Using the Fitbit Web API with Python, MongoDB and Streamlit

A Beginner's Guide for Storing and Visualizing FitBit data

Photo from FitBit site

In today's data-driven world, the ability to collect, store, and analyze data has become more critical than ever before. Public APIs offer an excellent way to acquire data from various sources, and the Fitbit Web API is a perfect example of this.

It provides access to a wealth of fitness-related data, including step counts, heart rate, and sleep patterns, making up a rich source of information for developers, data scientists, and fitness enthusiasts.

In this tutorial, we will explore how to use the Fitbit Web API to acquire data, store it in a NoSQL database and visualize it on a web platform using Streamlit, an open-source Python library for building beautiful custom web apps for machine learning and data science.

With this tutorial, you will learn how to extract insights from your fitness data and create a custom dashboard that displays your progress over time. Whether you are a fitness enthusiast or a data scientist, this tutorial is a valuable resource for anyone interested in working with Fitbit Web API data.

• • •

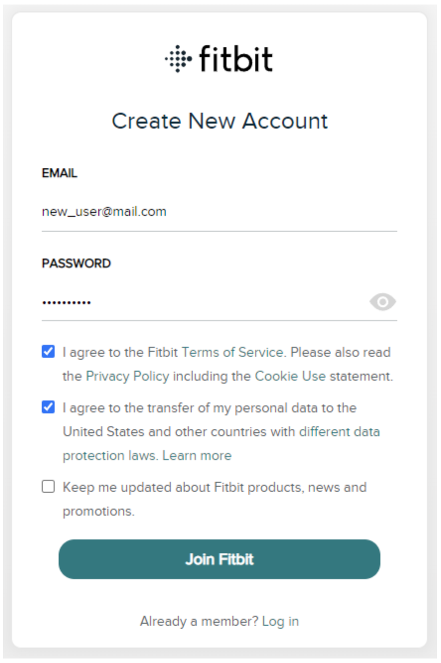
Creating a Fitbit developer account and registering an application

The wearable that was used to collect fitness data from Fitbit Web API is **Fitbit Sense 2**, a health and fitness-based smartwatch.

Before starting collecting data, we need to follow some steps, which are based on this tutorial.

Fitbit Sense 2

Firstly, we must create a Fitbit account from here. Just use a valid email and a password.



fitbit

Create New Account

EMAIL

new_user@mail.com

PASSWORD

....

☑ I agree to the Fitbit Terms of Service. Please also read the Privacy Policy including the Cookie Use statement.

☑ I agree to the transfer of my personal data to the United States and other countries with different data protection laws. Learn more

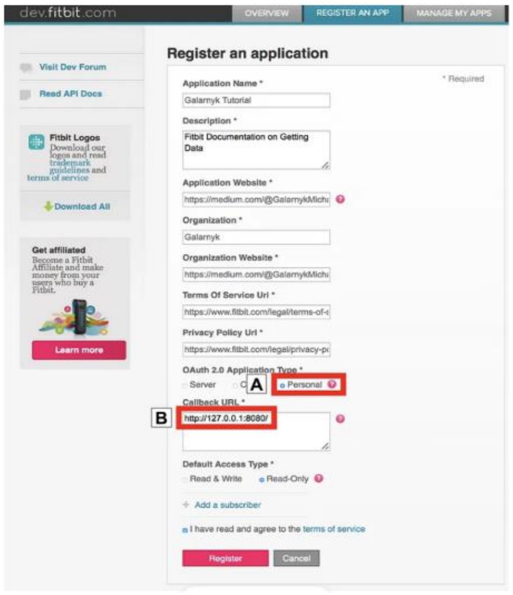
☐ Keep me updated about Fitbit products, news and promotions.

Join Fitbit

Already a member? Log in

After you successfully create an account, go to dev.fitbit.com and from the upper right corner of the page, select **Manage/Register An App.**

You will see a page like the below one (taken from the tutorial we attached before):



Register an application

* Required

Application Name *: Galarnyk Tutorial

Description *: Fitbit Documentation on Getting Data

Application Website *: https://medium.com/@GalarnykMich

Organization *: Galarnyk

Organization Website *: https://medium.com/@GalarnykMich

Terms Of Service Url *: https://www.fitbit.com/legal/terms-of-

Privacy Policy Url *: https://www.fitbit.com/legal/privacy-p

OAuth 2.0 Application Type *: Server C [✓ Personal]

Callback URL *: http://127.0.0.1:8080/

Default Access Type *: Read & Write [✓ Read-Only]

[✓] I have read and agree to the terms of service

You need to specify **Personal** Application Type to be able to ask to download intraday data. The callback URL is http://127.0.0.1:8080, because the Python API we will use, has it as the default redirect URL.

Next steps are pretty straightforward, so we will not mention them here. As mentioned in the tutorial, keep in mind to write down the CLIENT_ID and CLIENT_SECRET provided during the registration, as there are useful for the next step.

• • •

API authorization in Python

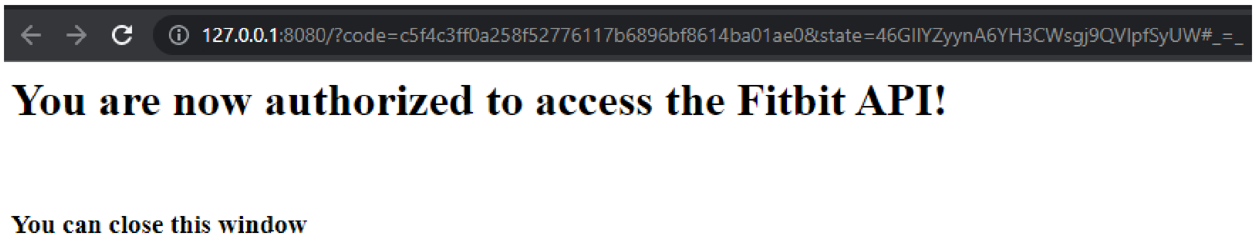
Fitbit authorizes users and authenticates API calls using OAuth 2.0. The OAuth 2.0 framework requires your application to obtain your own Access Token when the Fitbit user authorizes your app to access their data. The Access Token is used for making HTTPS requests to the Fitbit API.

The authorization process is enabled by the Python code below.

```
# YOU NEED TO PUT IN YOUR OWN CLIENT_ID AND CLIENT_SECRET
CLIENT_ID=''
CLIENT_SECRET=''

server=Oauth2.OAuth2Server(CLIENT_ID, CLIENT_SECRET)
server.browser_authorize()
ACCESS_TOKEN=str(server.fitbit.client.session.token['access_token'])
REFRESH_TOKEN=str(server.fitbit.client.session.token['refresh_token'])
auth2_client=fitbit.Fitbit(CLIENT_ID,CLIENT_SECRET,oauth2=True,access_token=ACCESS_TOKEN,refresh_token=REFRESH_TOKEN)
```

After authorization and login, a new page should appear as shown.


127.0.0.1:8080/?code=c5f4c3ff0a258f52776117b6896bf8614ba01ae0&state=46GIlYZyynA6YH3CWsgj9QVIpfSyUW#_=_

You are now authorized to access the Fitbit API!

You can close this window

• • •

API requests for data acquisition

Fitbit Web API provides a wide range of endpoints, including exercise details, activities summary (calories, steps, distance covered etc.), heart rate, sleep measures and many other interesting health categories.

For this tutorial, we will acquire data for **Sleep & User Engagement**. The code we used to make requests is the following:

```
def df_fitbit(activity, base_date, end_date, token):
    url = 'https://api.fitbit.com/1.2/user/-/'+activity+'/date/'+base_date+'/'+end_date+'.json'
    response = requests.get(url=url, headers={'Authorization':'Bearer ' + token}).json()

    return response
```

where ***base_date*** and ***end_date*** specify the data range in which we retrieve the data from Fitbit API and ***activity*** is the activity types, which include in our case:

activities-minutesSedentary	*activities-heart*
activities-minutesLightlyActive	*activities-steps*
activities-minutesFairlyActive	*sleep*
activities-minutesVeryActive	

• • •

Data storage in MongoDB

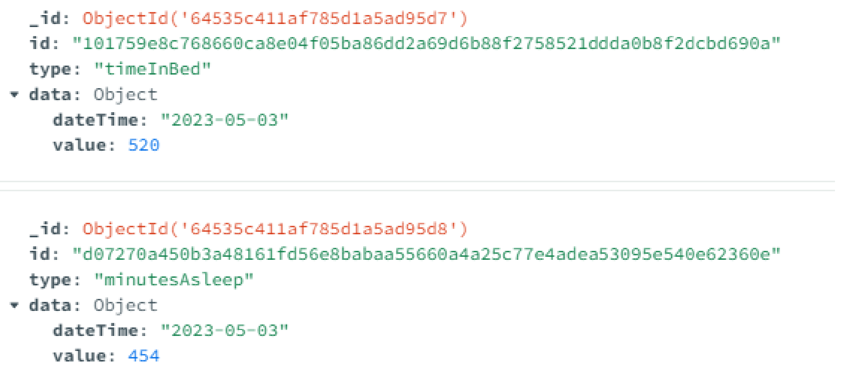
We built 3 main functions to get the information we need from the API response. In all these functions, for each metric we collect data for, we create a new dictionary with a random hash ID, the metric type, and the metric value for that day. Then, we insert each new dictionary into a database. We use a NoSQL database, the famous MongoDB. This is a document-based database, which uses JSON-like (BSON) documents to store data. You can download MongoDB from here and then, we recommend you to use MongoDB Compass or Studio 3T for visualizing the contents.

The following code is used to transform the data in the above-mentioned form.

```python
def create_data(dictItem,typeItem):
    data_list = []
    # create a random hash ID
    hash_object = hashlib.sha256(str(random.getrandbits(256)).encode())
    id = hash_object.hexdigest()
    # create a new dictionary with the id, type, and data fields
    new_dict = {
        'id': id,
        'type': typeItem,
        'data': dictItem
      }
        # append the new dictionary to the output list
    collection.insert_one(new_dict)
    return data_list
```

Figure 1: create_data function

An example of the data we acquire in MongoDB looks like this:



```
  _id: ObjectId('64535c411af785d1a5ad95d7')
  id: "101759e8c768660ca8e04f05ba86dd2a69d6b88f2758521ddda0b8f2dcbd690a"
  type: "timeInBed"
▾ data: Object
    dateTime: "2023-05-03"
    value: 520

  _id: ObjectId('64535c411af785d1a5ad95d8')
  id: "d07270a450b3a48161fd56e8babaa55660a4a25c77e4adea53095e540e62360e"
  type: "minutesAsleep"
▾ data: Object
    dateTime: "2023-05-03"
    value: 454
```

We have multiple types of records and each document contains **only one Fitbit record.**

To be able to store our data in MongoDB, we need to create a local database and a collection where our data will be stored. Then, we can connect to our MongoDB database collection using the following code.

```python
from pymongo import MongoClient

try:
    connect = MongoClient()
    print("Connected successfully!!!")
except:
    print("Could not connect to MongoDB")

# connecting or switching to the database
db = connect.fitbitDB

# creating or switching to fitbitCollection
collection = db.fitbitCollection
```

```
Connected successfully!!!
```

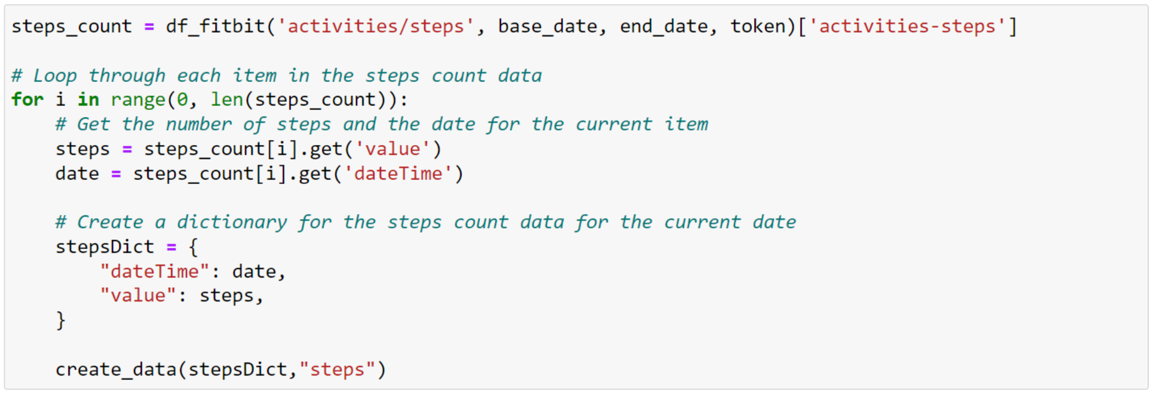
The first function used to get data from Fitbit API is the ***getSleepData( ).*** It retrieves the sleep data for each day, within the date range, and for each main sleep session on that day, it extracts various sleep metrics such as sleep duration, start time, end time, minutes asleep, minutes awake, minutes spent in deep, light, REM, and wake sleep stages etc.

The other two are ***getActivityData( )*** and ***getStepsData( )*** functions and we use them to get insights on the engament of the user.

The ***getActivityData()*** function retrieves activity data from the Fitbit API for a specified date range and activity types. The activity types include sedentary minutes, lightly active minutes, fairly active minutes, very active minutes, and heart rate. The function then loops through the date range and creates new data entries for each type of activity. It also creates a new data entry for the total wear time for the day.

Finally, the ***getStepsData()*** function retrieves the step count data from the Fitbit API for a specified date range. It then loops through each item and gets the steps count data for the current date.

The following figure shows an example of the code used to retrieve, transform, and store step data into MongoDB. The procedure is similar for the rest of the data. The full code can be found in our GitHub repository.



```python
steps_count = df_fitbit('activities/steps', base_date, end_date, token)['activities-steps']

# Loop through each item in the steps count data
for i in range(0, len(steps_count)):
    # Get the number of steps and the date for the current item
    steps = steps_count[i].get('value')
    date = steps_count[i].get('dateTime')

    # Create a dictionary for the steps count data for the current date
    stepsDict = {
        "dateTime": date,
        "value": steps,
    }

    create_data(stepsDict,"steps")
```

Setting up a Streamlit website with Python

After creating the database, it is interesting to create some visualizations to get a better understanding of the data. For a faster way to build and share our data app, we used Streamlit, an open-source Python library that makes it easy to build beautiful custom web apps for machine learning and data science.

It is quite simple to install Streamlit using pip in your dedicated Python environment.

```
pip install streamlit
```

Then, in a Python file, let's say app_st.py, import the library.

```
import streamlit as st
```

There are various functions of the library to define the layout of the application in Streamlit. For example, we can set the screen size, add a main title and a main text.

```
st.set_page_config(layout="wide", page_title="Fitbit Data")
st.title(':blue[Fitbit Data Visualization]')
st.subheader(
    'This is a Streamlit application that shows useful charts from Fitbit data stored in a MongoDB database')
```

After writing the code to create the visuals, it is easy to plot them in the Streamlit dashboard. In our case, we first created plots using matplotlib library and then displayed those plots in Streamlit using the following line of code.

```
st.pyplot(fig)
```

This is not the only way to create plots and display them in Streamlit. Streamlit library has some of its own functions that create graphs (see documentation: https://docs.streamlit.io/library/api-reference/charts/st.bar_chart)

When the code is ready, we can run the web application locally with Streamlit. We can do it by using the following command in the terminal.

```
streamlit run app_st.py
```

This code will open a new tab in your browser with the web application.

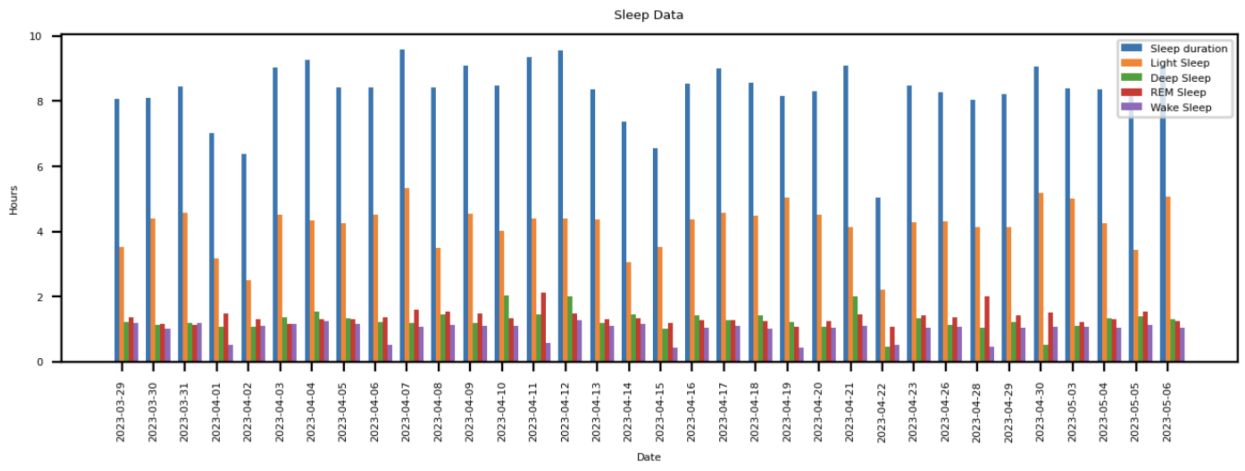
• • •

Visualizations with Streamlit for understanding Fitbit data

Now that we have obtained the data we want, we can make some interesting visualizations for better understanding of the data. This will also help us to reach into useful conclusions.

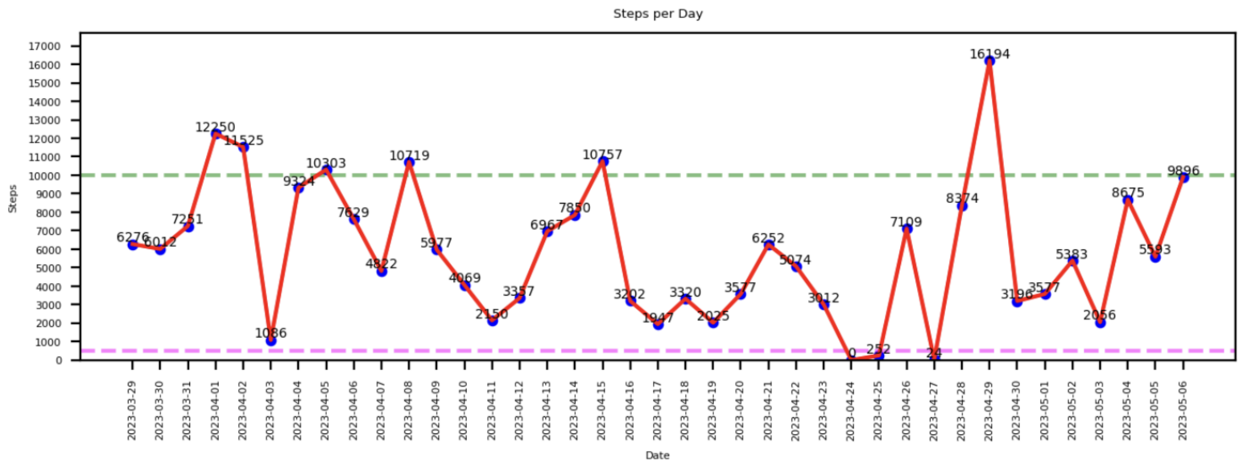
The following figures display some of the visualizations we created with our Fitbit data. Reader is advised to visit our GitHub repo for more visuals.

Sleep time



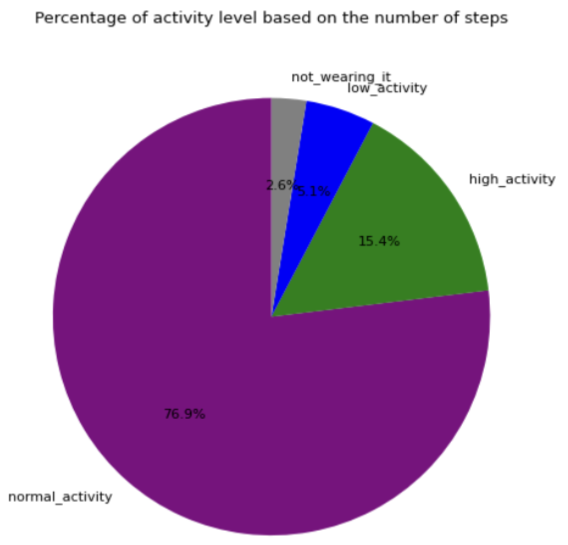
This bar chart provides a detailed breakdown of the different stages of sleep, including light sleep, deep sleep, REM sleep, and wake time, as well as the total sleep duration for a date range. This chart can be useful in identifying any irregularities or patterns in a person's sleep. By monitoring sleep patterns regularly, a person can identify any trends or improvements in their sleep and adjust their habits accordingly.

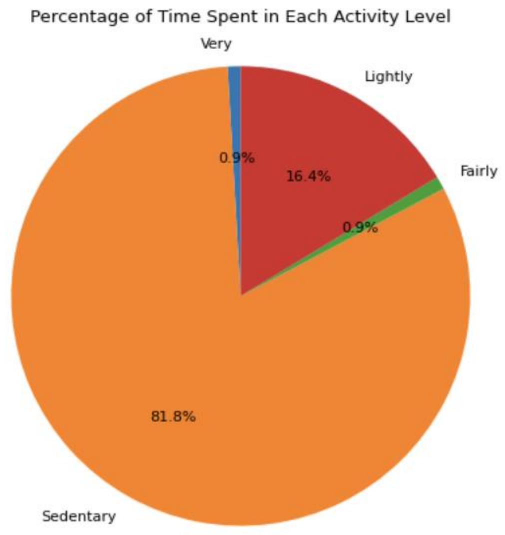
Steps per Day



This chart shows the number of steps the user took on each date. The green line shows the baseline to 10000 steps, where the user’s high activity starts. The pink line shows where the low activity ends (500 steps). We can assume that the user was wearing the smartwatch at a specific date, when the number of steps was > 500 for that date.

Steps and activity levels pipes





These two pie charts can give great insights related to the engagement of the user with the smartwatch.

The left pie illustrates the percentage of the days the user had a specific activity level. The ***low_activity*** *level* is the days that user did less than 500 steps, ***normal_activity*** the days the user did more than 500 steps but also less than 10000 and ***high_activity*** the days that user did more than 10000 steps. ***not_wearing_it*** is the days with 0 steps, which means that the user wasn't wearing the smartwatch at all.

The right pie chart provides a quick summary of how the user spends their time, by examining user's different activity levels. By breaking down the total time into four distinct categories - sedentary, lightly, fairly, and very active - it offers valuable insights into their daily physical activity patterns. Based on the pie chart, it appears that the user spends a significant amount of time in sedentary and lightly active activities. This could be an insightful and motivational observation for the user, to aim for a higher level of physical activity by increasing their time spent in fairly active level activities.

Insightful conclusions

We've discussed how to get beyond some of the common challenges encountered in:

- Requesting data from Fitbit API using Python
- Formatting the data into a JSON form
- Connecting to a MongoDB database and storing the data
- Plotting the data using Streamlit to get useful insights

This barely scratches the surface of what you can do with the data available in your Fitbit data archive. Any further exploration of the data could include:

- Analyzing timeseries data to identify historical trends and predict future trends
- Utilizing the Fitbit Subscription endpoints to create a “listener” waiting for live user updates.

I hope you find this tutorial useful. Please let us know if you have any thoughts or concerns.

Thanks for reading!