6.5.4 Map Vacation Criteria

Once the customers have filtered the database (DataFrame) based on their temperature preferences, show them a heatmap for the maximum temperature for the filtered cities. In addition, create a marker for each city that will display the name of the city, country code, maximum temperature, and name of a nearby hotel within three miles of the coordinates when the marker is clicked.

Using the coordinates from the `preferred_cities_df` DataFrame, find a hotel using our Google Places API and then retrieve that hotel information. Once we retrieve the hotel information, we'll need to store it so we can reference it and add the information to the pop-up marker.

Get Travel Destinations

Don't add the hotel information to the `preferred_cities_df` DataFrame because this DataFrame is our filtered DataFrame, and the customer will always filter it for each trip. We'll need to create a new DataFrame specifically for the data needed to create a heatmap and pop-up markers.

Make a copy of the `preferred_cities_df` DataFrame and name it `hotel_df`. For the `hotel_df`, keep the columns "City," "Country," "Max Temp," "Lat," and "Lng." Add a new column to the `hotel_df` DataFrame to hold the name of the hotel.

Add the following code to a new cell and run it to create the `hotel_df` DataFrame.

```
# Create DataFrame called hotel_df to store hotel names along with city, cou
hotel_df = preferred_cities_df[["City", "Country", "Max Temp", "Lat", "Lng"]
hotel_df["Hotel Name"] = ""
hotel_df.head(10)
```

Your `hotel_df` DataFrame should look like the following, with an empty Hotel Name column.

	City	Country	Max Temp	Lat	Lng	Hotel Name
7	Avarua	CK	77.00	-21.21	-159.78	
13	Vaini	IN	77.30	15.34	74.49	
16	Dingle	PH	75.43	11.00	122.67	
21	Vinh Yen	VN	77.00	21.31	105.60	
28	Mazagao	BR	84.20	-0.12	-51.29	
29	Marsaxlokk	MT	82.00	35.84	14.54	
30	Portland	US	82.00	43.66	-70.25	
32	Camacha	PT	77.00	33.08	-16.33	
34	Prata	BR	76.75	-19.31	-48.92	
45	Albany	US	75.99	42.65	-73.75	

Using the latitude and longitude and specific parameters, use the Google Places Nearby Search request to retrieve a hotel and add it to the Hotel Name column.

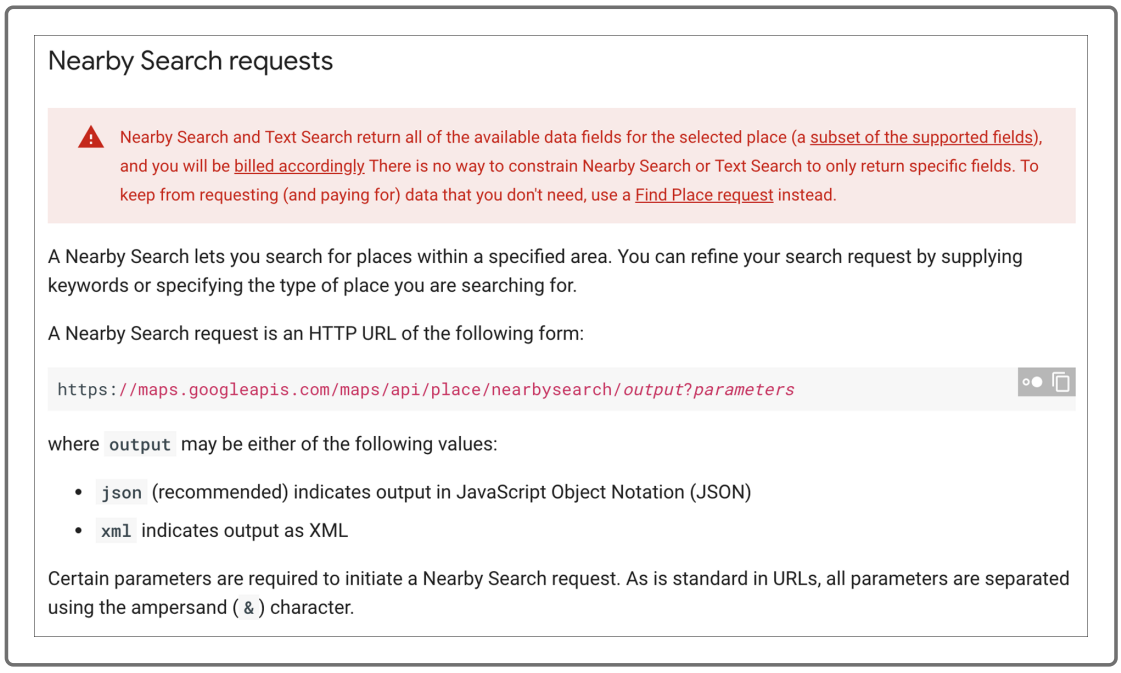
Retrieve Hotels from a Nearby Search

The first step for retrieving hotels from a Nearby Search is to set the parameters for the search.

Set the Parameters for a Nearby Search

To find the nearest establishment to geographic coordinates, use the Google Places Nearby Search request. First, navigate to the **Nearby Search requests page. (https://developers.google.com/places/web-service/search#PlaceSearchRequests%0D%0A)**

Review the following documentation from the page, as seen in the image:



Nearby Search requests

Nearby Search and Text Search return all of the available data fields for the selected place (a subset of the supported fields), and you will be billed accordingly There is no way to constrain Nearby Search or Text Search to only return specific fields. To keep from requesting (and paying for) data that you don't need, use a Find Place request instead.

A Nearby Search lets you search for places within a specified area. You can refine your search request by supplying keywords or specifying the type of place you are searching for.

A Nearby Search request is an HTTP URL of the following form:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/output?parameters
```

where `output` may be either of the following values:

- `json` (recommended) indicates output in JavaScript Object Notation (JSON)
- `xml` indicates output as XML

Certain parameters are required to initiate a Nearby Search request. As is standard in URLs, all parameters are separated using the ampersand (`&`) character.

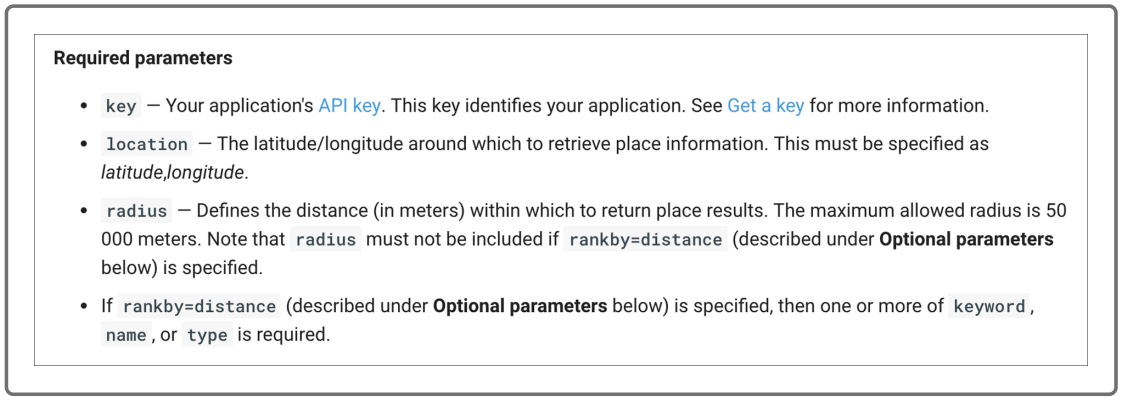
The Nearby Search request lets us perform these tasks:

1. Search for places within a specified area.

2. Refine the search using keywords and specifying the type of place we are searching for.
3. Use an API URL, where the output can be either XML or JSON format.

Although very similar to our OpenWeatherMap API search, with the Google Places Nearby Search, we'll add a specified area and keyword to search.

The documentation states we must provide specific parameters.



Required parameters

- `key` — Your application's API key. This key identifies your application. See Get a key for more information.
- `location` — The latitude/longitude around which to retrieve place information. This must be specified as *latitude,longitude*.
- `radius` — Defines the distance (in meters) within which to return place results. The maximum allowed radius is 50 000 meters. Note that `radius` must not be included if `rankby=distance` (described under **Optional parameters** below) is specified.
- If `rankby=distance` (described under **Optional parameters** below) is specified, then one or more of `keyword`, `name`, or `type` is required.

Specific parameters include:

1. Our API key
2. A location, which will be the latitude and longitude
3. A radius for the search. The radius can be up to 50,000 meters or approximately 31 miles. This distance is much too great for finding hotels, so we'll keep our search to 5,000 meters.
4. If we use the `rankby=distance` for a parameter, we need to add one or more of the three parameters above. We won't use the `rankby=distance` because we will be search based on the radius from a given latitude and longitude.

We can add optional parameters such as a keyword, a type of business, or the name of a business.

Optional parameters

- `keyword` — A term to be matched against all content that Google has indexed for this place, including but not limited to name, type, and address, as well as customer reviews and other third-party content.
- `language` — The language code, indicating in which language the results should be returned, if possible. See the list of supported languages and their codes. Note that we often update supported languages so this list may not be exhaustive.
- `minprice` and `maxprice` (*optional*) — Restricts results to only those places within the specified range. Valid values range between 0 (most affordable) to 4 (most expensive), inclusive. The exact amount indicated by a specific value will vary from region to region.
- `name` — A term to be matched against all content that Google has indexed for this place. Equivalent to `keyword`. The `name` field is no longer restricted to place names. Values in this field are combined with values in the `keyword` field and passed as part of the same search string. We recommend using only the `keyword` parameter for all search terms.
- `opennow` — Returns only those places that are open for business at the time the query is sent. Places that do not specify opening hours in the Google Places database will not be returned if you include this parameter in your query.
- `rankby` — Specifies the order in which results are listed. Note that `rankby` must not be included if `radius` (described under **Required parameters** above) is specified. Possible values are:
  - `prominence` (default). This option sorts results based on their importance. Ranking will favor prominent places within the specified area. Prominence can be affected by a place's ranking in Google's index, global popularity, and other factors.
  - `distance`. This option biases search results in ascending order by their distance from the specified `location`. When `distance` is specified, one or more of `keyword`, `name`, or `type` is required.
- `type` — Restricts the results to places matching the specified type. Only one type may be specified (if more than one type is provided, all types following the first entry are ignored). See the list of supported types.
- `pagetoken` — Returns the next 20 results from a previously run search. Setting a `pagetoken` parameter will execute a search with the same parameters used previously — all parameters other than `pagetoken` will be ignored.

For our hotel search, we'll use these parameters:

1. API key
2. Latitude and longitude
3. 5,000-meter radius
4. Type of place

To discover the support types we can use, click on the link "list of supported types" in the optional parameters list (shown in the image above) or navigate to the **Place Types guide (https://developers.google.com/places/web-service/supported_types)** .

On the webpage, Table 1 shows all the different Place types values. "Hotel" does not appear, but "lodging" does, so we will use the string "lodging" for the type parameter.

Next, we need to know how to add the other parameters in the search.

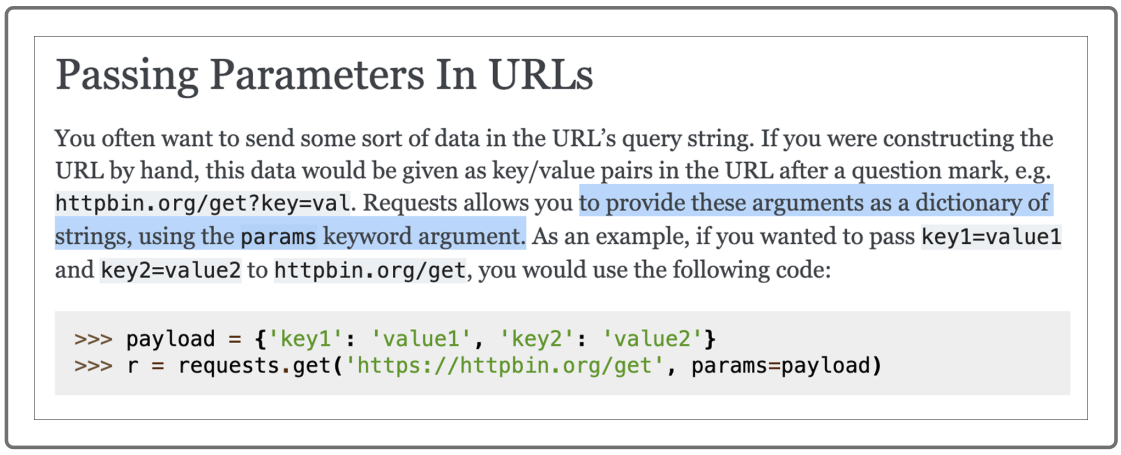
REWIND

Recall that when we made a request with the OpenWeatherMap API, we added the base URL with the city, `city_url`, to the request, `city_weather = requests.get(city_url)`.

We can use the same format to make a request with the Google Places API. The following base URL is provided to retrieve the JSON format of the data:

https://maps.googleapis.com/maps/api/place/nearbysearch/json

Next, we'll need to look at the **documentation on the Python Requests Library (https://requests.kennethreitz.org/en/master/)** provided earlier. The section "Passing Parameters In URLs" states that we can add the parameters as a dictionary of strings, using the `params` keyword argument `requests.get('base URL', params=parameters)`. This is highlighted in the following screenshot.



Passing Parameters In URLs

You often want to send some sort of data in the URL's query string. If you were constructing the URL by hand, this data would be given as key/value pairs in the URL after a question mark, e.g. `httpbin.org/get?key=val`. Requests allows you to provide these arguments as a dictionary of strings, using the `params` keyword argument. As an example, if you wanted to pass `key1=value1` and `key2=value2` to `httpbin.org/get`, you would use the following code:

```
>>> payload = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.get('https://httpbin.org/get', params=payload)
```

In a new cell, add the parameters as key-value pairs. Add the `params` dictionary, API key, `type`, and `radius` parameters with the following

values.

```
# Set parameters to search for a hotel.
params = {
    "radius": 5000,
    "type": "lodging",
    "key": g_key
}
```

The `g_key` is our Google API key, so be sure to import the `config.py` file.

Next, add the coordinates for the location parameter for each city, which need to be pulled from the `Lat` and `Lng` columns of the `hotel_df` DataFrame. We'll need to iterate through the columns of the DataFrame and add the coordinates to the `params` dictionary before making the request and retrieving the JSON data. However, before iterating through the `hotel_df` DataFrame, we need to know what the JSON data looks like from a search. Let's practice using a fixed latitude and longitude so we understand where to get the hotel name.

Practice Using a Fixed Latitude and Longitude

Create a new Jupyter Notebook file and name it `Google_Nearby_Search.ipynb`. In the first cell, import your dependencies.

```
# Dependencies and Setup
import requests
import gmaps

# Import API key
from config import g_key
```

In the next cell, add the following `params` dictionary to search for a hotel in Paris, the base URL, and the request for the JSON data.

```
# Set the parameters to search for a hotel in Paris.
params = {
    "radius": 5000,
    "types": "lodging",
    "key": g_key,
    "location": "48.8566, 2.3522"}
# Use base URL to search for hotels in Paris.
base_url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json"
# Make request and get the JSON data from the search.
hotels = requests.get(base_url, params=params).json()

hotels
```

When we call the hotels variable to look at the JSON data, we get the following at the top of the output.

Scroll through the information. Below the `next_page_token` is a `results` dictionary. To get all the hotels in the `results` dictionary we can get the length of this dictionary by using, `len(hotels["results"])`.

```
len(hotels["results"])
```

```
20
```

The output of the `len(hotels["results"])` returns 20 hotels. To retrieve more hotels, use the `next_page_token`, as it states in the Google Maps documentation.

Upon closer examination of the JSON file, we see that the first hotel is Hôtel Duo, which is a value for the key `name`.


```
{'html_attributions': [],
 'next_page_token': 'CpQEBQIAADTJuc40_-nuYAq
x1nY35GyVKx7jb3OWy8La3VxoEBP7sa4xvSEFvvP-eek
pRGswKdR76UaIp2PnI5KnEQ5Shwg2_9BszTCEdx9PxWs
m3evUqm8_QnHskrSyZenQVTGTGjUpWwBW2OdSPa5heBz
qQtpU7gHZeRV0bYfKWFXJxkX166xQAoE0oRbd-2GrcWC
RlOymDOaMVP1unu1CYTifytGjeeVXC4753QqGyZDMcv7
Gprs5YfgvhR6cfs5tOGUV0_uD3QJ3nX5xIQBqv2u3o7U
 'results': [{'geometry': {'location': {'lat
    'viewport': {'northeast': {'lat': 48.859
      'lng': 2.354348530291502},
     'southwest': {'lat': 48.8567422697085,
   'icon': 'https://maps.gstatic.com/mapfile
   'id': '7fa47cb61d7345cecc3b802d0e04fcff2b
   'name': 'Hôtel Duo',
   'opening_hours': {'open_now': True},
   'photos': [{'height': 3840,
     'html_attributions': ['<a href="https:/
>']
```

Retake

Now that we know how to retrieve the hotel name from the JSON data, we can iterate over the rows in the `hotel_df` DataFrame for the coordinates.

Iterate Through hotel_df DataFrame

We can use the `iterrows()` function to perform the iteration; however, we need to provide the `index` and the `row` in the `for` loop using this syntax: **`for index, row in df.iterrows()`**.

In our `VacationPy.ipynb` file and below the `params` dictionary, add the following code.

```
# Iterate through the DataFrame.
for index, row in hotel_df.iterrows():
    # Get the latitude and longitude.
    lat = row["Lat"]
    lng = row["Lng"]

    # Add the latitude and longitude to location key for the params dictiona
    params["location"] = f"{lat},{lng}"

    # Use the search term: "lodging" and our latitude and longitude.
    base_url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json
    # Make request and get the JSON data from the search.
    hotels = requests.get(base_url, params=params).json()
    # Grab the first hotel from the results and store the name.
    hotel_df.loc[index, "Hotel Name"] = hotels["results"][0]["name"]
```

In this code, we modified how we add the coordinates to the `location` key. We add the latitude and longitude to the `location` using the f-string format. The rest of the code is the same as when we practiced, but at the

end, we add the name of the first hotel to the Hotel Name column in the `hotel_df` DataFrame.

Let's run the cell and retrieve the data.

Oops! Our output says there is an `IndexError`.

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-20-2e5688034bf2> in <module>
     21
     22     # Grab the first hotel from the results and store the name.
---> 23     hotel_df.loc[index, "Hotel Name"] = hotels["results"][0]["name"]

IndexError: list index out of range
```

Retake

REWIND

Recall that when trying to parse the data from an API request, we need to use a `try-except` block to handle the error with a statement and continue the retrieval.

Let's modify the last piece of code and make a `try-except` block to handle the `IndexError`. Edit the last piece of code so that it looks like the following.

```
# Grab the first hotel from the results and store the name.
  try:
```

```
    hotel_df.loc[index, "Hotel Name"] = hotels["results"][0]["name"]
  except (IndexError):
    print("Hotel not found... skipping.")
```

NOTE

> If you encounter more errors, add them to the `except` block—like so, `(IndexError, NewError)`—and continue running your code until there are no errors.

In the output, the first 10 rows of the `hotel_df` DataFrame should look like the following.

	City	Country	Max Temp	Lat	Lng	Hotel Name
7	Avarua	CK	77.00	-21.21	-159.78	Paradise Inn
13	Vaini	IN	77.30	15.34	74.49	Dandeli Lake County
16	Dingle	PH	75.43	11.00	122.67	Vhiel Sebandra
21	Vinh Yen	VN	77.00	21.31	105.60	NGOC HA 3 HOTEL
28	Mazagao	BR	84.20	-0.12	-51.29	Casa Lima Santos
29	Marsaxlokk	MT	82.00	35.84	14.54	Water's Edge
30	Portland	US	82.00	43.66	-70.25	Hampton Inn Portland Downtown - Waterfront
32	Camacha	PT	77.00	33.08	-16.33	Hotel Porto Santo & Spa
34	Prata	BR	76.75	-19.31	-48.92	Hotel Pontal Aluga-se Quitinetes e suítes mens...
45	Albany	US	75.99	42.65	-73.75	Hilton Albany

Our final task will be to add pop-up markers with hotel information to a heatmap.

NOTE

> For more information, see the **Google Developers' Nearby Search documentation (https://developers.google.com/places/web-service/search#PlaceSearchRequests)**.

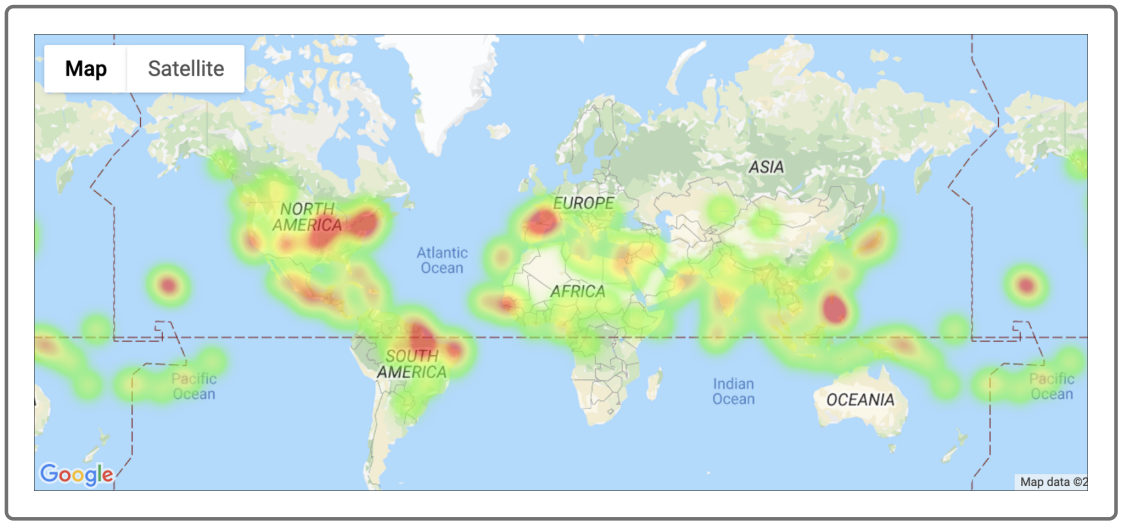
Create a Maximum Temperature Heatmap from a Hotel DataFrame

Before we add pop-up markers with hotel information, let's create a heatmap using the maximum temperature from our `hotel_df` DataFrame by reusing the code and changing the DataFrame name.

```
# Add a heatmap of temperature for the vacation spots.
locations = hotel_df[["Lat", "Lng"]]
max_temp = hotel_df["Max Temp"]
fig = gmaps.figure(center=(30.0, 31.0), zoom_level=1.5)
heat_layer = gmaps.heatmap_layer(locations, weights=max_temp, dissipating=Fa
             max_intensity=300, point_radius=4)

fig.add_layer(heat_layer)
# Call the figure to plot the data.
fig
```

For our `hotel_df` DataFrame, the heatmap looks like the following.



If we refer to the **gmaps documentation on how to add markers (https://jupyter-gmaps.readthedocs.io/en/latest/tutorial.html#markers-and-**

symbols)_, the syntax is `markers = gmaps.marker_layer(marker_locations)`, where the `marker_locations` are latitudes and longitudes.

Let's add the markers for each city on top of the heatmap. Edit the code we used to create the heatmap to look like the following:

```
# Add a heatmap of temperature for the vacation spots and marker for each ci
locations = hotel_df[["Lat", "Lng"]]
max_temp = hotel_df["Max Temp"]
fig = gmaps.figure(center=(30.0, 31.0), zoom_level=1.5)
heat_layer = gmaps.heatmap_layer(locations, weights=max_temp,
             dissipating=False, max_intensity=300, point_radius=4)
marker_layer = gmaps.marker_layer(locations)
fig.add_layer(heat_layer)
fig.add_layer(marker_layer)
# Call the figure to plot the data.
fig
```

The map in the output will look similar to the following:

Now we can add a pop-up marker for each city that displays the hotel name, city name, country, and maximum temperature.

From the **gmaps documentation on how to add markers (https://jupyter-gmaps.readthedocs.io/en/latest/tutorial.html#markers-and-symbols)**, we need to add an `info_box_template` that has the following syntax.

```
info_box_template = """
<dl>
<dt>Name</dt><dd>{column1}</dd>
<dt>Another name</dt><dd>{column2}</dd>
</dl>
"""
```

This is a new code block, so let's review its form and function.

First, the `info_box_template` variable is assigned to a multiline string using three quotes. The text inside the multiline string is HTML code. **HTML code** is defined by the opening and closing the angular brackets (e.g., `<tag>` and `<tag/>`). Angular brackets always come in pairs. The opening angular bracket is followed by some text inside, such as `dl`, `dt`, and `dd`. The closing angular bracket is preceded by a forward-slash ("/"). The text inside with the angular brackets is called a **tag**. We'll see more examples of HTML code in an upcoming module.

Here's what these tags mean:

- The `<dl>` tag is a description list (dl).
- The `<dt>` tag is a term or name in a description list that is nested under the `<dl>` tag.
- The `<dd>` tag is used to define the term or name or `<dt>` tag.

If we were to write out these tags on paper, it would look like this.

- Description List: `<dl>`
    - Description Term: `<dt>`
        - Description Definition: `<dd>`

- Description Term: `<dt>`
  - Description Definition: `<dd>`

For our purposes, we'll add the hotel name, city name, country code, and the maximum temperature values from the `hotel_df` DataFrame as the description definition. Our code will look like the following.

```
info_box_template = """
<dl>
<dt>Hotel Name</dt><dd>{Hotel Name}</dd>
<dt>City</dt><dd>{City}</dd>
<dt>Country</dt><dd>{Country}</dd>
<dt>Max Temp</dt><dd>{Max Temp} °F</dd>
</dl>
"""
```

Next, add the data to the code by iterating through the `hotel_df` DataFrame using the `iterrows()` function. Then add the information to the `gmaps.marker_layer()` attribute with the locations.

According to the **documentation (https://jupyter-gmaps.readthedocs.io/en/latest/tutorial.html#markers-and-symbols)**, we can use this example to create the data for the `info_box_template`:

From the documentation, we can take this code:

`plant_info = [info_box_template.format(**plant) for plant in nuclear_power_plants]`

Then edit this code by replacing "plant_info" with "hotel_info," and then use the `iterrows()` function to get the index and data in the row to add to the marker.

Add the following code below to the `info_box_template` code and run the cell.

```
# Store the DataFrame Row.
hotel_info = [info_box_template.format(**row) for index, row in hotel_df.ite
```

Let's review what we're doing with this code.

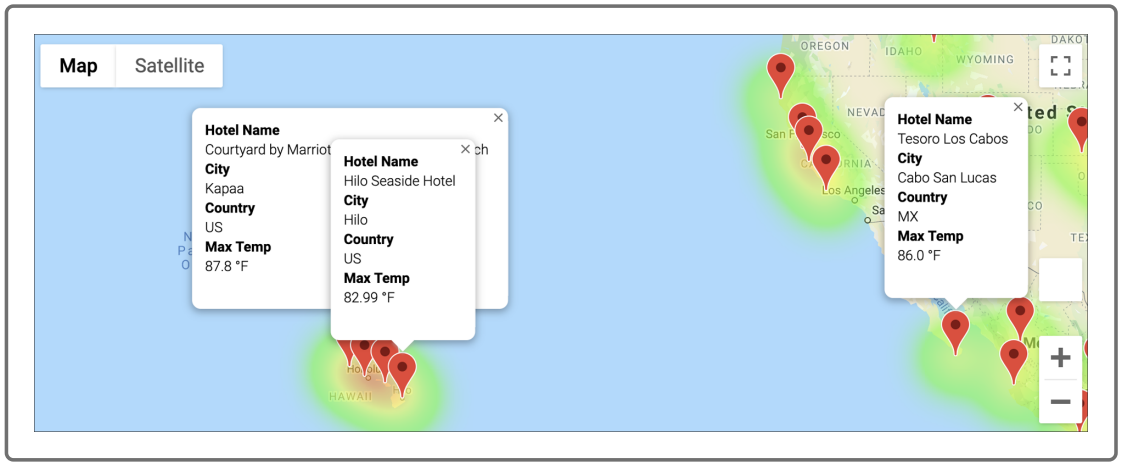
1. We set the `hotel_info` equal to the info_box_content.
2. In the list comprehension, `info_box_template.format(**row) for index, row in hotel_df.iterrows()`, we iterate through each "row" of the `hotel_df` DataFrame and then format the `info_box_template` with the data we set to populate the from each row. Remember, we are not using every row; we are only using the rows defined in the `info_box_template`, which are `Hotel Name`, `City`, `Country`, and `Max Temp`.

Next, in the code we used to create the heatmap with markers, add `info_box_content=hotel_info` to the `gmaps.marker_layer()` attribute with the locations. Our final cell should look like the following.

```
# Add a heatmap of temperature for the vacation spots and a pop-up marker fo
locations = hotel_df[["Lat", "Lng"]]
max_temp = hotel_df["Max Temp"]
fig = gmaps.figure(center=(30.0, 31.0), zoom_level=1.5)
heat_layer = gmaps.heatmap_layer(locations, weights=max_temp,dissipating=Fal
             max_intensity=300, point_radius=4)
marker_layer = gmaps.marker_layer(locations, info_box_content=hotel_info)
fig.add_layer(heat_layer)
fig.add_layer(marker_layer)

# Call the figure to plot the data.
fig
```

When your cell looks like the code above, run the cell. The map in the output cell will look similar to the following map when a marker is clicked.



Congratulations on adding pop-up markers to your heatmap!

ADD/COMMIT/PUSH

Add your `VacationPy.ipynb` file to your World_Weather_Analysis GitHub repository.