In this post, we will see how to extract step count, heart rate, and activity data from the Xiaomi Mi-Band 5 inexpensive personal tracker that was released in July of 2020. I bought one in August after my Fitbit die Fitbit, but was not very pleased the lifespan of the previous one (the devices are apparently designed not

One nice thing about the Mi-Band is that it’s relatively straightforward to extract the activity data from the follows: first, we will install a modified Mi-Fit app in order to get access to an “auth key” which allows othe install Gadgetbridge, an open source application that interfaces with and collects the data recorded by th export the raw data from the device, and then use R to extract the data and format them for analysis.

The steps outlined below are for Android mobile devices (because that’s the kind of phone I have, though phones). You can find the complete code on Github here. Let’s get started!

# Part 1: Install the Modified Mi-Band App and Extr

The first step is to install a modified Mi-Band app and extract the “Auth Key.” If you’re comfortable with Python, you can also apparently get the *auth key* progr myself), but for this post we won’t assume any Python knowledge.

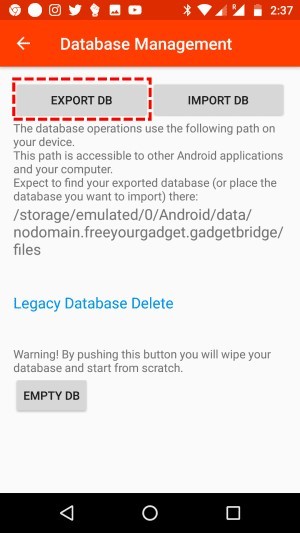
# Part 2: Install F-Droid & Gadgetbridge

Next, install F-Droid, a free software repository (like Google Play) for open source apps on Android. From an open source mobile application that can interface with a number of different tracking devices (Pebble,

You can find the information on how to connect the Mi-Band to Gadgetbridge using the auth key

# Part 3: Export the Data

It is very easy to export the data from the Mi-Band using Gadgetbridge.



When you navigate to the correct folder on your phone, you will find the database in SQLite format, simpl file as input.

# Part 4: Extract and Clean the Data

Because Gadgetbridge works with a number of different fitness trackers, the SQLite database contains m we are exporting data from the Mi-Band, we need to extract data from the table named *“MI\_BAND\_ACTI*

This table contains the following variables:

**TIMESTAMP:** This represents the time that a given row of data was recorded, using the Unix time **DEVICE\_ID:** This is an ID code that tracks the device from which a given row of data was devices linked to Gadgetbridge.

**USER\_ID:** This is an ID code that indicates which user’s information is recorded in the given row users using multiple devices linked to Gadgetbridge.

**RAW\_INTENSITY:** This variable is a code representing the activity that is recorded in the given and am not alone in that regard.

**STEPS:** This represents the number of steps recorded in a given row of data.

**RAW\_KIND:** This appears to have something to do with activity codes, particularly for sleep. As clear agreement on what these codes mean.

**HEART\_RATE:** The heart rate measurement for the given row of data.

## Granularity of the Data

One of the main issues is that the granularity in the recorded data is quite high. Specifically, data are writ are missing because not all parameters are measured every second. The maximum frequency for heart r step counts are also written to the database once per minute.

In our extraction of the data, therefore, we will not read the second-level data into R. If you’ve recorded m millions of rows if you don’t make some sensible exclusions in the query to the SQLite database.

In the code below, I make an immediate subset in the SQL query, selecting rows where the *HEART\_RAT RAW\_INTENSITY* is variable is not equal to -1 (which occurs when there are missing values for both of t complete data for the variables that I care the most about: steps and heart rate.

## Making Sense of the Time Stamp

The time stamp for each observation is recorded in Unix time, e.g. the number of seconds since January format, I use the lubridate package to convert the values to an R date-time format. I specify the time zone conversion to when the measurements were taken.

## Adding More Date Information

Because I often want to analyze step count data at the day/hour level, I extract the date and hour and ad lubridate package.

## The Code to Extract the Data

The following code performs all of the operations discussed above. It returns a dataset with 1 row per mi contained in the columns.

# load the libraries we'll use library(DBI) library(lubridate) library(plyr); library(dplyr)

# define the directory where the data are stored in\_dir <- 'D:\\Directory\\'

# function to read in the data

read\_gadgetbridge\_data <- function(in\_dir\_f, db\_name\_f){ # connect to the sqlite data base

con = dbConnect(RSQLite::SQLite(), dbname=paste0(in\_dir\_f, db\_name\_f)) # load the table with the Mi-Fit walking info

# (MI\_BAND\_ACTIVITY\_SAMPLE)

# the others contain other information not relevant for this exercise # select on HEART\_RATE and RAW\_INTENSITY to get non-missing observatio # otherwise, size of data is huge b/c it records 1 line / second raw\_data\_f = dbGetQuery(con,'select \* from MI\_BAND\_ACTIVITY\_SAMPLE whe

HEART\_RATE != -1 and RAW\_INTENSITY != -1' )

# close the sql connection dbDisconnect(con)

# Convert unix timestamp to proper R date object # make sure to set the timezone to your location!

raw\_data\_f$TIMESTAMP\_CLEAN <- lubridate::as\_datetime(raw\_data\_f$TIMEST "Europe/Paris")

# format the date for later aggregation

raw\_data\_f$hour <- lubridate::hour(raw\_data\_f$TIMESTAMP\_CLEAN) year\_f <- lubridate::year(raw\_data\_f$TIMESTAMP\_CLEAN)

month\_f <- lubridate::month(raw\_data\_f$TIMESTAMP\_CLEAN) day\_f <- lubridate::day(raw\_data\_f$TIMESTAMP\_CLEAN) raw\_data\_f$date <- paste(year\_f, month\_f, day\_f, sep = '-') return(raw\_data\_f)

}

# load the raw data with the function

raw\_data\_df <- read\_gadgetbridge\_data(in\_dir, 'Gadgetbridge')

Here is a sample of the data extracted from the SQLite database with the above function:

**TIMESTAMP DEVICE\_ID USER\_ID RAW\_INTENSITY STEPS RAW\_KIND HEART\_RATE TIMESTAM**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1598598000 | 1 | 1 | 36 | 0 | 96 | 74 | 2020-08-28 |
| 1598598060 | 1 | 1 | 57 | 21 | 1 | 76 | 2020-08-28 |
| 1598598120 | 1 | 1 | 82 | 37 | 1 | 89 | 2020-08-28 |
| 1598598180 | 1 | 1 | 29 | 7 | 16 | 87 | 2020-08-28 |
| 1598598240 | 1 | 1 | 48 | 0 | 96 | 81 | 2020-08-28 |
| 1598598300 | 1 | 1 | 26 | 0 | 96 | 79 | 2020-08-28 |
| 1598598360 | 1 | 1 | 49 | 0 | 80 | 78 | 2020-08-28 |
| 1598598420 | 1 | 1 | 72 | 12 | 80 | 90 | 2020-08-28 |
| 1598598480 | 1 | 1 | 78 | 39 | 17 | 83 | 2020-08-28 |
| 1598598540 | 1 | 1 | 40 | 0 | 16 | 74 | 2020-08-28 |

# Part 5: Aggregate to the Day / Hour Level

The raw data are recorded at the second level, and above we’ve done an extraction to obtain information still a bit too detailed for analyzing step count data.

The function below takes our second-level data frame and aggregates it to the day / hour level. It first set than 250 (it’s unrealistic to ever have a heart rate this high and 255 is a code representing missing readin

I then group the data by day and hour, and create three summary statistics: the total sum of steps within deviation of the heart rate measurements. I also add a column containing the cumulative steps per day (t device itself - the number of steps taken so far that day). Finally, I group the data by date and add a colu

The last step in this function adds information about the day of the week for each observation, and then c weekday or a weekend These data are now in a s previous devices - Accupedo and Fitbit.

The code to perform these steps looks like this:

# make the aggregation per day / hour make\_hour\_aggregation <- function(input\_data\_f){

# set values of greater than 250 to NA input\_data\_f$HEART\_RATE[input\_data\_f$HEART\_RATE > 250] <- NA

# aggregate to day / hour day\_hour\_agg\_f <- input\_data\_f %>%

group\_by(date, hour) %>%

summarize(hourly\_steps = sum(STEPS, na.rm = T), mean\_heart\_rate = mean(HEART\_RATE, na.rm = T), sd\_heart\_rate = sd(HEART\_RATE, na.rm = T)) %>%

# create column for cumulative sum mutate(cumulative\_daily\_steps = cumsum(hourly\_steps)) %>% # create column for daily total

group\_by(date) %>% mutate(daily\_total = sum(hourly\_steps))

# add day of the week

day\_hour\_agg\_f$dow <- wday(day\_hour\_agg\_f$date, label = TRUE)

# add a weekday/weekend variable day\_hour\_agg\_f$week\_weekend <- NA

day\_hour\_agg\_f$week\_weekend[day\_hour\_agg\_f$dow == 'Sun'] <- 'Weekend' day\_hour\_agg\_f$week\_weekend[day\_hour\_agg\_f$dow == 'Sat'] <- 'Weekend' day\_hour\_agg\_f$week\_weekend[day\_hour\_agg\_f$dow == 'Mon'] <- 'Weekday' day\_hour\_agg\_f$week\_weekend[day\_hour\_agg\_f$dow == 'Tue'] <- 'Weekday' day\_hour\_agg\_f$week\_weekend[day\_hour\_agg\_f$dow == 'Wed'] <- 'Weekday' day\_hour\_agg\_f$week\_weekend[day\_hour\_agg\_f$dow == 'Thu'] <- 'Weekday' day\_hour\_agg\_f$week\_weekend[day\_hour\_agg\_f$dow == 'Fri'] <- 'Weekday'

# put the columns in the right order

day\_hour\_agg\_f <- day\_hour\_agg\_f %>% select(date, daily\_total, hour,

hourly\_steps, cumulative\_ dow, week\_weekend, mean\_he

sd\_heart\_rate)

# add column meta-data for device day\_hour\_agg\_f$device <- 'MiBand'

return(day\_hour\_agg\_f)

}

omnibus\_mi\_band <- make\_hour\_aggregation(raw\_data\_df)

Which returns a dataset named *omnibus\_mi\_band* that looks like this:

### date daily\_total hour hourly\_steps cumulative\_daily\_steps dow week\_weekend mean\_heart\_

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2020-8-28 | 24800 | 9 | 3213 | 3747 | Fri | Weekday | 82.51 |
| 2020-8-28 | 24800 | 10 | 6010 | 9757 | Fri | Weekday | 89.53 |
| 2020-8-28 | 24800 | 11 | 1370 | 11127 | Fri | Weekday | 85.08 |
| 2020-8-28 | 24800 | 12 | 791 | 11918 | Fri | Weekday | 82.02 |
| 2020-8-28 | 24800 | 13 | 184 | 12102 | Fri | Weekday | 80.98 |
| 2020-8-28 | 24800 | 14 | 5827 | 17929 | Fri | Weekday | 89.14 |
| 2020-8-28 | 24800 | 15 | 771 | 18700 | Fri | Weekday | 84.68 |
| 2020-8-28 | 24800 | 16 | 4276 | 22976 | Fri | Weekday | 84.82 |

**date daily\_total hour hourly\_steps cumulative\_daily\_steps dow week\_weekend mean\_heart\_**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2020-8-28 | 24800 | 17 | 448 | 23424 | Fri | Weekday | 79.85 |
| 2020-8-28 | 24800 | 18 | 484 | 23908 | Fri | Weekday | 80.07 |

# Part 6: Data Checks

When doing this type of data extraction and aggregation, it’s always best to do some checks to make sur least 3 different checks that we can do.

1. Count the number of observations per day. If we have done things correctly, we should have 24 ob Gadgetbridge and the day I extracted the data). We can check this with the following code:

# we should have 24 observations / day

# except the first day using Gadgetbridge # and the date of data extraction table(omnibus\_mi\_band$date) table(table(omnibus\_mi\_band$date))

Which returns:

### 2020-10-1 2020-10-10 2020-10-2 2020-10-3 2020-10-4 2020-10-5 2020-10-6 2020-10-7 2020-10-8 202

24 18 24 24 24 24 24 24 24

and

### 18 22 24

1 1 47

Indeed, there are 24 observations for all days except the first day I used the Mi-Fit (2020-8-23) and the d

1. Check the step count figures in our data frame with the step count figures displayed in the Gadget total step counts extracted in the above data should match the totals given in the app. If this is the during our data extraction and transformations. In the example data shown above, the cumulative t the total step count given in the Gadgetbridge app for that date!
2. Check the step count figures in our data frame with the step count figures displayed on the device cumulative and total daily steps for the most recent observation in the data set should match the st when extracting the data and conducting the analysis). Every time I’ve done this, the figures have l is retrieving all the data correctly.

# Part 7: Make a Plot

One of the best ways of exploring data is by making graphs or plots of relationships among variables. The hours of the day, with separate loess regression lines for weekdays and weekends.

# load the ggplot2 package library(ggplot2)

# make the plot

ggplot(data = omnibus\_mi\_band, aes(x = hour, y = cumulative\_daily\_steps,

+

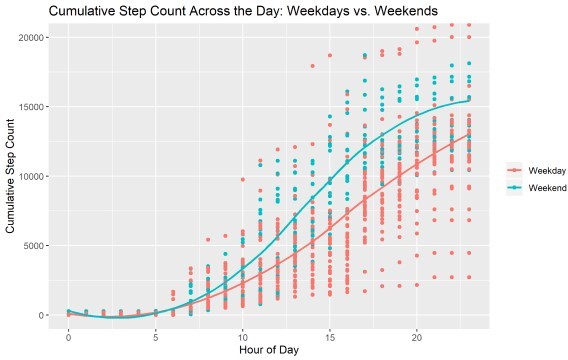
geom\_point() +

coord\_cartesian(ylim = c(0, 20000)) + geom\_smooth(method="loess", fill=NA) + theme(legend.title=element\_blank()) +

labs(x = "Hour of Day", y = "Cumulative Step Count",

title = 'Cumulative Step Count Across the Day: Weekdays vs. Weeke

Which returns the following plot:



This plot is very interesting - the patterns are quite different .I think this is partially because I walk a bit less than I did 3 years ago, and partially because Accup does.

Second, the patterns of step counts during weekdays and weekends have reversed! Three years ago, I w that point, I walked back and forth to work every day, which meant that my step count was rather high du between the end of August and the beginning of October 2020. This was during COVID time, and I was During this period, I tried to walk a fair amount every day, but it’s not easy to do 15,000 steps when your

# Summary and Conclusion

In this post, we saw how to extract data from the Mi-Band 5 with Gadgetbridge. We first used a modified connect the open source Gadgetbridge app to the Mi-Band device. We used Gadgetbridge to export an S Band. We used R to extract the raw data per minute, and then aggregated those data to the day / hour le and saw how my walking patterns differ on weekdays versus weekends. …