



## **INTRODUCTION TO EARTH SYSTEM DATA**

### **Project Report**

#### **Weather and environmental data via OpenWeatherMap**

*By Ralph Florent*

*January 26, 2019*

### **Introduction**

The course, *Introduction to the Earth System Data*, is all about Earth data and their handling [1]. Since the taxonomy of the Earth data can be a bit complex, scientists collaborate collectively to standardize the collected raw datasets. Thanks to this standardization, nowadays, we (scientists, students, related sub-disciplines, or, public interested in Earth data) can access a wide range of data collections through a paid service (sometimes free of charge), and use those data for our own purposes.

As for this project, the purpose is totally educational. We expect to use a web-based software as a service (SaaS), OpenWeatherMap (OWM) in this case, to collect the data programmatically and visualize them using our own web-based applications. Perhaps, in the future, we could try to find pattern in the collected data and infer some sustainable solutions to our daily challenges.

OpenWeatherMap is easy to use and provides a free subscription among other subscription plans. Actually, this is a good advantage for the public who want to assess OWM products and decide whether to upgrade to a fully-featured plan or simply move on.

Given the situation, we decide to go with OpenWeatherMap. Despite the limitations of their free subscription, it remains a good choice since we do not plan to overly expose our applications.

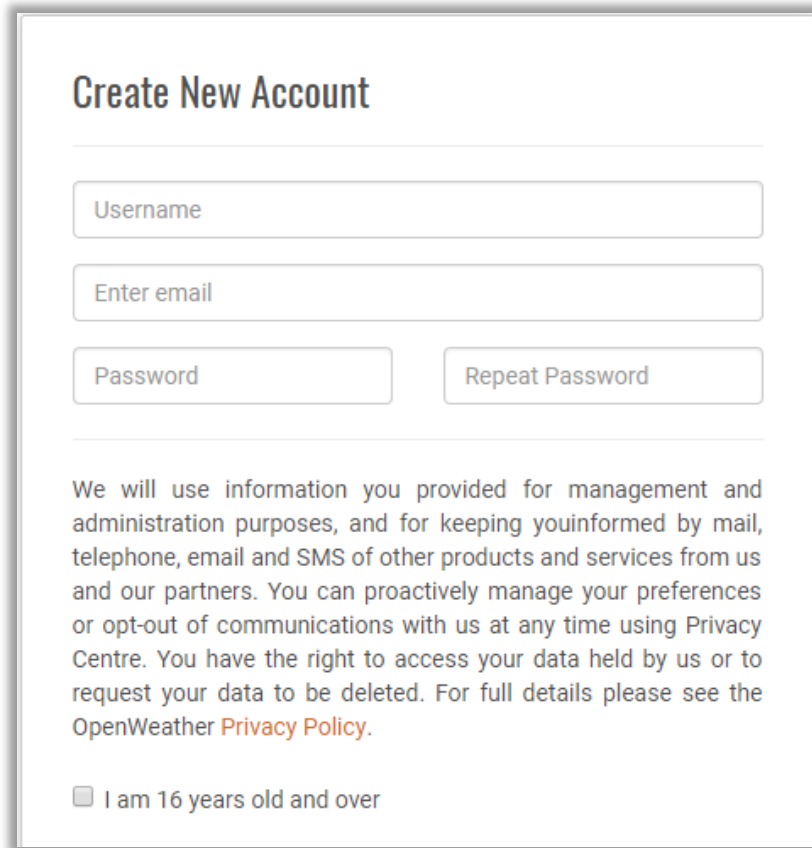
Exposing our application means that we would expect a massive use (a lot of API calls) that, in turn, would demand to upgrade our subscription plan. We don't want that.

Divided into four sections (*Account*, *Plan*, *API*, *Demo*), this document aims to guide the reader on how we proceed to successfully obtain the data collections from OpenWeatherMap and use them to provide useful information about the current weather and the daily forecasts in a city.

## Create an account

### Signup process

Creating an account on OpenWeatherMap is quite easy and straightforward. The basic steps require to locate the signup in the top-center of the page, and load the following page:



The image shows a web form titled "Create New Account". It contains four input fields: "Username", "Enter email", "Password", and "Repeat Password". Below the fields is a paragraph of text explaining the use of user information for management, administration, and communication. At the bottom, there is a checkbox labeled "I am 16 years old and over".

**Create New Account**

Username

Enter email

Password Repeat Password

We will use information you provided for management and administration purposes, and for keeping you informed by mail, telephone, email and SMS of other products and services from us and our partners. You can proactively manage your preferences or opt-out of communications with us at any time using Privacy Centre. You have the right to access your data held by us or to request your data to be deleted. For full details please see the OpenWeather [Privacy Policy](#).

☐ I am 16 years old and over

Figure 1 – Signup page interface of OpenWeatherMap

Then, fill out the form with the username, email, and password. And, as usual, the new user must agree with **Privacy Policy**, **Terms and conditions** of sale and **Websites terms and conditions** of use to complete the signup process.

If the typed-in information successfully passes the form validation, then the next stage of the account creation is the statistics that the website's owner runs for their own goal. Normally, the user is asked to give the purpose of registering a new account on OWM with the given options within a dropdown button. In our case, we specify that we stand for **Education/Science** at the **Jacobs University** organization.

## Confirmation via email

Once this step is completed, the user will immediately (a relatively short time) receive an email with the information about the API key, the endpoints, and some useful links for examples and test environment.

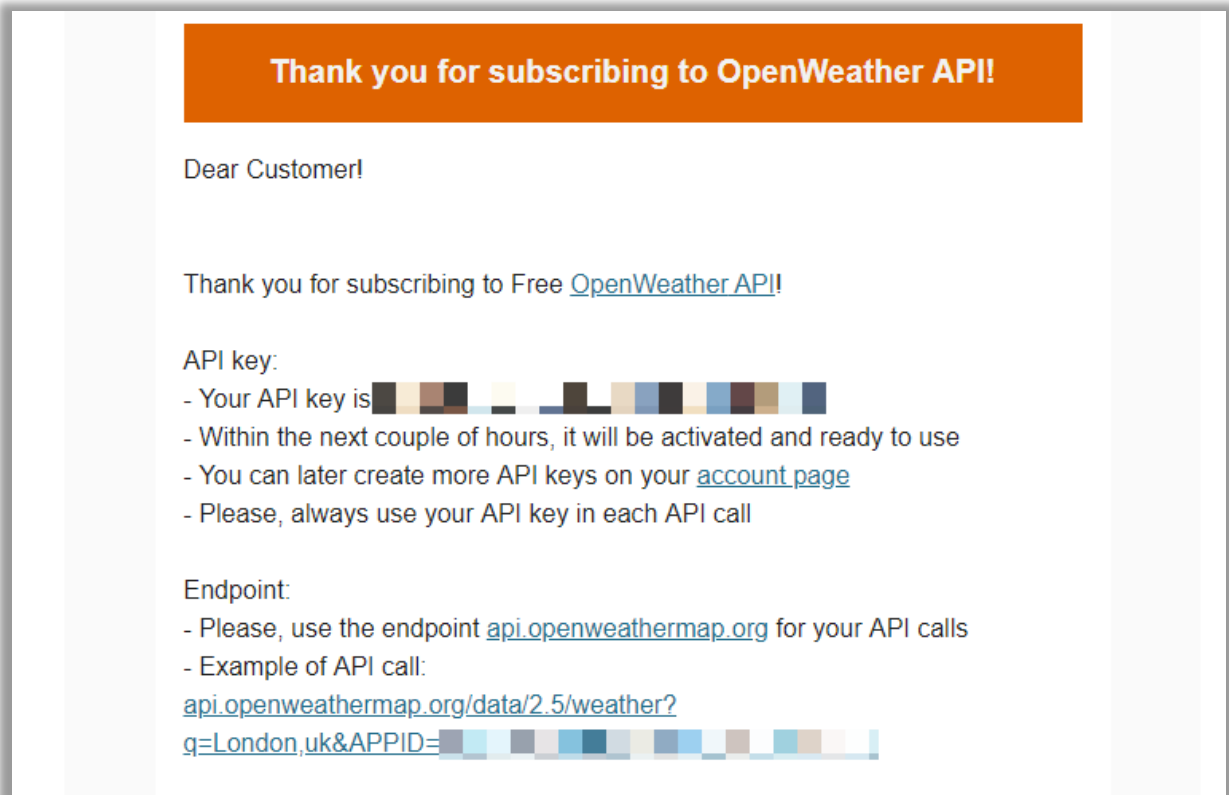


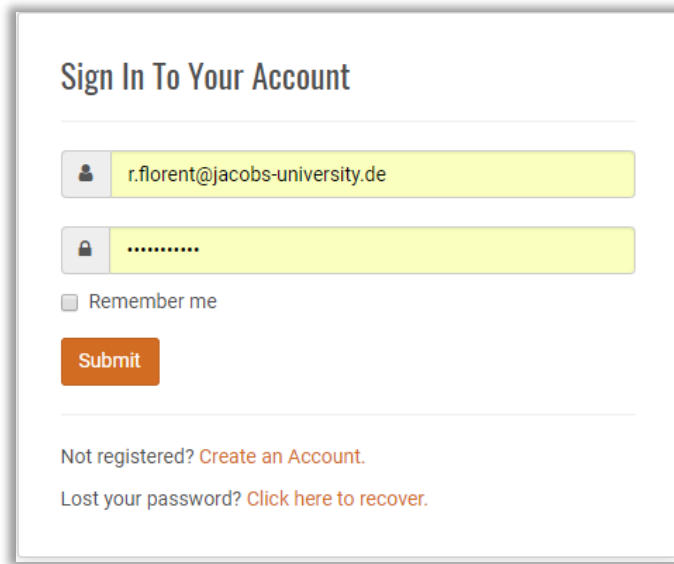
Figure 2 – *Email confirmation from OpenWeatherMap after creating an account*

A slight notice about the OWM website is that it does not use a rough, high-level authentication methodology. That is, it doesn't cover two-factor authentication, or any more advanced authentication method including the use of SMS code, email with token to verify a user. With that being said, a user can access the OWM website right after creating the account using his or her credentials.

**Note:** To avoid confusion with terminologies, a user is non-authenticated person browsing the OWM website; a customer is an authenticated user.

## Login to access

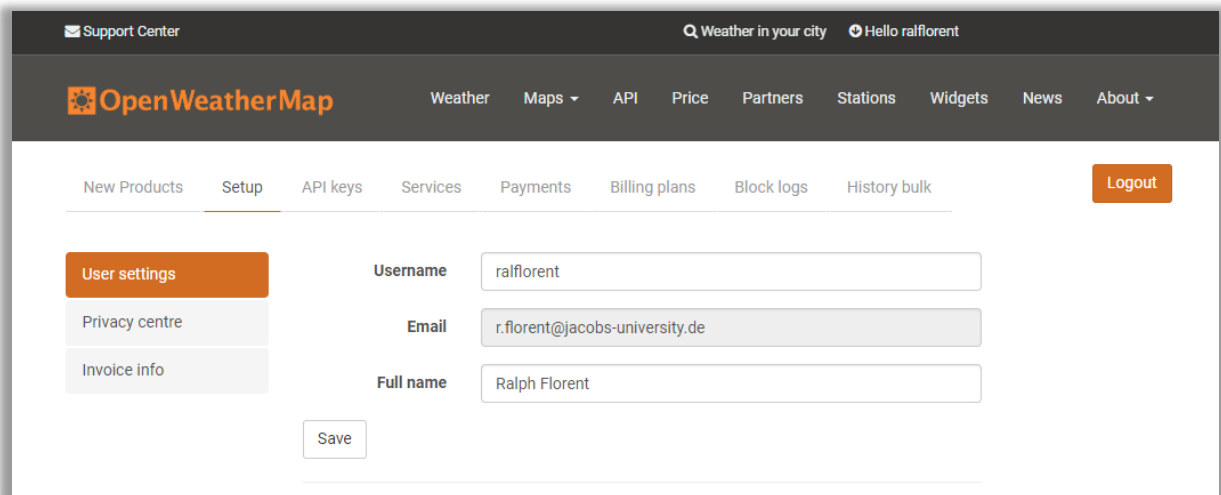
The user may want to log on to the website as a customer in order to see additional options that normally he or she could not see in the first place. Again, the sign-in button is located in the top-center header of the website and, by clicking on it, it opens a new panel that looks like the picture shown below:



The image shows a 'Sign In To Your Account' form. It has a title 'Sign In To Your Account' at the top. Below the title are two input fields: the first is for the email address, containing 'r.florent@jacobs-university.de', and the second is for the password, containing a series of dots. Below the password field is a checkbox labeled 'Remember me'. There is an orange 'Submit' button. At the bottom, there are two links: 'Not registered? Create an Account.' and 'Lost your password? Click here to recover.'

Figure 3 – Login page of the OpenWeatherMap website

Now that the customer (so-called authenticated user) is logged on, he or she can proceed to update details like name, username, password of his or her profile.



The image shows the 'User settings' page of the OpenWeatherMap website. The top navigation bar includes 'Support Center', 'Weather in your city', and 'Hello ralflorent'. The main navigation bar includes 'OpenWeatherMap', 'Weather', 'Maps', 'API', 'Price', 'Partners', 'Stations', 'Widgets', 'News', and 'About'. The sub-navigation bar includes 'New Products', 'Setup', 'API keys', 'Services', 'Payments', 'Billing plans', 'Block logs', 'History bulk', and a 'Logout' button. The 'Setup' tab is selected. On the left, there is a sidebar with 'User settings' (selected), 'Privacy centre', and 'Invoice info'. The main content area shows the 'User settings' form with fields for 'Username' (ralflorent), 'Email' (r.florent@jacobs-university.de), and 'Full name' (Ralph Florent). There is a 'Save' button at the bottom.

Figure 4 – *Customer profile on OpenWeatherMap*

## API key

The free subscription, despite its limitations in terms of services and benefits, remains a good start for a user willing to test the OWM service. In the next section, we will discuss the subscription plans in more details. For now, we will focus on the given (generated by OWM) API key. The API (*Application Programming Interface*) key is an alphanumeric hashed code associated to the user account and subscription. This is actually how the OWM website knows where and what the customer can access.

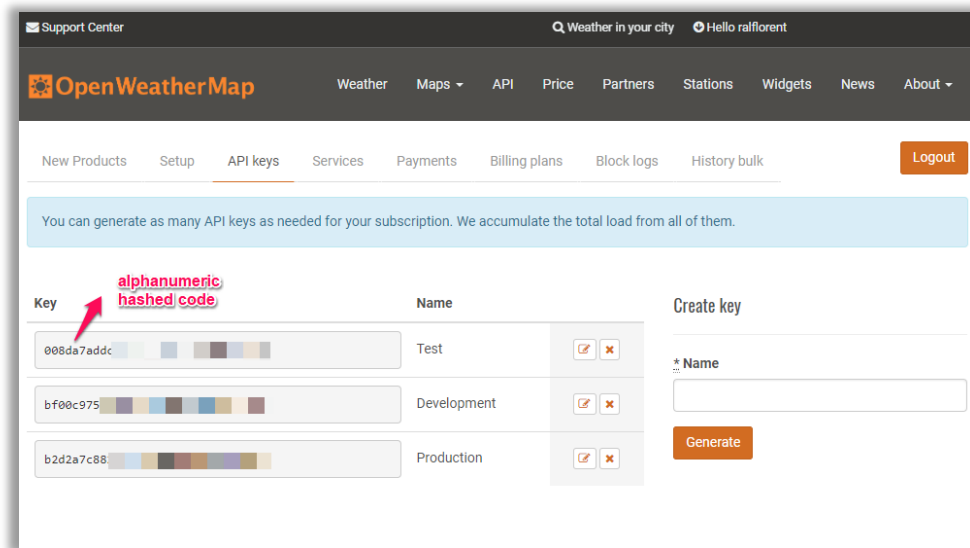


Figure 5 – *API keys for the free subscription*

## Subscription Plan

### Services

OpenWeatherMap offers its services in providing current weather, forecasts, and historical data collections. These services are sold in form of subscriptions with some limitations each. These subscription plans are:

- Free (Zero USD per month)
- Startup (40 USD per month)
- Developer (180 USD per month)

- Professional (470 USD per month)
- Enterprise (2000 USD per month)

| Current weather and forecasts collection                                    |                                       |                           |                           |                           |                           |
|---|---------------------------------------|---------------------------|---------------------------|---------------------------|---------------------------|
|   | Free                                  | Startup                   | Developer                 | Professional              | Enterprise                |
| <b>Price</b><br>Price is fixed, no other hidden costs (VAT is not included) | Free                                  | 40 USD / month            | 180 USD / month           | 470 USD / month           | 2,000 USD / month         |
| <b>Subscribe</b>  | <a href="#">Get API key and Start</a> | <a href="#">Subscribe</a> | <a href="#">Subscribe</a> | <a href="#">Subscribe</a> | <a href="#">Subscribe</a> |
| Calls per minute (no more than)   | 60                                    | 600                       | 3,000                     | 30,000                    | 200,000                   |
| Current weather API   | ✓                                     | ✓                         | ✓                         | ✓                         | ✓                         |
| 5 days/3 hour forecast API  | ✓                                     | ✓                         | ✓                         | ✓                         | ✓                         |
| 16 days/daily forecast API  | -                                     | ✓                         | ✓                         | ✓                         | ✓                         |
| Weather maps 2.0: Current, Forecast, Historical <sup>NEW</sup>              | -                                     | -                         | ✓                         | ✓                         | ✓                         |
| Relief maps <sup>NEW</sup>  | -                                     | -                         | ✓                         | ✓                         | ✓                         |
| Weather maps 1.0  | ✓                                     | ✓                         | ✓                         | ✓                         | ✓                         |
| Bulk download   | -                                     | -                         | -                         | ✓                         | ✓                         |
| UV index  | ✓                                     | ✓                         | ✓                         | ✓                         | ✓                         |
| Weather alerts  | ✓                                     | ✓                         | ✓                         | ✓                         | ✓                         |

Figure 6 – List of subscription plans

## Limitations

As we can see in Figure 6, the free subscription plan contains the basic options to test the OWM service. That is, the customer who subscribes to the *Free* subscription plan can request 60 API calls per minute to collect the forecast data and use this data for his or her own purpose. Moreover, the daily weather of the *Free* subscription plan is limited to 5 days per 3 hours (no more than that). It's clearly a business model to attract clients, however it's enough for the demo.

| Name    | Description                  | Price plan | Limits  | Details              |
|---------|------------------------------|------------|---|----------------------|
| Weather | Current weather and forecast | Free plan  | Threshold: 7,200<br>Hourly forecast: 5<br>Daily forecast: 0<br>Calls 1min: 60 | <a href="#">view</a> |

Figure 7 – Benefits and limitations of the free subscription

## Upgrade

The customer may want to upgrade his or her plan if he or she understands that it's convenient for him or her. But for our study case, we only use the free subscription to evaluate the OWM service and get access to the environmental data from a trusted source. We also hope to increase our experience of working with the earth data in its different formats (JSON, XML, HTML, HDF, etc.).

## API

As we mention in previous sections, API stands for *Application Programming Interface*. In other words, it is two software programs communicating with each other.

The OpenWeatherMap API is organized around *REST (Representational State Transfer)*. Their API has predictable resource-oriented URLs, accepts *form-encoded* request bodies, returns *JSON-encoded* responses, and uses standard HTTP response codes, no-authentication (at least for free subscription), and verbs.

The OpenWeatherMap API is also well documented. In their opinion, a customer can use their fast and easy-to-work weather APIs for free; he or she can also look at their monthly subscriptions for more options than a *Free account* can provide him or her; and finally, he or she can read how to



start first and take advantage of their powerful weather APIs [ref: <https://openweathermap.org/api>]. In other words, any customer should have full access to the how-to of their API.

**OpenWeatherMap** Weather Maps Guide API Price Partners Stations Widgets Blog

### Current weather data

[API doc](#) [Subscribe](#)

- Access current weather data for any location including over 200,000 cities
- Current weather is frequently updated based on global models and data from more than 40,000 weather stations
- Data is available in JSON, XML, or HTML format
- Available for Free and all other paid accounts

### 5 day / 3 hour forecast

[API doc](#) [Subscribe](#)

- 5 day forecast is available at any location or city
- 5 day forecast includes weather data every 3 hours
- Forecast is available in JSON and XML
- Available for Free and all other paid accounts

### 16 day / daily forecast

[API doc](#) [Subscribe](#)

- 16 day forecast is available at any location or city
- 16 day forecast includes daily weather
- Forecast is available in JSON and XML
- Available for all paid accounts

We have combined our Weather services and **Satellite imagery** in one simple and fast **Agricultural API**. **Satellite images** (True & False color, NDVI, EVI), Weather data (Current and Forecast), Historical data, Soil temperature and moisture, Accumulated temperature and precipitation, etc.

[Try it right now!](#)

### Historical data

[API doc](#) [Subscribe](#)

- Through our API we provide city historical weather data for 37,000+ cities
- In this service included Accumulated temperature and Accumulated precipitation data
- Historical data is available for 1 month previous in Starter account, for 1 year previous in Medium accounts, and for 6 years in History Bulk
- More opportunities for agriculture in our [Agricultural API](#)

### History Bulk

[API doc](#) [GET](#)

- We have recently launched History Bulk functionality that provides weather data for 37,000+ cities.
- Historical data is available for 6 years and Pricing is simple and easy - just \$10 for one city, no matter how much data will you receive.

### Weather maps 2.0 <sup>NEW</sup>

[API doc](#) [Subscribe](#)

- Forecast, Historical, Current weather maps
- 15 weather map layers
- Use as layers in Direct Tiles, OpenLayers, Leaflet, and Google Maps
- Available for Developer, Professional and Enterprise accounts

Figure 8 – API documentation for OpenWeatherMap

In this section, we skip the example part of the API documentation. We will take advantage of the **Demo** section to provide details about how we use the documentation to access the OWM data collections. That should be enough to serve as an example of the OWM API documentation.

## Demo

Recall that the project is about accessing programmatically weather and environmental data via OpenWeatherMap. Therefore, this demo allows users to read weather details in a specified city. These details mainly describe the current and daily weather and forecasts of that city.

### Tools and Software

It's important to mention the tools and software used to carry out this project. This will help to know the exact development environment put in place to realize the project and how to recreate or replicate that environment in the future, if needs be. Without getting into too many details, we use the following:

- Operating System: *Windows 10 Home, 64-bit*
- API development environment: *Postman (v6.7.2)*
- Text editor: *Notepad++ (7.5.9)*
- Screen snapshot tools: *Skitch (v2.3.2), Snipping tool*
- Versioning system: *Git, Git bash (v2.9.6), Github, @ralflorent*
- Development environment: *Jupyter Notebook*
- Programming language: *Python*

### First look at the data

We start by looking up the data and seeing what it looks like. Obviously, the OWM API documentation provides a short description of the API response for a simple request by *City Name*. But this is a good way to go since we actually want to test our given API key.

First, we set up a Postman collection name “OWM” for the requests and a development environment with the most important variables: `BASE_URL`, `API_KEY`. These two will respectively hold the values of the OpenWeatherMap API URL and the API key for our requests using Postman.

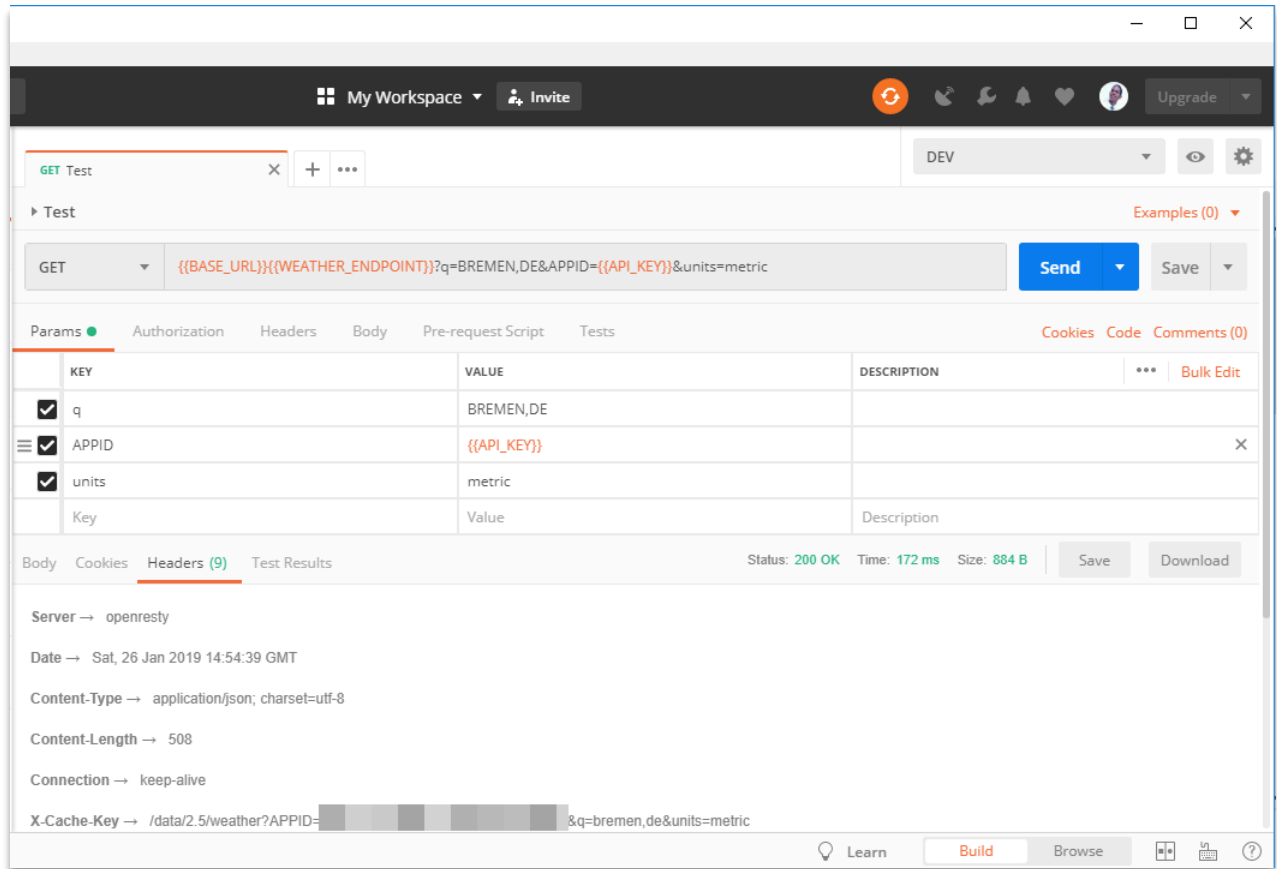


Figure 9 – Postman workspace for API test

Next, we perform a weather request using our API key for the city of Bremen, Germany. The API response looks like this:

```
{
  "coord": {
    "lon": 8.81,
    "lat": 53.08
  },
  "weather": [
    {
      "id": 310,
      "main": "Drizzle",
      "description": "light intensity drizzle rain",
      "icon": "09d"
    },
    {
      "id": 701,
      "main": "Mist",
      "description": "mist",
      "icon": "50d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 6,
    "pressure": 1000,
    "humidity": 100,
    "temp_min": 6,
    "temp_max": 6
  },
  "visibility": 2200,
  "wind": {
    "speed": 5.1,
    "deg": 250
  },
  "clouds": {
    "all": 75
  },
  "dt": 1548512400,
  "sys": {
    "type": 1,
    "id": 1281,
    "message": 0.0042,
    "country": "DE",
    "sunrise": 1548487038,
    "sunset": 1548518282
  },
  "id": 2944388,
  "name": "Bremen",
  "cod": 200
}
```

This is the raw JSON-oriented data from the request we just performed to obtain weather information for the city of Bremen, Germany. We can still use web-based online tools such as [www.jsoneditoronline.org](http://www.jsoneditoronline.org) to make the response data more human-readable.

## Scripts

Since we could successfully test via Postman our API calls, we are finally ready to write the Python scripts to automate the process and create a nice interface to view the details of the weather.

For this, we use the Jupyter Notebook, a web-based development environment for Python scripts. It's easy to use and, if all preconditions are satisfied, work smoothly from a standalone point of view. These preconditions are the availability of Python kernel and the internet access for package installation. Jupyter was previously installed alongside with other software like Spyder using Anaconda ([www.anaconda.com](http://www.anaconda.com)), a Python platform. So, no need to worry about these mentioned preconditions.

The written Python script is based on the following algorithm:

- Import relevant third-party libraries
- Mount and render the UI elements
- Read weather via API for default city
- Display human-readable weather details from the API response through (non-interactive) graphs to show daily forecasts

Figure 10 below shows the information from the JSON response and plotted accordingly.

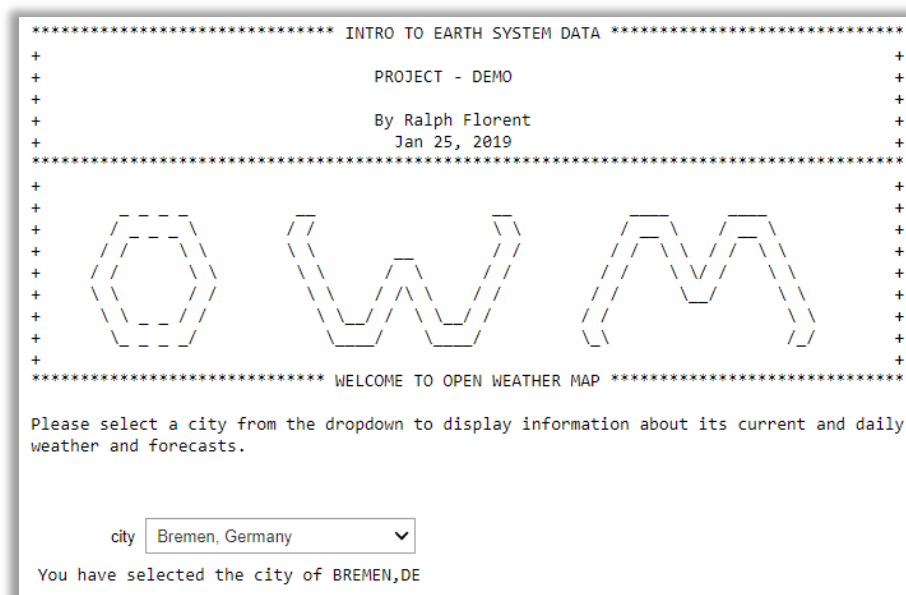


Figure 10 – *Display of welcome message when the application first runs*

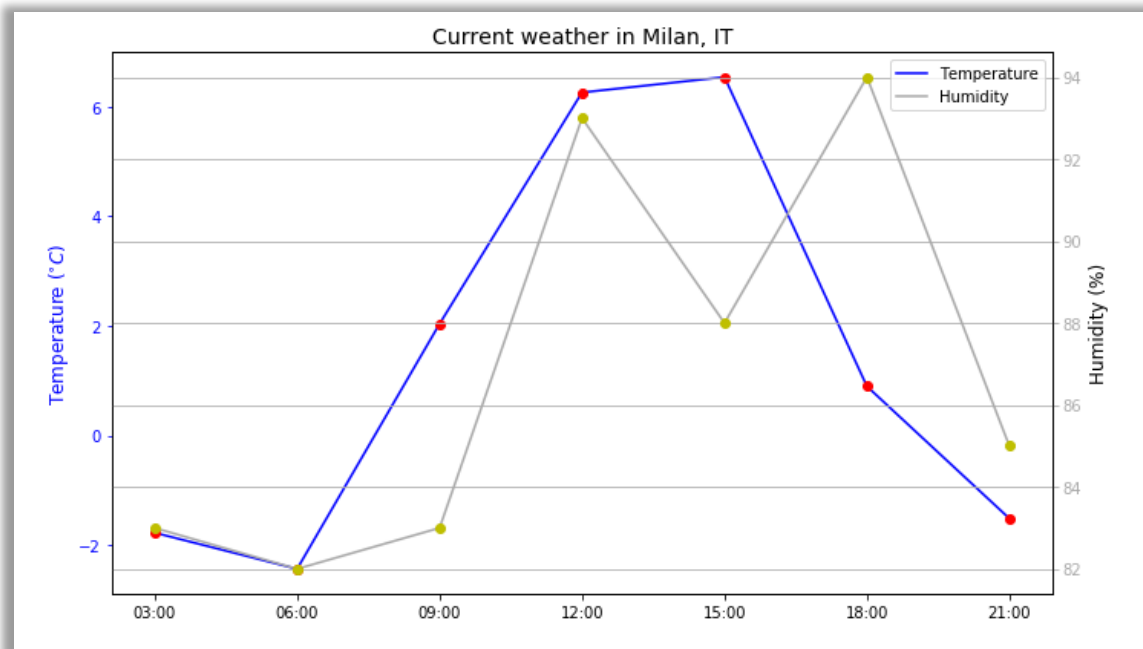


Figure 11 – Current weather in Milan, Italy via OpenWeatherMap

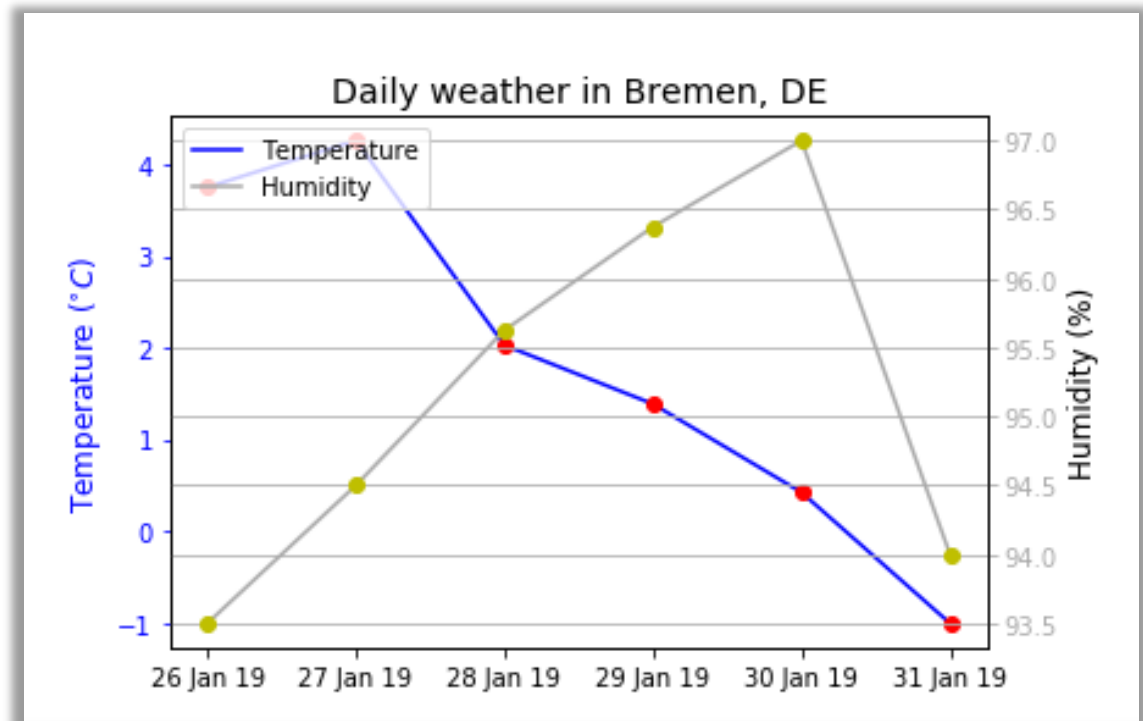


Figure 12 – Daily weather in Bremen, Germany via OpenWeatherMap

## **Conclusion**

The goal of this project is to be able to access and handle collected data programmatically. As we could go with any web-based application that offers the service of data collections, we decide to work with OpenWeatherMap to realize such project. It should be noted that the price varies according the customer's needs.

OpenWeatherMap has a well-documented API and an easy-to-use web interface. These two aspects are super important at the time of implementing our own application. The main goal is not to accentuate on the advantages and disadvantages of using OpenWeatherMap but being aware of such benefits can be valuable.

## References

- [1] Angelo P. Rossi. (2019). *The Introduction to Earth System Data Lecture*. Geoinformatics. Bremen: Jacobs University Bremen.
- [2] Postman Blog. (2014, February 20). *Using variables inside Postman and Collection Runner*. Retrieved January 25, 2019, from Postman: <https://bit.ly/2UMBTFW>
- [3] Dataquest Blog. (2015, September 7). *Python API Tutorial – An Introduction to using APIs*. Retrieved January 26, 2019, from Data Quest: <https://www.dataquest.io/blog/python-api-tutorial/>
- [4] Python Request Documentation. (2017, August 15). *Quickstart*. Retrieved January 26, 2019, from Python Request: <http://www.python-requests.org/en/latest/user/quickstart/#passing-parameters-in-urls>
- [5] W3 School. (2019). *Python Datetime*. Retrieved January 26, 2019, from W3 School: [https://www.w3schools.com/python/python\\_datetime.asp](https://www.w3schools.com/python/python_datetime.asp)
- [6] Open Weather Map. (2015 - 2019). *Weather API*. Retrieved January 27, 2019, from Open Weather Map: <https://openweathermap.org/api>
- [7] Jupyter Widgets Guide. (2015 - 2019). *User interact*. Retrieved January 27, 2019, from Jupyter Widgets: <https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html>

## Appendix

```
# -*- coding: utf-8 -*-

# Import relevant libraries
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime

import requests as http # HTTP request library
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets # useful for user interaction

### START: Scripts

# Global constants
OWM_BASE_URL      = 'http://api.openweathermap.org/data/2.5'
OWM_WEATHER_URL   = OWM_BASE_URL + '/weather'
OWM_FORECAST_URL  = OWM_BASE_URL + '/forecast'
OWM_API_KEY       = '008da7addc4cd8fc63053350fd71d595'

OWM_UNITS         = {'C': 'metric', 'F': 'imperial', 'K': ''}
OWM_COLORS        = {'grey': '#AAAAAA'}
OWM_ERR_MSG       = ''

DEFAULT_CITY_NAME = 'BREMEN'
"""
get_data_from_API - Perform API requests.
This function basically sends HTTP requests (GET mainly) to Open Weather Map and expects
a JSON-oriented weather data.
Args:
    url: the url of the Open Weather Map's endpoint to perform API calls
    city_name: name of the city (including the ISO-alpha2 code) of interests

Returns:
    the JSON data obtain from the HTTP response if successful (city found)

Exception:
    return an HTTP response (JSON) of NOT_FOUND (404)
"""
def get_data_from_API(url = OWM_WEATHER_URL, city_name = DEFAULT_CITY_NAME):
    params = { 'q': city_name, 'APPID': OWM_API_KEY, 'units': OWM_UNITS['C'] }
    return http.get(url, params = params)
```



```

"""
validate_data_from_API - Validate HTTP response once request is performed.
This function intends to standardize the response object so that some validation
rule can be used in the future. NOT IMPLEMENTED for the moment.
Args:
    response: plain HTTP response in JSON-like format

Returns:
    a new transformed dictionary with 'data' key, if successful or 'error', otherwise.
"""
def validate_data_from_API(response):
    # API response
    if(response.status_code == 200):
        return { 'data': response.json() }
    return { 'error': response.json() }

"""
get_current_weather - Get today's weather from loaded data

Args:
    data: JSON data (response) from Open Weather Map

Returns:
    a tuple of 'times' and 'temps' (temperatures) of today's date.
    additionally, 'filtered', the filtered 'main' object data for future use.
    finally, 'metadada', the story behind the filtered data.
"""
def get_current_weather(data):
    metadata = {
        'plot_title': 'Current weather in ' + data['city']['name'] + ', ' + data['city']['country']
    }
    today = datetime.today().strftime('%Y-%m-%d')
    times, temps, filtered = [], [], []

    # select today's date only
    for el in data['list']:
        day, time = el['dt_txt'].split(' ')
        if(day == today):
            times.append(time[:-3]) # get rid of seconds
            temps.append(el['main']['temp'])
            filtered.append(el['main'])

    return (times, temps, filtered, metadata)

```

```
def calculate_all_mean(objects):
    # mean accumulator
    acc = {
        "temp": 0,
        "temp_min": 0,
        "temp_max": 0,
        "pressure": 0,
        "sea_level": 0,
        "grnd_level": 0,
        "humidity": 0,
        "temp_kf": 0
    }

    """
    The idea is to look up the sibling elements distributed in each 'main' object
    of the array of objects and calculate the mean of all together. The data looks
    like this:
    [
        { 'main': {'temp': 1, 'temp_min': 2, ...} },
        { 'main': {'temp': 1, 'temp_min': 2, ...} }, ...
    ]
    Algorithm:
        1) sum them all accordingly (distributed)
        2) calculate the mean (Average / length)
    """

    for o in objects:
        for k, v in o['main'].items():
            acc[k] += v
    for k, v in acc.items():
        acc[k] = acc[k] / len(objects)
    return acc

"""
get_daily_weather - Get daily weather from loaded data
Recall that for the Free subscription plan, the daily basis is 5 days 3-hour wise.
This function groups temperatures by day and calculate the mean temperature for
that day.

Args:
    data: JSON data (response) from Open Weather Map

Returns:
    a tuple of 'times' and 'temps' (temperatures) on a daily basis.
    additionally, 'filtered', the filtered 'main' object data for future use.

```

```

    finally, 'metadada', the story behind the filtered data.
"""
def get_daily_weather(data):
    metadata = {
        'plot_title': 'Daily weather in ' + data['city']['name'] + ', ' + data['city']['country']
    }
    days = {}
    times, temps, filtered = [], [], []

    # group by same date: yyyy-MM-dd
    for el in data['list']:
        day, time = el['dt_txt'].split(' ')
        if day in days:
            days[day].append(el)
        else:
            days[day] = [el]

    # accumulate mean temperature on a daily basis
    for key, value in days.items():
        parsed_date = datetime.strptime(key, '%Y-%m-%d')
        formatted_date = parsed_date.strftime('%d %b %y')
        mean_values = calculate_all_mean(value)

        times.append(formatted_date)
        temps.append(mean_values['temp'])
        filtered.append(mean_values)

    return (times, temps, filtered, metadata)

# helper function to plot graphs
def plot_graph(data):
    times, temps, filtered, metadata = data
    humidities = [f['humidity'] for f in filtered]

    fig = plt.figure(figsize=(11, 6.5))
    # Set temperature y-axis on the left
    left_axis = fig.add_subplot(111)
    temp_line = left_axis.plot(times, temps, '-b')
    left_axis.plot(times, temps, 'ro')
    left_axis.set_ylabel('Temperature ($^{\circ}$C)', color='blue', fontsize=12)
    left_axis.tick_params(axis='y', colors='blue')
    # left_axis.set_xlabel('Time', fontsize=12)

    # Set humidity y-axis on the right

```

```

right_axis = left_axis.twinx()
hum_line = right_axis.plot(times, humidities, color=OWM_COLORS['grey'])
right_axis.plot(times, humidities, 'yo')
right_axis.set_ylabel('Humidity (%)', fontsize=12)
right_axis.tick_params(axis='y', colors=OWM_COLORS['grey'])

# Set additional/common properties for the figure
plt.legend(temp_line + hum_line, ['Temperature', 'Humidity'], loc='best')
plt.title(metadata['plot_title'], fontsize=14)
plt.grid()
plt.show()

# main function to run the entire process: fetch, prepare, and visualize data
def main(city):

    print('You have selected the city of', city)
    data = {}

    # Access data online via API
    api_response = get_data_from_API( OWM_FORECAST_URL, city)
    content = validate_data_from_API( api_response )

    if('data' in content):
        OWM_ERR_MSG = ''
        data = content['data']
    else:
        OWM_ERR_MSG = 'The weather for ' + city + ' could not be found. '
        OWM_ERR_MSG += 'Check for your internet connection and try again later.'
        print(OWM_ERR_MSG)
        # TODO: handle type of error info to display
        return

    current_weather = get_current_weather( data )
    plot_graph( current_weather )
    daily_weather = get_daily_weather( data )
    plot_graph( daily_weather )

# helper function to load static cities
def load_static_cities():
    return [
        ('Bremen, Germany', 'BREMEN,DE'), ('Mexico, Mexico', 'MEXICO,MX'),
        ('Milan, Italy', 'MILAN,IT'), ('New York, United States', 'NEW YORK,US'),
        ('London, United Kingdom', 'LONDON,UK'), ('Ottawa, Canada', 'OTTAWA,CA'),
        ('Paris, France', 'PARIS,FR'), ('Port-au-Prince, Haiti', 'PORT-AU-PRINCE,HT'),

```

```

        ('Santiago, Chile', 'SANTIAGO,CL'), ('Tokyo, Japan', 'TOKYO,JP')
    ]

# main application
def application():
    file = open('../assets/data/welcome-msg@project.txt', 'r')
    print(file.read())
    file.close()
    cities = load_static_cities()
    interact(main, city=cities)

# run application
application()
### END: Scripts

```