics/#app-registration) section I would direct the reader to better more-concise instructions [here](https://obrl-soil.github.io/fitbit-api-r/).

Now that you have an account we’re ready to do stuff in R.

Set your token up:

# You Found A Secret Area!  
token = "yourToken"

**Using the fitbitr package**

I had never made an HTTP request before and although the process is officially documented [here](https://moldach.github.io/fitbit-project/(https:/dev.fitbit.com/build/reference/web-api/oauth2/)) it can be a *tad* overwhelming. Therefore, I initially resorted to using an R package built to access the R API, under-the-hood, called fitbitr.

Unfortunately this would limit me to only accessing some basic user information, heart rate and step count data.

**Getting Basic User Info**

The first function in this package sends a GET request to the Get Profile [resource URL](https://dev.fitbit.com/build/reference/web-api/user/#get-profile).

# Extracting Resources

# Get userInfo  
user\_info <- fitbitr::getUserInfo(token)

# Hailing a Chocobo!

# What is my stride length in meters?  
strideLengthWalking <- user\_info$strideLengthWalking

My stride length is 68.5.

*Stride length is measured from heel to heel and determines how far you walk with each step. On average, a man’s walking stride length is 2.5 feet, or 30 inches, while a woman’s average stride length is 2.2 feet, or 26.4 inches, according to this*[*report*](https://livehealthy.chron.com/average-walking-stride-length-7494.html)*.*

# Hitting 80 MPH

# What is my running stride length  
strideLengthRunning <- user\_info$strideLengthRunning

My running stride length is 105.5.

*The Fitbit uses your sex and height by default to gauge your stride length which could potentially be inaccurate.*

# Looking for the fourth chaos emerald

# What is my average daily steps?  
averageDailySteps <- user\_info$averageDailySteps

My average daily steps is 14214.

Considering that the daily recommended steps is 10,000 I’d say that’s acceptable. That being said, there’s always room for improvement.

**Accessing heart rate and footsteps with the fitbitr pacakge**

I’m going to grab a week’s worth of data for a very preliminary EDA.

# Smashing buttons

days <- c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")

monday\_heart <- getTimeSeries(token, type = "heart", activityDetail = "1min", date = "2018-08-20", startTime = "00:00", endTime = "23:59")  
monday\_heart %<>% mutate(date = "2018-08-20")  
monday\_steps <- getTimeSeries(token, type = "steps", activityDetail = "1min", date = "2018-08-20")  
monday\_steps %<>% mutate(date = "2018-08-20")  
monday <- monday\_heart %>% full\_join(monday\_steps)  
monday %<>% mutate(week\_date = "Monday")  
monday %<>% mutate(day\_of\_week = "1")

tuesday\_heart <- getTimeSeries(token, type = "heart", activityDetail = "1min", date = "2018-08-21")  
tuesday\_heart %<>% mutate(date = "2018-08-21")  
tuesday\_steps <- getTimeSeries(token, type = "steps", activityDetail = "1min", date = "2018-08-21")  
tuesday\_steps %<>% mutate(date = "2018-08-21")  
tuesday <- tuesday\_heart %>% full\_join(tuesday\_steps)  
tuesday %<>% mutate(week\_date = "Tuesday")  
tuesday %<>% mutate(day\_of\_week = "2")

wednesday\_heart <- getTimeSeries(token, type = "heart", activityDetail = "1min", date = "2018-08-22")  
wednesday\_heart %<>% mutate(date = "2018-08-22")  
wednesday\_steps <- getTimeSeries(token, type = "steps", activityDetail = "1min", date = "2018-08-22")  
wednesday\_steps %<>% mutate(date = "2018-08-22")  
wednesday <- wednesday\_heart %>% full\_join(wednesday\_steps)  
wednesday %<>% mutate(week\_date = "Wednesday")  
wednesday %<>% mutate(day\_of\_week = "3")

thursday\_heart <- getTimeSeries(token, type = "heart", activityDetail = "1min", date = "2018-08-23")  
thursday\_heart %<>% mutate(date = "2018-08-23")  
thursday\_steps <- getTimeSeries(token, type = "steps", activityDetail = "1min", date = "2018-08-23")  
thursday\_steps %<>% mutate(date = "2018-08-23")  
thursday <- thursday\_heart %>% full\_join(thursday\_steps)  
thursday %<>% mutate(week\_date = "Thursday")  
thursday %<>% mutate(day\_of\_week = "4")

friday\_heart <- getTimeSeries(token, type = "heart", activityDetail = "1min", date = "2018-08-24")  
friday\_heart %<>% mutate(date = "2018-08-24")  
friday\_steps <- getTimeSeries(token, type = "steps", activityDetail = "1min", date = "2018-08-24")  
friday\_steps %<>% mutate(date = "2018-08-24")  
friday <- friday\_heart %>% full\_join(friday\_steps)  
friday %<>% mutate(week\_date = "Friday")  
friday %<>% mutate(day\_of\_week = "5")

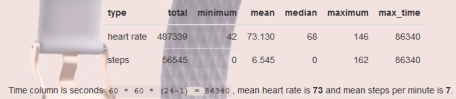
saturday\_heart <- getTimeSeries(token, type = "heart", activityDetail = "1min", date = "2018-08-24")  
saturday\_heart %<>% mutate(date = "2018-08-24")  
saturday\_steps <- getTimeSeries(token, type = "steps", activityDetail = "1min", date = "2018-08-24")  
saturday\_steps %<>% mutate(date = "2018-08-24")  
saturday <- saturday\_heart %>% full\_join(saturday\_steps)  
saturday %<>% mutate(week\_date = "Saturday")  
saturday %<>% mutate(day\_of\_week = "6")

week <- monday %>% bind\_rows(tuesday) %>% bind\_rows(wednesday) %>% bind\_rows(thursday) %>% bind\_rows(friday) %>% bind\_rows(saturday)  
   
week$date <- as.Date(week$date)

**Summary Statistics**

# Opening pod bay doors

week %>%   
 group\_by(type) %>%   
 summarise(  
 total = sum(value),  
 minimum = min(value),  
 mean = mean(value),  
 median = median(value),  
 maximum = max(value),  
 max\_time = max(time)  
 ) %>%  
 knitr::kable(digits = 3) %>%   
 kable\_styling(full\_width = F)



**Exploratory Data Analysis**

Since this is a post about gamification I decided to do something fun with my exploratory data visualizations. I wanted to use the Rokemon package which allows me to set the theme of ggplot2 (and ggplot2 extensions) to Game Boy and Game Boy Advance themes! When convenient, I’ve combined plots with cowplot.

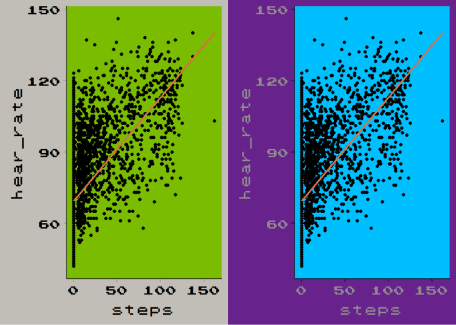
Let’s take a quick look at the relationship and distribution of heart rate and step count.

# Doing the thing...

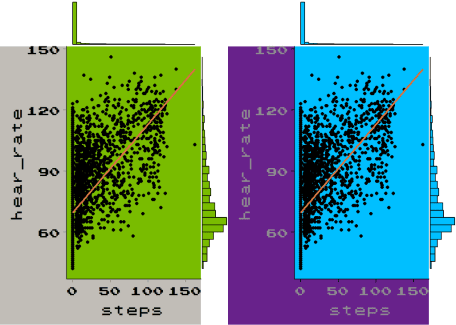
g <- week %>%   
 spread(type, value) %>%   
 rename(hear\_rate = "heart rate") %>%   
 na.omit() %>%   
 ggplot(aes(steps, hear\_rate)) + geom\_point() + geom\_smooth(method="lm", se=F, colour = "#DE7243")

gb <- g + theme\_gameboy()  
gba <- g + theme\_gba()

plot\_grid(gb, gba, labels = c("", ""), align = "h")



Alternatively, we could get a better look at the data by adding marginal density plots to the scatterplots with the ggMarginal() function from the ggExtra package.



**Weekly trends**

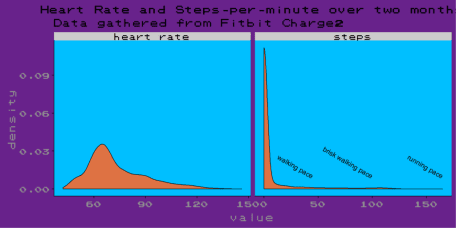
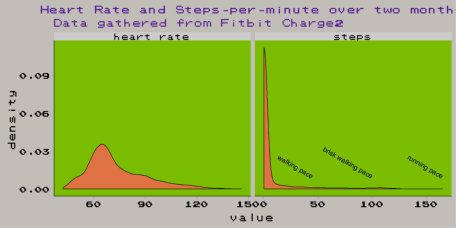
Let’s take a quick look at the distribution of the contiguous variables to get a better idea than the mean and median.

# Loading..... Wait, what else were you expecting?

annotations\_steps <- data\_frame(  
 x = c(45, 100, 165),  
 y = c(0.01, 0.01, 0.01),  
 label = c('walking pace', 'brisk walking pace', 'running pace'),  
 type = c('steps', 'steps', 'steps')  
)

g <- week %>%   
 ggplot(aes(value)) +   
 geom\_density(fill = "#DE7243") +  
 geom\_text(data = annotations\_steps, aes(x = x, y = y, label = label), angle = -30, hjust = 1) +  
 facet\_grid(.~type, scales = 'free\_x') +   
 labs(title = 'Heart Rate and Steps-per-minute over two months',  
 subtitle = 'Data gathered from Fitbit Charge2')

g + theme\_gameboy()  
g + theme\_gba()



Heart rate is a little right-skewed, probably due to sleep and sedentary work. Similarly, for step count you see that only a small bump under brisk walking pace from when I skateboarded to work.

**Exercise patterns**

This week I didn’t work out so I thought I’d at least look at when I was on my way to work. The figure below shows blue for heart rate/min and orange is the number of steps/min.

# You are carrying too much to be able to run

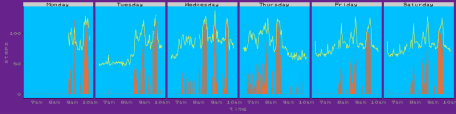
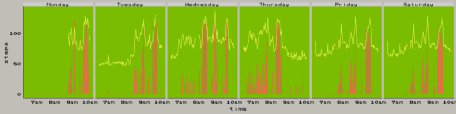
between\_six\_nine <- function(time) time > 7\*60\*60 & time < 10\*60\*60

is\_weekday <- function(day\_of\_week) day\_of\_week %in% 1:6

week$week\_date\_f <- factor(week$week\_date, levels=c("Monday","Tuesday","Wednesday", "Thursday", "Friday", "Saturday"))

g <- week %>%   
 filter(between\_six\_nine(time) & is\_weekday(day\_of\_week)) %>%   
 spread(type, value) %>%   
 ggplot(aes(x = time)) +  
 geom\_bar(aes(y = steps), color = '#DE7243', alpha = 0.3, stat = 'identity') +   
 geom\_line(aes(y = `heart rate`), color = '#E3F24D', size = 0.8) +   
 facet\_grid(~week\_date\_f) +  
 scale\_x\_continuous(breaks=c(27000, 30000, 33000, 36000), labels=c("7am", "8am", "9am", "10am"))

g + theme\_gameboy()  
g + theme\_gba()



My activity has been pretty much the same all week since I skateboard to work every morning.

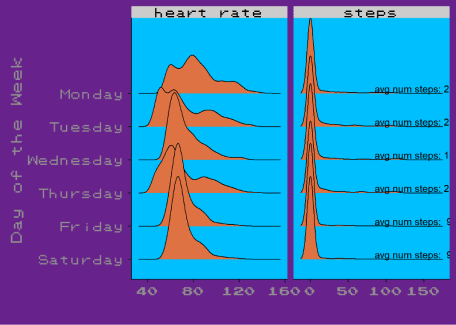
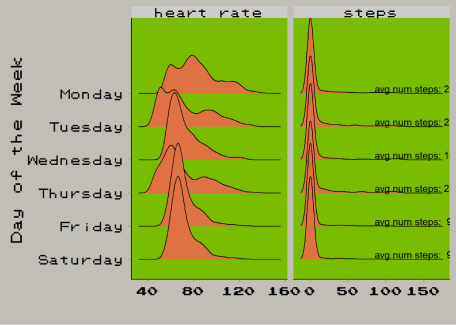
**Most active time**

# 60% of the time, it loads ALL the time

step\_counts <- week %>%   
 filter(type == 'steps') %>%   
 group\_by(day\_of\_week) %>%   
 summarise(  
 type = last(type),   
 avg\_num\_steps = sprintf('avg num steps: %3.0f', sum(value)/52)  
 )

g <- week %>%  
 ggplot(aes(x= value, y = fct\_rev(factor(day\_of\_week)))) +  
 geom\_density\_ridges(scale = 2.5, fill = "#DE7243") +  
 geom\_text(data = step\_counts, nudge\_y = 0.15, hjust = 0,   
 aes(x = 85, y = fct\_rev(factor(day\_of\_week)), label = avg\_num\_steps)) +  
 scale\_y\_discrete(breaks=1:6, labels = c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")) +  
 facet\_grid(.~type, scales = "free") +  
 labs(x = '', y = "Day of the Week")

g + theme\_gameboy()  
g + theme\_gba()



The distribution of steps per-minute was pretty constant because as I said I didn’t work-out; this likely reflects me shuffling to get tea.

It looks like Monday was the day I got my heart rate up the most, the bimodal peak is probably when I was running around looking for a rental property.

**Accessing the Fitbit API directly with the httr package**

Eventually, I found an excellent [tutorial](https://obrl-soil.github.io/fitbit-api-r/) by [obrl-soil](https://github.com/obrl-soil) which introduced me to the httr package and gave me the confidence I needed to peruse the [Fitbit DEV web API reference](https://dev.fitbit.com/build/reference/web-api/). Now I was able to gain access to far more sources of data.

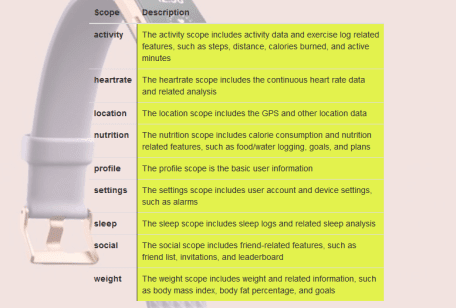
**What data is available?**

A brief overview of what data is available from the Fitbit API:

# Your Boko Club is badly damaged

# make a kable table for data you can access from Fitbit API  
dt01 <- data.frame(Scope = c("activity",  
 "heartrate",  
 "location",  
 "nutrition",  
 "profile",  
 "settings",  
 "sleep",  
 "social",  
 "weight"),  
 Description = c("The activity scope includes activity data and exercise log related features, such as steps, distance, calories burned, and active minutes",  
 "The heartrate scope includes the continuous heart rate data and related analysis",  
 "The location scope includes the GPS and other location data",  
 "The nutrition scope includes calorie consumption and nutrition related features, such as food/water logging, goals, and plans",  
 "The profile scope is the basic user information",  
 "The settings scope includes user account and device settings, such as alarms",  
 "The sleep scope includes sleep logs and related sleep analysis",  
 "The social scope includes friend-related features, such as friend list, invitations, and leaderboard",  
 "The weight scope includes weight and related information, such as body mass index, body fat percentage, and goals")  
)

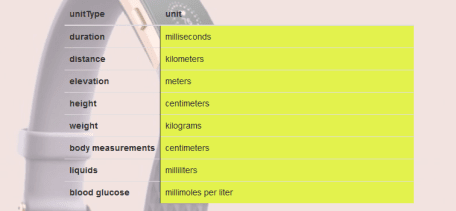
dt01 %>%  
 kable("html") %>%  
 kable\_styling(full\_width = F) %>%  
 column\_spec(1, bold = T, border\_right = T) %>%  
 column\_spec(2, width = "30em", background = "#E3F24D")



What are the units of measurement?

# Loading Cutscenes You Can't Skip

# make a Kable table or measurement information  
dt03 <- data.frame(unitType = c("duration",  
 "distance",  
 "elevation",  
 "height",  
 "weight",  
 "body measurements",  
 "liquids",  
 "blood glucose"),  
 unit = c("milliseconds",  
 "kilometers",  
 "meters",  
 "centimeters",  
 "kilograms",  
 "centimeters",  
 "milliliters",  
 "millimoles per liter"))  
dt03 %>%  
 kable("html") %>%  
 kable\_styling(full\_width = F) %>%  
 column\_spec(1, bold = T, border\_right = T) %>%  
 column\_spec(2, width = "30em", background = "#E3F24D")



Define a function for turning a json list into a dataframe.

# Inserting last-minute subroutines into program...

# json-as-list to dataframe (for simple cases without nesting!)  
jsonlist\_to\_df <- function(data = NULL) {  
 purrr::transpose(data) %>%  
 purrr::map(., unlist) %>%  
 as\_tibble(., stringsAsFactors = FALSE)  
}

**Investigating my 10Km run**

GET request to retrieve minute-by-minute heart rate data for my 10km run.

# Preparing for the mini-boss

get\_workout <- function(date = NULL, start\_time = NULL, end\_time = NULL,   
 token = Sys.getenv('FITB\_AUTH')) {  
GET(url =  
 paste0('<https://api.fitbit.com/1/user/-/activities/heart/date/>',  
 date, '/1d/1min/time/', start\_time, '/', end\_time, '.json'),   
 add\_headers(Authorization = paste0("Bearer ", token)))  
}

# Get the workout for my 10Km run   
got\_workout <- get\_workout(date = '2018-10-21', start\_time = '09:29', end\_time = '10:24')

workout <- content(got\_workout)

# summary

workout[['activities-heart']][[1]][['heartRateZones']] <- jsonlist\_to\_df(workout[['activities-heart']][[1]][['heartRateZones']])

# the dataset  
workout[['activities-heart-intraday']][['dataset']] <- jsonlist\_to\_df(workout[['activities-heart-intraday']][['dataset']])

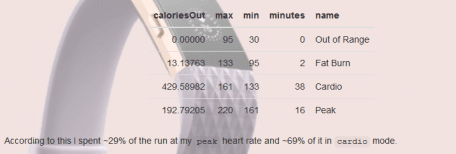
# format the time   
workout$`activities-heart-intraday`$dataset$time <- as.POSIXlt(workout$`activities-heart-intraday`$dataset$time, format = '%H:%M:%S')  
lubridate::date(workout$`activities-heart-intraday`$dataset$time) <- '2018-10-21'

# find time zone  
# grep("Canada", OlsonNames(), value=TRUE)  
lubridate::tz(workout$`activities-heart-intraday`$dataset$time) <- 'Canada/Eastern'

Let’s take a look at the summary for my 10Km run:

# Farming Hell Cows

workout$`activities-heart`[[1]]$heartRateZones %>% kable() %>% kable\_styling(full\_width = F)



[obrl-soil](https://obrl-soil.github.io/fitbit-api-r/) used the MyZone Efforts Points [(MEPS)](https://myzone.org/the-zones/) which is calculated minute-by-minute as a percentage of max heart rate. It measures the effort put in – The more points the better. Another example of gamification in action.

# Looting a chest

meps\_max <- function(age = NULL) { 207 - (0.7 \* age) }

Mine is 186.

Now is we create a tribble with 4 heart ranges showing the lower and higher bounds and use the mutate() function from above to calculate what my max heart rate is (with lower and upper bounds).

# Taking the hobbits to Isengard

my\_MEPS <- tribble(~MEPS, ~hr\_range, ~hr\_lo, ~hr\_hi,   
 1, '50-59%', 0.50, 0.59,  
 2, '60-69%', 0.60, 0.69,  
 3, '70-79%', 0.70, 0.79,  
 4, '>=80', 0.80, 1.00) %>%  
 mutate(my\_hr\_low = floor(meps\_max(30) \* hr\_lo),  
 my\_hr\_hi = ceiling(meps\_max(30) \* hr\_hi))  
my\_MEPS

## # A tibble: 4 x 6  
## MEPS hr\_range hr\_lo hr\_hi my\_hr\_low my\_hr\_hi  
##   
## 1 1 50-59% 0.5 0.59 93 110  
## 2 2 60-69% 0.6 0.69 111 129  
## 3 3 70-79% 0.7 0.79 130 147  
## 4 4 >=80 0.8 1 148 186

With the equation now defined let’s calculate my total MEPS:

# Checkpoint!

mep <- mutate(workout$`activities-heart-intraday`$dataset,  
 meps = case\_when(value >= 146 ~ 4,  
 value >= 128 ~ 3,  
 value >= 109 ~ 2,  
 value >= 91 ~ 1,  
 TRUE ~ 0)) %>%  
 summarise("Total MEPS" = sum(meps))

Wow it’s 216!

I’m not sure what that exactly means but apparently the maximum possible MEPS in a 42-minute workout is 168 and since I ran this 10Km in 54:35 I guess that’s good?

I’d like to post sub 50 minutes on my next 10Km run but I’m not sure if I should be aiming to shoot for a greater percentage of peak heart rate minutes or not – guess I will need to look into this.

**Minute-by-minute sleep data for one night**

Let’s examine my sleep patterns last night.

# Resting at Campfire

get\_sleep <- function(startDate = NULL, endDate = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1.2/user/-/sleep/date/>', startDate, "/", endDate, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}

# make sure that there is data for those days otherwise it tosses an error

got\_sleep <- get\_sleep(startDate = "2018-08-21", endDate = "2018-08-22")  
sleep <- content(got\_sleep)

dateRange <- seq(as.Date("2018-08-21"), as.Date("2018-08-22"), "days")

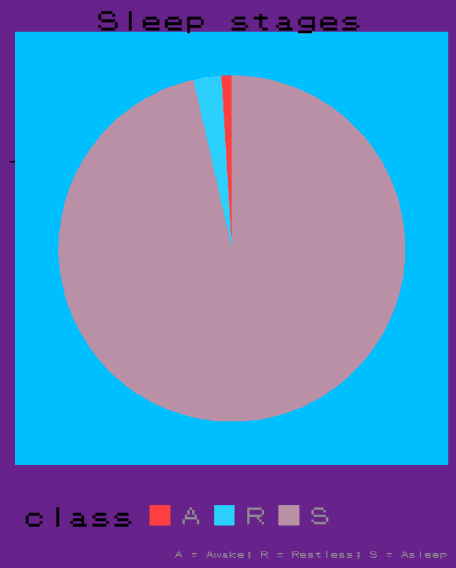
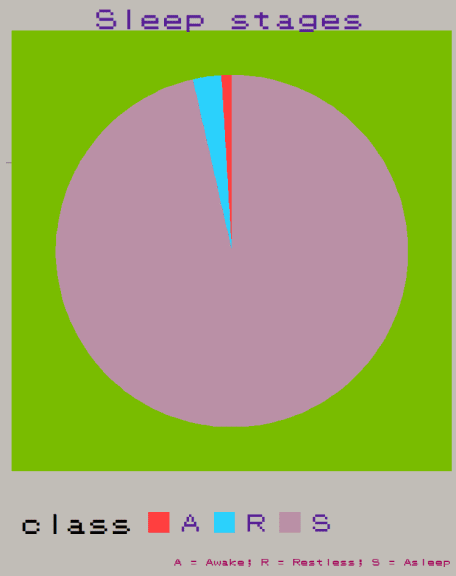
sleep\_pattern <- NULL  
for(j in 1:length(dateRange)){  
 sleep[['sleep']][[j]][['levels']][['data']] <- jsonlist\_to\_df(sleep[['sleep']][[j]][['levels']][['data']])  
 tmp <- sleep$sleep[[j]]$levels$`data`  
sleep\_pattern <- bind\_rows(sleep\_pattern, tmp)  
}

Okay now that the data munging is complete, let’s look at my sleep pattern.

# Now entering... The Twilight Zone

g <- sleep\_pattern %>% group\_by(level, seconds) %>%   
 summarise() %>%   
 summarise(seconds = sum(seconds)) %>%  
 mutate(percentage = seconds/sum(seconds)) %>%   
 ggplot(aes(x = "", y = percentage, fill = c("S", "A", "R"))) +  
 geom\_bar(width = 1, stat = "identity") +  
 theme(axis.text.y = element\_blank(),  
 axis.text.x = element\_blank(), axis.line = element\_blank(), plot.caption = element\_text(size = 5), plot.title = element\_blank()) +  
 labs(fill = "class", x = NULL, y = NULL, title = "Sleep stages", caption = "A = Awake; R = Restless; S = Asleep") +  
 coord\_polar(theta = "y", start = 0) +  
 scale\_fill\_manual(values = c("#FF3F3F", "#2BD1FC", "#BA90A6"))

g + theme\_gameboy()  
g + theme\_gba()

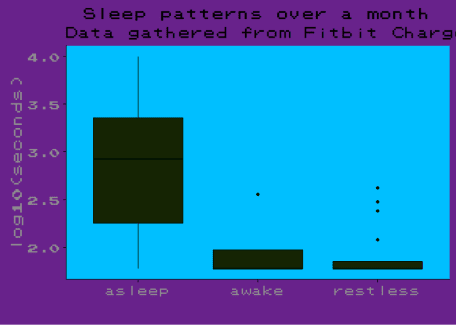


A pie chart is probably not the best way to show this data. Let’s visualize the distribution with a box plot.

# Entering Cheat Codes!

g <- ggplot(sleep\_pattern, aes(y=log10(seconds), x=level)) +   
 geom\_boxplot(color="#031300", fill='#152403') +  
 labs(x = "", title = 'Sleep patterns over a month',  
 subtitle = 'Data gathered from Fitbit Charge2') +  
 theme(legend.position = "none")

g + theme\_gameboy()  
g + theme\_gba()

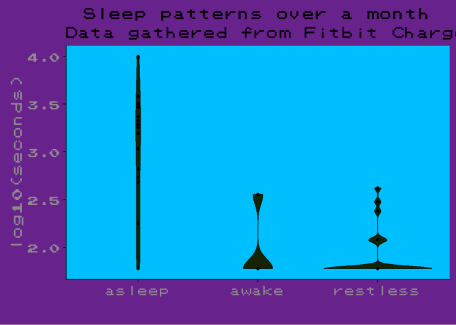
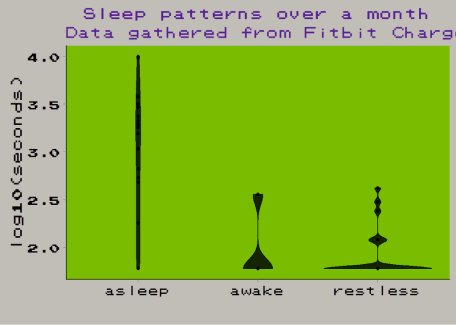


An even better way to visualize the distribution would be to use a violin plot with the raw data points overlaid.

# Neglecting Sleep...

g <- ggplot(sleep\_pattern, aes(y=log10(seconds), x=level)) +   
 geom\_violin(color="#031300", fill='#152403') +  
 geom\_point() +  
 labs(x = "", title = 'Sleep patterns over a month',  
 subtitle = 'Data gathered from Fitbit Charge2') +  
 theme(legend.position = "none")

g + theme\_gameboy()  
g + theme\_gba()



**Get Daily Activity Patterns For 3 Months**

You can do API requests for various periods from the Fitbit [Activity and Exercise Logs](https://dev.fitbit.com/build/reference/web-api/activity/) but since I’ve only had mine a couple months I’ll use the 3m period.

I will also need to trim off any day’s which are in the future otherwise they’ll appear as 0 calories in the figures. It’s best to use the Sys.Date() function rather than hardcoding the date when doing EDA, making a Shiny app, or [parameterizing a RMarkdown file](https://resources.rstudio.com/rstudio-connect-2/parameterized-r-markdown-reports-with-rstudio-connect-aron-atkins). This way you can explore different time periods without anything breaking.

I cannot remember when I started wearing my Fitbit but we can figure that out with the following code:

# ULTIMATE IS READY!

# Query how many days since you've had fitbit for  
inception <- user\_info$memberSince

I’ve had my Fitbit since 2018–08–20.

Let’s gather data from September 20th until November 6th 2018.

# Catching them all!

### Calories  
get\_calories <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/calories/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_calories <- get\_calories(baseDate = "2018-11-20", period = "3m")  
calories <- content(got\_calories)  
# turn into df  
calories[['activities-calories']] <- jsonlist\_to\_df(calories[['activities-calories']])  
# assign easy object and rename  
calories <- calories[['activities-calories']]  
colnames(calories) <- c("dateTime", "calories")

### STEPS  
get\_steps <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/steps/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_steps <- get\_steps(baseDate = "2018-11-20", period = "3m")  
steps <- content(got\_steps)  
# turn into df  
steps[['activities-steps']] <- jsonlist\_to\_df(steps[['activities-steps']])  
# assign easy object and rename  
steps <- steps[['activities-steps']]  
colnames(steps) <- c("dateTime", "steps")

### DISTANCE  
get\_distance <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/distance/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_distance <- get\_distance(baseDate = "2018-11-20", period = "3m")  
distance <- content(got\_distance)  
# turn into df  
distance[['activities-distance']] <- jsonlist\_to\_df(distance[['activities-distance']])  
# assign easy object and rename  
distance <- distance[['activities-distance']]  
colnames(distance) <- c("dateTime", "distance")

### FLOORS  
get\_floors <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/floors/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_floors <- get\_floors(baseDate = "2018-11-20", period = "3m")  
floors <- content(got\_floors)  
# turn into df  
floors[['activities-floors']] <- jsonlist\_to\_df(floors[['activities-floors']])  
# assign easy object and rename  
floors <- floors[['activities-floors']]  
colnames(floors) <- c("dateTime", "floors")

### ELEVATION  
get\_elevation <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/elevation/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_elevation <- get\_elevation(baseDate = "2018-11-20", period = "3m")  
elevation <- content(got\_elevation)  
# turn into df  
elevation[['activities-elevation']] <- jsonlist\_to\_df(elevation[['activities-elevation']])  
# assign easy object and rename  
elevation <- elevation[['activities-elevation']]  
colnames(elevation) <- c("dateTime", "elevation")

### minutesSedentary  
get\_minutesSedentary <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/minutesSedentary/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_minutesSedentary <- get\_minutesSedentary(baseDate = "2018-11-20", period = "3m")  
minutesSedentary <- content(got\_minutesSedentary)  
# turn into df  
minutesSedentary[['activities-minutesSedentary']] <- jsonlist\_to\_df(minutesSedentary[['activities-minutesSedentary']])  
# assign easy object and rename  
minutesSedentary <- minutesSedentary[['activities-minutesSedentary']]  
colnames(minutesSedentary) <- c("dateTime", "minutesSedentary")

### minutesLightlyActive  
get\_minutesLightlyActive <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/minutesLightlyActive/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_minutesLightlyActive <- get\_minutesLightlyActive(baseDate = "2018-11-20", period = "3m")  
minutesLightlyActive <- content(got\_minutesLightlyActive)  
# turn into df  
minutesLightlyActive[['activities-minutesLightlyActive']] <- jsonlist\_to\_df(minutesLightlyActive[['activities-minutesLightlyActive']])  
# assign easy object and rename  
minutesLightlyActive <- minutesLightlyActive[['activities-minutesLightlyActive']]  
colnames(minutesLightlyActive) <- c("dateTime", "minutesLightlyActive")

### minutesFairlyActive  
get\_minutesFairlyActive <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/minutesFairlyActive/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_minutesFairlyActive <- get\_minutesFairlyActive(baseDate = "2018-11-20", period = "3m")  
minutesFairlyActive <- content(got\_minutesFairlyActive)  
# turn into df  
minutesFairlyActive[['activities-minutesFairlyActive']] <- jsonlist\_to\_df(minutesFairlyActive[['activities-minutesFairlyActive']])  
# assign easy object and rename  
minutesFairlyActive <- minutesFairlyActive[['activities-minutesFairlyActive']]  
colnames(minutesFairlyActive) <- c("dateTime", "minutesFairlyActive")

### minutesVeryActive  
get\_minutesVeryActive <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/minutesVeryActive/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_minutesVeryActive <- get\_minutesVeryActive(baseDate = "2018-11-20", period = "3m")  
minutesVeryActive <- content(got\_minutesVeryActive)  
# turn into df  
minutesVeryActive[['activities-minutesVeryActive']] <- jsonlist\_to\_df(minutesVeryActive[['activities-minutesVeryActive']])  
# assign easy object and rename  
minutesVeryActive <- minutesVeryActive[['activities-minutesVeryActive']]  
colnames(minutesVeryActive) <- c("dateTime", "minutesVeryActive")

### activityCalories  
get\_activityCalories <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/activityCalories/date/>', baseDate, "/", period, '.json'),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
   
got\_activityCalories <- get\_activityCalories(baseDate = "2018-11-20", period = "3m")  
activityCalories <- content(got\_activityCalories)  
# turn into df  
activityCalories[['activities-activityCalories']] <- jsonlist\_to\_df(activityCalories[['activities-activityCalories']])  
# assign easy object and rename  
activityCalories <- activityCalories[['activities-activityCalories']]  
colnames(activityCalories) <- c("dateTime", "activityCalories")

##### Join multiple dataframes with purrr::reduce and dplyr::left\_join  
activity\_df <- list(calories, steps, distance, floors, elevation, activityCalories, minutesSedentary, minutesLightlyActive, minutesFairlyActive, minutesVeryActive) %>%   
 purrr::reduce(left\_join, by = "dateTime")

# Add the dateTime to this dataframe  
activity\_df$dateTime <- as.Date(activity\_df$dateTime)

names <- c(2:ncol(activity\_df))  
activity\_df[,names] <- lapply(activity\_df[,names], as.numeric)

# trim off any days that haven't happened yet  
activity\_df %<>% filter(dateTime <= "2018-11-06")

**Get Recent Activity Types**

# We're giving it all she's got!  
  
get\_frequentActivities <- function(baseDate = NULL, period = NULL, token = Sys.getenv('FITB\_AUTH')){  
 GET(url = paste0('<https://api.fitbit.com/1/user/-/activities/recent.json'>),  
 add\_headers(Authorization = paste0("Bearer ", token)))  
}  
  
got\_frequentActivities <- get\_frequentActivities(baseDate = "2018-11-20", period = "3m")  
frequentActivities <- content(got\_frequentActivities)

# This is a list object let's look at how many frequent activities are logged  
length(frequentActivities)  
## [1] 5

# Take a look at the object with str()  
str(frequentActivities)

## List of 5  
## $ :List of 6  
## ..$ activityId : int 2131  
## ..$ calories : int 0  
## ..$ description: chr ""  
## ..$ distance : int 0  
## ..$ duration : int 3038000  
## ..$ name : chr "Weights"  
## $ :List of 6  
## ..$ activityId : int 90009  
## ..$ calories : int 0  
## ..$ description: chr "Running - 5 mph (12 min/mile)"  
## ..$ distance : int 0  
## ..$ duration : int 1767000  
## ..$ name : chr "Run"  
## $ :List of 6  
## ..$ activityId : int 90013  
## ..$ calories : int 0  
## ..$ description: chr "Walking less than 2 mph, strolling very slowly"  
## ..$ distance : int 0  
## ..$ duration : int 2407000  
## ..$ name : chr "Walk"  
## $ :List of 6  
## ..$ activityId : int 90001  
## ..$ calories : int 0  
## ..$ description: chr "Very Leisurely - Less than 10 mph"  
## ..$ distance : int 0  
## ..$ duration : int 4236000  
## ..$ name : chr "Bike"  
## $ :List of 6  
## ..$ activityId : int 15000  
## ..$ calories : int 0  
## ..$ description: chr ""  
## ..$ distance : int 0  
## ..$ duration : int 1229000  
## ..$ name : chr "Sport"

I would never have considered myself a [Darwin](https://www.psychologytoday.com/us/blog/how-think-about-exercise/201501/charles-darwins-daily-walks) or a [Thoreau](https://en.wikipedia.org/wiki/Walking_(Thoreau)) but apparently *strolling very slowly* is my favorite activity in terms of time spent.

You can see that my Fitbit has also logged times for Weights, Sports and Biking which is likely from when I’ve manually logged my activities. There’s a possibility that Fitbit is registering Biking for when I skateboard.

**Correlogram of Activity**

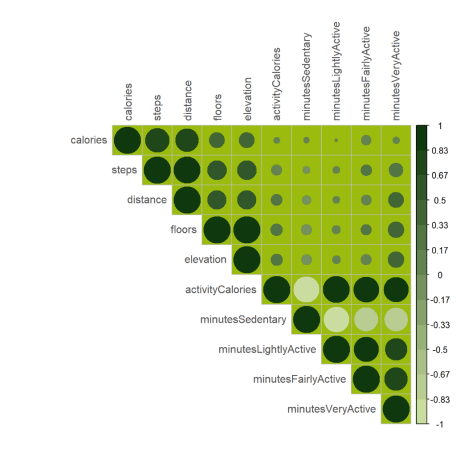
Previously I had always used the corrplot package to create a correlation plot; however, it doesn’t play nicely with ggplot meaning you cannot add Game Boy themes easily. Nonetheless, I was able to give it a retro-looking palette with some minor tweaking.

Since I had two colors in mind from the original gameboy, and knew their hex code, I was able to [generate a palette from this website](https://meyerweb.com/eric/tools/color-blend/#CADCA0:0F380E:10:hex).

# Aligning Covariance Matrices

# drop dateTime  
corr\_df <- activity\_df[,2:11]

# Correlation matrix  
corr <- cor(na.omit(corr\_df))  
corrplot(corr, type = "upper", bg = "#9BBB0E", tl.col = "#565656", col = c("#CADCA0", "#B9CD93", "#A8BE85", "#97AF78", "#86A06B", "#75915E", "#648350", "#537443", "#426536", "#315629", "#20471B", "#0F380E"))

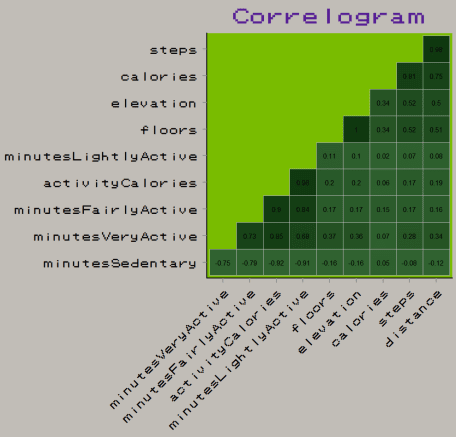


In a correlation plot the color of each circle indicates the magnitude of the correlation, and the size of the circle indicates its significance.

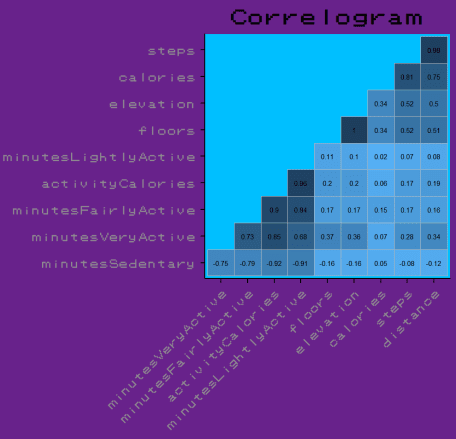
After a bit of searching for a ggplot2 extension I was able to use ggcorrplot which allowed me to use gameboy themes again!

# Generating textures...

ggcorrplot(corr, hc.order = TRUE,   
 type = "lower",   
 lab = TRUE,   
 lab\_size = 2,  
 tl.cex = 8,  
 show.legend = FALSE,  
 colors = c( "#306230", "#306230", "#0F380F" ),   
 title="Correlogram",  
 ggtheme=theme\_gameboy)

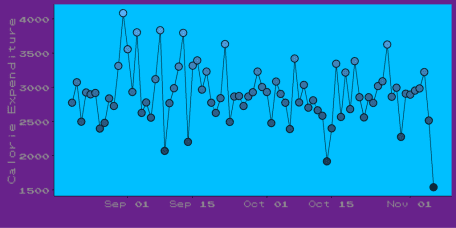
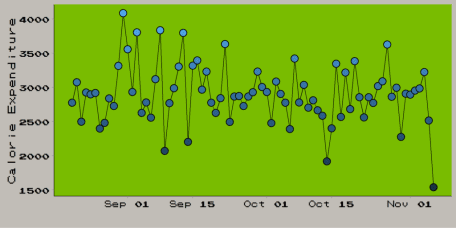


# Game Over. Loading previous save  
  
ggcorrplot(corr, hc.order = TRUE,   
 type = "lower",   
 lab = TRUE,   
 lab\_size = 2,  
 tl.cex = 8,  
 show.legend = FALSE,  
 colors = c( "#3B7AAD", "#56B1F7", "#1D3E5D" ),   
 title="Correlogram",  
 ggtheme=theme\_gba)

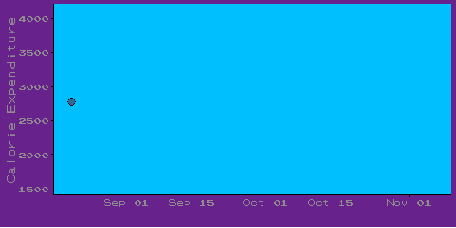


**Exploring Activity**

# Link saying "hyahhh!"  
  
# Static  
g <- activity\_df %>%   
 ggplot(aes(x=dateTime, y=calories)) +   
 geom\_line(colour = "black") +  
 geom\_point(shape = 21, colour = "black", aes(fill = calories), size = 5, stroke = 1) +  
 xlab("") +  
 ylab("Calorie Expenditure")  
  
g + theme\_gameboy() + theme(legend.position = "none")  
g + theme\_gba() + theme(legend.position = "none")



# Panick! at the Discord...  
  
# gganimate  
g <- activity\_df %>%   
 ggplot(aes(x=dateTime, y=calories)) +   
 geom\_line(colour = "black") +  
 geom\_point(shape = 21, colour = "black", aes(fill = calories), size = 5, stroke = 1) +  
 transition\_time(dateTime) +  
 shadow\_mark() +  
 ease\_aes('linear') +  
 xlab("") +  
 ylab("Calorie Expenditure")   
  
g + theme\_gba() + theme(legend.position = "none")



Distance is determined by using your steps and your estimated stride length (for the height you put in).

I’ve also made plots for Distance, Steps, Elevationand Floorsbut you’ll have to [check out this page to see them](https://moldach.github.io/fitbit-project/).

**Closing thoughts**

Even though Fitbit offers a nice dashboard for a single user it’s not scale-able. By accessing the data directly one can ask the questions they want from 200 individuals — or more. If one was inclined, they could even build a fancy Shiny dashboard with bespoke visualizations.

If you have any questions or comments you can always reach me on [LinkedIn](https://www.linkedin.com/in/matthewoldach/). Till then, see you in the next post!

# Wubba Lubba Dub Dub  
  
# <https://www.spriters-resource.com/game_boy_advance/kirbynim/sheet/15585/>  
sprite\_sheet <- png::readPNG("kirby.png")  
  
Nframes <- 11 # number of frames to extract  
width <- 29 # width of a frame  
sprite\_frames <- list() # storage for the extracted frames  
  
# Not equal sized frames in the sprite sheet. Need to compensate for each frame  
offset <- c(0, -4, -6, -7, -10, -16, -22, -26, -28, -29, -30)  
  
# Manually extract each frame  
for (i in seq(Nframes)) {  
 sprite\_frames[[i]] <- sprite\_sheet[120:148, (width\*(i-1)) + (1:width) + offset[i], 1:3]  
}  
  
# Function to convert a sprite frame to a data.frame  
# and remove any background pixels i.e. #00DBFF  
sprite\_frame\_to\_df <- function(frame) {  
 plot\_df <- data\_frame(  
 fill = as.vector(as.raster(frame)),  
 x = rep(1:width, width),  
 y = rep(width:1, each=width)  
 ) %>%  
 filter(fill != '#00DBFF')  
}  
  
sprite\_dfs <- sprite\_frames %>%  
 map(sprite\_frame\_to\_df) %>%  
 imap(~mutate(.x, idx=.y))  
  
fill\_manual\_values <- unique(sprite\_dfs[[1]]$fill)  
fill\_manual\_values <- setNames(fill\_manual\_values, fill\_manual\_values)  
  
mega\_df <- dplyr::bind\_rows(sprite\_dfs)  
  
p <- ggplot(mega\_df, aes(x, y, fill=fill)) +  
 geom\_tile(width=0.9, height=0.9) +  
 coord\_equal(xlim=c(1, width), ylim=c(1, width)) +  
 scale\_fill\_manual(values = fill\_manual\_values) +  
 theme\_gba() +  
 xlab("") +  
 ylab("") +  
 theme(legend.position = 'none', axis.text=element\_blank(), axis.ticks = element\_blank())  
  
panim <- p +  
 transition\_manual(idx, seq\_along(sprite\_frames)) +  
 labs(title = "gganimate Kirby")  
  
gganimate::animate(panim, fps=30, width=400, height=400)

