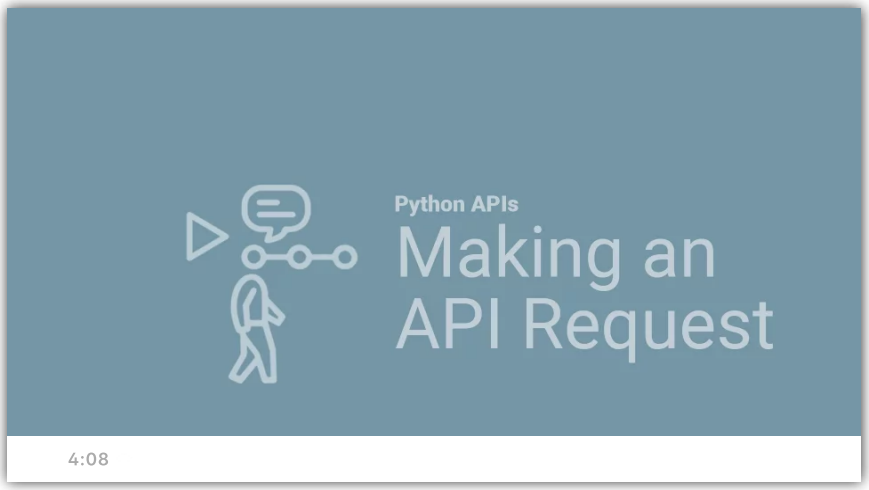
6.2.4 Make a Request for Data to an API

That tutorial was quick and painless. Now, a quick self-test: You want to see if you're really understanding by checking to see if you can retrieve the JSON data and display it in your Jupyter Notebook. If you can do this correctly, you'll know you can successfully make a request and retrieve the JSON file.



Although we have seen the data in a web browser, we need to retrieve this data from the JSON file so we can add it to a DataFrame and make our visualizations.

Retrieve a Response Using the get() Method

Use the `get()` method, a feature of the Requests Library, to request data from an API. The `get()` method is one of many HTTP methods that allow us to access, add, delete, get the headers, and perform other actions on the request.

Below is a list of HTTP methods and their uses. For more information, see the **Requests Library** (https://requests.kennethreitz.org/en/master/).

Request Method	Action
`get()`	Retrieves data from a web source.
`head()`	Retrieves the headers from a web source.
`post()`	Adds or annotates data on a web source. Used on mailing groups, message boards, or comments.
`put()`	Updates an existing resource. For example, if the date on a Wikipedia page is wrong, you can use the `put()` method to update that date.
`delete()`	Deletes data from a web source.
`options()`	Discovers what HTTP methods a web source allows.
`patch()`	Partially modifies a web source.

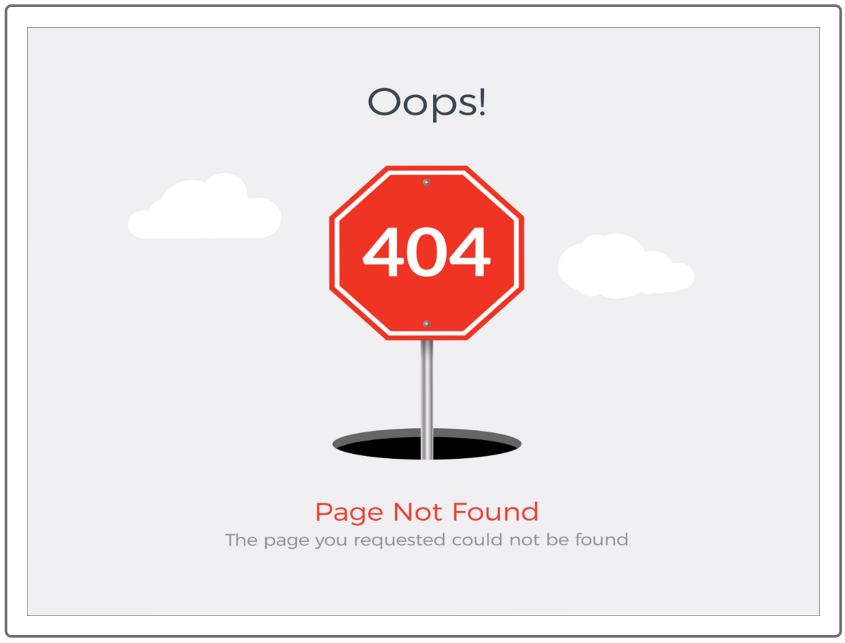
Inside the parentheses of the `get()` method, add the URL–in our case, the `city_url`. Let's make a request to get our weather data for Boston.

Add the following code in a new cell of your `API_practice` file and run the cell.

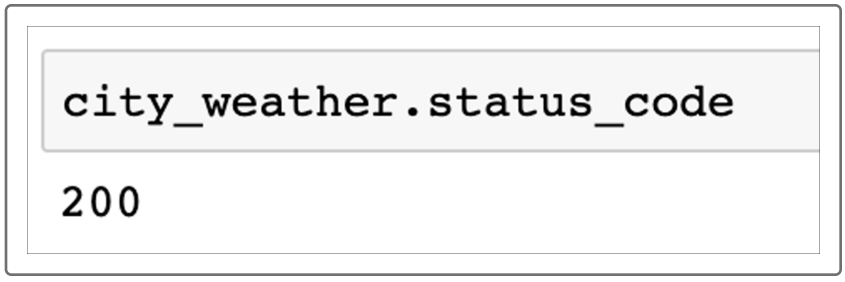
```
# Make a 'Get' request for the city weather.
city_weather = requests.get(city_url)
city_weather
```

When you run this code you will get a code response number.

The code output will be `<Response [200]>`, indicating a valid response. We won't see this code when a website appears in a browser. However, when a website does not appear, we'll see a 404 code, indicating a client error.



You can directly call the response code with the `get()` method using the `status_code`. If we chain the `status_code` to the `city_weather` variable, we get 200 as the output.



If we tried to get weather data from an unrecognized city, or if the weather data for a city wasn't available, we would get a 404 response.

Let's see what would happen if we misspelled a city name–"`Bston`" instead of "`Boston`." Add the following code to a new cell and run the cell.

```
# Create an endpoint URL for a city.
city_url = url + "&q=" + "Bston"
city_weather = requests.get(city_url)
city_weather
```

When we run this cell the output is a `<Response [404]>`.

```
# Create an endpoint URL for a city.
city_url = url + "&q=" + "Bston"
city_weather = requests.get(city_url)
city_weather
```

```
<Response [404]>
```

We'll review how to handle such response errors, but first, let's learn how to get data from a request.

Get Data from a Response

Now, we'll see what happens when we get a valid response.

In a new cell, correct the spelling for the city of "`Boston`" to create the `city_url`.

```
# Create an endpoint URL for a city.
city_url = url + "&q=" + "Boston"
city_weather = requests.get(city_url)
city_weather
```

When we run this cell the output is "`<Response [200]>`."

When we receive a valid response from the server, we have to decide on the data format. The options are text, JSON, XML, or HTML format. We can apply the format attributes to get the data into a useful format to parse.

One format that provides a preview of the JSON data is the `text` attribute. Let's get the content for the Boston weather data using the following code.

```
# Get the text of the 'Get' request.
city_weather.text
```

Running this cell, we'll get the following output:

```
'{"coord":{"lon":-71.06,"lat":42.36},"weather":[{"id":801,"main":"Clouds","description":"few
clouds","icon":"02d"}],"base":"stations","main":{"temp":61.7,"pressure":1020,"humidity":59,"t
emp_min":57,"temp_max":66.2},"visibility":16093,"wind":{"speed":14.99,"deg":80},"rain":{},"cl
ouds":{"all":20},"dt":1571678675,"sys":{"type":1,"id":3486,"country":"US","sunrise":157165581
8,"sunset":1571694830},"timezone":-14400,"id":4930956,"name":"Boston","cod":200}'
```

The text in the output is a dictionary of dictionaries and arrays, or a JSON file. We can work with this data, but it might be more challenging if we needed to retrieve temperature (`temp`) and humidity (`humidity`) from this output because the data is in a sentence format.

Let's use the `json()` attribute with our response and run the cell.

```
# Get the JSON text of the 'Get' request.
city_weather.json()
```

When we run the cell the output will be the weather data for Boston in JSON format.

```
{'coord': {'lon': -71.06, 'lat': 42.36},
 'weather': [{'id': 801,
   'main': 'Clouds',
   'description': 'few clouds',
   'icon': '02d'}],
 'base': 'stations',
 'main': {'temp': 61.7,
  'pressure': 1020,
  'humidity': 59,
  'temp_min': 57,
  'temp_max': 66.2},
 'visibility': 16093,
 'wind': {'speed': 14.99, 'deg': 80},
 'rain': {},
 'clouds': {'all': 20},
 'dt': 1571678675,
 'sys': {'type': 1,
  'id': 3486,
  'country': 'US',
  'sunrise': 1571655818,
  'sunset': 1571694830},
 'timezone': -14400,
 'id': 4930956,
 'name': 'Boston',
 'cod': 200}
```

With the JSON method, it is a lot easier to see the overall structure of the data, which will make it easier to retrieve data such as temperature and humidity.

Handle Request Errors

When we submit a `get` request for the `city_weather`, we want to make sure that we get a valid response, i.e., 200, before we retrieve any data. To check if we get a valid response, we can write a conditional expression that will evaluate whether the status code is equal to 200. If it is, then we can print out a statement that says the weather data was found. If there is a response other than 200, we can print out a statement that says the weather was not found, as in the following example:

```
# Create an endpoint URL for a city.
city_url = url + "&q=" + "Boston"
city_weather = requests.get(city_url)
if city_weather.status_code == 200:
    print(f"City Weather found.")
else:
    print(f"City weather not found.")
```

When the conditional expression is evaluated, it will print `City weather found` if true, or `City weather not found` if false. When we run the cell code above, the output is `City weather found.`

NOTE

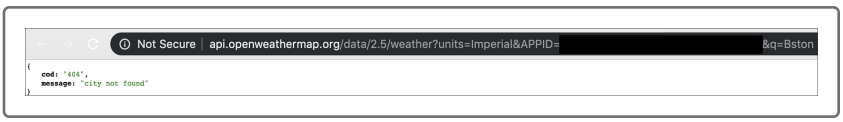
> If the `status_code` is something other than 200, JSON data will always be returned in the request. We can determine if the response was successful by checking the `status_code`, clicking the URL, or retrieving specific information from the JSON data.

Add the following code to a new cell and run the cell to test if we get the JSON formatted data.

```
# Create an endpoint URL for a city.
city_url = url + "&q=" + "Bston"
city_weather = requests.get(city_url)
if city_weather.status_code == 200:
    print(f"City Weather found.")
else:
    print(f"City weather not found.")
```

The output for this code is City weather not found.

However, if we type `print(city_url)` in a new cell and run the cell, the output will be a URL. If we click the URL, the web browser returns a 404 response and there is no data to retrieve.



We'll learn more about how to handle retrieving JSON data from a URL later in this module.