

Wearables Book

Merck and Data Mine Corporate Partnership Team

2020-07-28

Contents

1	Book Overview	5
2	Data Science Overview	7
3	Project Introduction	9
4	Using Git and GitHub	11
4.1	Git and GitHub Overview	11
4.2	Important Git Commands	11
4.3	Git and GitHub Simple Walkthrough	12
5	Fitbit Technology	17
6	Dependencies	21
7	Working in Python	23
7.1	Python Introduction	23
7.2	Reading Data, Conditionals, Loops	24
7.3	Pandas and Numpy	25
7.4	Vizualizations with Matplotlib	26
8	Tutorial - Data Capture in Python	29
8.1	Getting Started	29
8.2	Imports	29
8.3	Authentication	30

8.4	Accessing Data	31
8.5	Processing Data	32
8.6	Storing Data	34
9	Working in Bash	37
9.1	Introduction	37
9.2	Working with Data using Bash	38
9.3	Bash Scripts	42
10	Tutorial - Data Processing with Bash	45
11	Working in SQL	47
11.1	SQL Fundamentals	47
12	Tutorial - Data Storage with SQL	53
12.1	Create a new database	53
12.2	Create a new table	53
12.3	Import Data to table from CSV	54
13	Conlusion and More Resources	55
13.1	Front End Development Resources	55
13.2	Back End Development Resources	56

Chapter 1

Book Overview

A foundation of knowledge in the data sciences can prove advantageous for almost all scientists. Often, however, understanding the fundamentals of any new skill lacking proper instruction or recourses can be difficult. With ample guidance, all can develop a base knowledge of Data Science that can grow and provide new skills for improved research experiences.

The sets of instruction, examples, and tutorials in this walk-through are specifically designed for Merck Technology and can be distributed to Merck Scientists. The walk-through will be tailored to the needs of the pharmaceutical industry and will relate to work being done at Merck.

In this walkthrough, a complete, end-to-end, project will be outlined with instructional and applicational opportunities throughout.

Chapter 2

Data Science Overview

Data Science is an interdisciplinary field aimed to extract insight from structured and unstructured data. An intersection between computing, statistics, mathematics, and domain application, data science reveals new meaning to the extreme amounts of data being collected and stored every day.

Some of the foundational skills that the data sciences require include basic computing and data analysis knowledge. A range of technologies are used daily by data scientists, including Python, R, Hadoop, Bash, and more. Statistical concepts like descriptive statistics, probability, Bayesian theory, and modeling are used in combination with the previously mentioned technologies to gain insight from data.

Chapter 3

Project Introduction

An end-to-end project will be outlined in the remainder of the text. Primary concepts included are data capture, data visualization, and client-server communication. The technologies covered will include Python, Bash, SQL, R, and React Native along with many useful packages and dependencies.

Throughout each section of the text, there will be an introduction to a new technology. Basic programming skills and logic explanation related to the technology will be covered within this section. Using the skills taught, the walk-through section will follow, where instruction related to the overall project will be given.

The project being taught in this text is a data capture project that will evolve into a basic mobile application. Using Fitbit technology, users are collecting massive amounts of biometric data that, if captured properly, can be used to benefit pharmaceutical research. Patient activity, heart, sleep, and weight data can all be captured seamlessly. The medicinal benefits are great with a tool like this. It is through simple computing and foundational data science topics that a pipeline and tool can be engineered.

Chapter 4

Using Git and GitHub

4.1 Git and GitHub Overview

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

GitHub serves as a web-based interface for Git while also providing many other valuable features for version control and collaboration.

Downloading Git can be done at <https://git-scm.com/downloads>. GitHub is free to create to create an account at <https://github.com/>.

During this section, we will work through the basics of Git and GitHub!

4.2 Important Git Commands

```
git clone                # clone a repository into a new directory

git init                 # initialize local directory as a git repository

git add .                # add files to local repository

git commit- m "commit message" # commit files you staged in local repository
```

```
git remote add origin URL          # sets remote location of Github repository with URL
git remote -v                      # verifies remote location
git push                           # pushes changes from local repository to remote
git branch                         # check which branch you are currently working on
git checkout branch-name           # change branches
git checkout -b branch-name        # create a new branch if branch-name does not exist
git status                         # displays state of working directory and changes
git pull                           # update local version of a repository from remote
```

4.3 Git and GitHub Simple Walkthrough

Let's learn some basics of Git and GitHub through an example.

To start, make a new directory and cd into it.

```
mkdir myGitTutorial
cd myGitTutorial
```

Now, create a readme file and add a description within our new directory.

```
touch readme.md
echo this is the read me for my Git tutorial > readme.md
```

Initialize the Git repository.

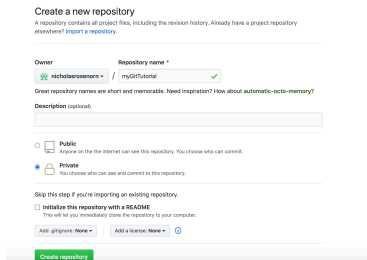
```
git init
```

Add and commit files to local git repository.

```
git add .

git commit -m "initial commit."
```

Now, if needed, create a GitHub account and then create a new repository.



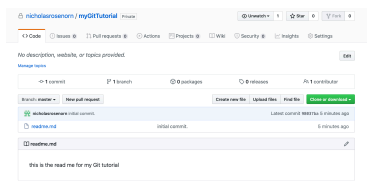
Back in command line, set remote location of repository

```
git remote add origin link/of/your/GitHub/repo
```

Now, we can push contents to the remote repository on GitHub.

```
git push -u origin master
```

The repository is now set up and can be viewed in Github! You will see the readme file we created in the repo.



Next, we can create a new branch.

```
git checkout -b tutorial-branch
```

Make a new file in the tutorial branch

```
touch branch.txt
echo this is for the tutorial branch > branch.txt
```

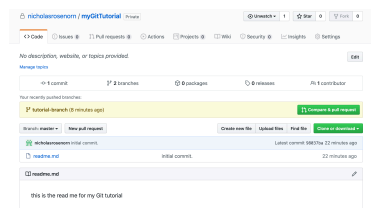
Stage and commit files to branch.

```
git add .
git commit -m "text file to branch"
```

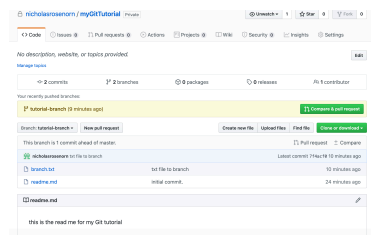
Push staged files to branch.

```
git push --set-upstream origin tutorial-branch
```

When we return to github we will notice that there are two branches!

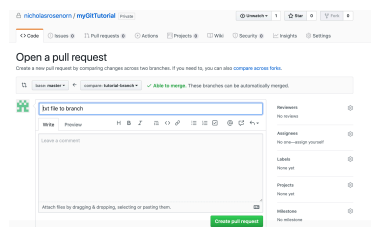


Notice that the master branch does not have the newly created branch.txt file. If we navigate to tutorial-branch, we will see the branch.txt file!



Let's update the master branch with the added files from the branch. To update the master branch, we will need to create a pull request.

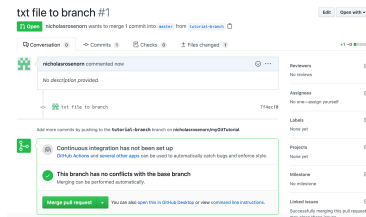
Click the green button that says 'compare and pull request'.



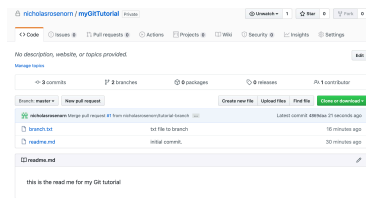
Then, click 'create pull request'. Notice how there are no conflicts with the master branch!

4.3. GIT AND GITHUB SIMPLE WALKTHROUGH

15



Now click ‘Merge pull request’ and then ‘confirm merge’.



Once the merge is done processing we can return to the master branch and we will see that branch.txt is now available on the master!

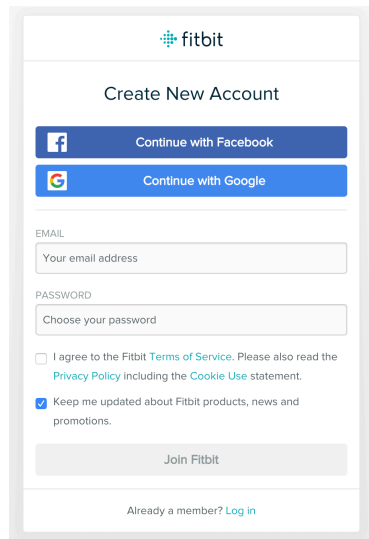
Chapter 5

Fitbit Technology

To begin the project, we will begin by discussing application programming interfaces (API). APIs are tools that allow developers to begin work easily. Often times, an API is a set of functions or methods that allow for replication of previously developed services. When released publicly, APIs allow for developers to gain access easily to features and data of large services.

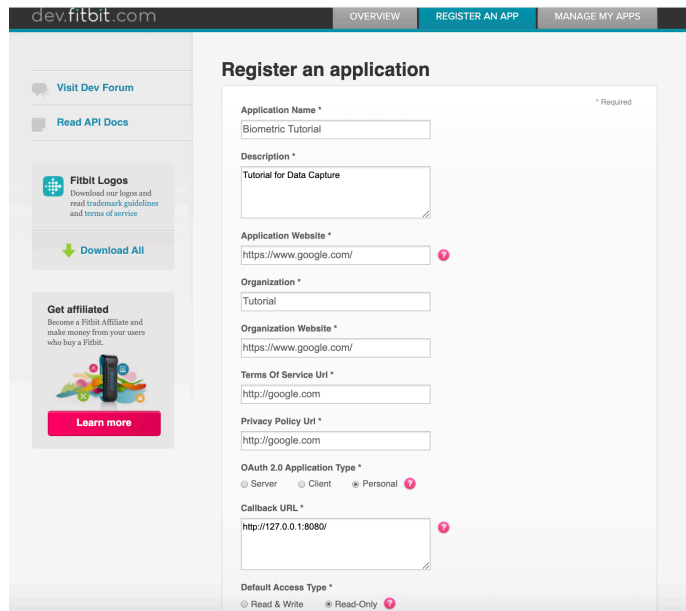
The API that this walk-through works with is the Fitbit Web API. The API can be found at <https://dev.fitbit.com/build/reference/web-api/basics/> and the accompanying documentation can be found at <https://python-fitbit.readthedocs.io/en/latest/>.

The first step to using the Fitbit API is to create a Fitbit account. Creating a new account can be found at <https://accounts.fitbit.com/signup>. You will be presented with the screen show below. Simply, enter an email and password of choice to create an account. You do not have to associate a Fitbit device to complete this tutorial. While no personal data will be collected due to lack of a device, the walk-through will still cover how properly use the API to access data. When data is necessary for later in the tutorial, dummy data will be provided.



The image shows the 'Create New Account' page on the Fitbit website. At the top is the Fitbit logo. Below it, the title 'Create New Account' is centered. There are two large blue buttons: 'Continue with Facebook' and 'Continue with Google'. Below these are input fields for 'EMAIL' (with placeholder 'Your email address') and 'PASSWORD' (with placeholder 'Choose your password'). There are two checkboxes: one for agreeing to the Fitbit Terms of Service and Privacy Policy, and another for staying updated about Fitbit products. At the bottom is a 'Join Fitbit' button and a link for 'Already a member? Log in'.

After registering your Fitbit account, proceed to <https://dev.fitbit.com/login> and login with the account credentials you just created. Click on 'manage' and 'register an app' in the navigation bar. You will then be presented with the screen show in figure 2. Fill out the necessary registration fields and click 'register' at the bottom of the page.



The image shows the 'Register an application' page on the Fitbit Developer website. The page has a dark header with 'dev.fitbit.com' and navigation links: 'OVERVIEW', 'REGISTER AN APP' (highlighted), and 'MANAGE MY APPS'. On the left is a sidebar with links like 'Visit Dev Forum', 'Read API Docs', 'Fitbit Logos', 'Download All', and 'Get affiliated'. The main content area is titled 'Register an application' and contains a form with the following fields: 'Application Name' (filled with 'Biometric Tutorial'), 'Description' (filled with 'Tutorial for Data Capture'), 'Application Website' (filled with 'https://www.google.com/'), 'Organization' (filled with 'Tutorial'), 'Organization Website' (filled with 'https://www.google.com/'), 'Terms Of Service Url' (filled with 'http://google.com'), 'Privacy Policy Url' (filled with 'http://google.com'), 'OAuth 2.0 Application Type' (radio buttons for 'Server', 'Client', and 'Personal' with 'Personal' selected), 'Callback URL' (filled with 'http://127.0.0.1:8080/'), and 'Default Access Type' (radio buttons for 'Read & Write' and 'Read-Only' with 'Read-Only' selected). Red question mark icons are present next to several fields.

Now that an application is registered to your account, you can navigate to your apps. When clicking on the recently created app you will see a similar display as in Figure 3.

Applications I registered

[Register a new app](#)

Application Biometric Tutorial

Tutorial for Data Capture

[Edit Application Settings](#)[Delete Application](#)

[Reset Client Secret](#)[Revoke Client Access Tokens](#)

OAuth 2.0 Client ID
22BLXS

Client Secret
a259368e5736a4171753dba8133f06d4

Callback URL
http://127.0.0.1:8080/

OAuth 2.0: Authorization URI
https://www.fitbit.com/oauth2/authorize

OAuth 2.0: Access/Refresh Token Request URI
https://api.fitbit.com/oauth2/token

[OAuth 2.0 tutorial page](#)

Subscribers endpoint state

The creation and registration of this app with fitbit will be necessary for the entirety of the tutorial. We will return to the information shown soon.

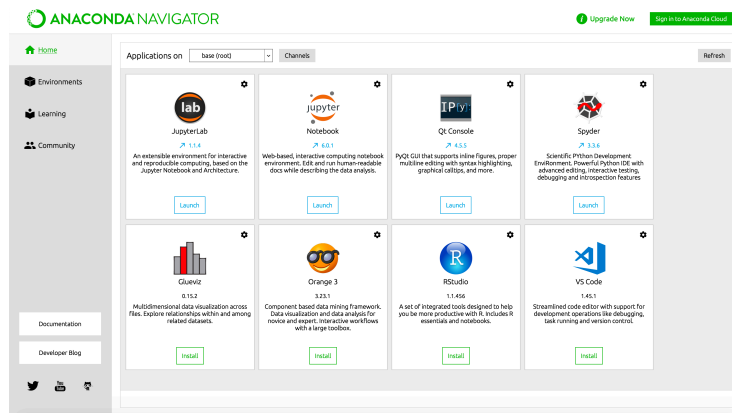
Chapter 6

Dependencies

For this project, it will be easiest for work in an Anaconda environment. Anaconda is an open-source Data Science toolkit. It allows users to easily work with thousands of open-source packages.

Install Anaconda here: <https://docs.anaconda.com/anaconda/install/>. Follow the steps provided to install for your machine.

After you have installed Anaconda, open the Anaconda Navigator. It will look similar to the image below.



On the left side navigation bar, click on environments and then search for 'cherry'. Check the box next to cherry and click the green apply button.

After, you be able to use the different applications offered through anaconda. Jupyter notebook is great for writing programs in Python and R.

Chapter 7

Working in Python

In this section, we will cover important computing fundamentals using the python programming language. These fundamentals are transferrable between many programming languages so laying a strong foundation will prove beneficial.

If you are new to python, it is in your best interest to review some of the material at <https://www.w3schools.com/python/>

7.1 Python Introduction

Python is an interpreted programming language that is known for its simplicity, readability, and small learning curve. Python is one of the most popular programming languages used today and learning to use Python will provide fundamental computing skills.

For this tutorial, it is easiest to use a Jupyter Notebook. Create a new notebook and select Python as the language. Before starting the tutorial, add the following imports at the top of your notebook:

```
# These are helpful packages that will aid our work through the tutorial!
import sys
import csv
import requests
from collections import defaultdict
```

7.2 Reading Data, Conditionals, Loops

The data for this tutorial can be found at <https://raw.githubusercontent.com/fivethirtyeight/data/master/us-weather-history/KNYC.csv>. The data is in CSV format meaning that each new value is separated by a comma. Take a look at the raw data by visiting the link. It is data on weather patterns in NYC during parts of 2014 and 2015.

Now that we know the format of the data, let's read it into our Notebook.

```
# open the file directly from the link
file = requests.get("https://raw.githubusercontent.com/fivethirtyeight/data/master/us-weather-history/KNYC.csv").text
```

Right now, the file is a string. To work with our data, we need to clean it and get it to a format for better analysis. To start, we will need to remove any unnecessary whitespace and create a list where each element is a new line.

```
# strip() - method that removes whitespace
# split("\n") - method that returns a list of values broken by input characters from text
file = file.strip().split('\n')
```

Our file is now a list, where each element is one row (run `print(file)` if you would like to check). Our objective is to make each element of the list, a new list! We can easily do this using a for loop. A for loop will iterate over a sequence (i.e. lists, dictionaries, tuples, sets) and perform iterative commands in the loop. Let's try to clean our data some more.

```
# for each element of the list, split the element by comma to make sublists
for row in range(len(file)):
    file[row] = file[row].split(",")
```

Great, our data is cleaned! Let's view our file with a loop:

```
# Print the first five file lines of the data
for row in file[0:5]:
    print(row)
```

```
## ['date', 'actual_mean_temp', 'actual_min_temp', 'actual_max_temp', 'average_min_temp', 'average_max_temp']
## ['2014-7-1', '81', '72', '89', '68', '83', '52', '100', '1943', '1901', '0.00', '0.00']
## ['2014-7-2', '82', '72', '91', '68', '83', '56', '100', '2001', '1966', '0.96', '0.00']
## ['2014-7-3', '78', '69', '87', '68', '83', '54', '103', '1933', '1966', '1.78', '0.00']
## ['2014-7-4', '70', '65', '74', '68', '84', '55', '102', '1986', '1949', '0.14', '0.00']
```


Notice that the first line is the header line and the rest of the lines are data for the corresponding headers. Let's take a look at the maximum temperature on the first five days listed in the data set.

```
for row in file[0:6]:
    print("%-20s %s" % (row[0],row[3])) # simple string formatting techniques!
```

```
## date                actual_max_temp
## 2014-7-1            89
## 2014-7-2            91
## 2014-7-3            87
## 2014-7-4            74
## 2014-7-5            81
```

7.3 Pandas and Numpy

While our data is already organized well, more complex analysis with data in this format can get tedious. A popular data manipulation and analysis software is called Pandas. Let's explore the functionality of Pandas!

To start, let's create the list of headers:

```
headers = file[0]
headers
```

```
## ['date', 'actual_mean_temp', 'actual_min_temp', 'actual_max_temp', 'average_min_temp', 'average_max_temp']
```

Next, we will reformat our original list of data so that the headers are removed:

```
import datetime
data = file[1:] # remove header

for row in data: # convert data from strings to floats
    for i in range(0,13):
        if i == 0:
            row[i] = datetime.datetime.strptime(row[i], '%Y-%m-%d')
        else:
            row[i] = float(row[i])
```

Lastly, we will place our data into a pandas dataframe:

```
import pandas as pd
df = pd.DataFrame(data, columns = headers)
df.head()      # .head() is a way to view the first 5 rows of the dataframe
```

	date	actual_mean_temp	...	average_precipitation	record_precipitation
## 0	2014-07-01	81.0	...	0.12	2.17
## 1	2014-07-02	82.0	...	0.13	1.79
## 2	2014-07-03	78.0	...	0.12	2.80
## 3	2014-07-04	70.0	...	0.13	1.76
## 4	2014-07-05	72.0	...	0.12	3.07

```
##
## [5 rows x 13 columns]
```

Great! Now we can perform analysis very easily on our data! For example, we can find some very valuable statistics with one simple command!

```
df.describe()
```

	actual_mean_temp	...	record_precipitation
## count	365.000000	...	365.000000
## mean	54.736986	...	2.386137
## std	18.679979	...	1.045702
## min	11.000000	...	0.860000
## 25%	39.000000	...	1.690000
## 50%	58.000000	...	2.160000
## 75%	72.000000	...	2.750000
## max	85.000000	...	8.280000

```
##
## [8 rows x 12 columns]
```

Pandas is a very powerful tool! More about pandas can be learned at <https://pandas.pydata.org/pandas-docs/version/0.25.3/>.

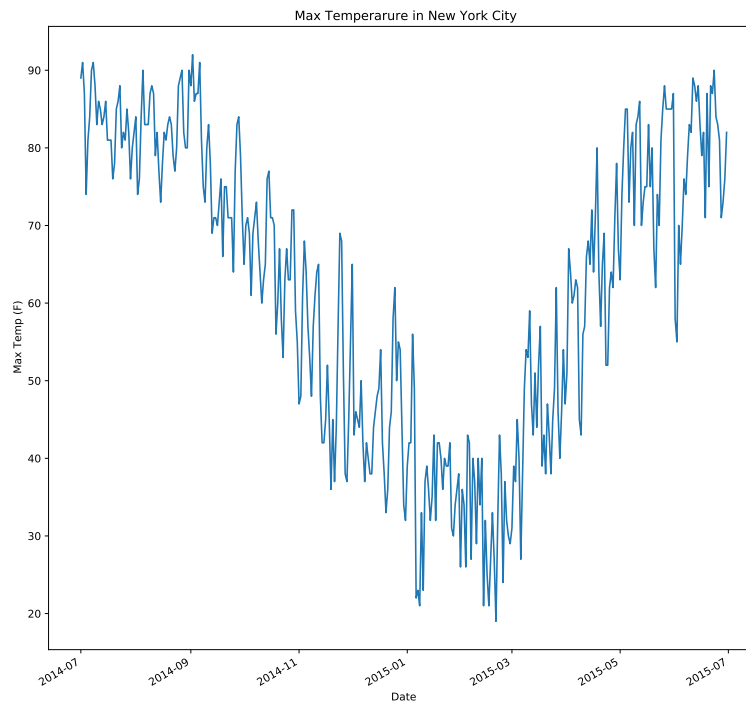
7.4 Vizualizations with Matplotlib

Lastly, we can make a simple vizualization of our data using another package called Matplotlib. It is a plotting library for the python programming language.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 12))
y = df['actual_max_temp']
x = df['date']
ax.plot(x,y)
ax.set(xlabel="Date", ylabel="Max Temp (F)", title="Max Temperatur in New York City")
```

```
## [Text(0.5, 0, 'Date'), Text(0, 0.5, 'Max Temp (F)'), Text(0.5, 1.0, 'Max Temperatur in New York City')]
```

```
plt.gcf().autofmt_xdate()  
plt.show()
```



Matplot has much more functionality than this simple example. Refer to the documentation at <https://matplotlib.org/3.2.1/contents.html> for more information!

Chapter 8

Tutorial - Data Capture in Python

8.1 Getting Started

Before starting this section of the tutorial, we need to download the Fitbit API.

Navigate to <https://github.com/orcasgit/python-fitbit> and click the green 'clone or download' button. Then, click download zip, and open the zip file in the same folder that you are using within Anaconda.

Next, download python on to your computer at <https://www.python.org/downloads/>. Click on the yellow 'download' button, open the downloaded file, and follow the steps to install.

Lastly, open a new terminal, and run the following commands:

```
pip install cherrypy (you can also use the Anaconda navigator to install cherrypy)
pip install requests-oauthlib
```

8.2 Imports

This project requires multiple packages. If the steps in the previous sections have been followed, these import statements should run seamlessly.

```
#Import the necessary packages
import fitbit
import gather_keys_oauth2 as OAuth2
```

```
import pandas as pd
import datetime
import json
```

8.3 Authentication

Remembering back to the Fitbit Technology section, we registered an app on the Fitbit Development site that will allow us to access our account data. When we completed the registration of the app, we were given a client id and client secret. These codes are unique to each individual app that is registered and are necessary for the authentication process.

Use the following code gain access to your account:

```
CLIENT_ID = '22BLXS' #ENTER CLIENT ID CODE HERE
CLIENT_SECRET = 'a259368e5736a4171753dba8133f06d4' #ENTER CLIENT SECRET CODE HERE

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)
```

When the code above is run for the first time, you will be directed to a new browser with the following screen:



Biometric Tutorial by [Tutorial](#) would like the ability to access the following data in your Fitbit account.

- ☒ Allow All
 - ☒ sleep
 - ☒ profile ⓘ
 - ☒ location and GPS
 - ☒ heart rate
 - ☒ activity and exercise
 - ☒ weight ⓘ
 - ☒ food and water logs ⓘ
 - ☒ Fitbit devices and settings
 - ☒ friends ⓘ

If you allow only some of this data, Biometric Tutorial may not function as intended. Learn more about these permissions [here](#).

Deny

Allow

The data you share with Biometric Tutorial will be governed by Tutorial's [Privacy Policy](#) and [Terms of Service](#). You can revoke this consent at any time in your Fitbit [account settings](#).

Select 'Allow All' and then 'Allow'. Your browser will then display the follow:

You are now authorized to access the Fitbit API!

You can close this window

When you have reached this screen, the authentication is complete and you can now access your account data. According the fitbit documentation, you can access your data, by default, for 8 hours until you will have to reauthenticate by running the authorization code above.

8.4 Accessing Data

Now that the authentication is complete, we can access our data.

To start, we need to declare a date to collect data from. Let's use today's date:

```
today = str(datetime.datetime.now().strftime("%Y-%m-%d")) #today's date
```

Now we can view our 'activities' data for today's date.

```
activities = auth2_client.activities(date = today)
```

Notice how the data is formatted in nested dictionary format, similar to json file format. Because of this format, we need to decompose the nested dictionaries to access our desired data.

For example, if we want to access the total distance traveled we can do the following:

```
activities = auth2_client.activities(date = today)['summary']
totalDistance = activities['distances'][0]['distance']
```

To access today's steps:

```
steps = activities['steps']
```

The process is almost identical for accessing other types of data (i.e. sleep, heart, and weight data).

8.5 Processing Data

Because we are now familiar with the structure of data, we can begin processing it!

To collect activities data:

```
#gets all Activities Data
def getActivities(myDate):
    activities = auth2_client.activities(date = myDate)['summary']

    totalDistance = activities['distances'][0]['distance']
    veryActiveDistance = activities['distances'][3]['distance']
    moderatleyActiveDistance = activities['distances'][4]['distance']
    lightlyActiveDistance = activities['distances'][5]['distance']
    veryActiveMinutes = activities['veryActiveMinutes']
    fairlyActiveMinutes = activities['fairlyActiveMinutes']
    lightlyActiveMinutes = activities['lightlyActiveMinutes']
    sedentaryMinutes = activities['sedentaryMinutes']
    floorsClimbed = activities['floors']
    daySteps = activities['steps']

    return totalDistance, veryActiveDistance, moderatleyActiveDistance, lightlyActiveD
```


To collect sleep data:

```
#gets all Sleep data
def getSleep(myDate):
    nightSleep = auth2_client.sleep(date = myDate)['sleep']

    sleepEfficiency = None
    minutesAsleep = None

    if len(nightSleep) != 0:
        sleepEfficiency = nightSleep[0]['efficiency']
        minutesAsleep = nightSleep[0]['minutesAsleep']

    return sleepEfficiency, minutesAsleep
```

To collect heart data:

```
#gets all Heart Data
def getHeart(myDate):
    heartRates = auth2_client.intraday_time_series('activities/heart', base_date=myDate, deta

    HRrange30to100 = None
    HRrange100to140 = None
    HRrange140to170 = None
    HRrange170to220 = None
    avgRestingHR = None

    if len(heartRates) == 3:
        HRrange30to100 = heartRates['heartRateZones'][0]['minutes']
        HRrange100to140 = heartRates['heartRateZones'][1]['minutes']
        HRrange140to170 = heartRates['heartRateZones'][2]['minutes']
        HRrange170to220 = heartRates['heartRateZones'][3]['minutes']
        avgRestingHR = heartRates['restingHeartRate']

    return HRrange30to100, HRrange100to140, HRrange140to170, HRrange170to220, avgRestingHR
```

To collect weight data:

```
#gets all Weight Data
def getWeight(myDate):
    grabWeight = auth2_client.get_bodyweight(base_date = myDate)['weight']
    weight = None
    BMI = None
    if len(grabWeight) > 0:
        weight = grabWeight[0]['weight']
```

```
BMI = grabWeight[0]['bmi']

return weight, BMI
```

Now that we have the functions to capture the data, we can process our data with a pandas data frame:

```
#creates empty data frame
biometricDF = pd.DataFrame(columns=["Date", "Steps", "Floors Climbed", "Total Miles", "Moderately Active Miles", "Very Active Miles", "Slightly Active Miles", "Lightly Active Minutes", "Fairly Active Minutes", "HR 30-100 Minutes", "HR 100-140 Minutes", "HR 140-170 Minutes", "HR 170-220 Minutes", "Average Resting HR"])
```

Next, we will need a function to collect all the data together and place it into a data frame:

```
#adds data to data frame
def getBiometricData(myDF, myDate):
    totalDistance, veryActiveDistance, moderatleyActiveDistance, lightlyActiveDistance = getDistance(myDate)

    sleepEfficiency, minutesAsleep = getSleep(myDate)

    HRrange30to100, HRrange100to140, HRrange140to170, HRrange170to220, avgRestingHR = getHeartRate(myDate)
    weight, BMI = getWeight(myDate)

    todaysData = {"Date" : myDate, "Steps" : daySteps, "Floors Climbed" : floorsClimbed, "Total Miles" : totalDistance, "Moderately Active Miles" : moderatleyActiveDistance, "Very Active Miles" : veryActiveDistance, "Slightly Active Miles" : lightlyActiveDistance, "Lightly Active Minutes" : minutesAsleep, "Fairly Active Minutes" : sleepEfficiency, "HR 30-100 Minutes" : HRrange30to100, "HR 100-140 Minutes" : HRrange100to140, "HR 140-170 Minutes" : HRrange140to170, "HR 170-220 Minutes" : HRrange170to220, "Average Resting HR" : avgRestingHR}

    biometricDF = myDF.append(todaysData, ignore_index=True)

    return biometricDF
```

And, finally, creating the database with todays data:

```
biometricDF = getBiometricData(biometricDF, today) #append to data frame
```

8.6 Storing Data

This process can be replicated each day so data must be stored appropriately. To start, we can export our data frame to a csv file and save the files to a directory like this (I created a new folder named bioDates to store all my Data):

```
biometricDF.to_csv('./bioDates/' + today + '.csv')
```

In the following chapters of this tutorial, we will learn how to concatenate the individual daily csv files into one master file as well as storing the data into a SQL database!

Chapter 9

Working in Bash

9.1 Introduction

Unix is simply a computer operating system.

Bash, on the other hand, is a shell. Shells are command line interfaces where you can communicate with the computer via certain commands. Lets explore some commands. Feel free to test these on your machine in a terminal.

9.1.1 Basic Commands

9.1.1.1 pwd

The command ‘pwd’ will display the current working directory. Directories are file systems that contain references to other directories and files.

9.1.1.2 ls

In order to view the files contained within the directory, we can use the command ‘ls’. This will list the files and sub-directories in the current working directory

9.1.1.3 cd

‘cd’ stands for change directory and will change the working directory to a specified directory using a file path. By just entering cd, you will be directed to your home directory

9.1.1.4 mkdir

With the ‘mkdir’ command, a new directory will be created with the name we provide within the current working directory.

9.1.1.5 touch and nano

The ‘touch’ command will create a new file within the current working directory.

Here's an example

```
pwd
mkdir Example
cd Example/
pwd
touch exampleTextFile.txt
ls
```

```
## /home/runner/work/wearables-book/wearables-book
## mkdir: cannot create directory 'Example': File exists
## /home/runner/work/wearables-book/wearables-book/Example
## exampleTextFile.txt
```

For more basic bash commands, refer to <https://www.educative.io/blog/bash-shell-command-cheat-sheet>!

9.2 Working with Data using Bash

Bash is very powerful when working with data. Let's explore an example of some data analysis using bash commands.

First, we will retrieve some data using the ‘wget’ command. This command will download data from a URL and save it to the current directory.

For
Mac
users:

```
If
the
wget
com-
mand
re-
turns
the
er-
ror
'wget:
com-
mand
not
found',
then
run
ruby
-e
"$(curl
-
fsSL
https:
//
raw.
githubusercontent.
com/
Homebrew/
install/
master/
install)"
followed
by
brew
in-
stall
wget
```

This
pro-
cess
in-
stalls
Home-
brew,
which
is
a
pack-
age
man-
ager
for
miss-
ing
pack-
ages
on
macOS.

```
wget https://raw.githubusercontent.com/fivethirtyeight/data/master/us-weather-history/1
```

```
## --2020-07-28 16:59:10-- https://raw.githubusercontent.com/fivethirtyeight/data/mas
## Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.208.133
## Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.208.133
## HTTP request sent, awaiting response... 200 OK
## Length: 20604 (20K) [text/plain]
## Saving to: 'KNYC.csv.5'
##
##      OK .....                               100% 2.64M=0.007s
##
## 2020-07-28 16:59:10 (2.64 MB/s) - 'KNYC.csv.5' saved [20604/20604]
```

Now, we can view the first *n* rows of the data. In this example, we will view the first five lines

```
head -n5 KNYC.csv
```

```
## date,actual_mean_temp,actual_min_temp,actual_max_temp,average_min_temp,average_max_t
## 2014-7-1,81,72,89,68,83,52,100,1943,1901,0.00,0.12,2.17
## 2014-7-2,82,72,91,68,83,56,100,2001,1966,0.96,0.13,1.79
## 2014-7-3,78,69,87,68,83,54,103,1933,1966,1.78,0.12,2.80
## 2014-7-4,70,65,74,68,84,55,102,1986,1949,0.14,0.13,1.76
```


Another helpful way to view data is to save subsets of data to separate files.

Here we save the first 10 lines to a new file called short.csv.

```
head -n10 KNYC.csv >short.csv
```

Now we can view the whole file using the ‘cat’ command.

```
cat short.csv
```

```
## date,actual_mean_temp,actual_min_temp,actual_max_temp,average_min_temp,average_max_temp,record
## 2014-7-1,81,72,89,68,83,52,100,1943,1901,0.00,0.12,2.17
## 2014-7-2,82,72,91,68,83,56,100,2001,1966,0.96,0.13,1.79
## 2014-7-3,78,69,87,68,83,54,103,1933,1966,1.78,0.12,2.80
## 2014-7-4,70,65,74,68,84,55,102,1986,1949,0.14,0.13,1.76
## 2014-7-5,72,63,81,68,84,53,101,1979,1999,0.00,0.12,3.07
## 2014-7-6,75,66,84,68,84,54,103,1979,2010,0.00,0.13,1.97
## 2014-7-7,81,72,90,68,84,56,100,1914,2010,0.04,0.13,3.13
## 2014-7-8,81,71,91,69,84,56,100,1894,1993,0.39,0.14,1.80
## 2014-7-9,80,71,88,69,84,54,106,1963,1936,0.09,0.14,1.09
```

We can also view specific columns from our data using ‘cut’.

```
cut -d, -f1,4 short.csv
```

```
## date,actual_max_temp
## 2014-7-1,89
## 2014-7-2,91
## 2014-7-3,87
## 2014-7-4,74
## 2014-7-5,81
## 2014-7-6,84
## 2014-7-7,90
## 2014-7-8,91
## 2014-7-9,88
```

Next, let's look at a specific row by using the ‘grep’ command to search by date.

```
grep 2015-2-23 KNYC.csv
```

```
## 2015-2-23,23,8,38,30,43,5,70,1889,1985,0.00,0.12,1.38
```

9.3 Bash Scripts

Now that we have covered some of the basics of bash, let's explore a powerful tool within bash called bash scripting. Writing bash scripts is similar to any other program where each new line is a command with an intent to build upon previous computations.

We can create files that will perform the tasks within. Consider this:

If we want to write some bash commands that will write the time of day to a file we can do something like

```
touch myDateFile.txt
date >>myDateFile.txt
```

running these commands each time we would like to write the date to the .txt file would begin to be tedious after time. Bash scripts provide an easy solution!

First, we will create a new .sh file.

```
touch myDateBash.sh
```

Then, using a text editor, we can edit the file and enter our code to write the date to myDateFile.txt.

Paste the following command into a terminal (be sure your working directory is the same directory as where you saved your bash file):

```
vi myDateBash.sh
```

You will be presented with a screen that looks like the following image.



Then, type the letter i to enter “Insert Mode” and write in date >>myDateFile.txt. Then, click escape on your keyboard and then type “:wq”. You should now be back to a normal Terminal setting.

Next, we need to set the file permissions. Paste “chmod +x myDateBash.sh” into the terminal.

Great! Now the setup is complete and we can run the bash script to write the current date and time to. myDateFile.txt like this:

```
./myDateBash.sh
```

Lastly, to view the contents of the date text file:

```
cat myDateFile.txt
```

For more information on Bash and scripting, refer to <https://linuxconfig.org/bash-scripting-tutorial>.

Chapter 10

Tutorial - Data Processing with Bash

Using the basics of bash and some more advanced commands we can now use the python script that we developed in the previous tutorial!

Let's start by creating a new bash script. We will call it bioBash.sh.

```
touch bioBash.sh
```

The first action we need to take is to run our python script from command line.

Using a text editor, either vim or nano, write

```
python /path/to/your/python/script
```

in your bash script.

This will run the python script!

Next, we can need to change directories to the folder where we are storing the data. In the previous tutorial section, I named this directory bioDates. Once again, using a text editor, write

```
cd /path/to/your/biometric/data
```

in bioBash.sh.

Lastly, we will need to concatenate all the individual dates into one master .csv. This can be done with the following line of code:

```
awk 'FNR==1 && NR!=1{next;}{print}' *.csv > bioFinal.csv.
```

With this line, we concatenate all csv files into one master csv file called bioFinal.csv.

Chapter 11

Working in SQL

In this section, we will cover and practice the basic skills associated with SQL.

SQL stands for structured query language and is a language that allows us to access and manipulate relational databases. It is common practice to have structured data stored in databases, and SQL derives information of the data through queries.

11.1 SQL Fundamentals

11.1.1 SELECT statement

The SELECT statement is a common start to many queries. It is used when retrieving data from a table. The returned data will be in a table format.

The FROM statement denotes from which table to perform the query. In each of these examples, the table is called 'nyc' because the data is New York City weather data.

The * denotes to select all columns from the table.

```
SELECT * FROM nyc
```

We can also denote specific columns to select by using the column names.

```
SELECT date, actual_max_temp FROM nyc
```

Table 11.1: Displaying records 1 - 10

date	actual_mean_temp	actual_min_temp	actual_max_temp	average_min_temp	av
2014-7-1	81	72	89	68	
2014-7-2	82	72	91	68	
2014-7-3	78	69	87	68	
2014-7-4	70	65	74	68	
2014-7-5	72	63	81	68	
2014-7-6	75	66	84	68	
2014-7-7	81	72	90	68	
2014-7-8	81	71	91	69	
2014-7-9	80	71	88	69	
2014-7-10	78	72	83	69	

Table 11.2: Displaying records 1 - 10

date	actual_max_temp
2014-7-1	89
2014-7-2	91
2014-7-3	87
2014-7-4	74
2014-7-5	81
2014-7-6	84
2014-7-7	90
2014-7-8	91
2014-7-9	88
2014-7-10	83

Table 11.3: Displaying records 1 - 10

date	actual_max_temp
2014-7-1	89
2014-7-2	91
2014-7-3	87
2014-7-5	81
2014-7-6	84
2014-7-7	90
2014-7-8	91
2014-7-9	88
2014-7-10	83
2014-7-11	86

11.1.2 WHERE clause

To perform conditional queries, we can use the WHERE clause. The WHERE clause acts similar to an if statement by denoting which conditions must be upheld in order for the row to be selected in the query.

For example, we can find the dates on which the maximum temperature was greater than 75 degrees:

```
SELECT date, actual_max_temp FROM nyc WHERE actual_max_temp > 75
```

11.1.3 AND operator

Often times, more than one conditional statement is necessary when writing queries. The AND clause allows for more than one conditional statement that must be upheld for the row to be selected.

Let's find the dates on which the maximum temperature was greater than 75 degrees but also less than or equal to 90 degrees:

```
SELECT date, actual_max_temp FROM nyc WHERE actual_max_temp > 75 AND actual_max_temp <= 90
```

11.1.4 OR operator

Similar to the AND operator, the OR operator assists with conditional statements. The OR operator will allow for the selection of rows that satisfy at least one of the provided conditions

Table 11.4: Displaying records 1 - 10

date	actual_max_temp
2014-7-1	89
2014-7-3	87
2014-7-5	81
2014-7-6	84
2014-7-7	90
2014-7-9	88
2014-7-10	83
2014-7-11	86
2014-7-12	85
2014-7-13	83

Table 11.5: Displaying records 1 - 10

date	actual_max_temp
2014-7-7	90
2014-8-5	90
2014-8-27	90
2014-8-31	90
2014-9-8	75
2014-9-20	75
2014-9-21	75
2015-5-15	75
2015-5-16	75
2015-5-18	75

To select the days where maximum temperature equaled 75 or 90, we can do the following:

```
SELECT date, actual_max_temp FROM nyc WHERE actual_max_temp = 75 OR actual_max_temp = 90
```

11.1.5 ORDER BY keyword

SQL offers a great keyword for sorting queries. The ORDER BY keyword will sort the rows by the provided column in the query followed by another keyword—most commonly DESC or ASC. DESC will order the rows from greatest to least and ASC does the opposite.

For example, we can sort max temperature from least to greatest:

Table 11.6: Displaying records 1 - 10

date	actual_max_temp
2015-2-20	19
2015-1-8	21
2015-2-13	21
2015-2-16	21
2015-1-6	22
2015-1-7	23
2015-1-10	23
2015-2-24	24
2015-2-15	25
2015-1-31	26

```
SELECT date, actual_max_temp FROM nyc ORDER BY actual_max_temp ASC;
```


Chapter 12

Tutorial - Data Storage with SQL

With the data from our Fitbit collected and placed properly in a csv file format, we can easily create a new table in our mySQL database and then import the data from the csv file into the proper table!

12.1 Create a new database

Creating a new database in mySQL is simple! Let's create a Database called 'Biometrics'.

```
CREATE DATABASE Biometrics;
```

Now we can designate the 'Biometrics' database as the database to refer to in our following queries using the use statement.

```
USE Biometrics;
```

12.2 Create a new table

Within our new database, we'll need to create a new table to store the data. We can do this using the CREATE TABLE statement.

```

create table Username (
myIndex INT,
date_collected DATE,
steps INT,
floors_climbed INT,
total_miles FLOAT,
lightly_active_miles FLOAT,
moderately_active_miles FLOAT,
very_active_miles FLOAT,
sedentary_minutes FLOAT,
lightly_active_minutes FLOAT,
fairly_active_minutes FLOAT,
very_active_minutes FLOAT,
HR30_100Minutes INT,
HR100_140Minutes INT,
HR140_170Minutes INT,
HR30170_220Minutes INT,
average_resting_HR INT,
bmi FLOAT,
minutes_asleep FLOAT,
sleep_efficiency FLOAT,
weight FLOAT,
username VARCHAR(20),
happiness_rating INT,
pain_rating INT
);

```

The first term in each line will be a new column name in the table. The second term is the SQL data type. Read about mySQL data types at <https://dev.mysql.com/doc/refman/8.0/en/data-types.html>.

12.3 Import Data to table from CSV

Lastly, we can import our data from the csv file into our table.

```

LOAD DATA LOCAL INFILE "/Path/To/File" INTO TABLE Username
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 rows
(myIndex, @date_collected, steps, floors_climbed, total_miles, lightly_active_miles, m
very_active_miles, sedentary_minutes, lightly_active_minutes, fairly_active_minutes, v
HR30_100Minutes, HR100_140Minutes, HR140_170Minutes, HR170_220Minutes, average_resting
sleep_efficiency, weight, username, happiness_rating, pain_rating) SET date_collected =

```

Chapter 13

Conlusion and More Resources

The entirety of the data capture, cleaning, and storage side of the project is complete!

There are plenty more opportunities to build upon the data that has been captured and stored. Specifically, there is a great opportunity to use your data in a mobile app setting. To do this, there are a few resources that will help.

13.1 Front End Development Resources

13.1.1 Expo

Expo is an open-source platform for making universal native apps for Android, iOS, and the web with JavaScript and React. It can be found at <https://expo.io/>

13.1.2 React Native

React Native is a framework for building native apps using React. Refer to <https://reactnative.dev/> for more information

13.2 Back End Development Resources

13.2.1 Flask

Flask is a micro web framework written in Python. This will help you in building a REST API to communicate data from your database to your mobile app! The documentation is at <https://flask.palletsprojects.com/en/1.1.x/>.

13.2.2 Postman

Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration so you can create better APIs—faster. Find the software for download at <https://www.postman.com/>.