Extracting personal activity data using the Fitbit API

Dave Ferguson

7 April 2017

The Fitbit API allows people who record their activity using Fitbit products to access their Fitbit data in more detail than that which is available via website intefaces. This document lays out the steps required to solve the problem of accessing and downloading personal Fitbit activity data and preparing it for analysis using R and the Fitbit API.

Before you start

In order to access the API, the following items are needed:

- A free account with fitbit (read the documentation and register at dev.fitbit.com)
- A registered 'application' in order to get API credentials

The following R packages are used as part of the extraction and cleaning:

- `httr` - to make and receive http requests
- `jsonlite` - to parse the JSON formatted repsonses
- `tidyverse` - for data management & manipulation
- `lubridate` - for date manipulation
- `stringr` - for string manipulation

Available data types & authentication

There are a range of data types available depending on the type of monitoring device you own. Before authenticating you need to specify which data types you woud like to access under the current auithentication request. I have chosen:

- Sleep
- Activity (steps)
- Heart rate
- my profile information

Authenticating with the API requires the secret and key provided when registering your application to be passed to the Fitbit oauth2 request URL.

```
# 1. Set up required data types and credentials
  scope <- c("sleep","activity","heartrate","profile")  # See
dev.fitbit.com/docs/oauth2/#scope
  fitbit_endpoint <- oauth_endpoint(
    request = "https://api.fitbit.com/oauth2/token",
    authorize = "https://www.fitbit.com/oauth2/authorize",
```

```
    access = "https://api.fitbit.com/oauth2/token")
  myapp <- oauth_app(
    appname = "app_data",
    key = "mykey",
    secret = "mysecret")

# 2. Get OAuth token
  fitbit_token <- oauth2.0_token(fitbit_endpoint, myapp,
                                 scope = scope, use_basic_auth = TRUE)
```

A browser page will open and prompt you to log into your fitbit developer account to finalise the authentication. You will also have the option use a local file ('.httr-oauth'), to cache the OAuth access credentials between R sessions. For this to work reliably, you should make sure that your working folder is the same one you are saving the file to.

Retreiving the data

Data is retreived via a series of calls to the API, with the URL structure varying depending on the data type you are trying to extract. The approach I have taken is to follow the following pattern:

1. specify a variable with a range of dates for which data is required
2. initialise one or more varailes in which retreived data will be held
3. create a loop that retreives the data for each of the dates in the range
4. parse the retreived JSON string to extract the specific data required
5. incrementally (before looping to the next date) add the fields to a data set for the variable

Sample code for the data types in this example are included below.

Heartrate

```
# Set sequence of dates for which data is required
  s <- seq.Date(from = as.Date("2017-03-01"), to = as.Date("2017-03-31"), by
= 1)

### Heartrate
  # api URL format: "https://api.fitbit.com/1/user/-
/activities/heart/date/2016-09-01/1d/1min.json"

  # create target dataframe
  HRMinuteData = NULL

  # Make API requests, looping through date sequence
  for (i in seq_along(s)) {
    # Form the request URL
    geturl <- paste(c("https://api.fitbit.com/1/user/-
/activities/heart/date/",as.character(s[[i]]),"/1d/1min.json"), sep ="",
collapse = "")
    # Request and parse the data
```

```
    resp <- GET(url = geturl, config(token = fitbit_token))
    get2json<- content(resp, as = "parsed")
    rawdata<- fromJSON(toJSON(get2json))

    # minute by minute data
    hrdf <- data.frame(rawdata[["activities-heart-intraday"]])
    # dateTime to add to dataframe
    hrdate <- toString(rawdata[["activities-heart"]][["dateTime"]])
    hrdf$dateTime <- hrdate
    # merge to combined dataframe
    HRMinuteData <- rbind(HRMinuteData, hrdf)
  }

  #cleanup
  HRMinuteData <- as_data_frame(HRMinuteData)
  rm(hrdf)
  HRMinuteData$dataset.value <- as.numeric(HRMinuteData$dataset.value)
  names(HRMinuteData)[names(HRMinuteData) == 'dataset.value'] <- 'value'

### END HEART
```

Sleep

Sleep has some intricacies for when you manage to get a power nap and record a second sleep period in the day. The JSON parsing becomes a little more challenging.

```
### SLEEP (Handles multiple sleep periods in a day)

  #Set up vecotrs to hold info
  vawakeCount = NULL
  vdateOfSleep = NULL
  vefficiency = NULL
  vstartTime = NULL
  vminutesAsleep = NULL
  vminutesAwake = NULL
  vrestlessCount = NULL
  vrestlessDuration = NULL
  SleepMinuteData = NULL

  # Loop through date sequence
  for (i in seq_along(s)) {
    # Form the request URL
    geturl <- paste(c("https://api.fitbit.com/1/user/-
/sleep/date/",as.character(s[[i]]),".json"), sep ="", collapse = "")
    # Request and parse the data
    resp <- GET(url = geturl, config(token = fitbit_token))
    get2json<- content(resp, as = "parsed")
    rawdata<- fromJSON(toJSON(get2json))

    # Need error checking in here for a response with an error
```

```
    # if there is an error then this will get the message, else be null
    #rawdata[["errors"]][["message"]][[1]]

    #check for multiple sleep periods
    sleeps <- rawdata[["summary"]][["totalSleepRecords"]]

    if (sleeps > 0) {
      for(j in 1:sleeps) {
        # add to vectors for the list of raw data summary items we want
        vawakeCount = append(vawakeCount,
rawdata[["sleep"]][["awakeCount"]][[j]])
        vdateOfSleep = append(vdateOfSleep,
rawdata[["sleep"]][["dateOfSleep"]][[j]])
        vefficiency = append(vefficiency,
rawdata[["sleep"]][["efficiency"]][[j]])
        vstartTime = append(vstartTime,
rawdata[["sleep"]][["startTime"]][[j]])
        vminutesAsleep = append(vminutesAsleep,
rawdata[["sleep"]][["minutesAsleep"]][[j]])
        vminutesAwake = append(vminutesAwake,
rawdata[["sleep"]][["minutesAwake"]][[j]])
        vrestlessCount = append(vrestlessCount,
rawdata[["sleep"]][["restlessCount"]][[j]])
        vrestlessDuration = append(vrestlessDuration,
rawdata[["sleep"]][["restlessDuration"]][[j]])


      # minute by minute data
      sleepdf <- data.frame(rawdata[["sleep"]][["minuteData"]][[j]])
      sleepdf$startTime <- rawdata[["sleep"]][["startTime"]][[j]]
      #merge to combined df
      SleepMinuteData <- rbind(SleepMinuteData, sleepdf)

      }
    }
  }

  #cleanup
  sleepSummary = cbind(vdateOfSleep, vawakeCount, vefficiency, vstartTime,
vminutesAsleep, vminutesAwake, vrestlessCount, vrestlessDuration)
  sleepSummary <- as_data_frame(sleepSummary)
  SleepMinuteData <- as_data_frame(SleepMinuteData)
  sleepSummary$vawakeCount <- as.numeric(sleepSummary$vawakeCount)
  sleepSummary$vefficiency <- as.numeric(sleepSummary$vefficiency)
  sleepSummary$vminutesAsleep <- as.numeric(sleepSummary$vminutesAsleep)
  sleepSummary$vminutesAwake <- as.numeric(sleepSummary$vminutesAwake)
  sleepSummary$vrestlessCount <- as.numeric(sleepSummary$vrestlessCount)
  sleepSummary$vrestlessDuration <-
as.numeric(sleepSummary$vrestlessDuration)
```

```
    rm(sleepdf)

### END SLEEP
```

Steps

```
### STEPS

#geturl <- ""https://api.fitbit.com/1/user/-/activities/log/steps/date/2016-
09-01/2016-09-01.json", options"

  StepMinuteData = NULL

  # Make API requests, looping through date sequence
  for (i in seq_along(s)) {
    # Form the request URL
    geturl <- paste(c("https://api.fitbit.com/1/user/-
/activities/steps/date/",as.character(s[[i]]),"/1d/1min.json"), sep ="",
collapse = "")
    # Request and parse the data
    resp <- GET(url = geturl, config(token = fitbit_token))
    get2json<- content(resp, as = "parsed")
    rawdata<- fromJSON(toJSON(get2json))

    # minute by minute data
    stepday <- data.frame(rawdata[["activities-steps-
intraday"]][["dataset"]])
    # dateTime to add to dataframe
    stepday$dateTime <- rawdata[["activities-steps"]][["dateTime"]]
    # merge to combined dataframe
    StepMinuteData <- rbind(StepMinuteData, stepday)
  }

  # cleanup
  rm(stepday)
  StepMinuteData <- as_data_frame(StepMinuteData)
  StepMinuteData$value <- as.numeric(StepMinuteData$value)

### END Steps
```

Data cleaning

The parsing of the JSON responses and the limited data fields included in them results in some additional data cleaning being required.

Cleaning included:

- converting the data into dataframes

- converting lists to numeric variables and factors
- creating a timestamp field from the date and time components that was common across the individual data sets
- creating some summary data sets for plotting

```
  HRMinuteData <- mutate(HRMinuteData, Time_Stamp = make_datetime(
    str_sub(HRMinuteData$dateTime,1,4),
    str_sub(HRMinuteData$dateTime,6,7),
    str_sub(HRMinuteData$dateTime,9,10),
    str_sub(HRMinuteData$dataset.time,1,2),
    str_sub(HRMinuteData$dataset.time,4,5)))

  HRMinuteData <- mutate(HRMinuteData, hrhour = make_datetime(
    str_sub(HRMinuteData$dateTime,1,4),
    str_sub(HRMinuteData$dateTime,6,7),
    str_sub(HRMinuteData$dateTime,9,10),
    str_sub(HRMinuteData$dataset.time,1,2),
    0))
  names(HRMinuteData)[names(HRMinuteData) == "value"] = "heartRate"

  StepMinuteData <- StepMinuteData %>%
    mutate( Time_Stamp = make_datetime(
    str_sub(dateTime,1,4),
    str_sub(dateTime,6,7),
    str_sub(dateTime,9,10),
    str_sub(time,1,2),
    str_sub(time,4,5)))
  StepMinuteData$time <- hms(StepMinuteData$time)
  names(StepMinuteData)[names(StepMinuteData) == "value"] = "stepCount"

  SleepMinuteData$Time_Stamp <- make_datetime(
      str_sub(SleepMinuteData$startTime,1,4),
      str_sub(SleepMinuteData$startTime,6,7),
      as.numeric(str_sub(SleepMinuteData$startTime,9,10)) + 1 -
as.numeric(hms(str_sub(SleepMinuteData$startTime,12,19)) <
hms(SleepMinuteData$dateTime)),
      str_sub(SleepMinuteData$dateTime,1,2),
      str_sub(SleepMinuteData$dateTime,4,5)
    )
  names(SleepMinuteData)[names(SleepMinuteData) == "value"] = "SleepValue"

  # #convert numeric sleep type to string factor
  SleepTypes <- c(asleep = 1, restless = 2, awake = 3)
  SleepMinuteData$SleepType <- factor(SleepMinuteData$SleepValue, levels =
SleepTypes, labels = names(SleepTypes))


  # Create some summaries for use in plotting
    by_day <- group_by(HRMinuteData, dateTime, hrhour)
```

```
    HRSummary <- summarise(by_day,
                           count = n(),
                           hrmean = mean(heartRate, na.rm = TRUE),
                           hrmedian = median(heartRate, na.rm = TRUE),
                           hrmin = min(heartRate, na.rm = TRUE),
                           hrmax = max(heartRate, na.rm = TRUE)
    )


    StepMinuteData$dateTime <-ymd(StepMinuteData$dateTime)
    by_day_steps <- group_by(StepMinuteData, dateTime)
    StepSummary <- summarise(by_day_steps,
                             count = n(),
                             stepmean = mean(stepCount, na.rm = TRUE),
                             stepmedian = median(stepCount, na.rm = TRUE),
                             stepmin = min(stepCount, na.rm = TRUE),
                             stepmax = max(stepCount, na.rm = TRUE),
                             stepcount = sum(stepCount, na.rm = TRUE)

    )

    sleeptidy <- sleepSummary %>%
      gather(`vminutesAsleep`, `vminutesAwake`, `vrestlessDuration`, key =
"sleeptype", value = "minutes")
    sleeptidy$sleeptype[sleeptidy$sleeptype=="vrestlessDuration"] <-
"Restless"
    sleeptidy$sleeptype[sleeptidy$sleeptype=="vminutesAsleep"] <- "Asleep"
    sleeptidy$sleeptype[sleeptidy$sleeptype=="vminutesAwake"] <- "Awake"

    sleeptypes <- c("Asleep", "Restless", "Awake")
    sleeptidy$sleeptype <- factor(sleeptidy$sleeptype, levels = sleeptypes)
```

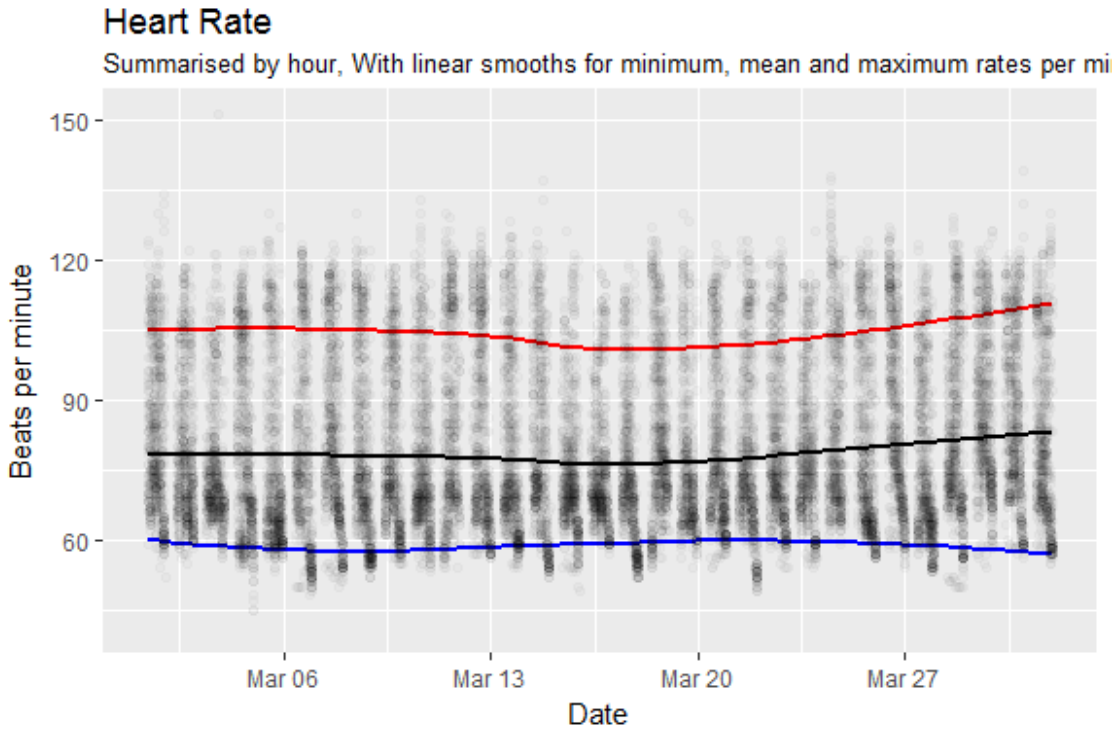
Some sample plots

While some more interesting observations might come from joining the extracted activity data with other data sets (exercise sessions or qualitative observatiosn such as tiredness, coffee consumption, relationship observations), there may still be some value in plotting the individual data types.

Heartrate

```
  # Heart rate
    ggplot(data = HRSummary) +
      geom_smooth(mapping = aes(x = hrhour, y = hrmean), colour = "black" ,
se = FALSE) +
      geom_smooth(mapping = aes(x = hrhour, y = hrmin), colour = "blue", se =
FALSE) +
      geom_smooth(mapping = aes(x = hrhour, y = hrmax), colour = "red", se =
FALSE) +
      geom_point(data = HRMinuteData, mapping = aes(x = Time_Stamp, y =
```

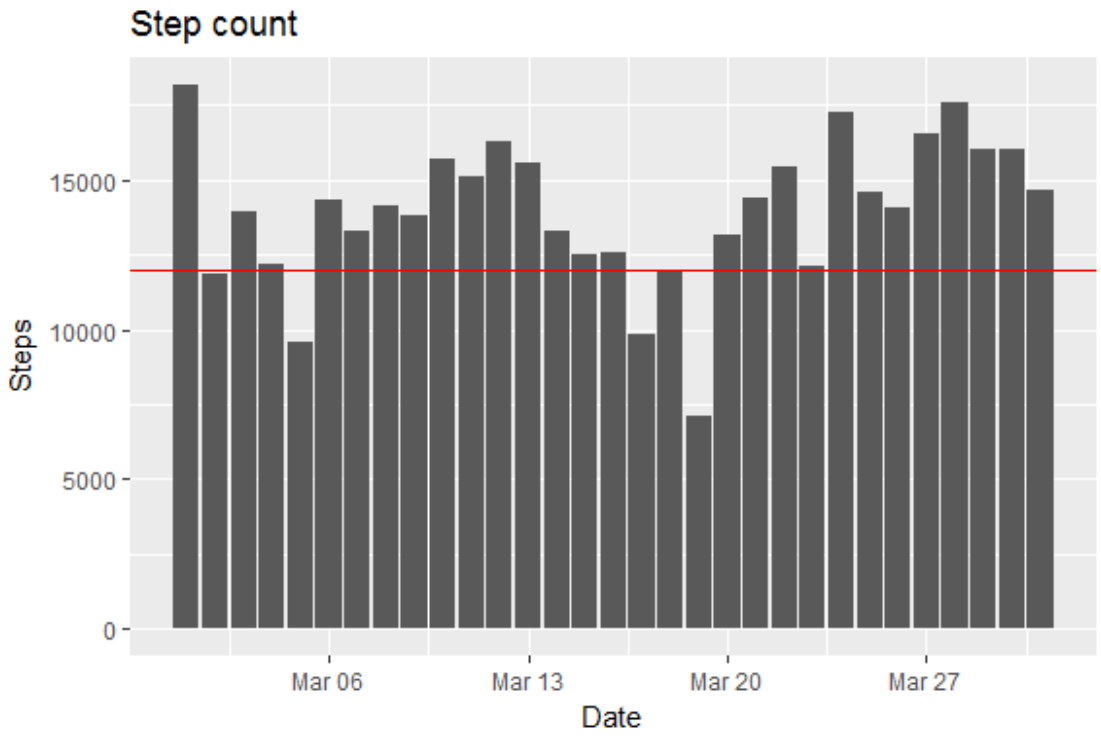
```
heartRate), alpha = 0.02) +
      labs(title = "Heart Rate", x = "Date", y = "Beats per minute",
           subtitle = "Summarised by hour, With linear smooths for minimum,
mean and maximum rates per minute")
```



Hourly heart rate summary over the period.

Steps

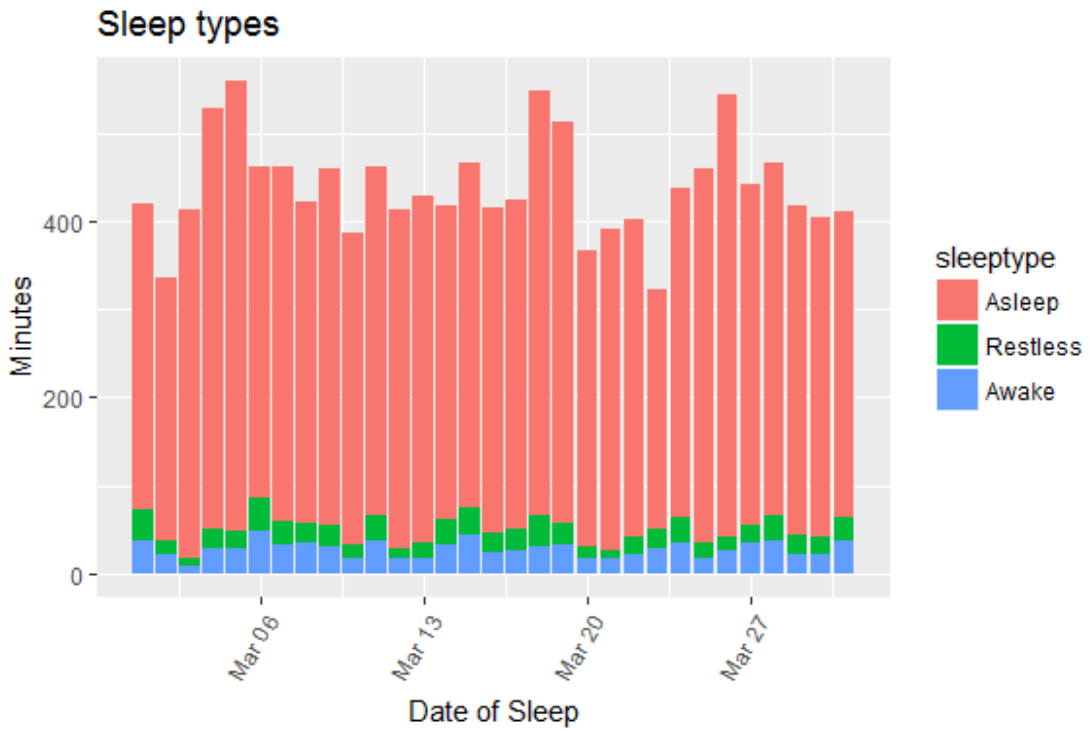
```
  #Steps
  ggplot(data = StepSummary) +
    geom_bar(aes(x=dateTime, weight = stepcount)) +
    geom_hline(aes(yintercept = 12000), colour = "red") +
    labs(title = "Step count", x = "Date", y = "Steps")
```



Daily step count with target.

Sleep

```
#Sleep
ggplot(data = sleeptidy) +
  geom_bar(aes(x = vdateOfSleep,  weight = minutes, fill = sleeptype)) +
  labs(title = "Sleep types", x = "Date of Sleep", y = "Minutes") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)
        )
```



Daily sleep by sleep type.

Future enhancements

There are a number of future enhancements which could be integrated into this process. They include more robust handling of the JSON results and improvements in the data cleaning steps. Another improvement would be to integrate the data manipulation used inthe plots into the ggplot calls.

Reference list

Fitbit, Inc 2017, *Fitbit Web API Documentaion*, viewed 5 April 2017, (https://dev.fitbit.com/docs/)

Wickham, H. & Grolemund, G. 2016, *R for Data Science*, viewed 7 April 2017, (http://r4ds.had.co.nz/)