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." This YouTube video gives a the modified app here). 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.

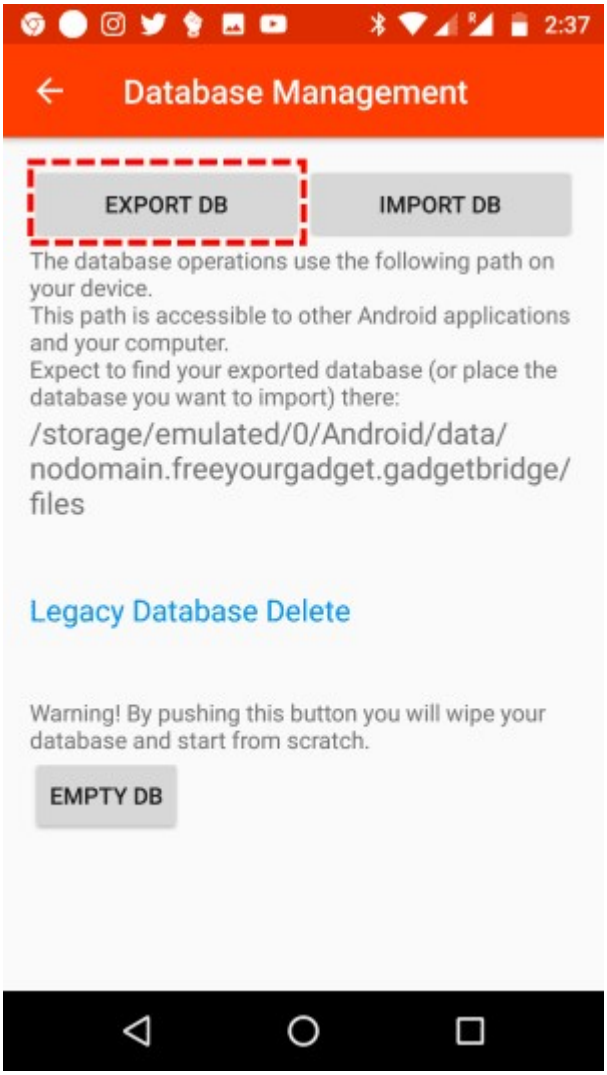
Part 2: Install F-Droid & Gadgetbridge

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 in the Yo

Part 3: Export the Data

It is very easy to export the data from the Mi-Band using Gadgetbridge. You can check out this web-page "Database Management" in the app and select "Export DB" (see the picture below). The data will be cont



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 recorded devices linked to Gadgetbridge.
- **USER_ID:** This is an ID code that indicates which user's information is recorded in the given row o users using multiple devices linked to Gadgetbridge.
- **RAW_INTENSITY:** This variable is a code representing the activity that is recorded in the given rov 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 wi clear agreement on what these codes mean (see here for more info).
- **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 writt
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 th
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 mir
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	TIMESTAMI
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. In previous blog posts, I've done quite a bit of data an (instead of steps per minute).

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 t 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 colur

The last step in this function adds information about the day of the week for each observation, and then c weekday or a weekend (I know from previous analyses of my steps that the patterns are quite different or columns and add a meta-data column called "device" which is set to 'MiBand'. These data are now in a si 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_hea
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	2020
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 da

2. Check the step count figures in our data frame with the step count figures displayed in the Gadgetb 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!

3. 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

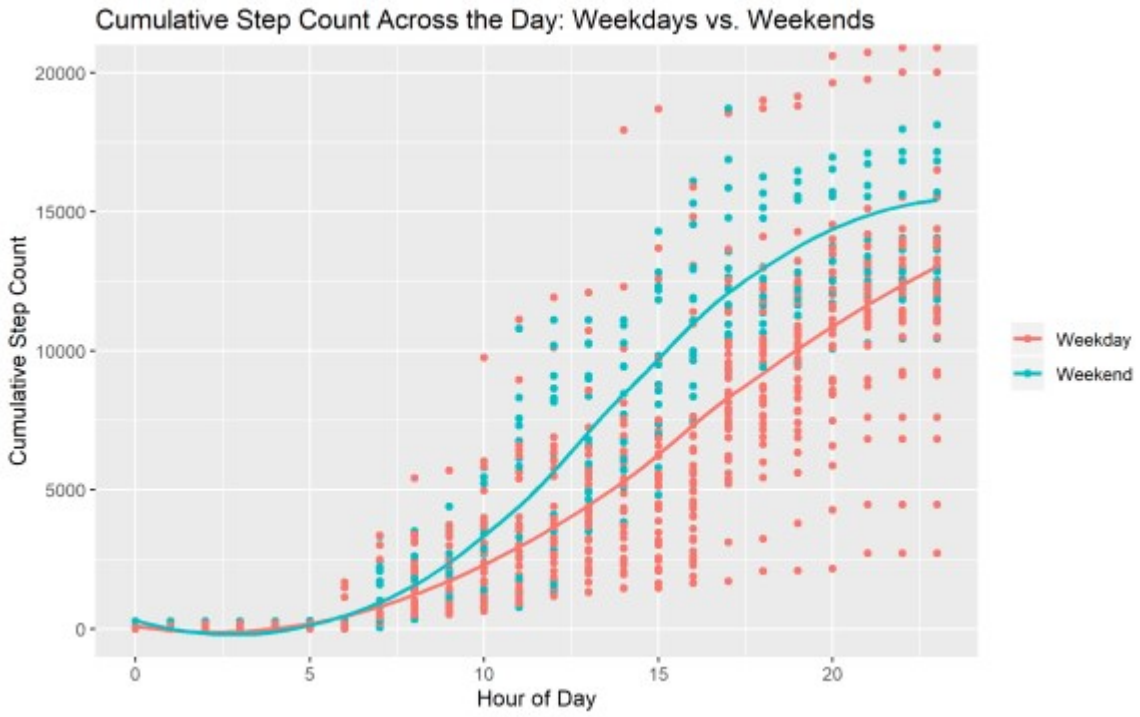
One of the best ways of exploring data is by making graphs or plots of relationships among variables. Let here. We will borrow some plotting code from the very first blog post I ever did about step count data. Spe the hours of the day, with separate loess regression lines for weekdays and weekends.

```
# 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. Weeker
```

Which returns the following plot:



This plot is very interesting - the patterns are quite different than they were in the first blog post about my
lower. 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 dur
between the end of August and the beginning of October 2020. This was during COVID time, and I was w
During this period, I tried to walk a fair amount every day, but it's not easy to do 15,000 steps when your c

Summary and Conclusion

In this post, we saw how to extract data from the Mi-Band 5 with Gadgetbridge. We first used a modified l
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. …