In this post, we will see how to download personal Fitbit data histories for step counts, heart rate, and sleep via the Fitbit API. We will use a combination of existing R packages and custom calls to the Fitbit API to get all of the data we are interested in.

This post won’t focus on data analysis per se, but rather data collection. As I was going about the exercise of retrieving my own Fitbit data, I noticed that there were no good examples of collecting one’s entire data history (but a number of [descriptions](http://benjaminnelson.strikingly.com/blog/download-fitbit-data-in-r) of getting a [single day’s](https://obrl-soil.github.io/fitbit-api-r/) [worth](https://erle.io/blog/working-with-fitbit-data-in-r/) of data). Because data gathering is such a fundamental part of the data science exercise, I think it’s worth it to go into detail about how to access one’s personal Fitbit data history via API!

**Step 1: Getting Set Up**

*Create a Developer Account*

The first thing we need to do is create a developer account with Fitbit, in order to get a key and secret to access the API. I won’t go into the details here, but you can check out [these](https://github.com/teramonagi/fitbitr) [two](https://obrl-soil.github.io/fitbit-api-r/) very clear and detailed descriptions of how to go through this process – the guides make it very easy. Make sure to select “Personal” for the *OAuth 2.0 Application Type*, or you won’t be able to access intra-day time series information (e.g. step or heart rate data at the minute-level).

*Set Up the R Environment*

Next, we will make sure we have everything loaded in our R environment. Below, I specify the Fitbit key and secret I obtained in the above procedure. I also specify the directory to which I will save the data I’ve downloaded, and create a vector containing the dates for which I’ve had my Fitbit. We will use this vector of dates in our calls to the API.

# obtain the key and secret for your fitbit account  
# procedure described here: https://github.com/teramonagi/fitbitr  
# and here: https://obrl-soil.github.io/fitbit-api-r/  
# make sure to choose personal usage for access  
# to intra-day time series data!  
FITBIT\_KEY <- "XXXYYY"  
FITBIT\_SECRET <- "xxxxxxyyyyyyyyxxxxxxxxxyyyyyyxxx"  
FITBIT\_CALLBACK <- "http://localhost:1410/"  
  
# load the packages we will use  
# make sure you install the fitbit r package (from github)  
# (you'll need to install the devtools package if you haven't already)  
# devtools::install\_github("teramonagi/fitbitr")  
library(fitbitr)  
library(plyr);library(dplyr)  
library(lubridate)  
library(httr)  
  
# choose folder to which we will save the data we extract  
out\_dir = 'C:\\Directory\\Fitbit\\Data\\Raw\\'  
  
# Get token  
token <- fitbitr::oauth\_token(language="en\_US")  
token$token  
  
# make list of dates: these are the dates that   
# I had the fitbit and wore it...  
# (uses lubridate package)  
fitbit\_dates <- seq(ymd('2018-03-20'),ymd('2018-12-18'), by = '1 day')

The workhorse package for most of this exercise is the excellent [**fitbitr**](https://github.com/teramonagi/fitbitr) package. This package comes with commands to easily obtain a token to access the Fitbit API, and to easily compose queries to request data. When you get the token, a separate window from your internet browser will open and you will have to specify which data you want to access via the API (see the guides above for more details).

I’ve created a vector of dates (called *fitbit\_dates*) that correspond to the period of time that I have had the Fitbit (ending at the time this blog post was prepared – December 2018). The vector contains 274 dates. We will use this as input for our code below, extracting step count and heart rate data for every date in the *fitbit\_dates* vector.

Note that you will need to replace the key and secret with the ones you obtain from the Fitbit developer platform!

**Step 2: Obtaining Step Count Data**

*Testing with a Single Day*

We will use the “*get\_activity\_intraday\_time\_series*” function included in the **fitbitr** package to download the intraday time series (at the minute-level) for the step data. The function in the **fitbitr** package is simply a wrapper that takes an activity (e.g. steps, calories, etc.), a date, a level of granularity for the requested information (1 or 15 minutes) and executes an API call, returning a cleaned data frame with the requested data. For more information about the data available through the intraday time series API, you can check out the [**fitbitr**](https://github.com/teramonagi/fitbitr#activity) and [Fitbit API documentation](https://dev.fitbit.com/build/reference/web-api/activity/#get-activity-intraday-time-series).

Let’s test a single day with the *get\_activity\_intraday\_time\_series* function in the **fitbitr** package. We will adapt the test code described in the [**fitbitr**](https://github.com/teramonagi/fitbitr#how-to-use) example.

The code looks like this:

# test date  
fitbit\_dates[1]  
# [1] "2018-03-20"  
# test out the function  
df <- get\_activity\_intraday\_time\_series(token, "steps", fitbit\_dates[1], detail\_level="1min")  
df$time <- as.POSIXct(strptime(paste0(df$dateTime, " ", df$dataset\_time), "%Y-%m-%d %H:%M:%S"))

And the resulting dataframe looks like this:

|  | **dateTime** | **value** | **dataset\_time** | **dataset\_value** | **datasetInterval** | **datasetType** | **time** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2018-03-20 | 15562 | 00:00:00 | 0 | 1 | minute | 2018-03-20 00:00:00 |
| 2 | 2018-03-20 | 15562 | 00:01:00 | 0 | 1 | minute | 2018-03-20 00:01:00 |
| 3 | 2018-03-20 | 15562 | 00:02:00 | 0 | 1 | minute | 2018-03-20 00:02:00 |
| 4 | 2018-03-20 | 15562 | 00:03:00 | 0 | 1 | minute | 2018-03-20 00:03:00 |
| 5 | 2018-03-20 | 15562 | 00:04:00 | 0 | 1 | minute | 2018-03-20 00:04:00 |
| 6 | 2018-03-20 | 15562 | 00:05:00 | 0 | 1 | minute | 2018-03-20 00:05:00 |
| 7 | 2018-03-20 | 15562 | 00:06:00 | 0 | 1 | minute | 2018-03-20 00:06:00 |
| 8 | 2018-03-20 | 15562 | 00:07:00 | 0 | 1 | minute | 2018-03-20 00:07:00 |
| 9 | 2018-03-20 | 15562 | 00:08:00 | 0 | 1 | minute | 2018-03-20 00:08:00 |
| 10 | 2018-03-20 | 15562 | 00:09:00 | 0 | 1 | minute | 2018-03-20 00:09:00 |

The data set contains information on the date, the total number of steps walked on that day (15,562 for 2018-03-20), and the number of steps (called *value* in the data set) for the given minute. We also have meta data describing the granularity of the measurement – we are recording steps in 1-minute intervals. Finally, the last line in the code (taken directly from the **fitbitr** documentation) creates an R date object called *time* with the day, hour, minute and second level information all combined in one variable. There are 1,440 lines in our dataset – 1 line for each minute of the day.  
    
*Getting Data for All Days*

We now have working code that obtains data for a single day. We simply need a way to programmatically execute this procedure for the 274 dates in our “*fitbit\_dates*” vector.

Below, I accomplish this via a function which will be applied to our vector of dates. For each date in the vector, we execute the *get\_activity\_intraday\_time\_series* command, obtaining the data in the format shown above.

# function to download data for all dates  
intraday\_steps\_list\_df <- lapply(fitbit\_dates, function(x){  
 df <- get\_activity\_intraday\_time\_series(token, "steps", x, detail\_level="1min")  
 # where are we in the list?  
 print('just downloaded:')  
 print(x)  
 print(Sys.time())  
 Sys.sleep(30)  
 df$time <- as.POSIXct(strptime(paste0(df$dateTime, " ", df$dataset\_time), "%Y-%m-%d %H:%M:%S"))  
 return(df)  
})  
  
# make a single dataframe from the list of dataframes  
# dplyr solution  
intraday\_steps\_df <- bind\_rows(intraday\_steps\_list\_df)  
  
# Save the dataframe  
saveRDS(intraday\_steps\_df, file = paste0(out\_dir,"intraday\_steps\_df.rds"))

Note that I include a [Sys.sleep](http://sys.sleep/) command in the function. This causes the program to pause for 30 seconds before continuing. I included this because the Fitbit API has a [rate limit of 150 calls per hour](https://dev.fitbit.com/build/reference/web-api/basics/#rate-limits). With the above function, we should make around 2 calls per minute, or 120 calls per hour. We are guaranteed not to go above the limit!

I also include some *print* statements in the function. As the function executes, we get an update on where the function is in the list of dates to download, and the time at which the last download occurred. If we encounter an error, this information will help us debug the problem.

The function returns a list (because we use the [*lapply*](https://www.rdocumentation.org/packages/base/versions/3.5.1/topics/lapply?tap_a=5644-dce66f&tap_s=10907-287229) command) of data frames, one data frame for each date in our *fitbit\_dates* vector. I then make a single data frame from the list of data frames, and save the merged step data (called *intraday\_steps\_df*) to the directory specified above.

The output returned to the console during the execution of the function looks like this:

[1] "just downloaded:"  
[1] "2018-03-20"  
[1] "2018-12-29 15:25:27 CET"  
[1] "just downloaded:"  
[1] "2018-03-21"  
[1] "2018-12-29 15:25:57 CET"

...

*Some Basic Checks*

Below, I do some basic checks on the data we have obtained via the API. Our *intraday\_steps\_df* has 274 unique dates, with 1,440 observations for each day (this matches the number of observations from our test case above). The final check confirms that our master data frame contains data for each day in our *fitbit\_dates* vector!

# our fitbit\_dates vector contains 274 dates  
# our final df has 274 days  
length(unique(intraday\_steps\_df$dateTime))  
  
# how many observations do we have per day?   
# we have 1440 for every day  
table(table(intraday\_steps\_df$dateTime))  
  
# extract the days for which we have some data  
unique\_days\_in\_step\_data <- as.Date(unique(intraday\_steps\_df$dateTime))  
# compare to original list  
# which dates in original are NOT in extracted data?  
# none- we have them all!  
fitbit\_dates[!fitbit\_dates %in% unique\_days\_in\_step\_data]

**Step 3: Obtaining Heart Rate Data**

*Testing with a Single Day*  
   
In order to obtain minute-level heart rate data, we will use the [built in function](https://github.com/teramonagi/fitbitr#heart-rate) from the **fitbitr** package called “*get\_heart\_rate\_intraday\_time\_series*“. Unfortunately, this function returns a data set that is formatted much less nicely than the comparable function for steps used above. Specifically, the basic call using this command for our test date looks like this:

# heart rate intraday time series data  
# for our test date  
df <- get\_heart\_rate\_intraday\_time\_series(token, date=fitbit\_dates[1], detail\_level="1min")

And it returns a data set that is formatted like this:

|  | **time** | **value** |
| --- | --- | --- |
| 1 | 06:34:00 | 96 |
| 2 | 06:35:00 | 95 |
| 3 | 06:36:00 | 98 |
| 4 | 06:37:00 | 69 |
| 5 | 06:38:00 | 61 |
| 6 | 06:39:00 | 68 |
| 7 | 06:40:00 | 82 |
| 8 | 06:41:00 | 83 |
| 9 | 06:42:00 | 85 |
| 10 | 06:43:00 | 84 |

We are missing a lot of the great meta-data we have in the steps data. We don’t even have the date on which the measurements were taken!

*Getting Data for All Days*

I wrote a simple function to cycle through each date in our *fitbit\_dates* vector, and download and add important meta-data. There was one date (“2018-06-05”) that was problematic. On that date, there was no heart rate data, and so the call returned just a string with the date.

The function below contains an exception to handle this error, and returns a list of data frames (or date strings in the case of errors):

# function to download data for all dates  
intraday\_heart\_list\_df <- lapply(fitbit\_dates, function(x){  
 # get the heart data for 1 minute level  
 # for the date specified  
 df <- get\_heart\_rate\_intraday\_time\_series(token, date=x, detail\_level="1min")  
 # where are we in the list?  
 print('just downloaded:')  
 print(x)  
 print(Sys.time())  
 # pause so we don't exceed our api limit  
 Sys.sleep(30)  
 # exception: if there's no data, the length of the df  
 # is zero. If this is the case, we return the df, which  
 # is just a character string of the date  
 if (length(df)== 0){  
 return(df)  
 }  
 # if we have data for the date, we format them  
 else {  
 # rename the columns  
 names(df) <- c('dataset\_time', 'dataset\_value')  
 # add columns to match format of step dataframe  
 # interval: 1-minute  
 df$datasetInterval <- '1'  
 # type  
 df$datasetType <- 'minute'  
 # datetime (year, month, day)  
 df$dateTime <- x  
 # explicit R date object for day, hour, minute, second  
 df$time <- as.POSIXct(strptime(paste0(df$dateTime, " ", df$dataset\_time), "%Y-%m-%d %H:%M:%S"))  
 # re-arrange the columns to match format of step dataframe  
 df <- df[c("dateTime","dataset\_time","dataset\_value","datasetInterval" ,"datasetType","time")]  
 # return the cleaned data frame  
 return(df)  
 }  
})  
  
# make a single dataframe from the list of dataframes  
# dplyr solution  
intraday\_heart\_df <- bind\_rows(intraday\_heart\_list\_df)  
  
# Save the dataframe  
saveRDS(intraday\_heart\_df, file = paste0(out\_dir,"intraday\_heart\_df.rds"))

The head of our master data frame (called *intraday\_heart\_df*) looks like this:

|  | **dateTime** | **dataset\_time** | **dataset\_value** | **datasetInterval** | **datasetType** | **time** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2018-03-20 | 06:34:00 | 96 | 1 | minute | 2018-03-20 06:34:00 |
| 2 | 2018-03-20 | 06:35:00 | 95 | 1 | minute | 2018-03-20 06:35:00 |
| 3 | 2018-03-20 | 06:36:00 | 98 | 1 | minute | 2018-03-20 06:36:00 |
| 4 | 2018-03-20 | 06:37:00 | 69 | 1 | minute | 2018-03-20 06:37:00 |
| 5 | 2018-03-20 | 06:38:00 | 61 | 1 | minute | 2018-03-20 06:38:00 |
| 6 | 2018-03-20 | 06:39:00 | 68 | 1 | minute | 2018-03-20 06:39:00 |
| 7 | 2018-03-20 | 06:40:00 | 82 | 1 | minute | 2018-03-20 06:40:00 |
| 8 | 2018-03-20 | 06:41:00 | 83 | 1 | minute | 2018-03-20 06:41:00 |
| 9 | 2018-03-20 | 06:42:00 | 85 | 1 | minute | 2018-03-20 06:42:00 |
| 10 | 2018-03-20 | 06:43:00 | 84 | 1 | minute | 2018-03-20 06:43:00 |

It matches much more closely the format of the step count data we downloaded above!

*Some Basic Checks*

As we did above, let’s do some basic checks on the data we downloaded.

I first look and see how many of the dates in the *fitbit\_dates* vector were problematic. Only one date contained no data. For the problematic date identified above (“2018-06-05”), there is no data in the master data frame – this is as it should be. The one date from the *fitbit\_dates* vector that is not in our master data is “2018-06-05”, which is exactly what we would expect.

# how many of the lists have no observations?  
# of the 274 days, only 1 has length of zero  
table(sapply(intraday\_heart\_list\_df, function(x){  
 return(length(x))  
}))  
  
# our list of dfs had 274 entries, but one was empty  
# the resulting df has 273 entries (1 less - makes sense)  
length(unique(intraday\_heart\_df$dateTime))  
# and there are no rows in our df for the  
# date that we know was problematic  
intraday\_heart\_df[intraday\_heart\_df$dateTime == "2018-06-05",]  
  
# how many observations do we have per day? It depends... on days  
# where I wore it overnight, there could be close to 1400... other days where  
# I wore it for less - as low as 500  
table(table(intraday\_heart\_df$dateTime))  
  
# extract the days for which we have some data  
unique\_days\_in\_heart\_data <- unique(intraday\_heart\_df$dateTime)  
# compare to original dates we used to fetch the data  
# from the API  
# which dates in originals are NOT in extracted data?  
fitbit\_dates[!fitbit\_dates %in% unique\_days\_in\_heart\_data]  
# [1] "2018-06-05"  
# just the one we already knew was missing.  
# everything looks ok!

Unlike the step count data above, we do not have observations for each minute of each day. We seem only to have data for the minutes for which there was a measurement recorded by the Fitbit. When examining the number of observations per day, the lowest is just under 500, and the highest is 1437. It seems clear that, even when I wear the Fitbit all day, there are some moments of the day where my heart rate is not recorded (perhaps because the placement of the Fitbit on my wrist was not optimal during those points).

This does not seem particularly problematic. In further data analysis, we should simply keep this difference between the data structures in mind.

**Step 4: Obtaining Sleep Data**

The final piece of information which we will obtain in this post is the sleep data. These data were the least straightforward to obtain, for a number of reasons.

1. In the time since the **fitbitr** package was released, the Fitbit API has been updated (to version 1.2 as of end December 2018). The **fitbitr** package calls Version 1.0 of the API, which could be discontinued at any moment.

2. This change in the API coincides with a change in the sleep measurements calculated by Fitbit. In Version 1.0 of the API, the data returned are in the “classic” format, which contains three values: minutes *restless*, minutes *asleep*, and minutes *awake*. In version 1.2 of the API, the data are mostly returned in the “stages” format, which contains 4 values: *wake*, *light*, *deep*, and *rem*.

3. However, the “stages” data are not calculated for periods of sleep that are shorter than 3 hours. Therefore, Version 1.2 of the API returns “stages” data for periods of sleep that are longer than 3 hours, and “classic” data for periods of sleep that are shorter than 3 hours.

4. It is not possible to manually harmonize the data between the “classic” and “stages” formats.

In sum, it’s a bit of a mess (as of the writing of this blog post – end December 2018). There’s lots of discussion on the Fitbit developer forums, but I didn’t see any great solutions and I also noticed that other people are struggling with this same issue.

Therefore, we will simply request summary totals per night of the few variables that are measured consistently between the “classic” and “stages” data formats. We will build the API calls ourselves, without the interface of the **fitbitr** package, and will use version 1.2 of the API (the most recent version at the time of this writing).

We can call up to 100 days with each API request. I constructed the API calls using the Fitbit sleep API [documentation](https://dev.fitbit.com/build/reference/web-api/sleep/), and created two chunks of codes to download the data. I didn’t start wearing my Fitbit at night until “2018-06-19” and so we will use that as the start date for our requests.

The following code downloads the sleep data in two chunks (so we don’t exceed the API limits), makes a selection of the data which is consistent across “classic” and “stages” data formats, binds the data frames together, and saves the master file.

# load library for making API calls  
library(httr)  
# June 19 was when I started wearing it at night...  
# we can request 100 days from the API with a single call  
# first chunk: 2018-06-19 to 2018-09-25  
sleep\_1\_raw <- GET('https://api.fitbit.com/1.2/user/-/sleep/date/2018-06-19/2018-09-25.json',   
 httr::config(token = token$token))  
# extract the data  
# this returns a list with a dataframe  
# (which itself contains nested data frames)  
sleep\_transform\_1 <- jsonlite::fromJSON(httr::content(sleep\_1\_raw, as = "text"))  
# extract the sleep data frame from the list  
# this contains some basic pieces  
# of information that match across   
# classic and stages formats  
# e.g. efficiency, minutes awake & asleep, time in bed  
sleep\_transform\_1\_df <- sleep\_transform\_1$sleep  
# deselect the columns with "levels" in the title  
# these are the more granular data that are   
# formatted differently across classic and stages formats  
sleep\_transform\_1\_df$levels <- NULL  
  
# get the second chunk  
# dates between 2018-09-26 and 2018-12-18  
# we use the same procedure here  
sleep\_2\_raw <- GET('https://api.fitbit.com/1.2/user/-/sleep/date/2018-09-26/2018-12-18.json',   
 httr::config(token = token$token))  
sleep\_transform\_2 <- jsonlite::fromJSON(httr::content(sleep\_2\_raw, as = "text"))  
sleep\_transform\_2\_df <- sleep\_transform\_2$sleep  
sleep\_transform\_2\_df$levels <- NULL  
  
# merge the datasets together  
# and arrange by sleep date  
sleep\_master\_df <- bind\_rows(sleep\_transform\_1\_df, sleep\_transform\_2\_df) %>% arrange(dateOfSleep)  
  
# save the dataset  
saveRDS(sleep\_master\_df, file = paste0(out\_dir,"sleep\_master\_df.rds"))

The head of our master sleep data frame looks like this:

| **dateOfSleep** | **duration** | **efficiency** | **endTime** | **infoCode** | **logId** | **minutesAfterWakeup** | **minutesAsleep** | **minutesAwake** | **minutesToFallAsleep** | **startTime** | **timeInBed** | **type** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2018-06-19 | 29820000 | 93 | 2018-06-19T07:28:00.000 | 0 | 18620515648 | 1 | 403 | 94 | 0 | 2018-06-18T23:10:30.000 | 497 | stages |
| 2018-06-20 | 24780000 | 89 | 2018-06-20T07:13:00.000 | 0 | 18620515649 | 0 | 334 | 79 | 0 | 2018-06-20T00:20:00.000 | 413 | stages |
| 2018-06-21 | 28020000 | 86 | 2018-06-21T07:30:30.000 | 0 | 18620515650 | 0 | 386 | 81 | 0 | 2018-06-20T23:43:00.000 | 467 | stages |
| 2018-06-23 | 3840000 | 92 | 2018-06-23T17:42:00.000 | 2 | 18651040599 | 0 | 59 | 5 | 0 | 2018-06-23T16:38:00.000 | 64 | classic |
| 2018-06-26 | 17880000 | 91 | 2018-06-26T05:55:00.000 | 0 | 18686114178 | 0 | 257 | 41 | 0 | 2018-06-26T00:57:00.000 | 298 | stages |
| 2018-06-27 | 25440000 | 90 | 2018-06-27T06:39:00.000 | 0 | 18686114179 | 0 | 367 | 57 | 0 | 2018-06-26T23:35:00.000 | 424 | stages |
| 2018-06-28 | 23640000 | 92 | 2018-06-28T05:05:30.000 | 0 | 18697091620 | 0 | 328 | 66 | 0 | 2018-06-27T22:31:30.000 | 394 | stages |
| 2018-06-29 | 21300000 | 93 | 2018-06-29T05:08:00.000 | 0 | 18706624030 | 0 | 311 | 44 | 0 | 2018-06-28T23:12:30.000 | 355 | stages |
| 2018-06-30 | 33600000 | 93 | 2018-06-30T07:31:30.000 | 0 | 18717877597 | 0 | 504 | 56 | 0 | 2018-06-29T22:11:30.000 | 560 | stages |
| 2018-07-02 | 32460000 | 91 | 2018-07-02T07:52:30.000 | 0 | 18740188120 | 0 | 469 | 72 | 0 | 2018-07-01T22:51:00.000 | 541 | stages |

We have some basic information about each night’s sleep – the date, number of minutes asleep, the number of minutes awake, and the time spent in bed. The information is not very granular in comparison with the step count and heart rate data, but given the circumstances, this is the most consistent information we can extract across all sleep episodes as of the writing of this blog post. (Any suggestions or improvements are welcome – please let me know in the comments section below!)

*Some Basic Checks*

Finally, let’s do some basic checks on the sleep data that we’ve downloaded. There are 193 lines in the data set, corresponding to 193 sleep episodes across the time I’ve been wearing the Fitbit at night.

Below, I check the number of sleep observations across the days. There were 138 days with 1 recorded sleep episode, 23 with 2 sleep episodes (naps, clearly), and 3 days where I apparently slept 3 different times!

# how many days have what number of observations?  
table(table(sleep\_master\_df$dateOfSleep))  
# 1 2 3   
# 138 23 3   
  
# make a vector of dates corresponding  
# to those used to call the API  
# (using lubridate package)  
sleep\_dates <- seq(ymd('2018-06-19'),ymd('2018-12-18'), by = '1 day')  
# there are 183 different dates in the range  
length(sleep\_dates)  
  
# extract the days for which we have some data  
unique\_sleep\_dates <- as.Date(unique(sleep\_master\_df$dateOfSleep))  
# compare to original dates we used to fetch the data  
# from the API  
# which dates in originals are NOT in extracted data?  
sleep\_dates[!sleep\_dates %in% unique\_sleep\_dates]  
# [1] "2018-06-22" "2018-06-24" "2018-06-25" "2018-07-01" "2018-07-06" "2018-07-09" "2018-07-15" "2018-08-05" "2018-09-01" "2018-09-23" "2018-09-29"  
# [12] "2018-10-05" "2018-10-12" "2018-10-19" "2018-11-01" "2018-11-16" "2018-11-23" "2018-11-30" "2018-12-07"  
# there are 19 missing dates...  
length(sleep\_dates[!sleep\_dates %in% unique\_sleep\_dates])

I also checked to see how many of the dates were missing data. There were 183 days in the time window we used to retrieve the data, but 19 of these dates were missing from the data we got back from the API. This doesn’t seem crazy – when the battery on my Fitbit is low, I usually charge it overnight, meaning that it cannot record any information about my sleeping patterns.

All in all, it looks like the retrieval of the summary sleep data was successful!

**Summary and Conclusion**

In this blog post, we saw how to download data from a Fitbit step counter. The first step was to register a developer account at the Fitbit website. With the key and secret from this process, it is possible to request one’s own individual data from Fitbit via their API.

We used the **fitbitr** package to extract the step count and heart rate data at the most granular level possible. We used a custom function to download the data for each day, pausing after downloading each day’s data to avoid exceeding the API limits. We created one master data set for the step count data, and one for the heart rate data. The sleep data were more complicated to gather correctly, given the current formats of the data and their availability from the API. Nevertheless, we were able to extract summary statistics for each night by making calls directly to the API.

We’ll save the analysis of these data for a future post. However, I’m glad to have taken the time to talk in detail about the data retrieval process, as this is a critical but under-appreciated aspect of data science. It is my hope that this post will be helpful to anyone who is looking to extract and analyze the complete history of the data from their Fitbits.