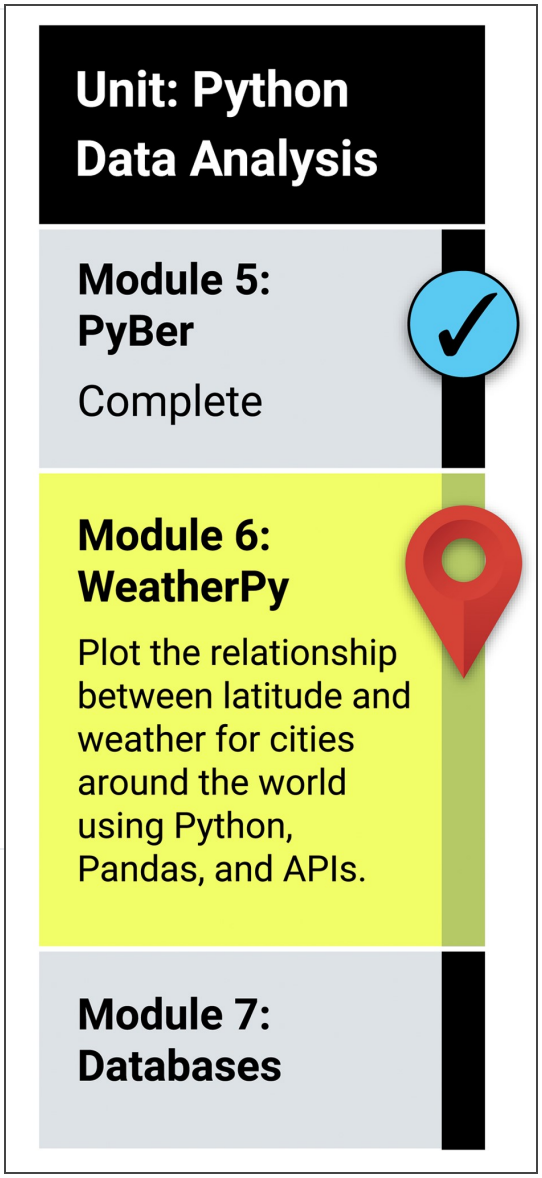
6.0.1: Using APIs to Visualize Weather Data

6.0.2: Module 6 Roadmap

Looking Ahead

In this module, you'll practice your analysis, visualization, and statistical skills by retrieving and analyzing weather data for a hypothetical travel company, PlanMyTrip. Successfully completing the tasks will draw on your knowledge of Python, decision and repetition statements, data structures, Pandas, Matplotlib, and SciPy statistics.



What You Will Learn

By the end of this module, you will be able to:

- Perform tasks using new Python libraries and modules.

- Retrieve and use data from an API “get” request to a server.
- Retrieve and store values from a JSON array.
- Use `try` and `except` blocks to resolve errors.
- Write Python functions.
- Create scatter plots using the Matplotlib library, and apply styles and features to a plot.
- Perform linear regression, and add regression lines to scatter plots.
- Create heatmaps, and add markers using the Google Maps API.

Planning Your Schedule

Here’s a quick look at the lessons and assignments you’ll cover in this module. You can use the time estimates to help pace your learning and plan your schedule.

- Introduction to Module 6 (1 hour)
- Generate Random Coordinates of World Cities (1–2 hours)
- Retrieve, Collect, and Clean Weather Data (1–2 hours)
- Plot Weather Data (1–2 hours)
- Determine Correlations (2–3 hours)
- Use a Google API to Create Heatmaps (2–3 hours)
- Application (5 hours)

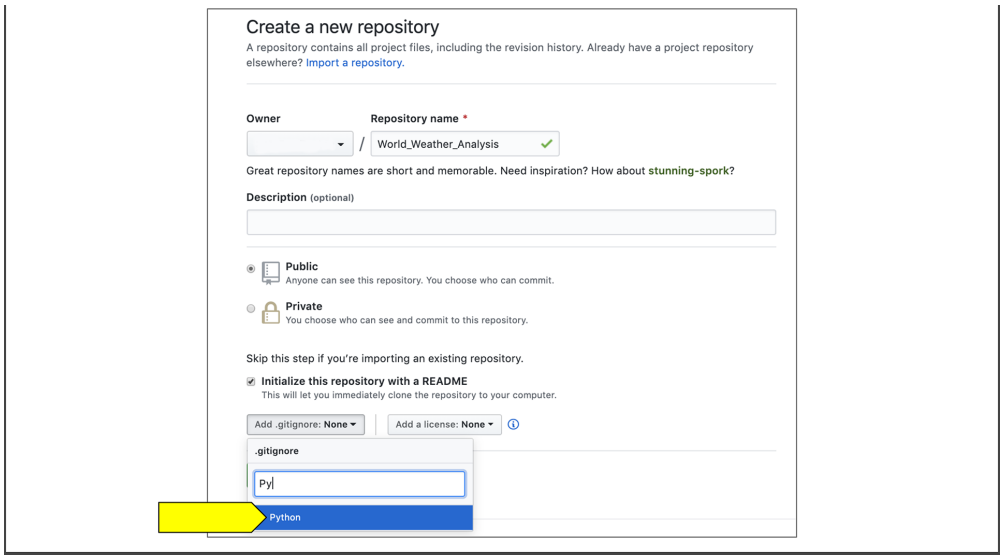
6.0.3: Welcome to APIs and World Wide Weather

6.1.1: Create and Clone a New GitHub Repository

Jack's email was a bit mysterious—something about a kickoff meeting for a fun new project on world weather analysis that he wanted help with. But he's always come up with complex, rewarding challenges before, so you know you want to be at that meeting. As you accept the invitation, you decide to get a bit ahead of the game and create your GitHub repository.

Before we get started, create a GitHub repository for your project and clone it onto your work computer.

1. Log in to your GitHub account.
2. Create a "New repository."
3. Name the repository for this module, "World_Weather_Analysis".
4. Make the repository public.
5. Check the box that states "Initialize the repository with a README."
6. Next, we'll add a `.gitignore` file to the repository. GitHub does not track files or files with extensions that are added to the `.gitignore` file.
7. Click "Add .gitignore" and begin typing "Python", and then select Python as the option.



8. Click the green “Create repository” box.
9. You should see the `.gitignore` file in the repository. Later in this module we’ll edit the `.gitignore` file.

10. On the repository’s main page, click the green button (top right) “Clone or download.”
11. Select “Clone with HTTPS” and click the clipboard icon to copy the clone URL for the repository.
12. Open the command line (macOS) or Git Bash (Windows).
13. Navigate to the Class folder.
14. Once in the Class folder, type `git clone` followed by a space, and then paste the URL that you copied.
15. Press Enter.

6.1.2: Overview of the Project

Good thing you got your repo set up and out of the way prior to that kickoff meeting. This project is going to be no joke, but it should be a lot of fun.

At the most fundamental level, Jack needs help answering a question: How might we provide real-time suggestions for our client's ideal hotels? Your first task was to define what you meant by "ideal." So, over the course of the conversation, you narrowed that to hotels that were (1) within a given range of latitude and longitude and that (2) provided the right kind of weather for the client.

When you get back to your desk, your first step is to write out a basic plan for how you'll write code that can do this fairly complex task.

Basic Project Plan

Here's an outline of your project plan:

- **Task:** Collect and analyze weather data across cities worldwide.
- **Purpose:** PlanMyTrip will use the data to recommend ideal hotels based on clients' weather preferences.
- **Method:** Create a Pandas DataFrame with 500 or more of the world's unique cities and their weather data in real time. This process will entail collecting, analyzing, and visualizing the data.

Your analysis of the data will be split into three main parts, or stages.

1. **Collect the Data**
    - Use the NumPy module to generate more than 1,500 random latitudes and longitudes.
    - Use the citipy module to list the nearest city to the latitudes and longitudes.
    - Use the OpenWeatherMap API to request the current weather data from each unique city in your list.
    - Parse the JSON data from the API request.
    - Collect the following data from the JSON file and add it to a DataFrame:
        - City, country, and date
        - Latitude and longitude
        - Maximum temperature
        - Humidity
        - Cloudiness
        - Wind speed

2. **Exploratory Analysis with Visualization**
    - Create scatter plots of the weather data for the following comparisons:
        - Latitude versus temperature

        - Latitude versus humidity
        - Latitude versus cloudiness
        - Latitude versus wind speed
    - Determine the correlations for the following weather data:
        - Latitude and temperature
        - Latitude and humidity
        - Latitude and cloudiness
        - Latitude and wind speed
    - Create a series of heatmaps using the Google Maps and Places API that showcases the following:
        - Latitude and temperature
        - Latitude and humidity
        - Latitude and cloudiness
        - Latitude and wind speed

3. **Visualize Travel Data**

   Create a heatmap with pop-up markers that can display information on specific cities based on a customer's travel preferences. Complete these steps:

   1. Filter the Pandas DataFrame based on user inputs for a minimum and maximum temperature.
   2. Create a heatmap for the new DataFrame.
   3. Find a hotel from the cities' coordinates using Google's Maps and Places API, and Search Nearby feature.
   4. Store the name of the first hotel in the DataFrame.

5. Add pop-up markers to the heatmap that display information about the city, current maximum temperature, and a hotel in the city.

6.1.3: Review the Geographic Coordinate System

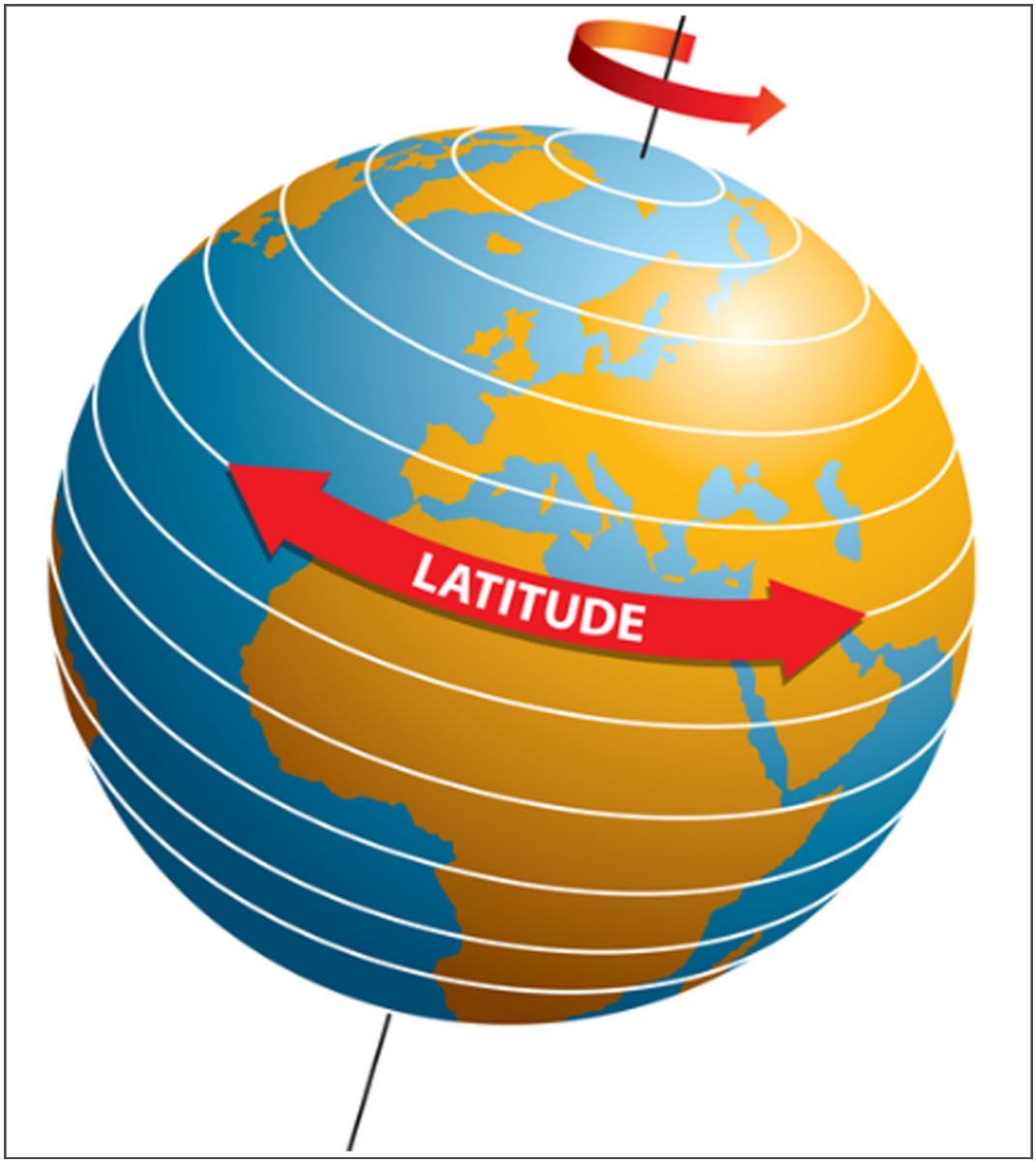
Your project planning document is prepared, approved, and pinned above your computer. You're ready to get going on the first step: creating a list of over 1,500 latitudes and longitudes.

But wait. Which one runs north to south? Latitude or longitude? And which one runs east to west? And how exactly do these work, again? As a professional in the world of data, you are no stranger to employing Google-fu when faced with a problem, so you decide to do a quick self-taught review.

> We use the **geographic coordinate system** (GCS) to reference any point on Earth by its latitude and longitude coordinates.
>
> **Latitudes** are imaginary lines on Earth that run parallel east to west and are measured in angular units called degrees, minutes, and seconds, with 60 minutes in a degree and 60 seconds in a minute. Sometimes a latitude is referred to as a **parallel**. Consider, for example, the embattled 38th

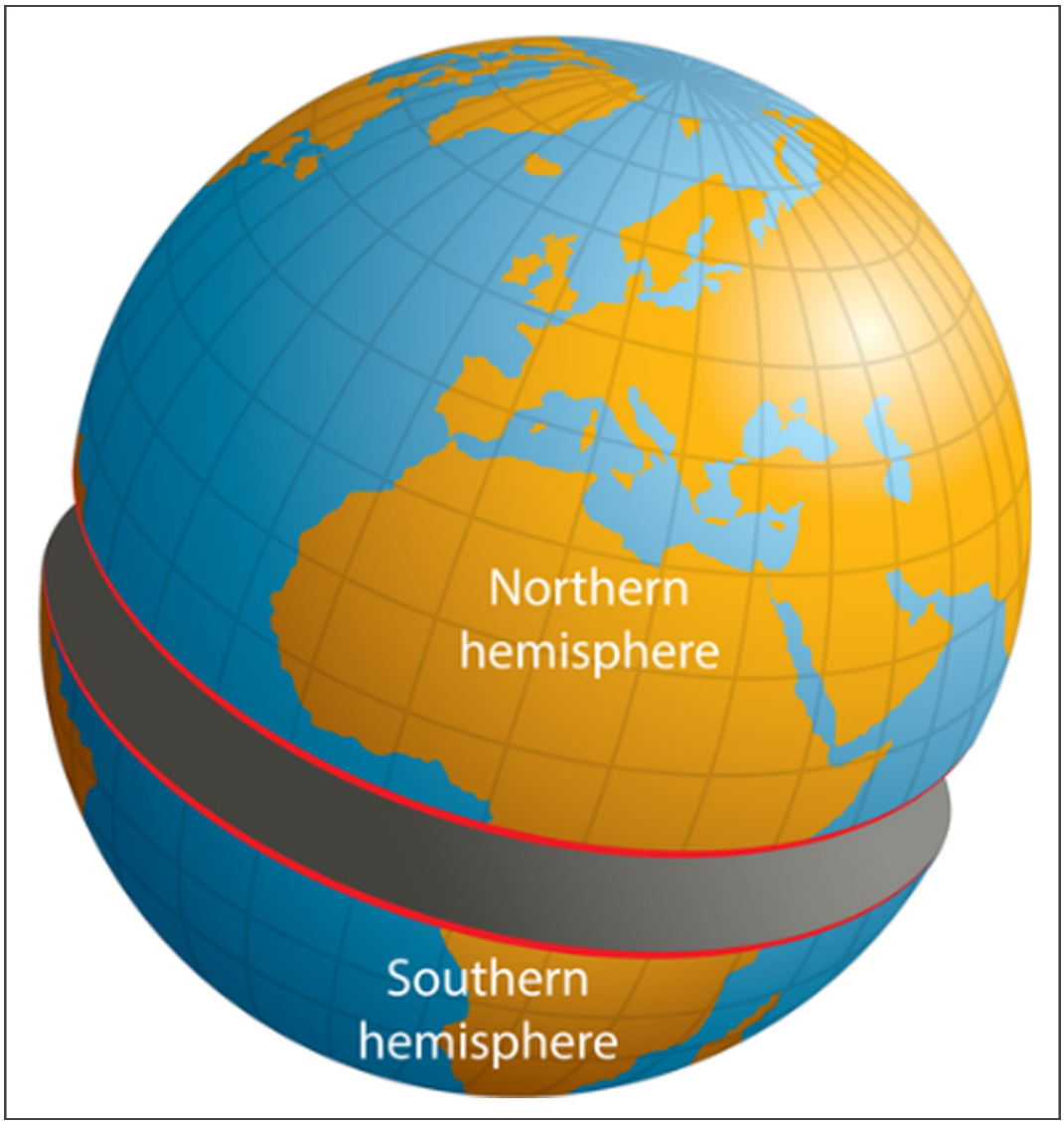
parallel (38° north) in East Asia that roughly demarcates North Korea and South Korea.

The **equator** is an imaginary line around the middle of the earth that is equidistant from the North and South Poles and has a latitude of 0°. The equator splits Earth into Northern and Southern Hemispheres.

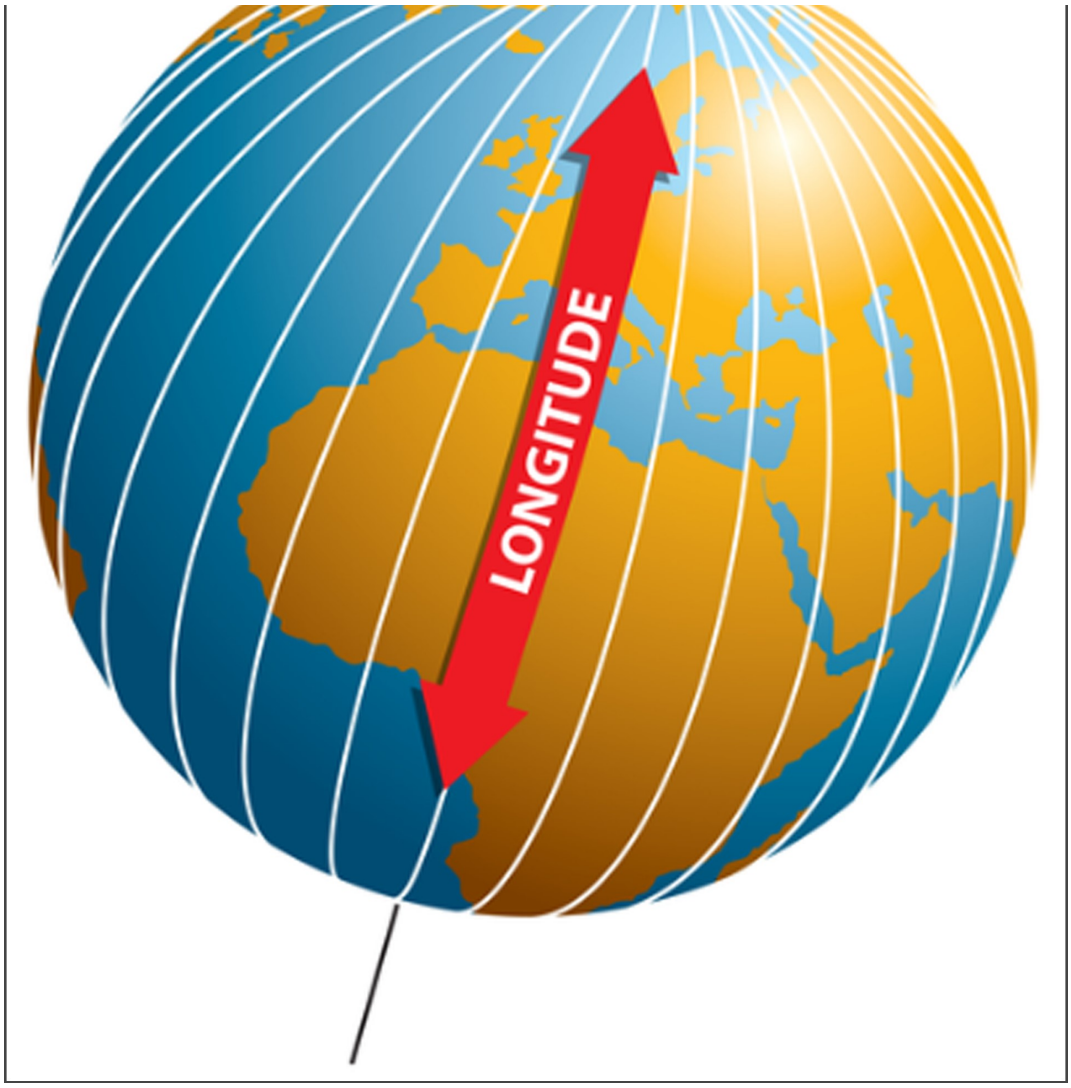


All latitude lines above the equator are measured northward and considered positive, after 0° (the equator) and up to 90°, or 90° north (the North Pole). All latitude lines below the equator are measured southward

and considered negative, before 0° (the equator) and down to −90°, or 90° south (the South Pole).

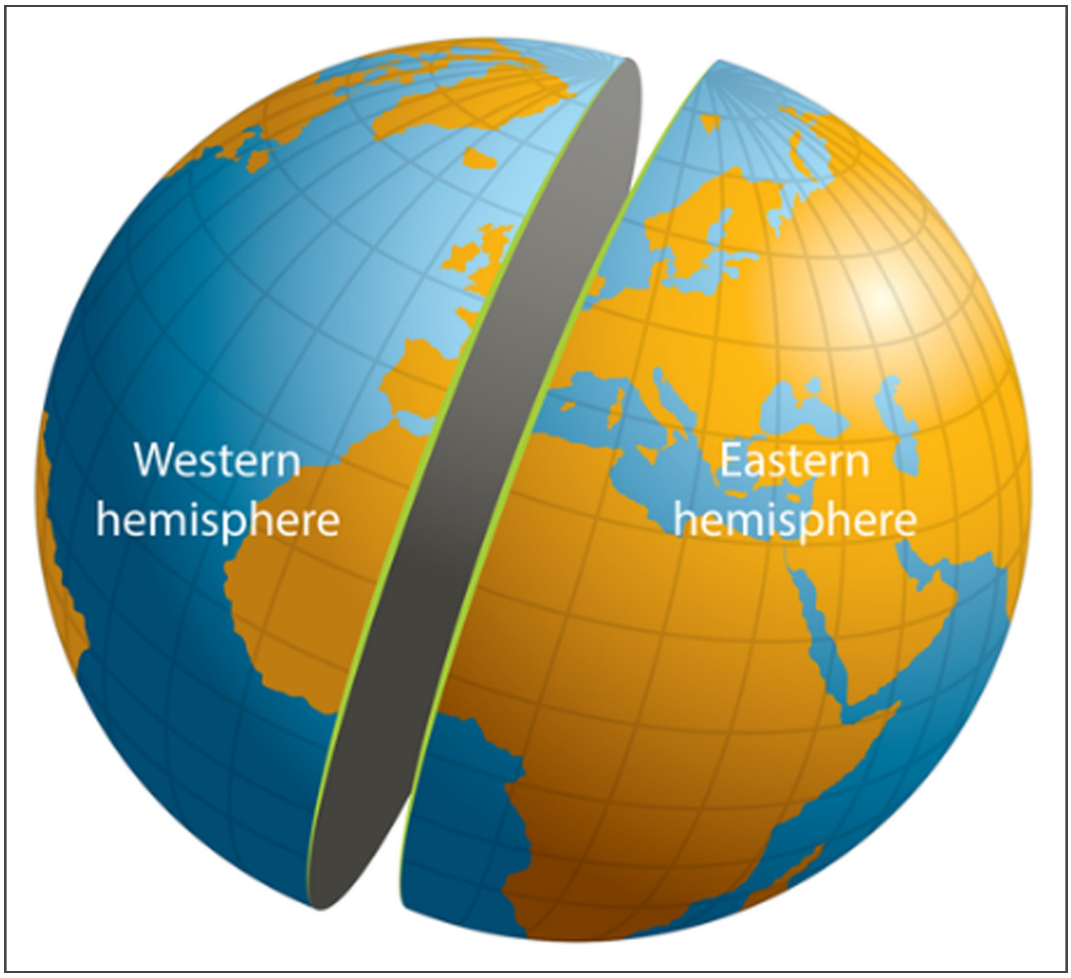


Longitudes are imaginary lines on Earth that run from the North to the South Poles and are called **meridians**. The **prime meridian** represents zero meridian, the origin for longitude coordinates, and splits Earth into the Eastern and Western Hemispheres.



The prime meridian passes through Greenwich, England, from which longitude east and west is measured.

All meridians east of the prime meridian are considered positive, after 0° and up to 180°. All meridians west of the prime meridian are considered negative, before 0° and down to –180°.



All together, the lines of latitude (parallels) and longitude (meridians) make up a geographic grid, as if the Earth were wrapped in graph paper with intersecting horizontal and vertical lines mapping to specific locations.

GCS makes it possible to pinpoint any place on Earth by providing its precise address, which is the intersection of its latitude and longitude lines.

Now, after having our refresher course on GCS, let’s generate random latitudes and longitudes and retrieve the nearest city to those coordinates.

6.1.4: Generate Random Latitudes and Longitudes

Latitude and longitude—got it. Now it's time to code. You know you want to generate enough coordinates that you'll have a good covering of common travel destinations, so you'll need coordinates that are close to a city.

For the development phase of the project, you and Jack decide that about 500 cities should be enough. Of course, if the algorithm you're about to write works, you'll be able to use and reuse it to get many more cities.

> Before we write the algorithm to generate the latitudes and longitudes, we need to refresh our memory of where people live in the world. Time for another quick geography lesson!
>
> Earth's surface is covered by 70% water while the rest is covered by land. So, we can assume 70% of the latitude and longitude coordinates we generate are positioned over a body of water, whether an ocean, major lake (e.g., Lake Superior), or major river (e.g., Amazon). Geographic coordinates over a body of water may not be close to a city, especially if in the middle of an ocean.

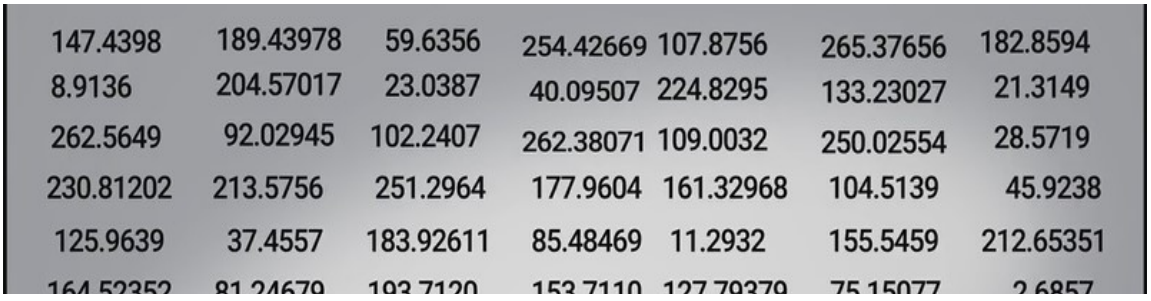
Seven continental landmasses comprise 30% of Earth's surface. Some land is uninhabitable or sparsely populated due to extreme terrain and climes (e.g., Sahara, Siberia, the Himalayas, and areas of the western United States).

First consider the bodies of water. Start with at least 1,500 latitudes and longitudes, because 500 divided by 0.3 (30% land mass) equals 1,666 latitudes and longitudes.

We'll generate random latitudes and longitudes to ensure coordinates are fairly distributed around the world. An algorithm will pick random numbers between the low and high values for latitudes and longitudes. Also, the latitudes and longitudes must be floating-point decimal numbers, as each angular unit of degrees, minutes, and seconds can be represented by a decimal number. For example, Kailua-Kona, Hawaii has the angular coordinates 19° 38' 23.9784" north and 155° 59' 48.9588" west and can be written as a decimal number as follows: 19.639994, -155.996933.

To generate random numbers, we can use the Python `random` module. This module is part of the Python and Anaconda installation, so we don't need to install it. Let's test some `random` module functions to find one that can help us.

The random Module

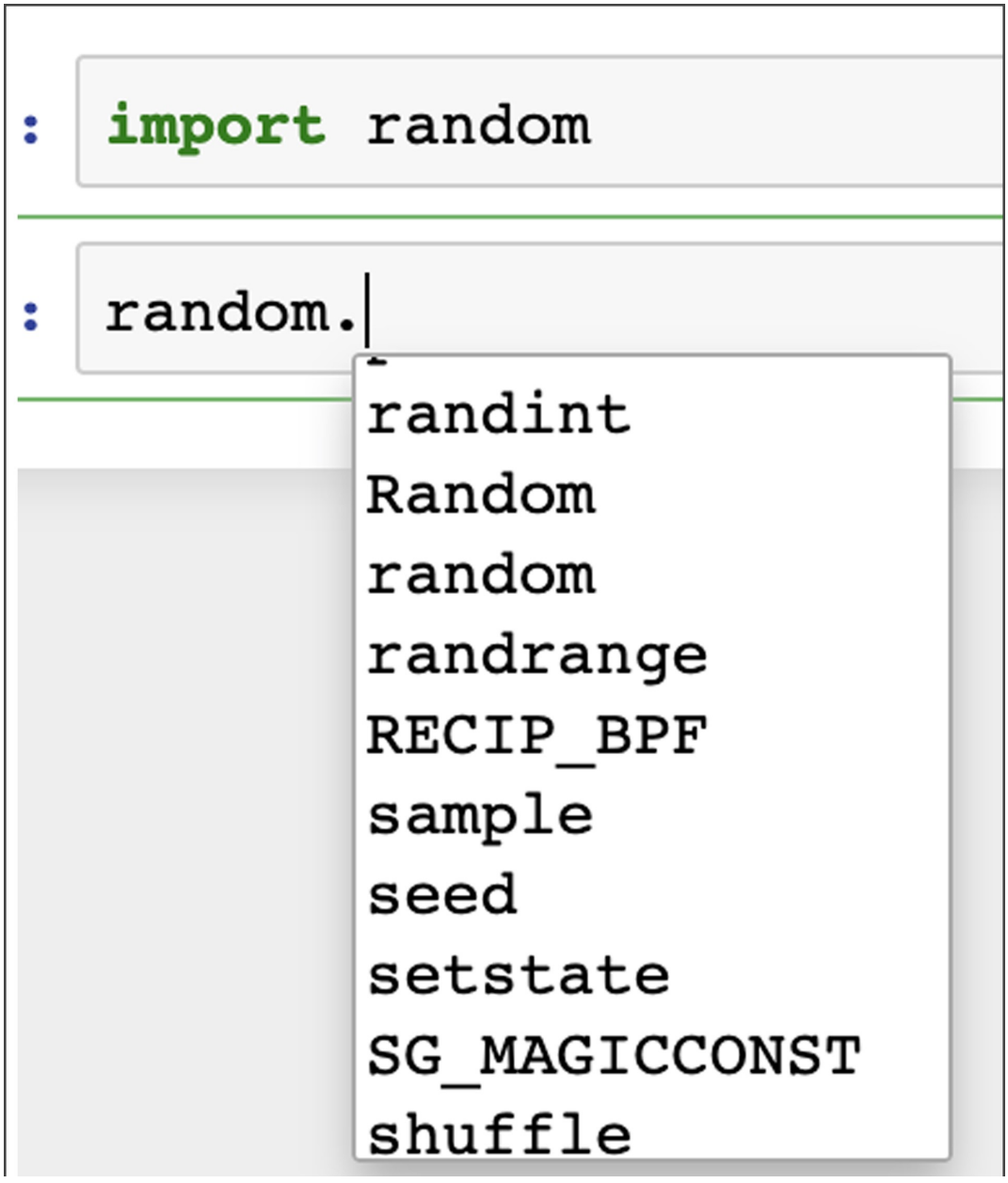


First, create a new Jupyter Notebook file named `random_numbers.ipynb`.

In the first cell, import the `random` module and run the cell.

```
# Import the random module.
import random
```

In the next cell, type `random.`, and after the period, press the Tab key for a list of available `random` module functions.



For testing, we'll use the `randint()`, `random()`, `randrange()`, and `uniform()` functions.

The randint() Function

`randint` is short for "random integer." In the second cell, after `random.`, type `randint(-90, 90)`, as shown below.

```
random.randint(-90, 90)
```

When we run this cell, we'll get a single integer between −90 and 90 because we need two latitudes between −90 and 90.

```
random.randint(-90, 90)
```

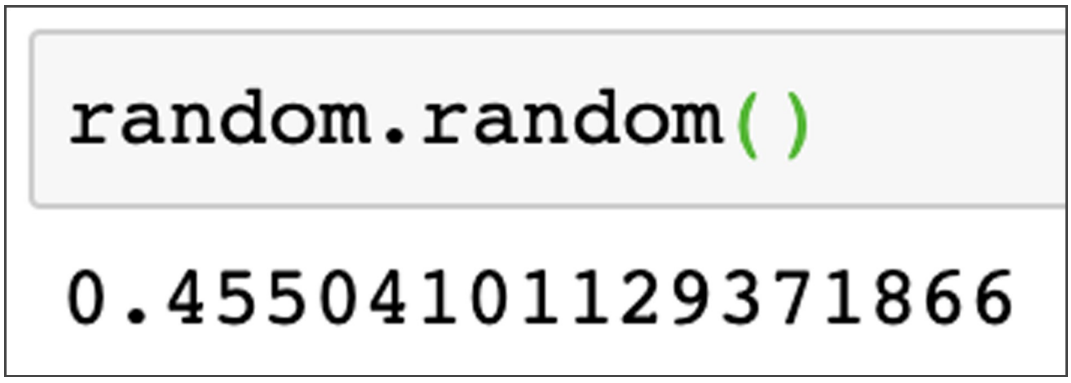
```
-66
```

This is useful information, but it doesn't get the job done for us. Remember, we need 1,500 random decimal numbers. This function will only return one integer, not a floating-point decimal, between the given intervals. Let's try the `random()` function.

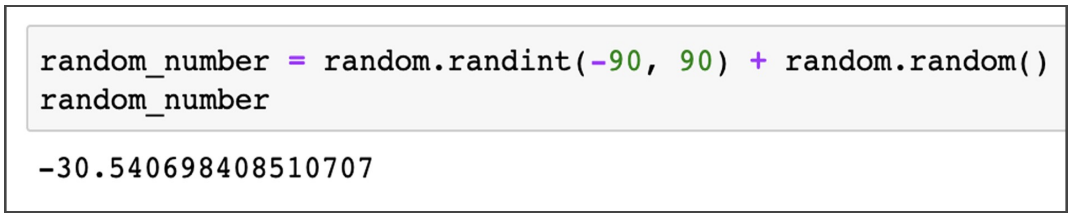
The random() Function

Using the `random()` function, we can get a single floating-point decimal number between 0 and 1.0.

Add `random.random()` to a new cell and run the cell. Your output should be a decimal point number between 0 and 1.0, as shown below.



The `random()` function may help us. This function returns only a floating-point decimal number between 0 and 1. If we combined `random.randint(-90, 90)` and `random.random()` to generate a floating-point decimal between −90 and 90, we can generate a random latitude.



Using these two functions, we can write an algorithm that will generate latitudes between -90 and 90. Here is a small sample of what it might take to generate ten random floating-point decimal latitudes between −90 and 90.

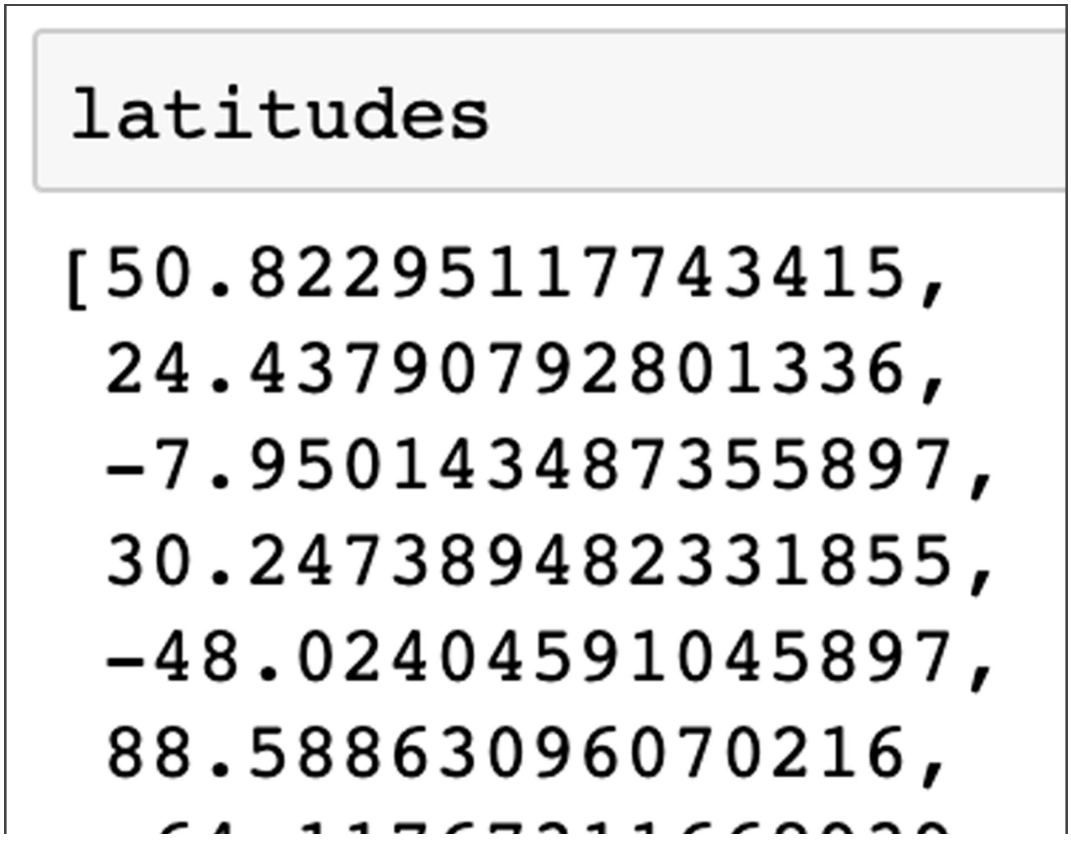
```
x = 1
latitudes = []
```

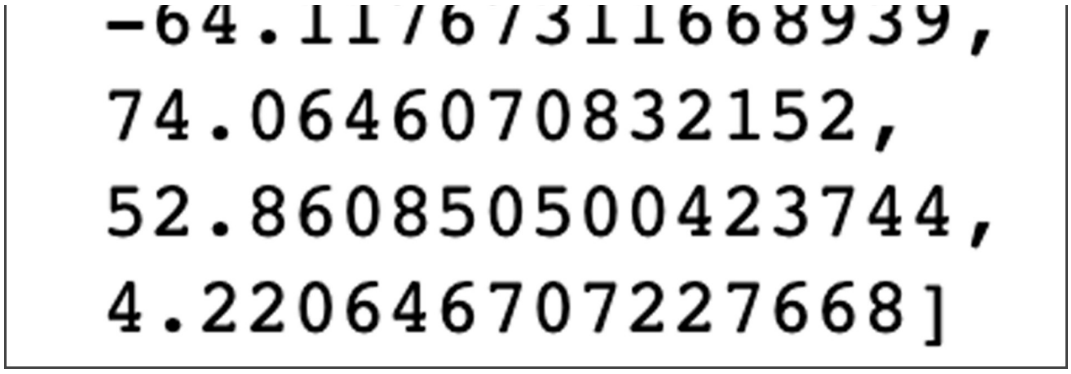
```
while x < 11:
    random_lat = random.randint(-90, 90) + random.random()
    latitudes.append(random_lat)
    x += 1
```

In the code block above, we:

1. Assign the variable x to 1.
2. Initialize an empty list, `latitudes`.
3. We create a while loop where we generate a random latitude and add it to the list.
4. After the random latitude is added to the list we add one to the variable “x”.
5. The while loop condition is checked again and will continue to run as long as x is less than 11.

The output from running this code might look like this:





Next, we would have to use a similar method to get random longitudes between 180 and -180 and pair them with the latitudes. This looks promising but the code to generate these latitude is a little long.

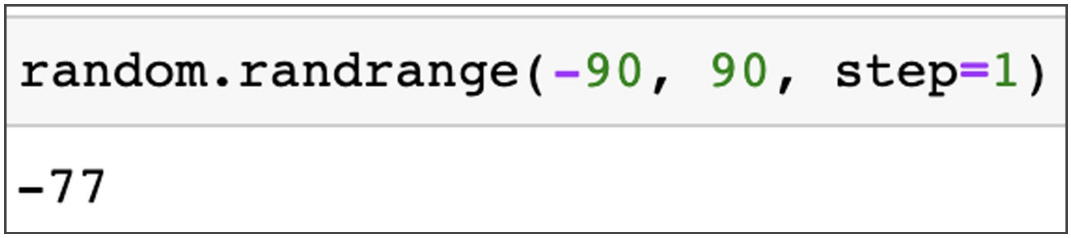
Let's try another function, the `randrange()` function.

The randrange() Function

The `randrange()` function behaves differently than the previous two functions. Inside the parentheses, we need to add two numbers, a lower and upper limit, separated by a comma.

For the `randrange()` function, there is an option to add a `step` parameter and set it equal to an integer, which will generate increments of a given integer value, from the lower to the upper limit.

For example, add `random.randrange(-90, 90, step=1)` to a new cell and run the cell. The output is a number between -90 and 90, where the step is the difference between each number in the sequence.



Now add `random.randrange(-90, 90, step=3)` to a new cell and run the cell. The output is a number between -90 and 90, where the difference between each number in the sequence is 3.

```
random.randrange(-90, 90, step=3)
```

```
72
```

> **Note**
> If you don't add the step parameter, the output will be a number with an increment of 1, which is the default integer value.

This function might help us by combining the `random.randrange()` and `random.random()` functions to generate a floating-point decimal between –90 and 90, like we did with the `random.randint()` and `random.random()` functions.

Let's look at one last function, the `uniform()` function.

The uniform() Function

The `uniform()` function will allow us to generate a floating-point decimal number between two given numbers inside the parentheses.

Add `random.uniform(-90, 90)` to a new cell and run the cell. The output should look like the following:

```
random.uniform(-90, 90)
```

```
-16.992740452265 65
```

The `uniform()` function could prove to be quite useful because it will return a floating-point decimal number! The table below reviews the functions' outputs and limitations:

Function	Output	Limitation
`randint(-90, 90)`	Returns a whole integer between the interval, -90 and 90.	Will not generate a floating-point decimal number.
`random()`	Returns a floating-point decimal number between 0 and 1.	Will not generate a whole integer.
`randrange(-90, 90, step=1)`	Returns a whole integer between the interval, -90 and 90 where the step is the difference between each number in the sequence.	Will not generate a floating-point decimal number.
`uniform(-90, 90)`	Returns a floating-point decimal	Will not generate a whole integer.

	number between the interval, -90 and 90.

Remember, we need to get more than a thousand latitudes and longitudes, and running one of these functions using a while loop or other methods may take more programming than needed.

To help us generate the 1500 latitudes and longitudes, we can combine the NumPy module with one of the random module functions.

The NumPy and random Modules

One way to generate more than a thousand latitudes and longitudes is to chain the NumPy module to the random module to create an array of latitudes or longitudes between the lowest and highest values, or –90° and 90°, and –180° and 180°, respectively. To accomplish this, we'll use the `uniform()` function from the random module.

> **Rewind**
> Recall that the NumPy module is a numerical mathematics library that can be used to make arrays or matrices of numbers.

Let's import the NumPy module in a new cell and run the cell.

```
# Import the NumPy module.
import numpy as np
```

> **Note**
> The NumPy module has a built-in random module, and supplements the built-in Python random module. There is no need to import the random module if we import the NumPy module, as it's redundant.

In the next cell add `np.random.uniform(-90.000, 90.000)` to generate a floating-point decimal number between −90.000 and 90.000. Adding the zeros past the decimal places is optional.

```
np.random.uniform(-90.000, 90.000)
```

```
44.43699159272211
```

When we use the NumPy module with the `random.uniform()` function, the parenthetical parameters contain a lower boundary (low value) and an upper boundary (high value) that are floating-point decimal numbers.

> **Note**
> Another option is to write the parameters as `np.random.uniform(low=-90, high=90)`.

To generate more than one floating-point decimal number between −90 and 90, we can add the `size` parameter when we use the NumPy module and set that equal to any whole number.

To see how this works, add the code `np.random.uniform(-90.000, 90.000, size=50)` to a new cell and run the cell. The output is an array of 50 floating-point decimal numbers between −90.000 and 90.000.

```
np.random.uniform(-90.000, 90.000, size=50)
```

```
array([ 17.96687837,  14.36195784,  69.32809477, -55.49345622,
         1.75422576,  43.05565827, -71.77162584, -14.42567115,
        51.79523949,  43.23458966, -41.37785462, -23.62406857,
       -13.37855367,  -6.8691874 , -50.84192853,  87.28360625,
       -60.18972261,  73.24902693,  41.48800538, -63.44483297,
        73.32733193, -68.74690001,  70.15217544, -55.02030612,
        50.58837816, -11.19406658,  61.59836396,  38.76352194,
        34.70962514,  82.55273652,  30.97675505, -47.36772117,
        10.59141067, -17.46764942, -26.84302643,  12.80951266,
        -4.06464157, -72.06059609,   3.29231855, -71.50703124,
        87.93813796, -81.63540186,  72.92436787,  15.4066607 ,
        55.84949676, -74.57043875,  74.93885592, -39.98419312,
        75.70113548, -25.56786225])
```

Now we are getting somewhere—all we need to do is increase the parameter size to 1,500.

Is this method faster than creating a while loop like we did before? Let's test this for a size of 1,500.

To test how long a piece of code or function takes to run, we can import the "timeit" module and use the `%timeit` magic command when we run our code or call the function.

First, import the timeit module in a new cell, and run the cell.

```
# Import timeit.
import timeit
```

Next, add the `%timeit` magic command before the `np.random.uniform(-90.000, 90.000, size=1500)` in a new cell. The cell should look like this:

```
%timeit np.random.uniform(-90.000, 90.000, size=1500)
```

When we run the cell, the output is the amount of time it took to run the code for 7 runs and 1,000 loops per run.

```
%timeit np.random.uniform(-90.000, 90.000, size=1500)
```

```
14.6 µs ± 526 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

The output is the amount of time it took to run this code, which is an average of 14.6 microseconds. The amount of time it takes to run this code depends on the processing speed and the RAM of your computer.

Now, let's run the `while` loop as a function. Copy the following code in a new cell and run the cell.

```
def latitudes(size):
    latitudes = []
    x = 0
    while x < (size):
        random_lat = random.randint(-90, 90) + random.random()
        latitudes.append(random_lat)
        x += 1
    return latitudes
# Call the function with 1500.
%timeit latitudes(1500)
```

The output is 1.45 milliseconds.

```
# Call the function with 1500.
%timeit latitudes(1500)
```

```
1.45 ms ± 5.39 us per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Using the `np.random.uniform(-90.000, 90.000, size=1500)` is 100 times faster than using the function, and our code is one line, whereas the function uses eight lines!

> **Skill Drill**
>
> Refactor the code for the while loop with the `%timeit` magic command and write a for loop that will generate the 1,500 latitudes.

Create Latitude and Longitude Combinations

Let's apply our new knowledge to this project. Create a new Jupyter Notebook file called

`WeatherPy.ipynb`, import the Pandas, Matplotlib, and NumPy dependencies in the first cell, and run the cell.

```
# Import the dependencies.
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In the next cell, we'll add the code that generates the latitudes and longitudes, but first, they need to be stored so that we can access them later. Since we are creating arrays of latitudes and longitudes, we'll declare each array as a variable.

In the next cell, add the following code that we used to generate the random latitudes. Also, we'll create a similar code snippet that will generate longitudes. To ensure enough latitudes and longitudes, we'll start with 1,500. In addition, we'll pack the latitudes (`lats`) and

longitudes (`lngs`) as pairs by zipping them (`lat_lngs`) with the `zip()` function.

```
# Create a set of random latitude and longitude combinations.
lats = np.random.uniform(low=-90.000, high=90.000, size=1500)
lngs = np.random.uniform(low=-180.000, high=180.000, size=1500)
lat_lngs = zip(lats, lngs)
lat_lngs
```

When we run this cell, the output is a zip object in memory.

```
# Create a set of random latitude and longitude combinations.
lats = np.random.uniform(low=-90.000, high=90.000, size=1500)
lngs = np.random.uniform(low=-180.000, high=180.000, size=1500)
lat_lngs = zip(lats, lngs)
lat_lngs
```

```
<zip at 0x113dc66c8>
```

The zip object packs each pair of `lats` and `lngs` having the same index in their respective array into a tuple. If there are 1,500 latitudes and longitudes, there will be 1,500 tuples of paired latitudes and longitudes, where each latitude and longitude in a tuple can be accessed by the index of 0 and 1, respectively.

Let's practice zipping a small number of latitudes and longitudes and then unpacking the zipped tuple to see how the packing and unpacking work.

In a new Jupyter Notebook file called `API_practice.ipynb`, add the following lists and pack them into the zipped tuple. Then, run the cell.

```
# Create a practice set of random latitude and longitude combinat
x = [25.12903645, 25.92017388, 26.62509167, -59.98969384, 37.3057
y = [-67.59741259, 11.09532135, 74.84233102, -76.89176677, -61.13
coordinates = zip(x, y)
```

In a new cell, display the coordinate pairs with the following code.

```
# Use the tuple() function to display the latitude and longitude
for coordinate in coordinates:
    print(coordinate[0], coordinate[1])
```

When we run the cell above, the output is ordered pairs of our x and y coordinates.

```
25.12903645 -67.59741259
25.92017388 11.09532135
26.62509167 74.84233102
-59.98969384 -76.89176677
37.30571269 -61.13376282
```

Next, let's unpack our `lat_lngs` zip object into a list. This way, we only need to create a set of random latitudes and longitudes once. In a new cell in the `WeatherPy.ipynb` file, add the following code and run the cell.

```
# Add the latitudes and longitudes to a list.
coordinates = list(lat_lngs)
```

Now that we have our ordered pairs of latitudes and longitudes in a list, we can iterate through the list of tuples and find the nearest city to those coordinates.

> **NOTE**

> For more information, see the **documentation on numpy.random.uniform() (https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.random.uniform.html)** .

6.1.5: Generate Random World Cities

You lean back from your desk and reflect on the project thus far. Not only have you successfully navigated a quick return to the geographic coordinate system and the Earth's geography, you also wrote code to generate 1,500 latitudes and longitudes. Now, as cool as this is, your clients aren't going to want information or suggestions provided to them in this format. So, it's time to get started on the next step in your project plan: match those coordinates up with cities.

We are making great progress. With our list of random latitudes and longitudes, we'll use the coordinates in our `lat_lngs` tuple to find the nearest city using Python's citipy module.

Since we haven't worked with the citipy module yet, let's import and test it. Citipy doesn't come with the Anaconda module, so we'll install it in our PythonData environment.

The **citypy documentation (https://pypi.org/project/citipy/)** instructs us to install the citipy module by typing `pip install citipy`. To complete the installation, follow the instructions for your operating system.

---

macOS

To install the citipy module on macOS, complete the following steps:

1. Click the clipboard icon to copy `pip install citipy`.
2. Launch the command line and make sure you are in your PythonData environment. You should see the following in your terminal:

```
(PythonData) your_computer_name:~ your_home_directory$
```

3. Paste `pip install citipy` and press Enter.

The citipy module will probably take a few minutes to download into your PythonData environment.

Windows

To install the citipy module on Windows, complete the following steps:

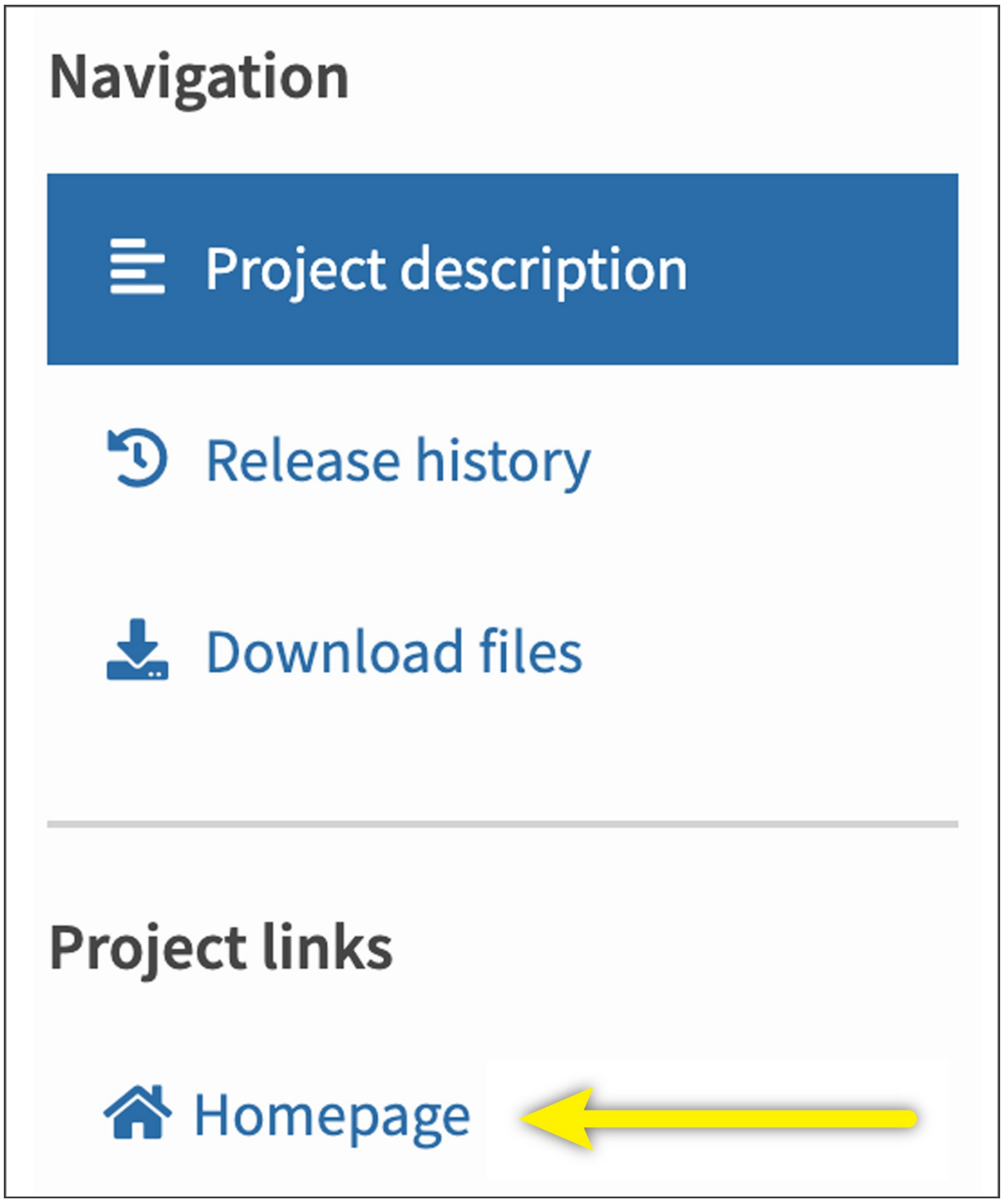
1. Click the clipboard to copy `pip install citipy`.
2. Launch your PythonData Anaconda Prompt. You should see the following:

```
(PythonData) C:\Users\your_computer_name>
```

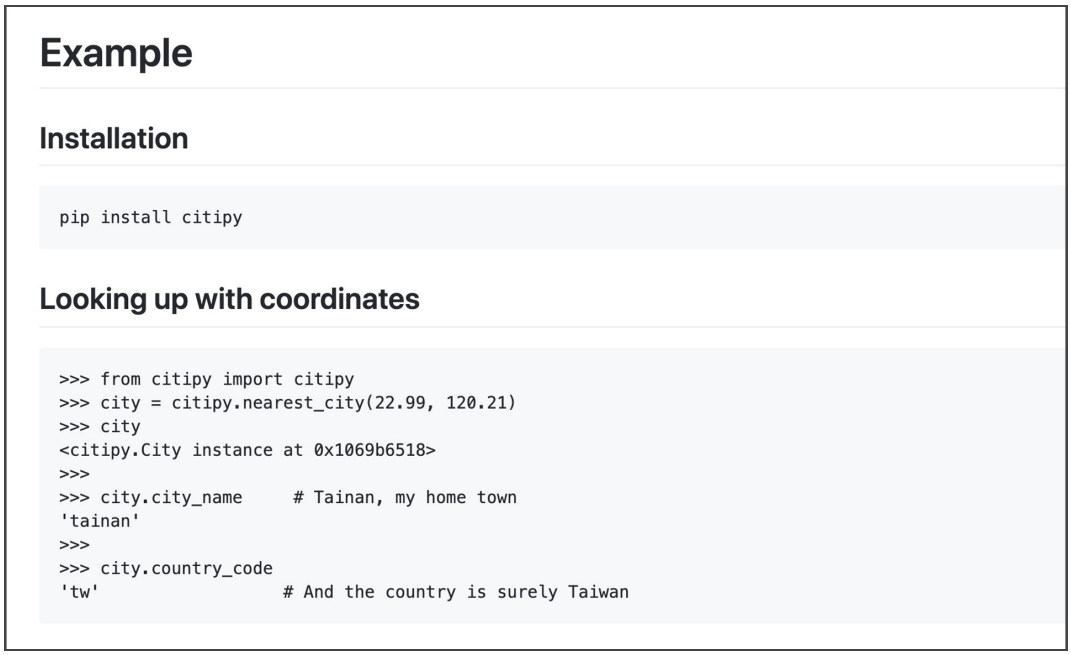
3. Paste `pip install citipy` and press Enter.

The citipy module will probably take a few minutes to download into your PythonData environment.

To learn how to use citypy, click “Homepage” on the module webpage or see the [GitHub citipy repository (https://github.com/wingchen/citipy)](https://github.com/wingchen/citipy).



Select the `README.md` file on the citipy GitHub page for an example of how to use citipy to locate the nearest city and its country code from a pair of latitude and longitude coordinates.



Example

Installation

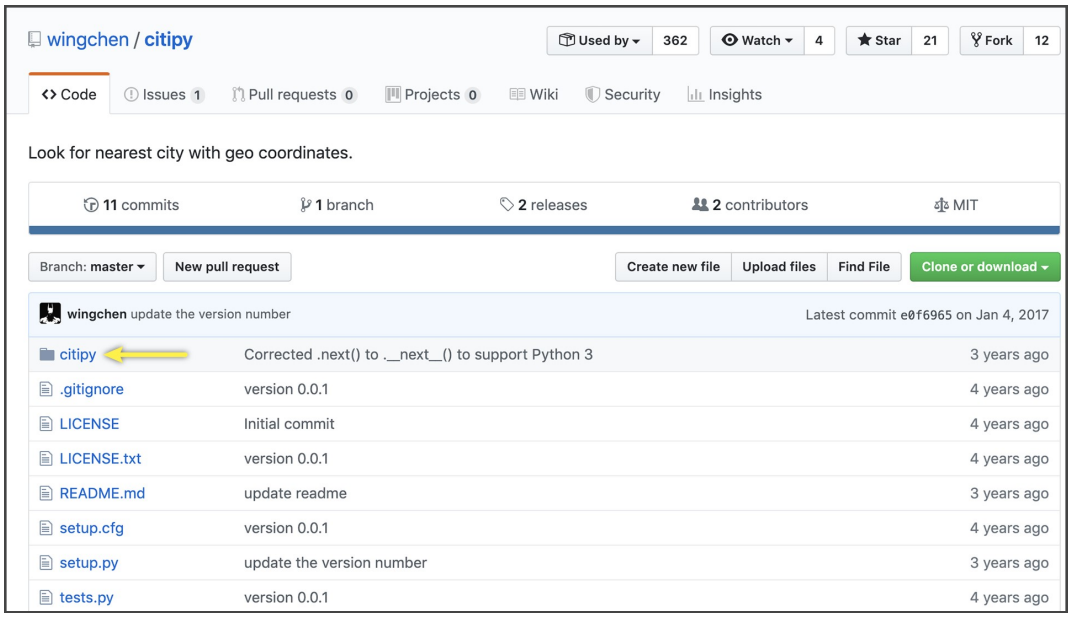
```
pip install citipy
```

Looking up with coordinates

```
>>> from citipy import citipy
>>> city = citipy.nearest_city(22.99, 120.21)
>>> city
<citipy.City instance at 0x1069b6518>
>>>
>>> city.city_name     # Tainan, my home town
'tainan'
>>>
>>> city.country_code
'tw'                   # And the country is surely Taiwan
```

Under “Looking up with coordinates,” the first line says `from citipy import citipy`, meaning we’ll import the `citipy` script from the citipy module.

This script is located in the GitHub repository as indicated by the arrow in the following image.



Note

> When a Python file containing a script is imported to use in another Python script, the `.py` extension does not need to be added to the name of the file when using the import statement.

Let's import the `citipy` script and practice using it. In our `API_practice` file, add a new cell and import the `citipy.py` script from the citipy module.

```
# Use the citipy module to determine city based on latitude and l
from citipy import citipy
```

Next, use the five pairs of latitudes and longitudes we used from our zip practice to get a city and country code from the citipy module.

In a new cell, create a `for` loop that will do the following:

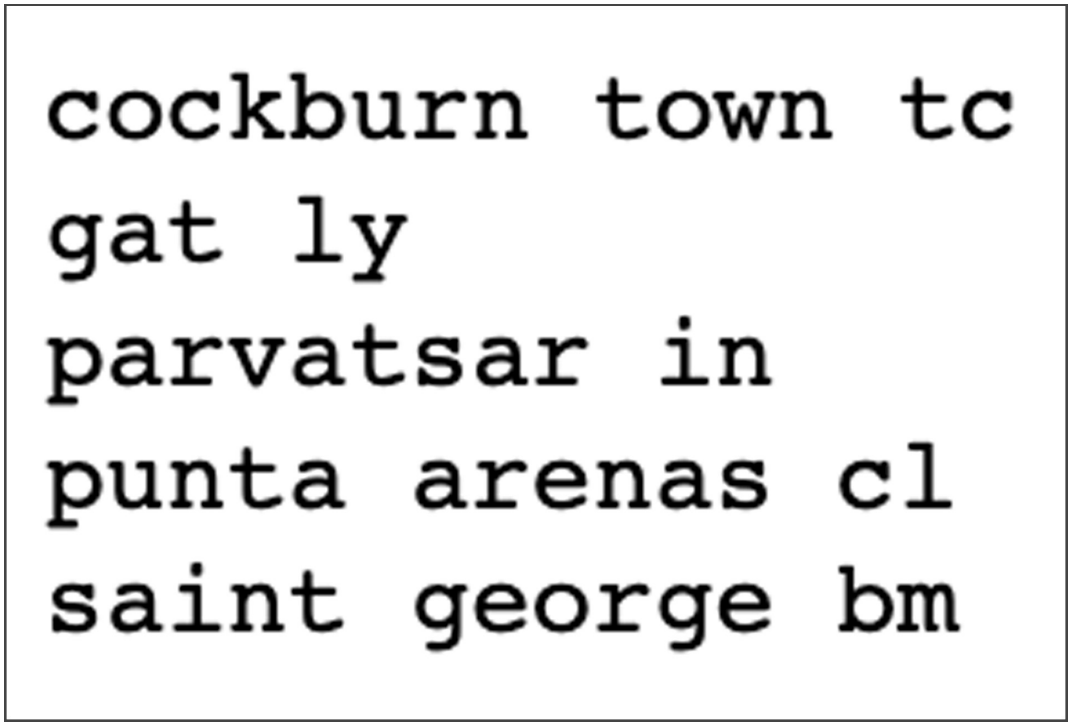
1. Iterate through the coordinates' zipped tuple.
2. Use `citipy.nearest_city()` and inside the parentheses of `nearest_city()`, add the latitude and longitude in this format: `coordinate[0], coordinate[1]`.
3. To print the city name, chain the `city_name` to the `nearest_city()` function.
4. To print the country name, chain the `country_code` to the `nearest_city()` function.

Your code should look like the following. Add the code to a new cell and run the cell.

```
# Use the tuple() function to display the latitude and longitude
for coordinate in coordinates:
```

```
    print(citipy.nearest_city(coordinate[0], coordinate[1]).city_
          citipy.nearest_city(coordinate[0], coordinate[1]).count
```

When we run this cell, the output will show five cities with their associated country codes.



> **Note**
> You can only unzip a zipped tuple once before it is removed from the computer's memory. If you didn't print the cities and their country codes to your output, you'll have to rerun the cell with the list of five latitudes and longitudes to pack them into a zipped file.

Now that we are familiar with using the citipy module, we can iterate through our zipped `lat_lngs` tuple and find the nearest city. When we find a city, we'll need to add it to a list so that we can use the cities to get the weather data.

First, import the `citipy` module in our `WeatherPy` file. In a new cell, add the following code.

```
# Create a list for holding the cities.
cities = []
# Identify the nearest city for each latitude and longitude combi
for coordinate in coordinates:
    city = citipy.nearest_city(coordinate[0], coordinate[1]).city

    # If the city is unique, then we will add it to the cities li
    if city not in cities:
        cities.append(city)
# Print the city count to confirm sufficient count.
len(cities)
```

Some of this code should look familiar, but let's break it down:

1. We create a `cities` list to store city names.
2. We iterate through the `coordinates`, as in our practice, and retrieve the nearest city using the latitude and longitude pair.
3. We add a decision statement with the logical operator `not in` to determine whether the found city is already in the `cities` list. If not, then we'll use the `append()` function to add it. We are doing this because among the 1,500 latitudes and longitudes, there might be duplicates, which will retrieve duplicate cities, and we want to be sure we capture only the unique cities.

> **Note**
>
> The citipy module finds the nearest city to the latitude and longitude pair with a population of 500 or more.

Finding

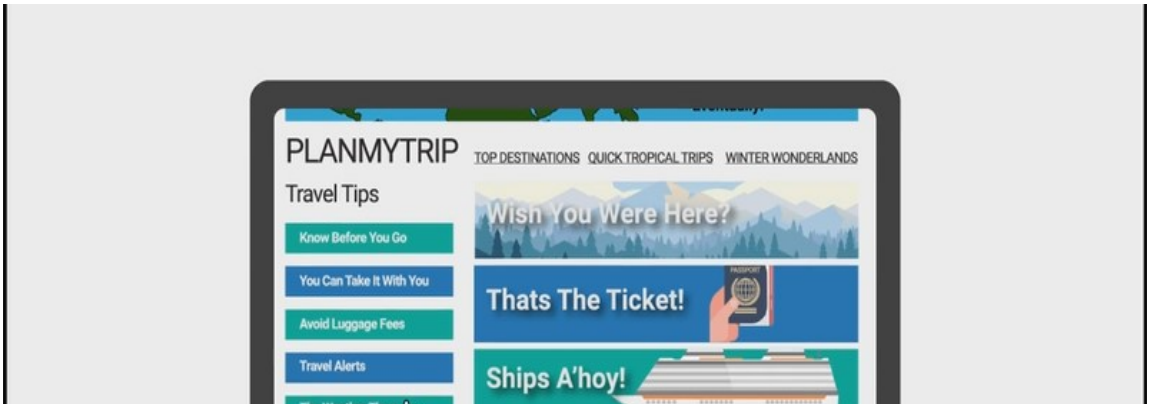
When you run the code block, you should get slightly more than 500 unique cities. If you get fewer than 500, increase your `size` limit on the `np.random.uniform()` function.

Add, Commit, Push

Update or add your code to your GitHub repository for this module.

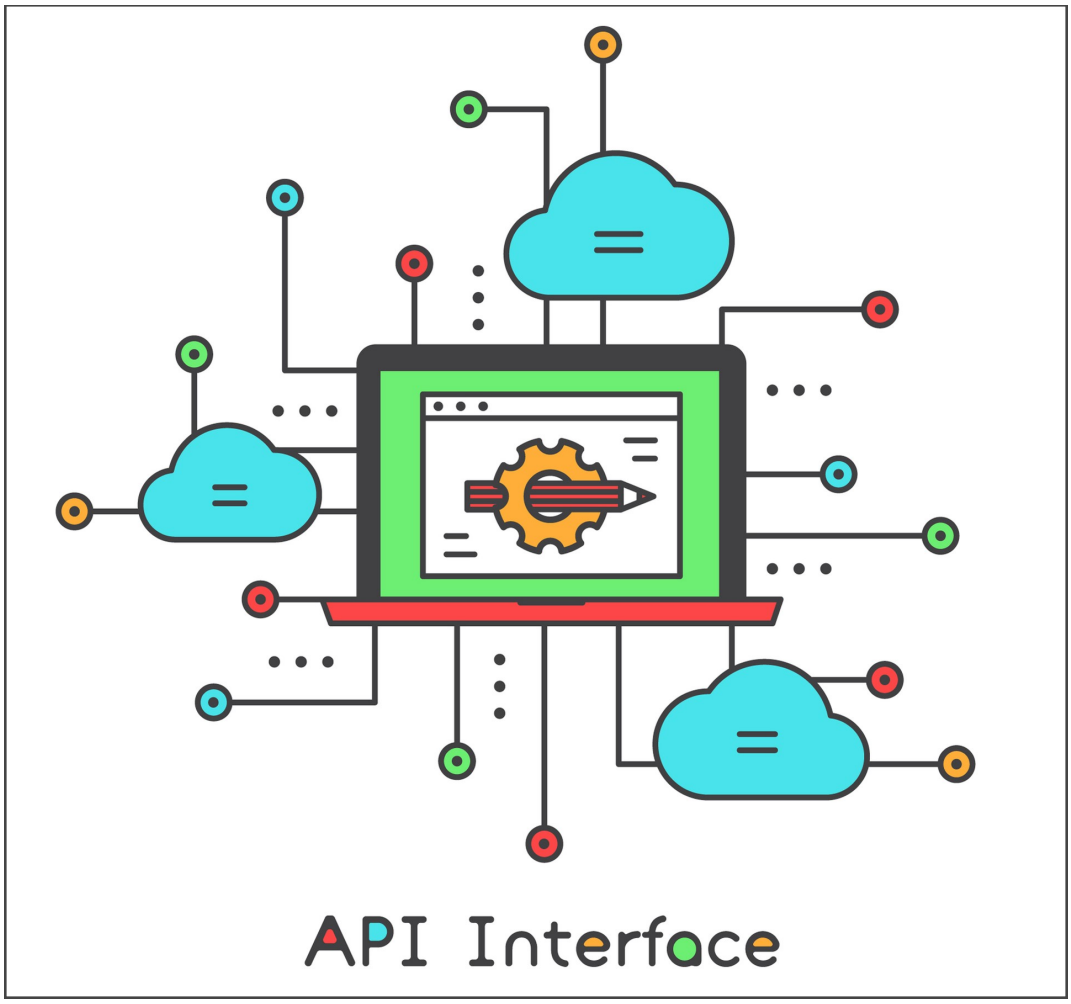
6.2.1: Understanding APIs

Time to get some weather data. During a brainstorming session with Jack, you decide to use the OpenWeatherMap Application Programming Interface (API) to get the weather data for your database. Of course, now comes the tricky part: actually getting the data. You know you can do this using an API—but how do you use an API? Time to dig back in. The weather data is visually displayed on maps for the customer. But, in order for that to happen, you'll need to retrieve the weather from each city in your database. Since this is your first time using an API to get data from a server, your manager would like you to review APIs and how your company retrieves the data.



When a client uses our company's website to search for hotels, our search engine will gather information from a variety of websites based on the client's preferences through **APIs**. An API call is very similar to navigating to a website. An API points to a URL and collects some data from the webpage or server.

When clients request information from our server through our website, they are making an API call. Once our database has the client's search criteria, our servers search the web for hotels on behalf of the client. Now the roles are reversed: our company is the client requesting information, and all the websites where we derive information are the servers.



Using an API has its limitations because not all information from a server is accessible. Most APIs have tiered services, from free to paid. Free services allow access to limited information, and paid subscriptions provide more access based on the payment plan. Our company has a paid subscription for APIs, but we can only get certain information from websites on hotels such as location, accessibility, rooms, prices, services, and amenities, as well as regional weather data.

Now that you have a general concept of how APIs work, let's register for an OpenWeatherMap **API key**, a token granting access, and use it to retrieve weather data.

6.2.2: Get Started with OpenWeatherMap API

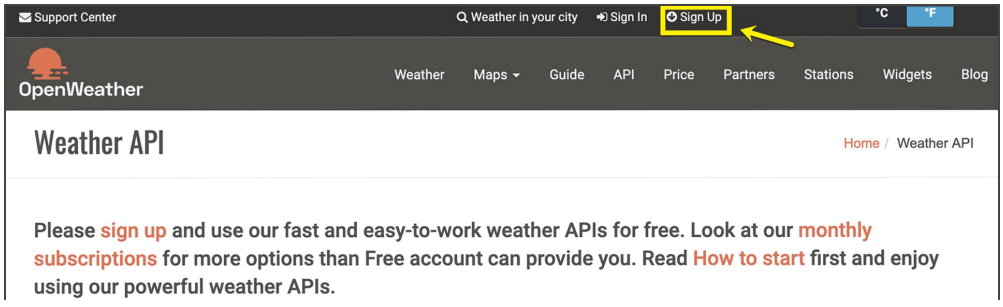
Awesome. You get it—mostly. We'll use the API setup to go out and get information when our clients ask us for it. So, now it's time to download the Python Requests Library and register for an API key.

Register for an API key, then review a short example of how to retrieve weather data from an API call.

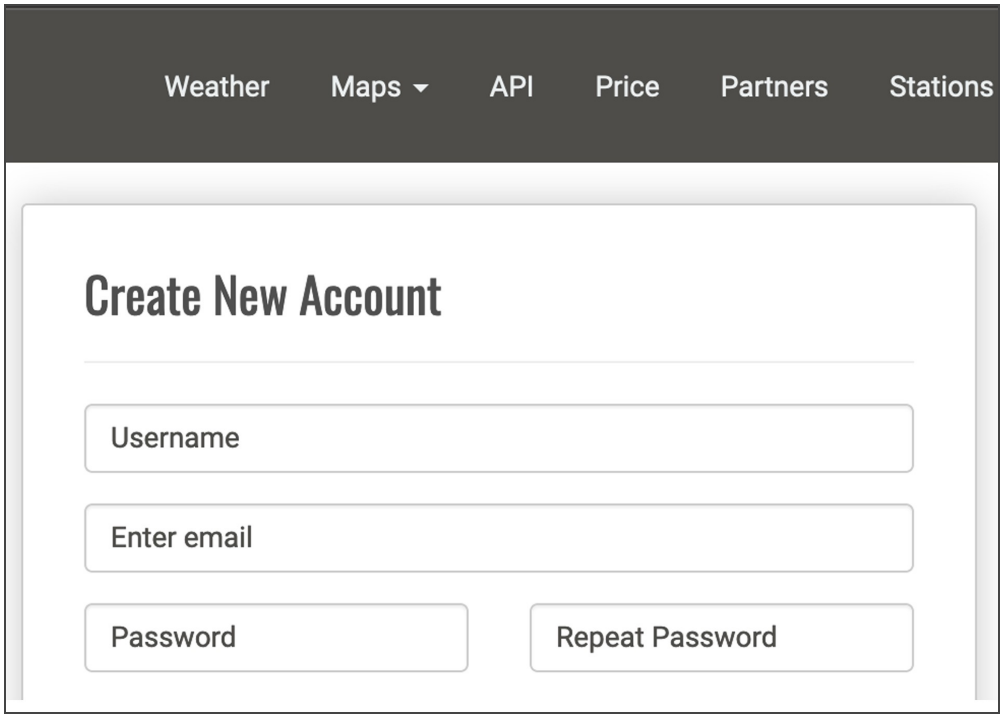
Register for an API Key

Follow these steps to register for an OpenweatherMap API key:

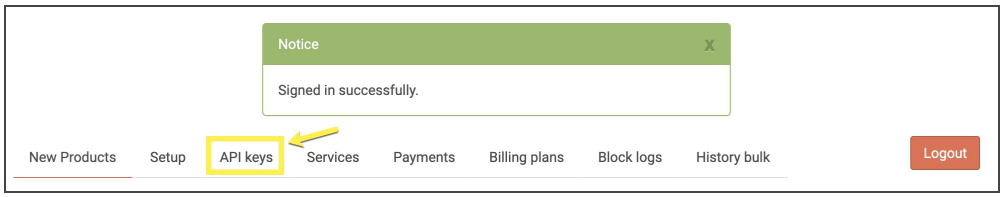
1. Navigate to the [OpenWeatherMap website (https://openweathermap.org/api)](https://openweathermap.org/api) .
2. Click "Sign Up."



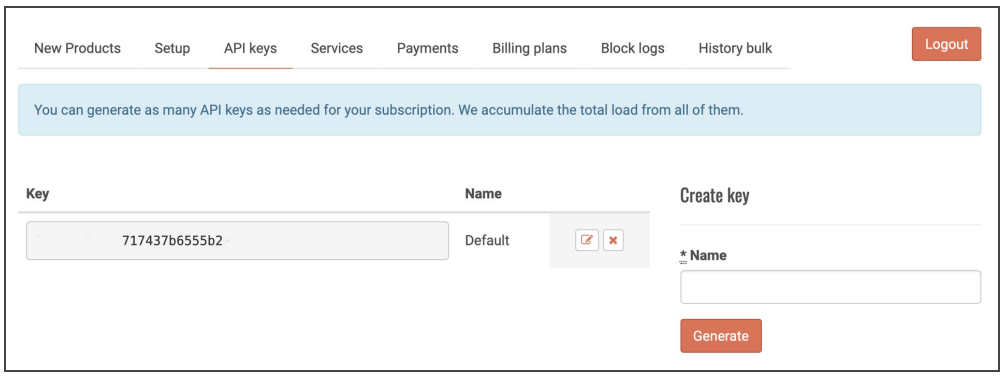
3. Complete the form Create New Account.



4. Once you have a new account, sign in and click on “API keys.”



5. The site will likely generate an API key automatically. If not, under “Create key,” type a name in the available cell and click “Generate” to create an API key.

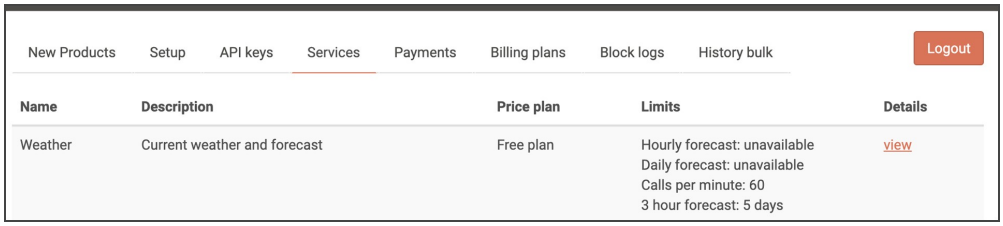


6. Save your API key to a Python file, which we'll add as a dependency to your `WeatherPy.ipynb` file.
    - Navigate to your World_Weather_Analysis folder and launch Jupyter Notebook.
    - Click the New button and select Text File.
    - Rename the text file `config.py`.
    - On the first line, type `weather_api_key=""` and add your API key between the double quotation marks.
    - Save and close the `config.py` file.

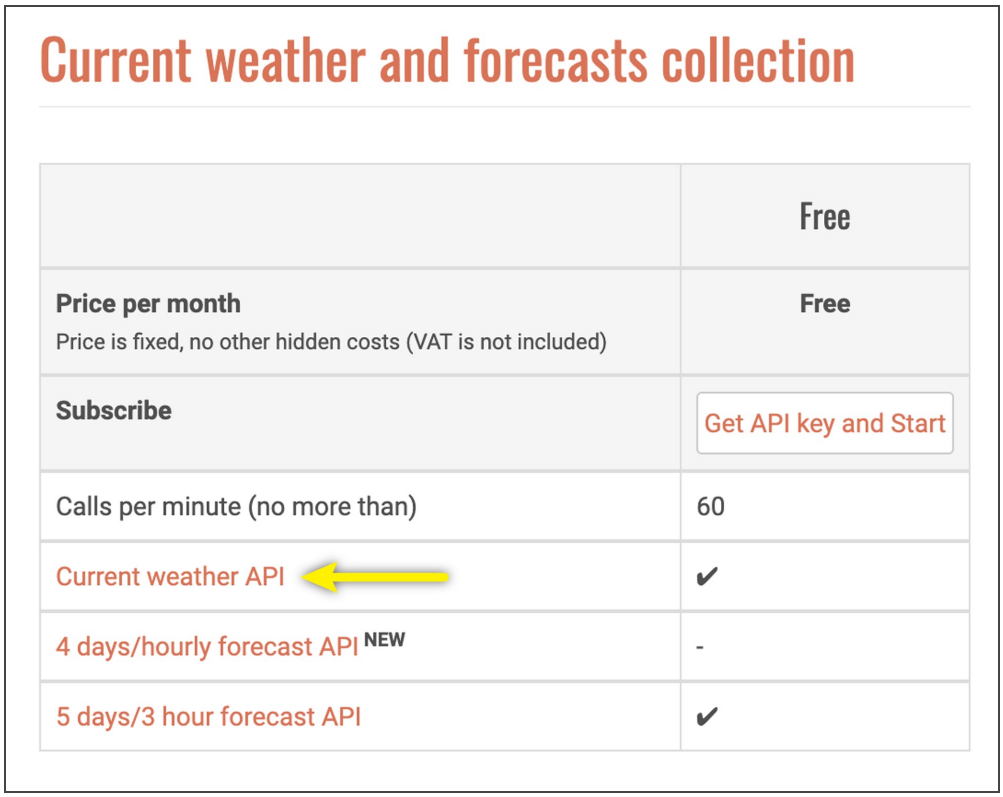
> **Note**
> You can also create the `config.py` file using VS Code.

> **Important!**
> Don't share your API key with anyone and do not add the `config.py` file to your GitHub repository—someone might copy and use it and you could incur charges on your credit card.

7. Click on "Services" for details on your free subscription and its limitations.



8. Click on "View" to see more options on your plan, and then click on "Current weather API" to see how to get the current weather from a city.



We'll refer to this documentation when we make an API call to the server.

The JavaScript Object Notation Format for API Data

The API has reached the website or server, its endpoint, and now we can retrieve data from the website. When we retrieve data from a website, we have to make a "request," which returns data in a text format, not in a tab- or comma-separated file.

One format we can use to parse data is **JavaScript Object Notation (JSON)**. The JSON format is also referred to as an "object" or "JSON object." The data inside a JSON object opens and closes with curly braces, much like a Python dictionary. Inside the JSON object is a collection of dictionaries and arrays.

Below is an example of what weather data looks like in the JSON format when we request it from the OpenWeatherMap website. There are curly braces that wrap the data, and inside the curly braces are dictionaries and arrays.


```
{
  - coord: {
        lon: -0.13,
        lat: 51.51
    },
  - weather: [
      - {
            id: 300,
            main: "Drizzle",
            description: "light intensity drizzle",
            icon: "09d"
        }
    ],
    base: "stations",
  - main: {
        temp: 280.32,
        pressure: 1012,
        humidity: 81,
        temp_min: 279.15,
        temp_max: 281.15
    },
    visibility: 10000,
  - wind: {
        speed: 4.1,
        deg: 80
    },
  - clouds: {
        all: 90
    },
    dt: 1485789600,
  - sys: {
        type: 1,
        id: 5091,
        message: 0.0103,
        country: "GB",
```

```
        sunrise: 1485762037,
        sunset: 1485794875
    },
    id: 2643743,
    name: "London",
    cod: 200
}
```

The Python Requests Library

To request JSON data over the internet, we use the Requests Library in Python. The Anaconda installation comes with version 2.22 of the Requests Library.

Confirm you have the latest version of the Requests Library using the command line, or in the Jupyter Notebook environment. Follow the instructions for your operating system.

---

macOS

1. Launch the command line and activate your PythonData environment.
2. After the prompt, type `$ python` to launch Python.
3. After the Python prompt, `>>>`, type `import requests` and press Enter.
4. On the next line, type `requests.__version__` and press Enter.
5. The output should be version `2.22.0` or later.

Alternatively, you can check the version of request in Jupyter Notebook.

In Jupyter Notebook, create a new file if one hasn't been created. Add the following code to the new cell and run it.

```
import requests
requests.__version__
```

The output should be `2.22.0` or later.

If you have an older version, please upgrade it in your PythonData environment by typing `conda install -c conda-forge requests` at the command prompt and press Enter.

Windows

1. Launch the Anaconda Prompt for your PythonData environment.
2. After the Python prompt, `>`, type `python` to launch Python.
3. At the Python prompt, `>>>`, type `import requests` and press Enter.
4. On the next line type `requests.__version__` and press Enter.
5. The output should be version `2.22.0` or later.

Alternatively, you can check the version of request in Jupyter Notebook.

In Jupyter Notebook, create a new file if one hasn't been created. Add the following code to the new cell and run it.

```
import requests
request.__version__
```

The output should be `2.22.0` or later.

If you have an older version, please upgrade it in your PythonData environment by typing `conda install -c conda-forge requests`, at the command prompt and press Enter.

> **Note**
>
> For more information about the Requests Library, see the following documentation:
>
> **Requests: HTTP for Humans**
> **(https://requests.kennethreitz.org/en/master/)**
>
> **Quickstart**
> **(https://requests.kennethreitz.org/en/master/user/quickstart/#make-a-request)**

6.2.3: Make an API Call

You're about to start just trying random functions to see what works when Jack texts you and tells you your company has a free in-house API tutorial that was designed to teach new team members how to do this very thing. And it gets better—he sent you the link to the tutorial, too. Excited, you click through to access the API tutorial.

In our `API_practice` file, add a new cell after the code we wrote to get the unique cities from with the citipy module. In the new cell, we will import the Requests Library and your API key from the `config.py` file.

> **Note**
> Your `config.py` file should be in the same folder as your `API_practice.ipynb` file or any Jupyter Notebook file that is accessing the `config.py` file.

Add the following code to the new cell and run the cell.

```
# Import the requests library.
import requests
```

```
# Import the API key.
from config import weather_api_key
```

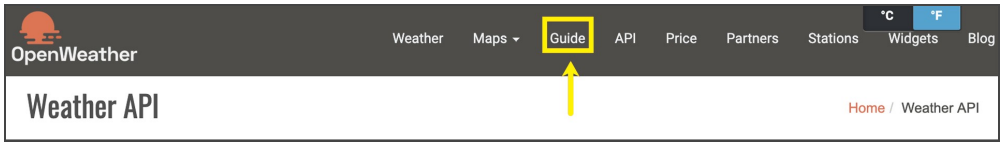
> **Note**
>
> If you get a `ModuleNotFoundError` message, then either your `config.py` file is not in the same folder as the Jupyter Notebook file, or the name of your `config.py` file is not the same in the import statement.
>
> If you get a `ImportError` message, then the variable for your API key in the code statement is not the same as the variable in your `config.py` file.

Make an API Call

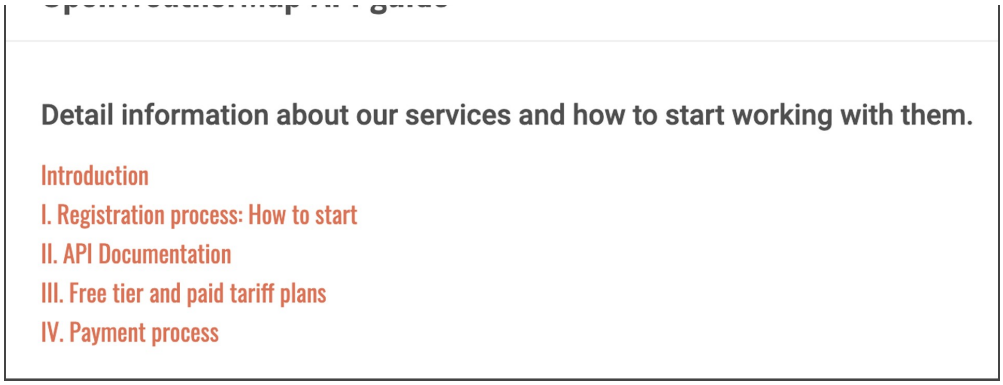
Before we make an API call for the OpenWeatherMap, we need to use the URL provided on the OpenWeatherMap website.

Let's look at the documentation on the OpenWeatherMap website.

1. Navigate to the [OpenWeatherMap website (https://openweathermap.org/api)](https://openweathermap.org/api) .
2. Click "Guide" in the top navigation menu.



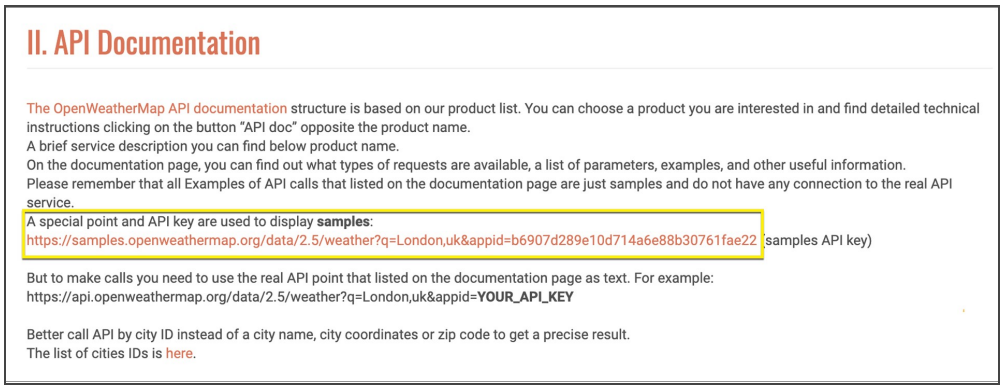
3. From the Guide menu, select the "II. API Documentation" section.



The API documentation section provides instructions on how to make the API call by city name. The structure of our URL should look like the following:

`api.openweathermap.org/data/2.5/weather?q=city&appid=b6907d289e10d714a6e88b30761fae22`

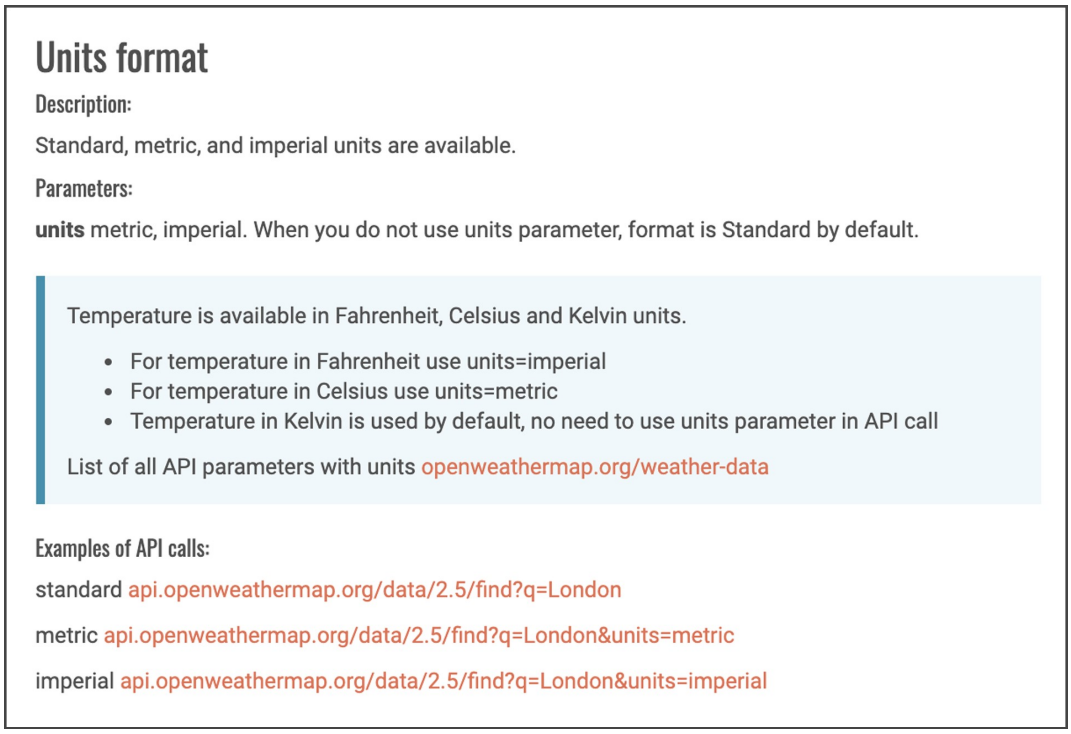
4. Add your API key and the city from the cities array for each call.



Next let's practice making an API call and look at the data returned from the API call. Add the following code to a new cell and run the cell.

```
# Starting URL for Weather Map API Call.
url = "http://api.openweathermap.org/data/2.5/weather?units=Imper
print(url)
```

If you notice, we added another feature to the URL: `units=Imperial`. There are three unit options: standard, metric, and imperial. Navigating to **current weather data page (https://openweathermap.org/current#name)** will show you the options for the unit format.



Units format

Description:

Standard, metric, and imperial units are available.

Parameters:

units metric, imperial. When you do not use units parameter, format is Standard by default.

> Temperature is available in Fahrenheit, Celsius and Kelvin units.
>
> - For temperature in Fahrenheit use units=imperial
> - For temperature in Celsius use units=metric
> - Temperature in Kelvin is used by default, no need to use units parameter in API call
>
> List of all API parameters with units openweathermap.org/weather-data

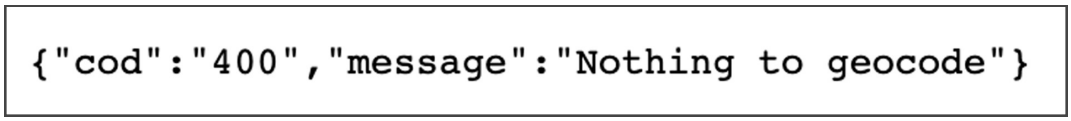
Examples of API calls:

standard api.openweathermap.org/data/2.5/find?q=London

metric api.openweathermap.org/data/2.5/find?q=London&units=metric

imperial api.openweathermap.org/data/2.5/find?q=London&units=imperial

When we run the cell, the output will be a URL. Click the URL, and a new window will open in your default web browser. The URL will return a 400 message because we haven't added a city to our URL.



```
{"cod":"400","message":"Nothing to geocode"}
```

Now let's add a city to the URL to get the current weather data. Add the following code to a new cell and run the cell.

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

In the code, we are creating a string to get the weather data for Boston by using the `city_url`. To create the `city_url` we add the parameter, `&q=` and “Boston” to the `url`.

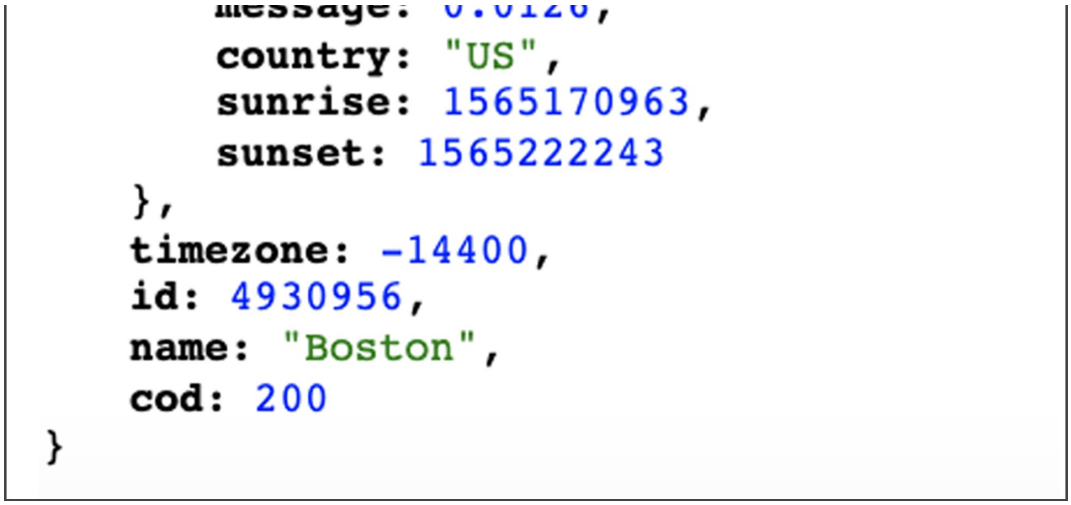
The output of this cell will also be a URL. Click the URL and a new window will open in your default web browser that shows the current weather data for Boston.


```
'{"coord":{"lon":-71.06,"lat":42.36},"weather":[{"id":804,"main":"Clouds","description":"over
cast clouds","icon":"04d"}],"base":"stations","main":{"temp":80.01,"pressure":1011,"humidit
y":65,"temp_min":77,"temp_max":82.99},"visibility":16093,"wind":{"speed":11.41,"deg":200},"cl
ouds":{"all":90},"dt":1565193597,"sys":{"type":1,"id":3486,"message":0.0097,"country":"US","s
unrise":1565170963,"sunset":1565222243},"timezone":-14400,"id":4930956,"name":"Boston","cod":
200}'
```


To make the JSON format readable, [add the Chrome JSONView extension (https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc)](https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc) to your Chrome web browser. The Boston weather data will appear as follows.


```
{
  - coord: {
        lon: -71.06,
        lat: 42.36
    },
  - weather: [
      - {
            id: 501,
            main: "Rain",
            description: "moderate rain",
            icon: "10d"
        },
      - {
            id: 211,
            main: "Thunderstorm",
            description: "thunderstorm",
            icon: "11d"
```

```
        },
      - {
            id: 721,
            main: "Haze",
            description: "haze",
            icon: "50d"
        },
      - {
            id: 701,
            main: "Mist",
            description: "mist",
            icon: "50d"
        }
    ],
    base: "stations",
  - main: {
        temp: 79.68,
        pressure: 1009,
        humidity: 74,
        temp_min: 73.4,
        temp_max: 82.99
    },
    visibility: 16093,
  - wind: {
        speed: 5.82,
        deg: 210
    },
  - rain: {
        1h: 2.54
    },
  - clouds: {
        all: 75
    },
    dt: 1565207600,
  - sys: {
        type: 1,
        id: 3486,
        message: 0.0126
```



Note

You will need to use an extension on Safari to view JSON shown in the above image. Firefox does not need a JSON extension to view JSON files.

Since we're retrieving current weather data, the JSON data shown in the above image will differ from that in your endpoint printout. Instead, your weather data will reflect the specific time you ran the code.

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.

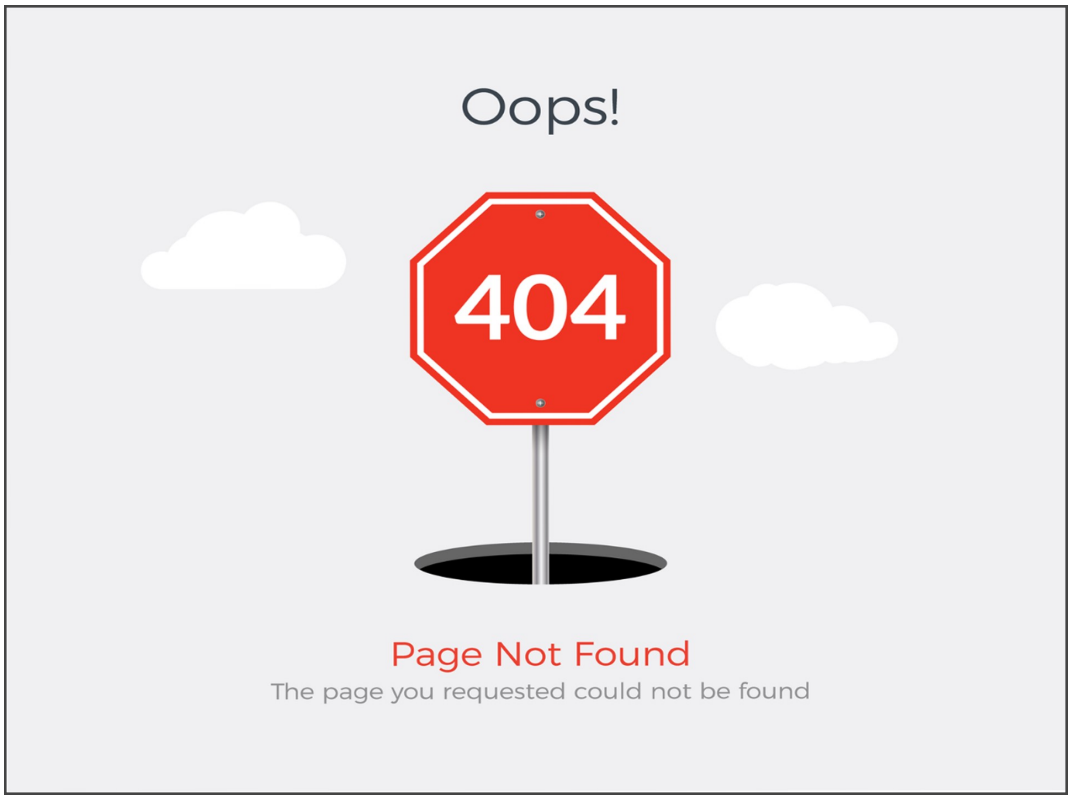
```
# Make a 'Get' request for the city weather.
city_weather = requests.get(city_url)
city_weather
```

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

What is the code response when you run the code, `city_weather = requests.get(city_url)` ?

- ○ `<Response [404]>`
- ○ `<Response [200]>`

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.

```
city_weather.status_code
```

```
200
```

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.

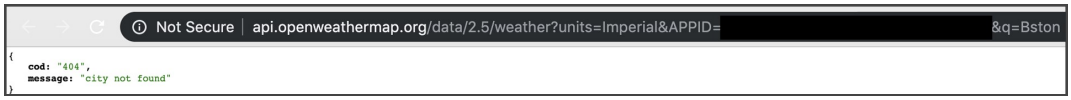
Add the following code to a new cell and run the cell to test if we get the JSON formatted data.

```
# Create an endpoint URL for a city.
city_url = url + "&q=" + "Bston"
city_weather = requests.get(city_url)
if city_weather.json():
    print(f"City Weather found.")
```

```
else:
    print(f"City weather not found.")
```

The output for this code is City weather 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.

6.2.5: Parse a Response from an API

Congratulations on passing the test! You can pat yourself on the back for passing your own test with flying colors. Now you can confidently start mining the JSON file to retrieve specific weather data for each city and add it to a DataFrame.

For each city in our `lats_lngs` list, we need to retrieve the following data and add it to a DataFrame:

- City, country, and date
- Latitude and longitude
- Maximum temperature
- Humidity
- Cloudiness
- Wind speed

Our final DataFrame should look like the following.

	City	Country	Date	Lat	Lng	Max Temp	Humidity	Cloudiness	Wind Speed
0	Tuktoyaktuk	CA	2019-08-08 22:26:20	69.44	-133.03	42.80	87	75	5.82
1	Hermanus	ZA	2019-08-08 22:26:21	-34.42	19.24	54.00	73	0	1.01
2	Bluff	AU	2019-08-08 22:26:21	-23.58	149.07	56.08	54	1	4.94
3	Port Alfred	ZA	2019-08-08 22:26:21	-33.59	26.89	54.00	77	0	4.00
4	Outjo	NA	2019-08-08 22:26:21	-20.11	16.16	62.60	12	26	1.12
5	Agadez	NE	2019-08-08 22:26:21	16.97	7.99	98.24	25	0	4.79
6	Paso De Los Toros	UY	2019-08-08 22:26:22	-32.81	-56.52	48.20	93	90	19.46

7	Avarua	CK	2019-08-08 22:26:22	-21.21	-159.78	77.00	78	90	6.93
8	Taltal	CL	2019-08-08 22:26:22	-25.41	-70.49	54.14	67	33	5.32
9	Airai	TL	2019-08-08 22:26:23	-8.93	125.41	67.40	82	12	4.72

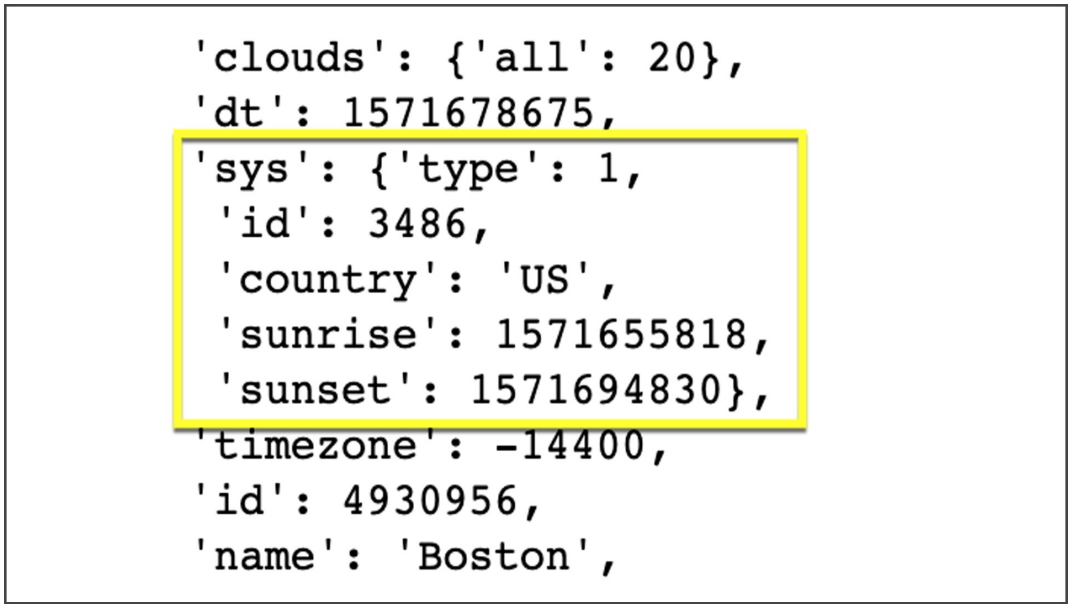
Before we collect weather data from more than 500 cities, we'll walk through how to get the weather data from Boston. First, correct the spelling for the city of `Boston` to get a valid URL. Then, in a new cell, add the following code and run the cell.

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

After running the cell, the output will be the JSON-formatted data from the city of Boston.


```
{'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}
```

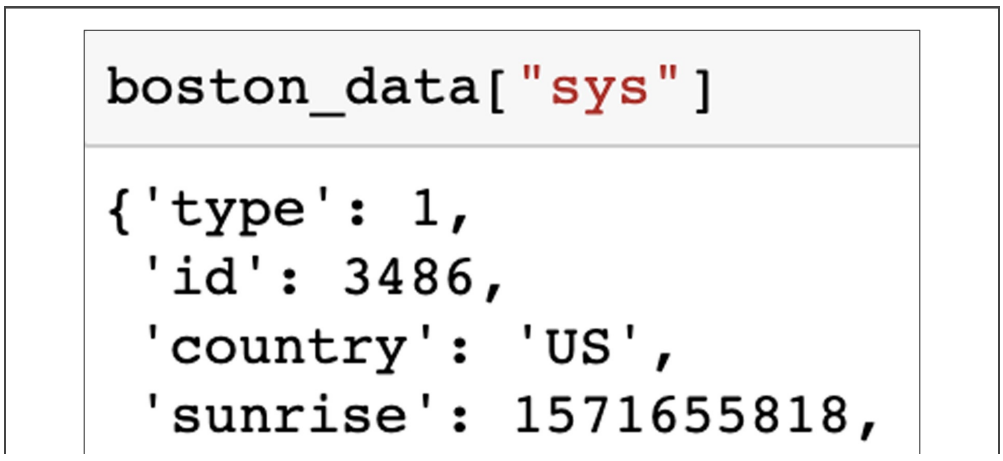
First, let's get something simple, like the country code from the JSON formatted data, which is in a nested dictionary where the first dictionary starts with `sys`.



1. In a new cell, let's assign a variable to the `city_weather.json()` data to the variable "boston_data" and run the cell.

```
# Get the JSON data.
boston_data = city_weather.json()
```

2. Next, using the `sys` key to get the corresponding value, we type `boston_data['sys']` in a new cell and run the cell. The output is another dictionary as shown in the following image.



```
'sunset': 1571694830}
```

3. If we add the `country` key in brackets after the `sys` key, and run the cell again, `'US'` will be returned in the output.

```
boston_data["sys"]["country"]
```

```
'US'
```

> **Note**
> When we used `boston_data["sys"]`, there was a key for `sunrise` and a key for `sunset` in the output. The value for these keys is the time of day in seconds in a database timestamp format.

If we want to retrieve the date in the weather data, we would add the `dt` key to the `boston_data` variable like this: `boston_data["dt"]`.

```
boston_data["dt"]
```

```
1571678675
```

How would you get the latitude value from the Boston JSON data?

- ○ `boston_data["lat"]`
- ○ `boston_data["coord"]["lat"]`

How would you get the maximum temperature value from the Boston JSON data?

- ○ `boston_data["maximum_temp"]`
- ○ `boston_data["temp_max"]`
- ○ `boston_data["main"]["temp_max"]`
- ○ `boston_data["main"]["max_temp"]`

Check Answer

Finish ▸

Using similar syntax to get the time of day, we can get the latitude, longitude, maximum temperature, humidity, percent cloudiness, and wind speed. Add the following code to a new cell and run the cell.

```
lat = boston_data["coord"]["lat"]
lng = boston_data["coord"]["lon"]
max_temp = boston_data["main"]["temp_max"]
humidity = boston_data["main"]["humidity"]
clouds = boston_data["clouds"]["all"]
wind = boston_data["wind"]["speed"]
print(lat, lng, max_temp, humidity, clouds, wind)
```

The output will be all of the weather parameters, with the units for maximum temperature in degrees Fahrenheit, the humidity and clouds as a percentage, and the wind in miles per hour.

```
print(lat, lng, max_temp, humidity, clouds, wind)
```

```
42.36 -71.06 66.2 59 20 14.99
```

Convert the Date Timestamp

The date format will appear in seconds, as we saw when we ran this code.

```
boston_data["dt"]
```

```
1571678675
```

This format is called Coordinated Universal Time (UTC) or Greenwich Mean Time (GMT). If we want to convert the timestamp to the International Organization for Standardization (ISO) format, or YYYY-MM-DD-HH-MM-SS, we need to use the Python datetime module.

Let's convert the date from the Boston weather data in the JSON format to the ISO format.

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

```
# Import the datetime module from the datetime library.
from datetime import datetime
# Get the date from the JSON file.
date = boston_data["dt"]
# Convert the UTC date to a date format with year, month, day, ho
datetime.utcfromtimestamp(date)
```

When we run this code, the output is now in the ISO format with the, year, month, date, hour, minute, and seconds offset by commas.

```
datetime.datetime(2019, 10, 21, 17, 24, 35)
```

We can convert this datetime format to 2019-08-08 15:47:30 using the Python string format method `strftime()` and adding how we want the string to look inside the parentheses. In our case, we would use `strftime('%Y-%m-%d %H:%M:%S')`.

Add `.strftime('%Y-%m-%d %H:%M:%S')` to the end of the conversion: `datetime.utcfromtimestamp(date).strftime('%Y-%m-%d %H:%M:%S')`. Rerun the cell. The output should look like the following.

```
datetime.utcfromtimestamp(date).strftime('%Y-%m-%d %H:%M:%S')
'2019-10-21 17:24:35'
```

Now that we know how to get all the weather data from a JSON response, we can iterate through our cities list and retrieve the data from each city.

> **Note**
>
> For more information about the datetime library and `strftime()`, see the documentation:
>
> - **Datetime** (https://docs.python.org/3.6/library/datetime.html)
> - `strftime()` (https://docs.python.org/3.6/library/datetime.html#strftime-and-strptime-behavior)

6.2.6: Get the City Weather Data

You and Jack have a quick catch-up over coffee the next morning, and he can hardly believe his eyes. You have already learned how to use APIs to retrieve data from a JSON file. As you look at the project plan, you realize it's now time for an even bigger challenge: retrieving the weather data from 500+ cities.

You and Jack know this weather data will be able to enhance our client's search capabilities and hopefully draw more clients to your website. This is a huge step for you, but you know you can build on what you've learned thus far to pull it off.

Let's use pseudocode to map out, at a high level, how we will get the weather data for each city for the website.

We will need to do the following:

1. Import our dependencies and initialize counters and an empty list that will hold the weather data.
2. Loop through the cities list.

3. Group the cities in sets of 50 to log the process as we find the weather data for each city.
    - Two counters will be needed here: one to log the city count from 1 to 50, and another for the sets.
4. Build the `city_url` or endpoint for each city.
5. Log the URL and the record and set numbers.
6. Make an API request for each city.
7. Parse the JSON weather data for the following:
    - City, country, and date
    - Latitude and longitude
    - Maximum temperature
    - Humidity
    - Cloudiness
    - Wind speed
8. Add the data to a list in a dictionary format and then convert the list to a DataFrame.

Import Dependencies, and Initialize an Empty List and Counters

At the top of our code block, we are going to declare an empty list, `city_data = []`; add a print statement that references the beginning of the logging; and create counters for the record numbers, 1–50; and the set counter.

Before adding new code to our `WeatherPy` file, make sure the following tasks are completed:

- Import your Requests Library and the weather_api_key.
- Build the basic URL for the OpenWeatherMap with your weather_api_key added to the URL.

Also, import the datetime module using the following code:

```
# Import the datetime module from the datetime library.
from datetime import datetime
```

Next, add the following code to a new cell, but don't run the cell. Instead, continue to add on to this code block.

```
# Create an empty list to hold the weather data.
city_data = []
# Print the beginning of the logging.
print("Beginning Data Retrieval     ")
print("-----------------------------")

# Create counters.
record_count = 1
set_count = 1
```

In the code block, we have initialized the counters at 1 because we want the first iteration of the logging for each recorded response and the set to start at 1.

Loop Through the List of Cities and Build the City URL

Next, we need to iterate through our list of cities and begin building the URL for each city, while grouping our records in sets of 50. To do this, use `for i in range(len(cities))` and the index to tell us when we get to 50. We can also retrieve the city from the `cities` list and add it to the `city_url` by using indexing, as shown in the following code:

```
# Loop through all the cities in our list.
for i in range(len(cities)):

    # Group cities in sets of 50 for logging purposes.
    if (i % 50 == 0 and i >= 50):
        set_count += 1
        record_count = 1
    # Create endpoint URL with each city.
    city_url = url + "&q=" + "i"
```

Every time we want to reference the city in our code, we need to use the indexing on the `cities_list`. Unfortunately, this will cause programming errors when we are building the `city_url` because it adds the index, not the city name, to the `city_url`. To fix this issue, we need to create another `for` loop to get the city from the `cities` list.

Instead of using two `for` loops, we can use the `enumerate()` method as an alternative way to iterate through the list of cities and retrieve both the index, and the city from the list. The syntax for the `enumerate()` method is the following:

```
for i, item in enumerate(list):
```

Let's use the `enumerate()` method to get the index of the city for logging purposes and the city for creating an endpoint URL. Add the following code below our counters.

```
# Loop through all the cities in the list.
for i, city in enumerate(cities):

    # Group cities in sets of 50 for logging purposes.
    if (i % 50 == 0 and i >= 50):
        set_count += 1
        record_count = 1
    # Create endpoint URL with each city.
    city_url = url + "&q=" + city

    # Log the URL, record, and set numbers and the city.
    print(f"Processing Record {record_count} of Set {set_count} |
    # Add 1 to the record count.
    record_count += 1
```

Let's break down the code so we understand fully before continuing:

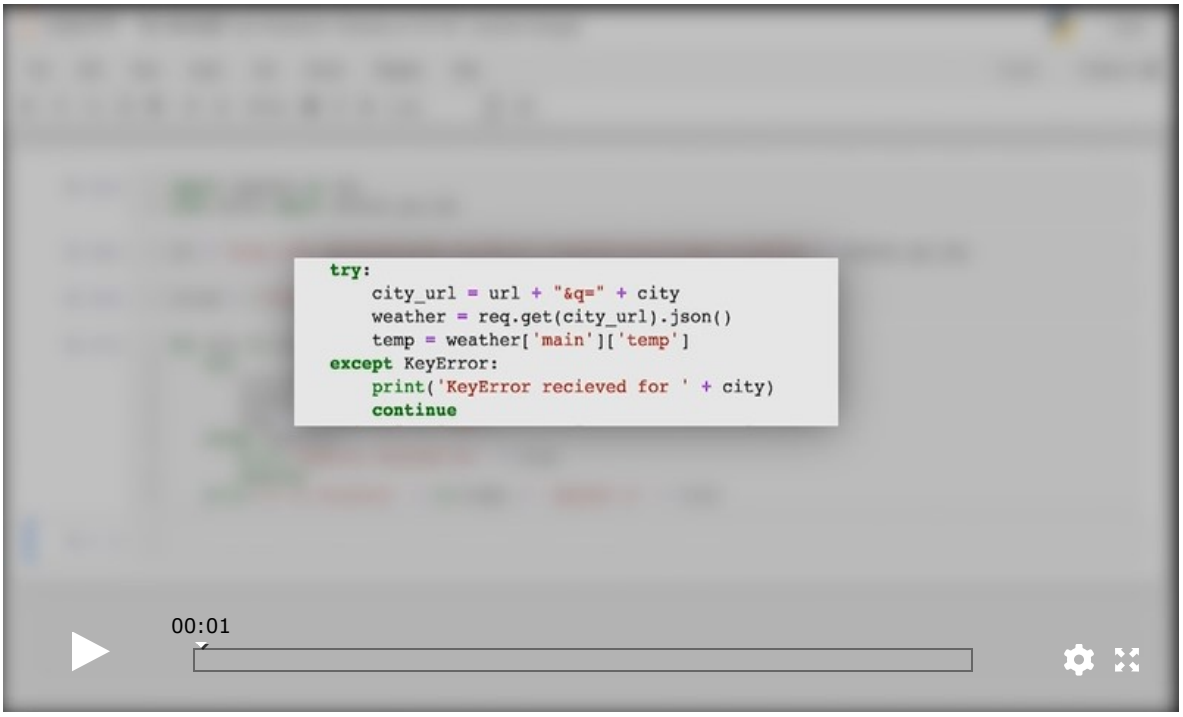
- We create the `for` loop with the `enumerate()` method and reference the index and the city in the list.
- In the conditional statement, we check if the remainder of the index divided by 50 is equal to 0 and if the index is greater than or equal to 50. If the statement is true, then the `set_count` and the `record_count` are equal to 1.
- Inside the conditional statement, we create the URL endpoint for each city, as before.
- Also, we add a print statement that tells us the record count and set count, and the city that is being processed.

- Then we add one to the record count before the next city is processed.

Next, we will retrieve the data from the JSON weather response for each city.

> **Note**
> When retrieving data from an API, or even when scraping a webpage, make sure there is data to parse. If not, the script might stop at that moment and not finish getting all the data we need.

Handle API Request Errors with try-except Blocks



We have handled request errors for getting the response from a valid city with an API call using conditional statements. Now we'll learn how to handle errors while parsing weather data from a JSON file.

We'll add a `try-except` block to our code to prevent the API request from stopping prematurely if the `city_weather` request isn't a valid response. If the request isn't valid, the code will not find the first item requested, which is the dictionary `"coord"` with the code `city_lat = city_weather["coord"]["lat"],` and skip the city and continue to run.

The `try-except` block has similar syntax and structure as the if-else statement. The basic format is as follows:

```
try:
  Do something
except:
  print("An exception occurred")
```

We can add a `try-except` block to our code and, below the `try` block, we will parse the data from the JSON file and add the data to the cities list.

Let's add a `try` block. Then, below the `try` block, do the following:

1. Parse the JSON file.
2. Assign variables for each piece of data we need.
3. Add the data to the cities list in a dictionary format.

Add the following code after `record_count += 1`.

```
# Run an API request for each of the cities.
    try:
        # Parse the JSON and retrieve data.
```

```
        city_weather = requests.get(city_url).json()
        # Parse out the needed data.
        city_lat = city_weather["coord"]["lat"]
        city_lng = city_weather["coord"]["lon"]
        city_max_temp = city_weather["main"]["temp_max"]
        city_humidity = city_weather["main"]["humidity"]
        city_clouds = city_weather["clouds"]["all"]
        city_wind = city_weather["wind"]["speed"]
        city_country = city_weather["sys"]["country"]
        # Convert the date to ISO standard.
        city_date = datetime.utcfromtimestamp(city_weather["dt"])
        # Append the city information into city_data list.
        city_data.append({"City": city.title(),
                          "Lat": city_lat,
                          "Lng": city_lng,
                          "Max Temp": city_max_temp,
                          "Humidity": city_humidity,
                          "Cloudiness": city_clouds,
                          "Wind Speed": city_wind,
                          "Country": city_country,
                          "Date": city_date})

# If an error is experienced, skip the city.
    except:
        print("City not found. Skipping...")
        pass

# Indicate that Data Loading is complete.
print("-----------------------------")
print("Data Retrieval Complete      ")
print("-----------------------------")
```

Let’s review the code:

- We parse the JSON file for the current city.

- We get the latitude, longitude, maximum temperature, humidity, cloudiness, and wind speed and assign those values to variables.
- We append the cities list with a dictionary for that city, where the key-value pairs are the values from our weather parameters.
- Finally, below the `try` block and after the `except` block, we add the closing print statement, which will let us know the data retrieval has been completed. Make sure that your `except` block is indented and in line with the `try` block, and that the print statements are flush with the margin.
- Under the print statement in the `except` block, we add the `pass` statement, which is a general purpose statement to handle all errors encountered and to allow the program to continue.

> **IMPORTANT!**
> Generally, it isn't good coding practice to add the `pass` statement to the `except` block. Ideally, we want to handle or catch each error as it happens and do something specific (e.g., add another `try` block or print out the error).

Now you have all your code to perform the API calls for each city and parse the JSON data. Let's run the cell!

As your code is running, your output should be similar to the following image:

```
Beginning Data Retrieval
-----------------------------
Processing Record 1 of Set 1 | tuktoyaktuk
Processing Record 2 of Set 1 | hermanus
Processing Record 3 of Set 1 | bluff
Processing Record 4 of Set 1 | port alfred
Processing Record 5 of Set 1 | outjo
Processing Record 6 of Set 1 | agadez
```

```
Processing Record 7 of Set 1 | paso de los toros
Processing Record 8 of Set 1 | avarua
Processing Record 9 of Set 1 | mancio lima
City not found. Skipping...
Processing Record 10 of Set 1 | taltal
Processing Record 11 of Set 1 | airai
Processing Record 12 of Set 1 | samusu
City not found. Skipping...
```

After collecting all our data, we can tally the number of cities in the `city_data` array of dictionaries using the `len()` function.

IMPORTANT!

If you didn't get more than 500 cities, run the code to generate random latitude and longitude combinations and all the code below it. Or increase the size of the latitude and longitude combinations.

NOTE

For more information about the `try-except` blocks, see the [documentation on errors and exceptions (https://docs.python.org/3.6/tutorial/errors.html)](https://docs.python.org/3.6/tutorial/errors.html).

6.2.7: Create a DataFrame of City Weather Data

You have the data in a list of dictionaries, which is a format that you can use to create a Pandas DataFrame. You will also need to export the DataFrame as a CSV file for Jack.

Our next steps will entail converting the array of dictionaries to a DataFrame, ensuring the columns are in the correct order, and exporting the DataFrame to a comma-separated (CSV) file.

> **Rewind**
> Recall that we can convert a list of dictionaries to a Pandas DataFrame using `df = pd.DataFrame(list with dictionaries)`.

In a new cell, add the following code to convert the array of dictionaries to a Pandas DataFrame and run the cell.

```
# Convert the array of dictionaries to a Pandas DataFrame.
city_data_df = pd.DataFrame(city_data)
city_data_df.head(10)
```

The DataFrame should appear like the following.

0	Tuktoyaktuk	75	CA	2019-08-08 22:26:20	87	69.44	-133.03	42.80	5.82
1	Hermanus	0	ZA	2019-08-08 22:26:21	73	-34.42	19.24	54.00	1.01
2	Bluff	1	AU	2019-08-08 22:26:21	54	-23.58	149.07	56.08	4.94
3	Port Alfred	0	ZA	2019-08-08 22:26:21	77	-33.59	26.89	54.00	4.00
4	Outjo	26	NA	2019-08-08 22:26:21	12	-20.11	16.16	62.60	1.12
5	Agadez	0	NE	2019-08-08 22:26:21	25	16.97	7.99	98.24	4.79
6	Paso De Los Toros	90	UY	2019-08-08 22:26:22	93	-32.81	-56.52	48.20	19.46
7	Avarua	90	CK	2019-08-08 22:26:22	78	-21.21	-159.78	77.00	6.93
8	Taltal	33	CL	2019-08-08 22:26:22	67	-25.41	-70.49	54.14	5.32
9	Airai	12	TL	2019-08-08 22:26:23	82	-8.93	125.41	67.40	4.72

Next, we'll reorder the columns as City, Country, Date, Lat, Lng, Max Temp, Humidity, Cloudiness, and Wind Speed, so they are easy to read.

Rewind

Recall that to reorder the columns, we assign a variable to an array of the columns in the order we want them to appear:

```
new_column_order = ["column2", "column4", "column1"]
```

Then, we assign a new or the same DataFrame with new column order:

```
df = df[new_column_order]
```

Once you reorder the columns and run the cell, the DataFrame should look like the following.

	City	Country	Date	Lat	Lng	Max Temp	Humidity	Cloudiness	Wind Speed
0	Tuktoyaktuk	CA	2019-08-08 22:26:20	69.44	-133.03	42.80	87	75	5.82
1	Hermanus	ZA	2019-08-08 22:26:21	-34.42	19.24	54.00	73	0	1.01
2	Bluff	AU	2019-08-08 22:26:21	-23.58	149.07	56.08	54	1	4.94
3	Port Alfred	ZA	2019-08-08 22:26:21	-33.59	26.89	54.00	77	0	4.00
4	Outjo	NA	2019-08-08 22:26:21	-20.11	16.16	62.60	12	26	1.12
5	Agadez	NE	2019-08-08 22:26:21	16.97	7.99	98.24	25	0	4.79
6	Paso De Los Toros	UY	2019-08-08 22:26:22	-32.81	-56.52	48.20	93	90	19.46
7	Avarua	CK	2019-08-08 22:26:22	-21.21	-159.78	77.00	78	90	6.93
8	Taltal	CL	2019-08-08 22:26:22	-25.41	-70.49	54.14	67	33	5.32

9	Airai	TL	2019-08-08 22:26:23	-8.93	125.41	67.40	82	12	4.72

Lastly, following the instructions below, we'll create an output file to save the DataFrame as a CSV in a new folder for that file.

In our World_Weather_Analysis folder, create a new folder called "weather_data." Add the following code to a new cell, run the cell, then confirm your CSV file is in the folder.

```
# Create the output file (CSV).
output_data_file = "weather_data/cities.csv"
# Export the City_Data into a CSV.
city_data_df.to_csv(output_data_file, index_label="City_ID")
```

The last line in the code block will export the DataFrame to a CSV file, with the index label (or column A) header as "City_ID." If we ever need to export the CSV file to a DataFrame, that header will be present in the DataFrame.

We've completed our tasks for making API calls, parsing the response, and collecting the data for our project. Before we move on to graphing and statistical analysis, let's update our GitHub repository.

Modify the .gitignore File.

We don't want the `config.py` file containing the API key to be exposed to the public on GitHub, as this would mean anyone could copy and use our API key, possibly causing us to incur charges.

When we type `git status` in the command line, we can see all the files we have created so far that are untracked.

```
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
    (use "git add <file>..." to include in what will be committed)
          API_practice.ipynb
          WeatherPy.ipynb
          config.py
          random_numbers.ipynb

nothing added to commit but untracked files present (use "git add" to track)
```

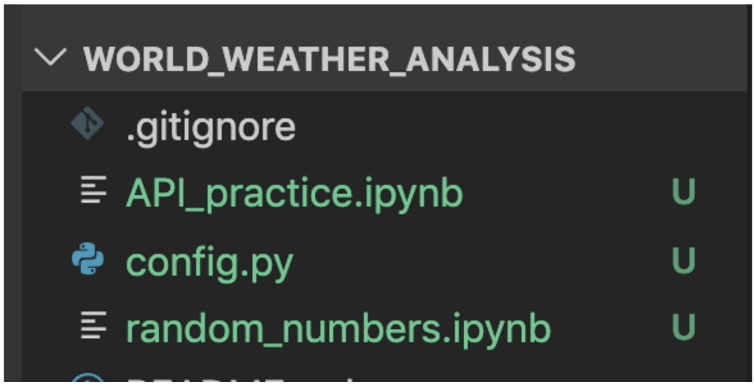
If we only wanted to add the `WeatherPy.ipynb` file to GitHub we could type

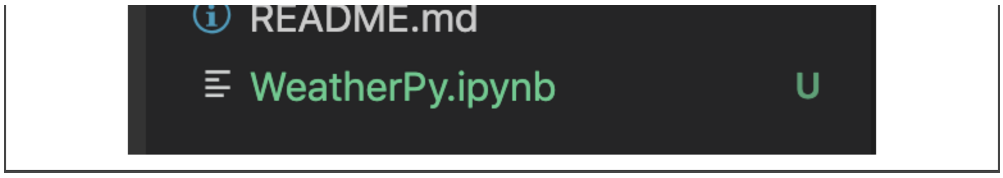
`git add WeatherPy.ipynb`. However, every time we want to add a new file or update current files to the repository, we have to add each file individually, which is time-consuming and cumbersome. Instead, we can add the files we don't want to track to the `.gitignore` file.

> **Rewind**
> GitHub does not track files and extensions that are added to the `.gitignore` file.

Before we add our files to GitHub, let's add `config.py` to the `.gitignore` file. Follow these steps:

1. Open your World_Weather_Analysis GitHub folder in VS Code.



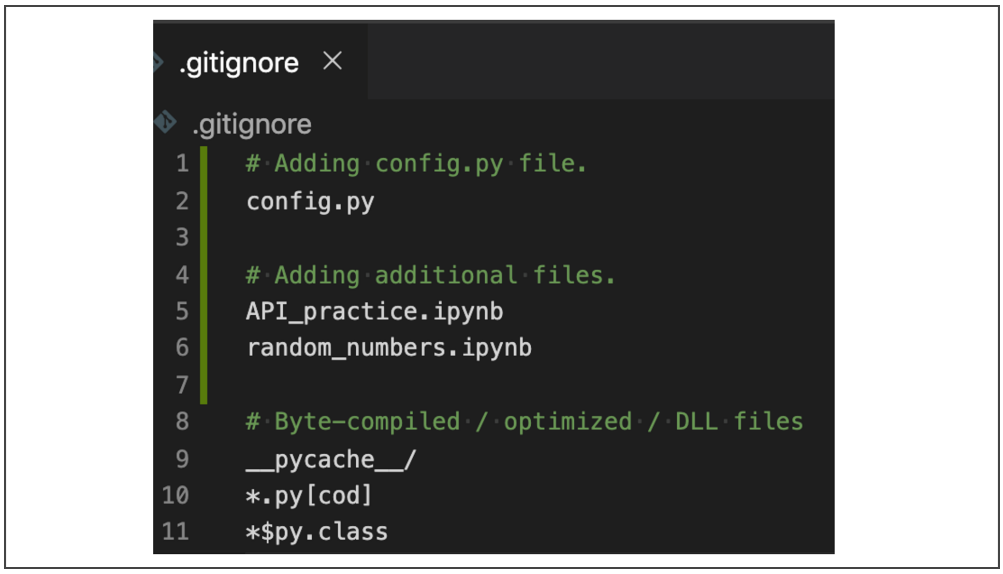


2. Open the `.gitignore` file, and on the first line type the following:

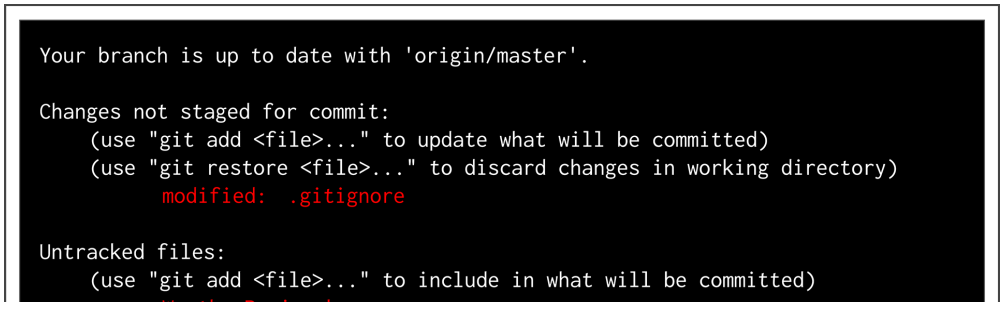
```
# Adding config.py file.
config.py
```

3. While the `.gitignore` file is open, add the `API_practice.ipynb` and `random_numbers.ipynb` files and save the file.

   Your `.gitignore` file should look similar to the following.



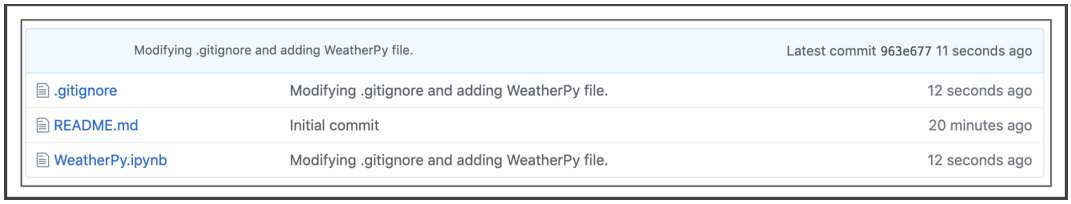
4. In the command line, type `git status` and press Enter. The output should say the `.gitignore` file has been modified and the `WeatherPy.ipynb` file is untracked.



```
        WeatherPy.ipynb

no changes added to commit (use "git add" and/or "git commit -a")
```

5. Use `git add` and `git commit` to commit the modifications to `.gitignore` and the `WeatherPy.ipynb` file to GitHub.

On GitHub, the only new file you should see is the `WeatherPy.ipynb` file.



Congratulations on modifying your `.gitignore` file!

6.3.1: Plot Latitude vs. Temperature

After exporting your CSV file, you closed your laptop and headed home, confident in a good day's work.

And, when you come into work the next day, you hear that someone else has been impressed with your work as well—Jack! In fact, Jack is so impressed that he wants to give you the opportunity to take the day to dedicate some time to a different project: creating a community outreach website for middle school STEM students.

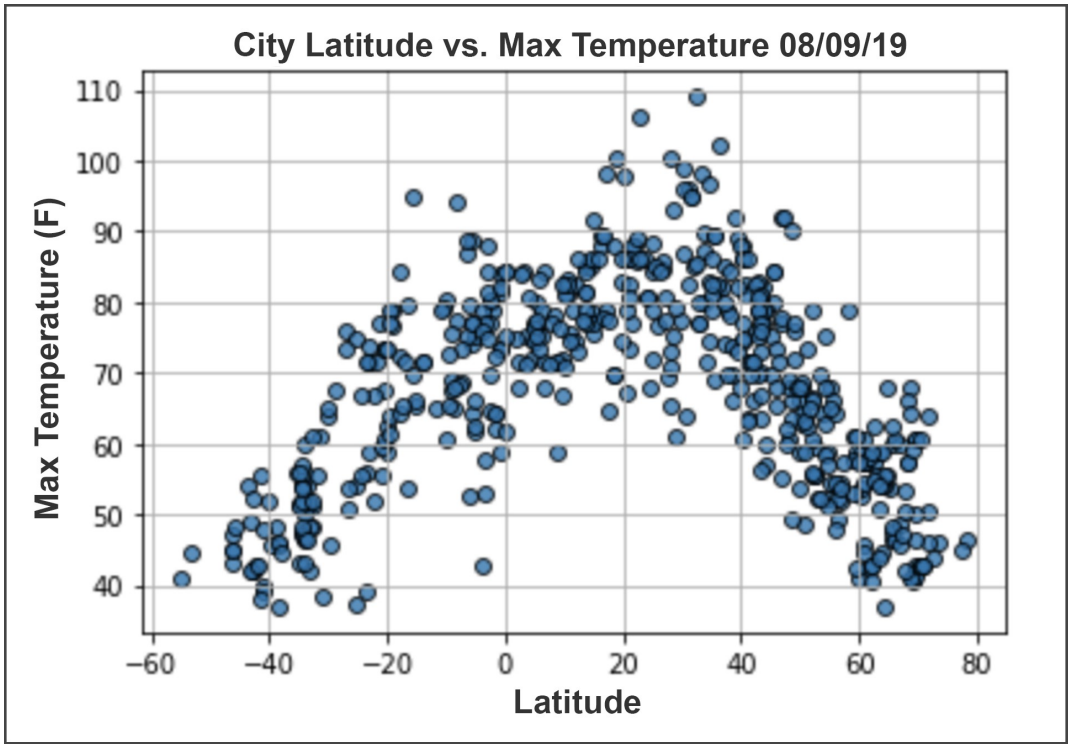
This is an exciting project. These community-focused opportunities don't come around as often as you would like, and they are a great chance to get kids involved in STEM. Additionally, you work with other tech companies in the area, each taking on one aspect of the project.

Your company is going to focus on climate change, and since you know how to use Matplotlib, you decide to create some visualizations that showcase the weather parameters you

retrieved with changing latitude for the 500-plus cities from all over the world. The students will then be able to use these visualizations to explore how weather parameters change based on latitude.

We are going to create a series of scatter plots for each weather parameter against the latitude for all the cities. The students will use these scatter plots to write a summary report on how different weather parameters change based on the latitude.

We'll create scatter plots for latitude vs. maximum temperature, humidity, cloudiness, and wind speed. The first, latitude vs. maximum temperature, should look like the following scatter plot.



Rewind

> Recall that to create a scatter plot, we collect then add x- and y-axis data to `plt.scatter()`.

Get Data for Plotting

First, we'll retrieve the data we need to create our scatter plots. We will need latitude, maximum temperature, humidity, cloudiness, and wind speed from all the cities. Add the following code to a new cell and run the cell.

```
# Extract relevant fields from the DataFrame for plotting.
lats = city_data_df["Lat"]
max_temps = city_data_df["Max Temp"]
humidity = city_data_df["Humidity"]
cloudiness = city_data_df["Cloudiness"]
wind_speed = city_data_df["Wind Speed"]
```

Your final scatter plot will need the current date in the title. To add the current date, we will need to import the `time` module, rather than the `datetime` module that we used to convert the date. The `time` module is a standard Python library, so there is no need to install it.

In a new cell in our `API_practice.ipynb` file, we will import the time module and some code to practice how to use this module. Add the following code in a new cell and run the cell.

```
# Import the time module.
import time
# Get today's date in seconds.
```

```
today = time.time()
today
```

When we call the `time()` function with the time module, we get the output of today's time in seconds since January 1, 1970, as a floating-point decimal number.

```
# Get today's date in seconds.
today = time.time()
today
```

```
1565377180.5886168
```

> **Note**
> Your time will be different when you run the code.

The format for time appears like the datetime stamp for the JSON weather data. We can convert this using the string format method, `strftime()` and pass the formatting parameters for our date in parentheses. To get the format for today, we can add `%x` inside the parentheses.

In the `API_practice` file, add `strftime("%x")` to the time module for our `today` variable and run the cell. The output will be today's date.

```
today = time.strftime("%x")
today
```

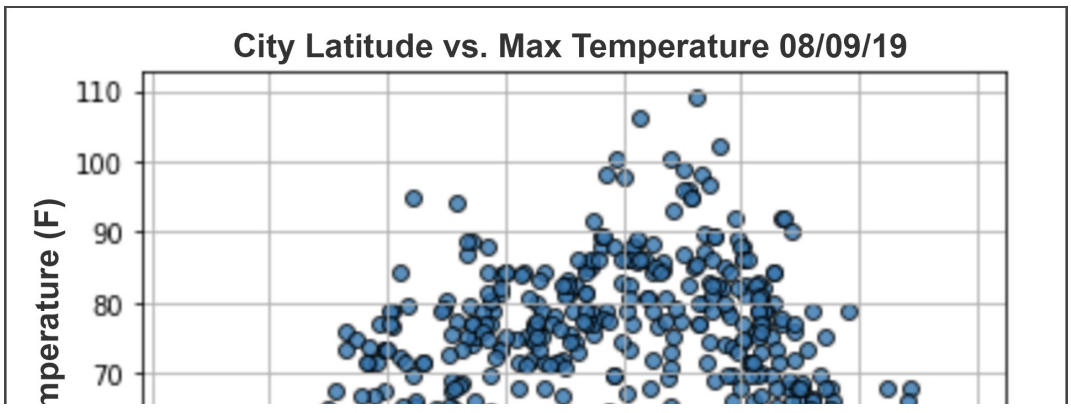
```
'08/09/19'
```

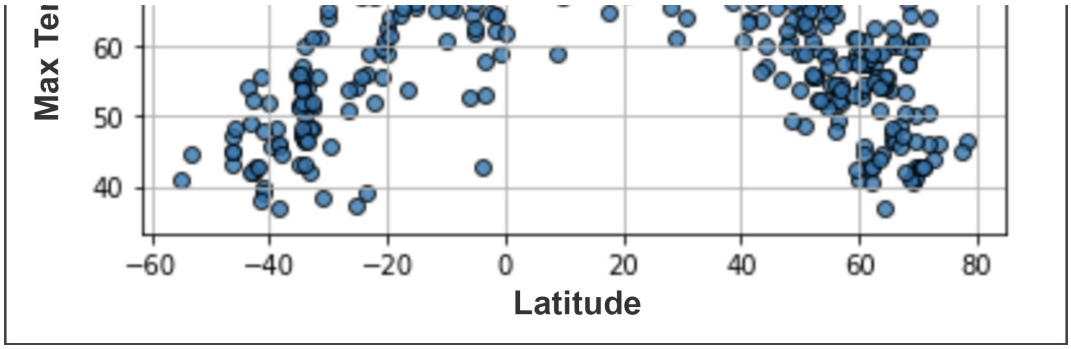
Now, we can add `time.strftime("%x")` to our `plt.title()` function in our scatter plot.

In a new cell, add the following code to create a scatter plot for the latitude vs. maximum temperature and run the cell.

```
# Build the scatter plot for latitude vs. max temperature.
plt.scatter(lats,
            max_temps,
            edgecolor="black", linewidths=1, marker="o",
            alpha=0.8, label="Cities")

# Incorporate the other graph properties.
plt.title(f"City Latitude vs. Max Temperature "+ time.strftime("%
plt.ylabel("Max Temperature (F)")
plt.xlabel("Latitude")
plt.grid(True)

# Save the figure.
plt.savefig("weather_data/Fig1.png")

# Show plot.
plt.show()
```

Our scatter plot will look like the following.





The balance of the scatter plots will share the same format. All we need to do is change the y-axis variable for each weather parameter. Let's create the scatter plots quickly by copying the code and changing the y-axis variable.

> **Note**
>
> For more information, see the **documentation on the time module (https://docs.python.org/3.6/library/time.html#functions)** .

6.3.2: Plot Latitude vs. Humidity

Great job on the first scatter plot! Now, you need to create a scatter plot that compares the latitude vs. the humidity.

We can repurpose our code for the maximum temperature scatter plot and create a scatter plot for the latitude versus humidity.

Add the correct variable to complete the code to create a scatter plot for latitude versus humidity.

```
plt.scatter(lats,
            [          ],
            edgecolor="black", linewidths=1, marker="o",
            alpha=0.8, label="Cities")
```

Check Answer

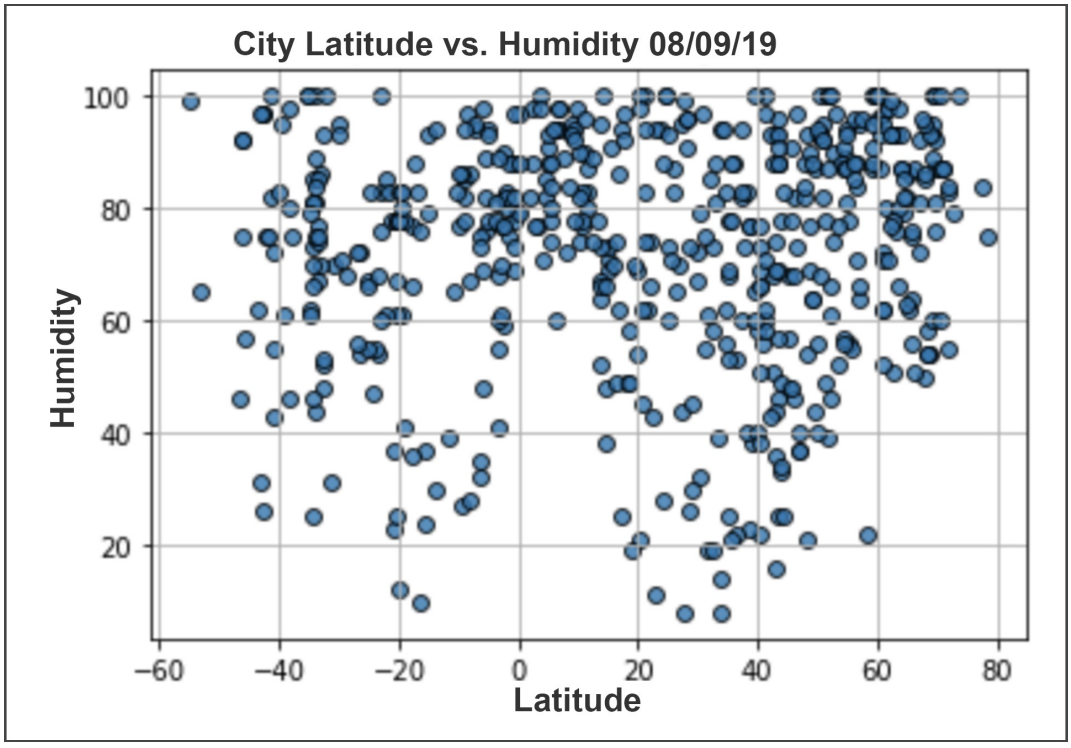
Finish ▸

In addition to changing the y-axis variable to “humidity,” we need to change the title to “Humidity,” and the y-axis label to “Humidity (%).”

In a new cell, add the following code and run the cell.

```
# Build the scatter plots for latitude vs. humidity.
plt.scatter(lats,
            humidity,
            edgecolor="black", linewidths=1, marker="o",
            alpha=0.8, label="Cities")

# Incorporate the other graph properties.
plt.title(f"City Latitude vs. Humidity "+ time.strftime("%x"))
plt.ylabel("Humidity (%)")
plt.xlabel("Latitude")
plt.grid(True)
# Save the figure.
plt.savefig("weather_data/Fig2.png")
# Show plot.
plt.show()
```

Our scatter plot will look like the following:



6.3.3: Plot Latitude vs. Cloudiness

You have a few more plots to create, and you know that it's time to stay hyperfocused. Whenever you start reusing code, it gets a bit easier to make a mistake. These charts will be on a public website for all to see, so the stakes are high.

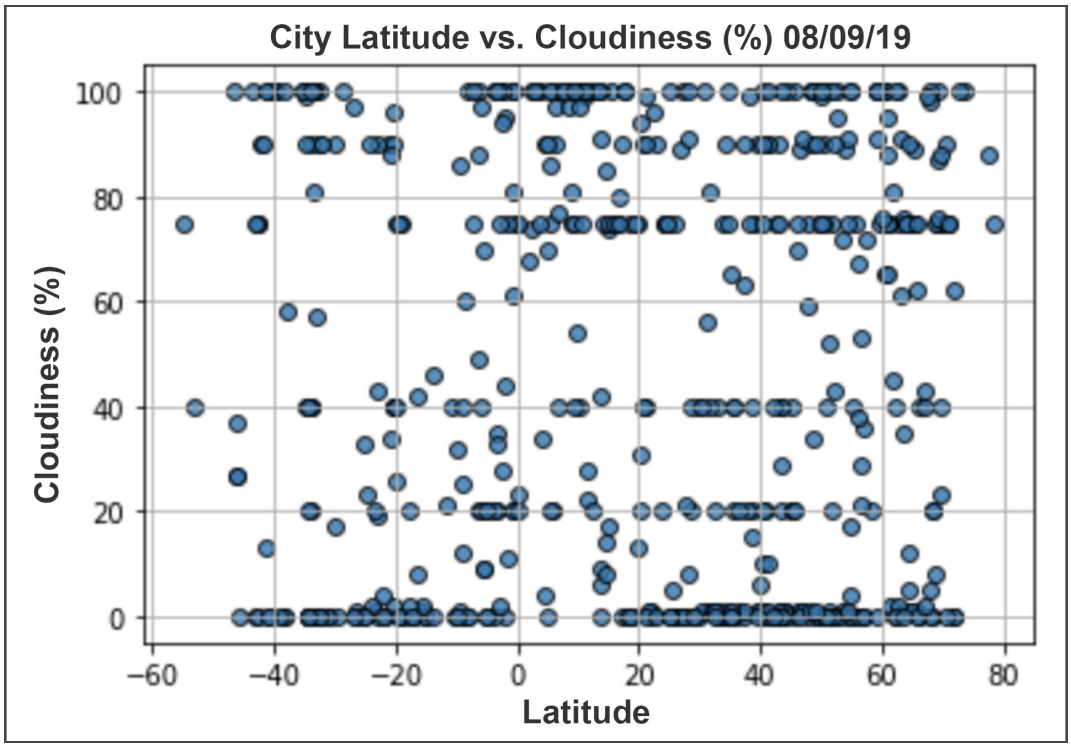
Let's refactor the code for our scatter plots by changing the y-axis variable to "cloudiness," the title to "Cloudiness (%)," and the y-axis label to "Cloudiness (%)."

In a new cell, add the following code and run the cell.

```
# Build the scatter plots for latitude vs. cloudiness.
plt.scatter(lats,
            cloudiness,
            edgecolor="black", linewidths=1, marker="o",
            alpha=0.8, label="Cities")

# Incorporate the other graph properties.
plt.title(f"City Latitude vs. Cloudiness (%) "+ time.strftime("%x
plt.ylabel("Cloudiness (%)")
plt.xlabel("Latitude")
plt.grid(True)
# Save the figure.
plt.savefig("weather_data/Fig3.png")
```

```
# Show plot.
plt.show()
```

Our scatter plot will look like the following.



6.3.4: Plot Latitude vs. Wind Speed

You have one last scatter plot to make. If you can make it through this one, you'll be done with scatter plots for the day. Just be sure to upload them to GitHub so the team working on the STEM project can get them for the community outreach website!

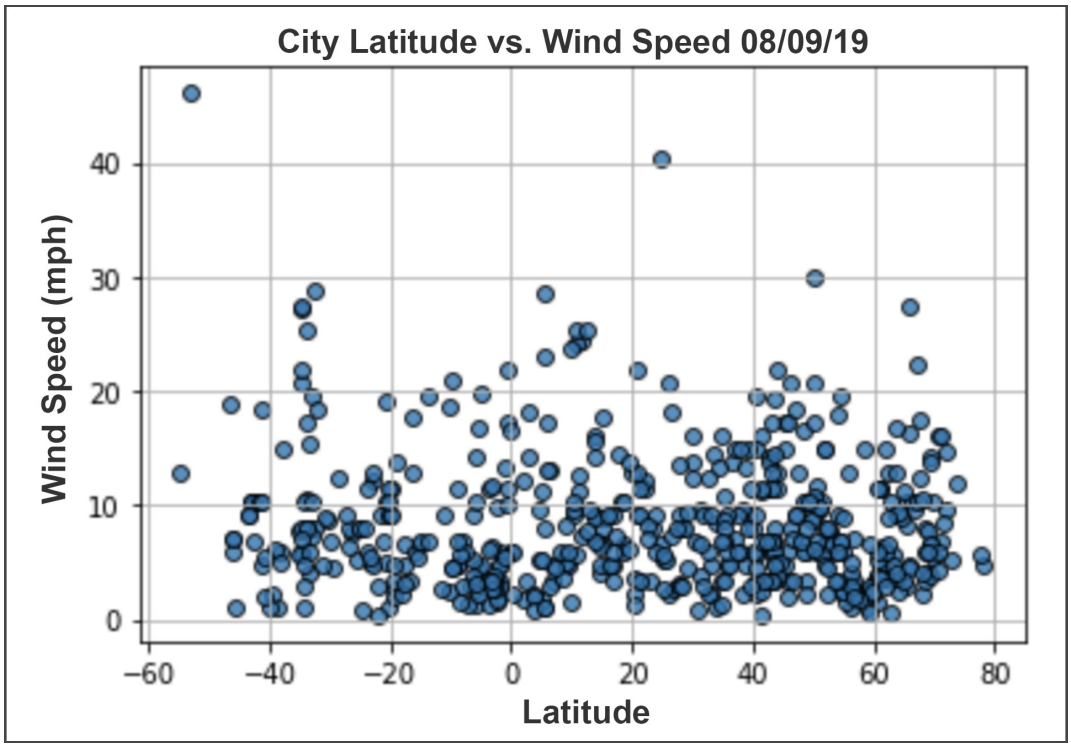
Now, we can create our last scatter plot! Let's repurpose the code we have been using and change the y-axis variable to "wind speed," the title to "Wind Speed," and the y-axis label to "Wind Speed (mph)."

In a new cell, add the following code and run the cell.

```
# Build the scatter plots for latitude vs. wind speed.
plt.scatter(lats,
            wind_speed,
            edgecolor="black", linewidths=1, marker="o",
            alpha=0.8, label="Cities")

# Incorporate the other graph properties.
plt.title(f"City Latitude vs. Wind Speed "+ time.strftime("%x"))
plt.ylabel("Wind Speed (mph)")
plt.xlabel("Latitude")
plt.grid(True)
# Save the figure.
```

```
plt.savefig("weather_data/Fig4.png")
# Show plot.
plt.show()
```

Our scatter plot will look like the following.



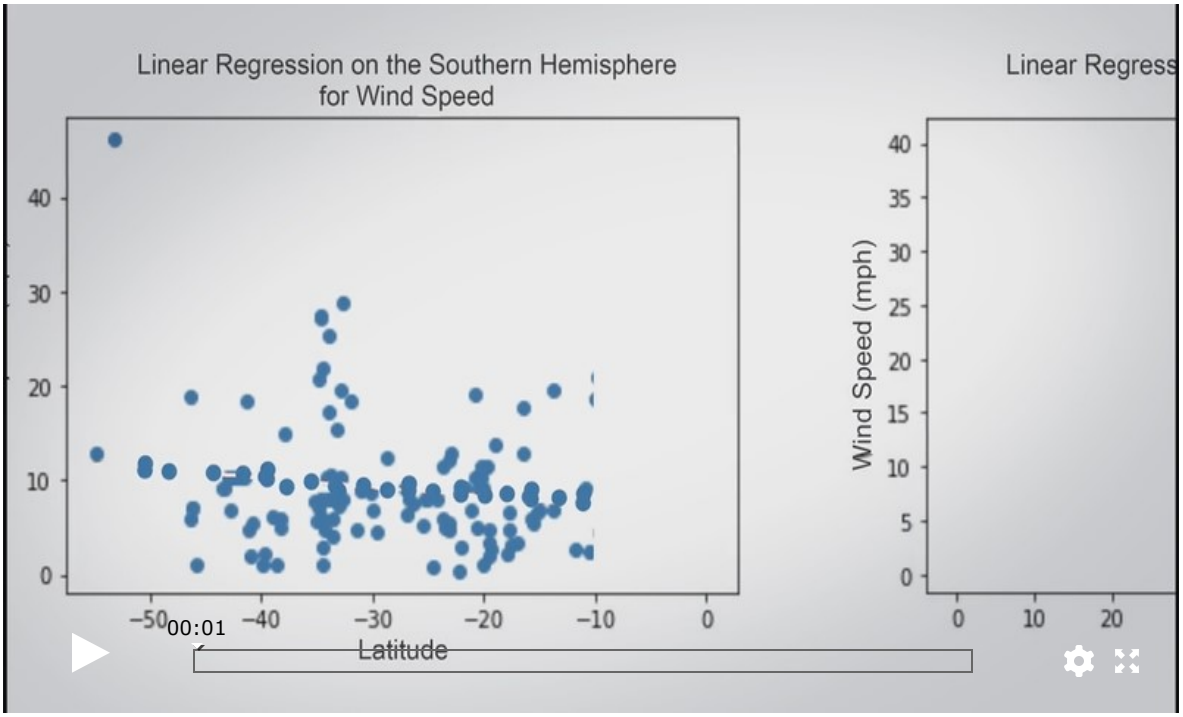
Add, Commit, Push

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

6.4.1: Use Linear Regression to Find the Relationship Between Variables

Jack loves your work on the scatter plots thus far, and the other tech companies were impressed, too. They ask you to put together something that can help the students explore how to determine correlations between weather data and latitude.

You and Jack decide to divide and conquer this task. He will write the piece on the scientific method and describe how to use linear regression on the scatter plots. You volunteer to use your scatter plot skills to create scatter plots for each weather parameter on the Northern and Southern Hemispheres. You'll need to also add a regression line equation and correlation coefficient to each scatter plot, so your first step is to brush up on linear regression. You know this is something Jack uses regularly, so you'll need to really understand it if you want to continue impressing the team.



Linear regression is used to find a relationship between a dependent variable and one or more independent variables. The trick is to find something (a **dependent variable**) that depends on something else (the **independent variable**) and plot that relationship.

For example, let's say we wanted to understand how weather affects ice cream sales. We would model the linear regression between temperature (the independent variable) and ice cream sales (the dependent variable). Our hypothesis about the relationship would be that as temperatures rise, as they do in summer, more ice cream is sold. We will learn more hypothesis testing and building models in later modules.

For your project, you've already been working with independent and dependent variables. We have enough data to test relationships by creating scatter plots as we've done for each weather parameter vs. latitude. Plotting the data is the first step in determining if there might be an association between the two variables. For our scatter plots, the independent variable is the latitude, plotted on the x-axis, as its value is

fixed. When we change the latitude, temperature changes, making it the dependent variable.

> **IMPORTANT**
> **Independent variable:** the variable changed by the analyst to observe how it affects the dependent variable
>
> **Dependent variable:** the variable tested by the analyst to observe how it is affected by the independent variable

To determine if maximum temperature correlates to latitude, we can plot a linear regression line, a straight trendline predicting the average y-value, or dependent variable, for a given x-value, or independent variable. This line can be plotted using the equation y = mx + b, where "m" is the slope of the line and "b" is the y-intercept. For every x-value, or latitude we use in the equation, we will get a predicted temperature value.

To determine how strong the relationship is between the fitted line and the data, we find the correlation coefficient, or r-value. A correlation coefficient close to 1 shows a strong positive correlation, whereas close to –1 shows a strong negative correlation. A correlation coefficient close to zero is no correlation.

Let's practice using linear regression on fake weather data.

Practice Using Linear Regression

In a new cell of our `random_numbers` Jupyter Notebook file, we'll import the linear regression function from the SciPy statistics module.

```
# Import linear regression from the SciPy stats module.
from scipy.stats import linregress
```

Next, we'll generate random latitudes as we did earlier. However, this time, latitudes will be in the Northern Hemisphere and therefore positive. In addition, we'll generate an equal number of random temperatures. Add the following list to a new cell and run the cell.

```
# Create an equal number of latitudes and temperatures.
lats = [42.5, 43.9, 8.1, 36.8, 79.9, 69.1, 25.7, 15.3, 12.7, 64.5
temps = [80.5, 75.3, 90.9, 90.0, 40.4, 62.3, 85.4, 79.6, 72.5, 72
```

Next, use the `linregress` function to calculate the slope, y-intercept, correlation coefficient (r-value), *p*-value, and standard deviation, and then we'll print out the equation for the line.

```
# Perform linear regression.
(slope, intercept, r_value, p_value, std_err) = linregress(lats,
# Get the equation of the line.
line_eq = "y = " + str(round(slope,2)) + "x + " + str(round(inter
print(line_eq)
print(f"The p-value is is: {p_value:.3f}")
```

In the code to perform linear regression, the `linregress` function takes only two arguments, the x- and y-axes data (`lats` and `temps`) in the form of arrays. And it returns the following:

- Slope of the regression line as `slope`
- y-intercept as `intercept`
- Correlation coefficient as `r_value`

- *p*-value as `p_value`
- Standard error as `std_err`

> **Important**
> The `slope`, `intercept`, `r_value`, `p_value`, and `std_err` are always returned when we run the `linregress` function. If you don't want to calculate one of these values but do not add it inside the parentheses, you'll get a `ValueError: too many values to unpack`.
>
> To prevent this error, add a comma and underscore for each value you don't want to calculate.
>
> For instance, if you don't want to print out the p-value and the standard error, write your function as `(slope, intercept, r_value, _, _) = linregress(x, y)`.

When we run the cell, we get the equation of a line for the data and the calculated probability (*p*-value).

```
y = -0.45x + 92.94
The p-value is: 0.011
```

> **Note**
> In statistics, the p-value is used to determine significance of results. In most cases, data scientists like to use a significance level of 0.05, which means:
>
> A linear regression with a p-value > 0.05 is not statistically

> significant.
>
> A linear regression with a p-value < 0.05 is statistically significant.
>
> P-values can also be used to justify rejecting a null hypothesis. We will discuss *p*-values and hypothesis testing in more detail later in the course.

Now we can calculate the ideal temperatures (y-values) using the slope and intercept from the equation of the regression line. To do this, perform list comprehension on the latitudes by multiplying each latitude by the slope and adding the intercept.

Add the following code in a new cell and run the cell.

```
# Calculate the regression line "y values" from the slope and int
regress_values = [(lat * slope + intercept) for lat in lats]
```

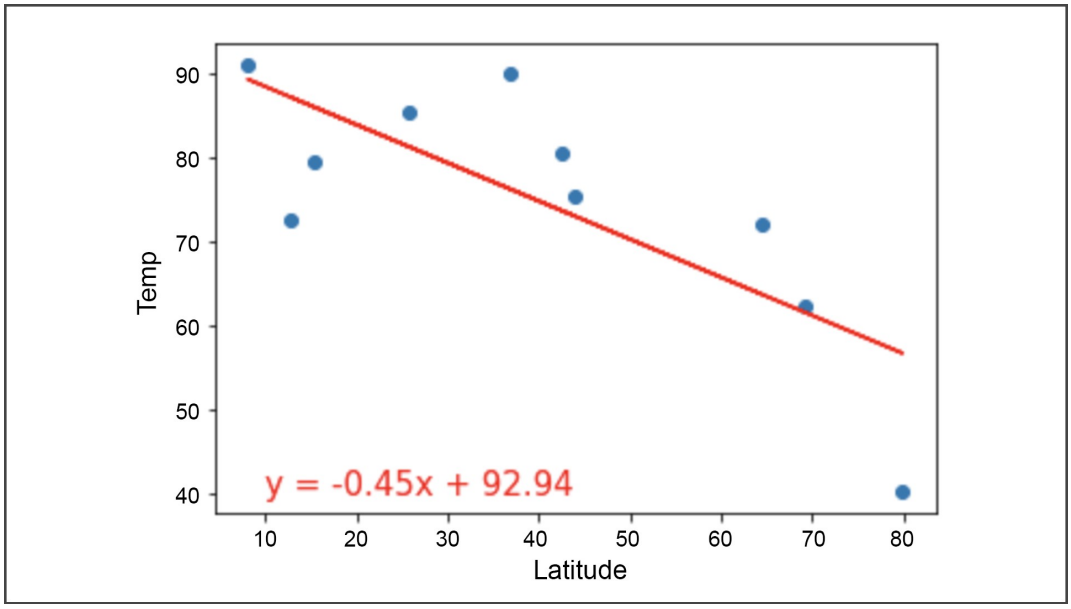
Add the following code to a new cell and run the cell to plot our data.

```
# Import Matplotlib.
import matplotlib.pyplot as plt
# Create a scatter plot of the x and y values.
plt.scatter(lats,temps)
# Plot the regression line with the x-values and the y coordinate
plt.plot(lats,regress_values,"r")
# Annotate the text for the line equation and add its coordinates
plt.annotate(line_eq, (10,40), fontsize=15, color="red")
plt.xlabel('Latitude')
plt.ylabel('Temp')
plt.show()
```

Let's review what this code does:

- We plot the latitudes and temperatures on a scatter plot.
- We create a line plot of our regression line with the ideal temperatures.
- We annotate the line plot by adding the equation of our regression line, where the x-axis is 10 and the y-axis is 40, and specify the font and color.
- We create x- and y-axes labels.

Now let's run the cell and plot our data.



Now that you are familiar with how to perform linear regression, let's use our data from the scatter plots.

> **Note**
>
> For more information, see the [documentation on the linear regression function (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html#scipy.stats.linregress)](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html#scipy.stats.linregress) .

6.4.2: Find the Correlation Between Latitude and Maximum Temperature

Now that you are familiar with generating linear regression lines and equations, you can put this knowledge to work by generating a regression line for latitude and maximum temperature for the Northern and Southern Hemispheres.

You send Jack a quick text to let him know you'll be back to APIs soon—and you're excited to fill him in on what you've learned about this side project.

> Using the data from the Northern and Southern Hemispheres, we are going to perform linear regression on all four weather parameters: maximum temperature, humidity, cloudiness, and wind speed.
>
> We have an algorithm that performs the linear regression; returns the equation of the regression line, and correlation coefficient, and *p* value; and adds the regression line to a scatter plot of city weather data. Below, the code looks like what we have used before.

```
# Perform linear regression.
(slope, intercept, r_value, p_value, std_err) = linregress(x_valu

# Calculate the regression line "y values" from the slope and int
regress_values = x_values * slope + intercept

# Get the equation of the line.
line_eq = "y = " + str(round(slope,2)) + "x + " + str(round(inter

# Create a scatter plot of the x and y values.
plt.scatter(x_values,y_values)
# Plot the regression line with the x-values and the y coordinate
plt.plot(x_values,regress_values,"r")
# Annotate the text for the line equation and add its coordinates
plt.annotate(line_eq, (10,40), fontsize=15, color="red")
plt.xlabel('Latitude')
plt.ylabel('Temp')
plt.show()
```

We will reuse this code with minor changes for each weather parameter in each hemisphere. The variables for each graph are as follows:

1. The *x* values, the latitudes
2. The *y* values, each of the four weather parameters
3. The *y* label, the weather parameter being plotted
4. The x- and y-values given as a tuple, `(10,40)`, for the regression line equation to be placed on the scatter plot.

With only four small changes to the code, this is a great time to convert our linear regression calculation and plotting to a function! In the function,

we can add these four parameters as variables, and when we call the function, pass values to those variables.

Create a Linear Regression Function

In a new cell of our `WeatherPy.ipynb` Jupyter Notebook file, let's create a function,

"plot_linear_regression" and add the four parameters inside the parentheses. Our function should look like the following.

```
# Create a function to create perform linear regression on the we
# and plot a regression line and the equation with the data.
def plot_linear_regression(x_values, y_values, title, y_label, te
```

In our function, we have four parameters: `x_values`, `y_values`, `y_label`, and `text_coordinates`, and we will add a fifth parameter for the title, called, `title`. Now, add the algorithm we use to perform the linear regression underneath the function. Our function should look like the following.

```
# Create a function to create perform linear regression on the we
# and plot a regression line and the equation with the data.
def plot_linear_regression(x_values, y_values, title, y_label, te

    # Run regression on hemisphere weather data.
    (slope, intercept, r_value, p_value, std_err) = linregress(x_

    # Calculate the regression line "y values" from the slope and
    regress_values = x_values * slope + intercept
```

```
    # Get the equation of the line.
    line_eq = "y = " + str(round(slope,2)) + "x + " + str(round(i
    # Create a scatter plot and plot the regression line.
    plt.scatter(x_values,y_values)
    plt.plot(x_values,regress_values,"r")
    # Annotate the text for the line equation.
    plt.annotate(line_eq, text_coordinates, fontsize=15, color="r
    plt.xlabel('Latitude')
    plt.ylabel(y_label)
    plt.show()
```

If we run this code there will be no output until we call the function with five parameters.

> **Rewind**
>
> To get an output from a function, we need to call the function with the correct number of parameters or arguments for the function.

Create the Hemisphere DataFrames

We will add some code to perform regression analysis on the maximum temperatures in the Northern and Southern Hemispheres. To do this, we will need to create Northern Hemisphere DataFrames from the `city_data_df` DataFrame.

To create a new DataFrame from a current DataFrame, we can use the `loc` method on the current DataFrame. The `loc` method accesses a group of rows and columns in the current DataFrame by an index, labels,

or a Boolean array. The syntax to get a specific row from a current DataFrame is `row = df.loc[row_index]`.

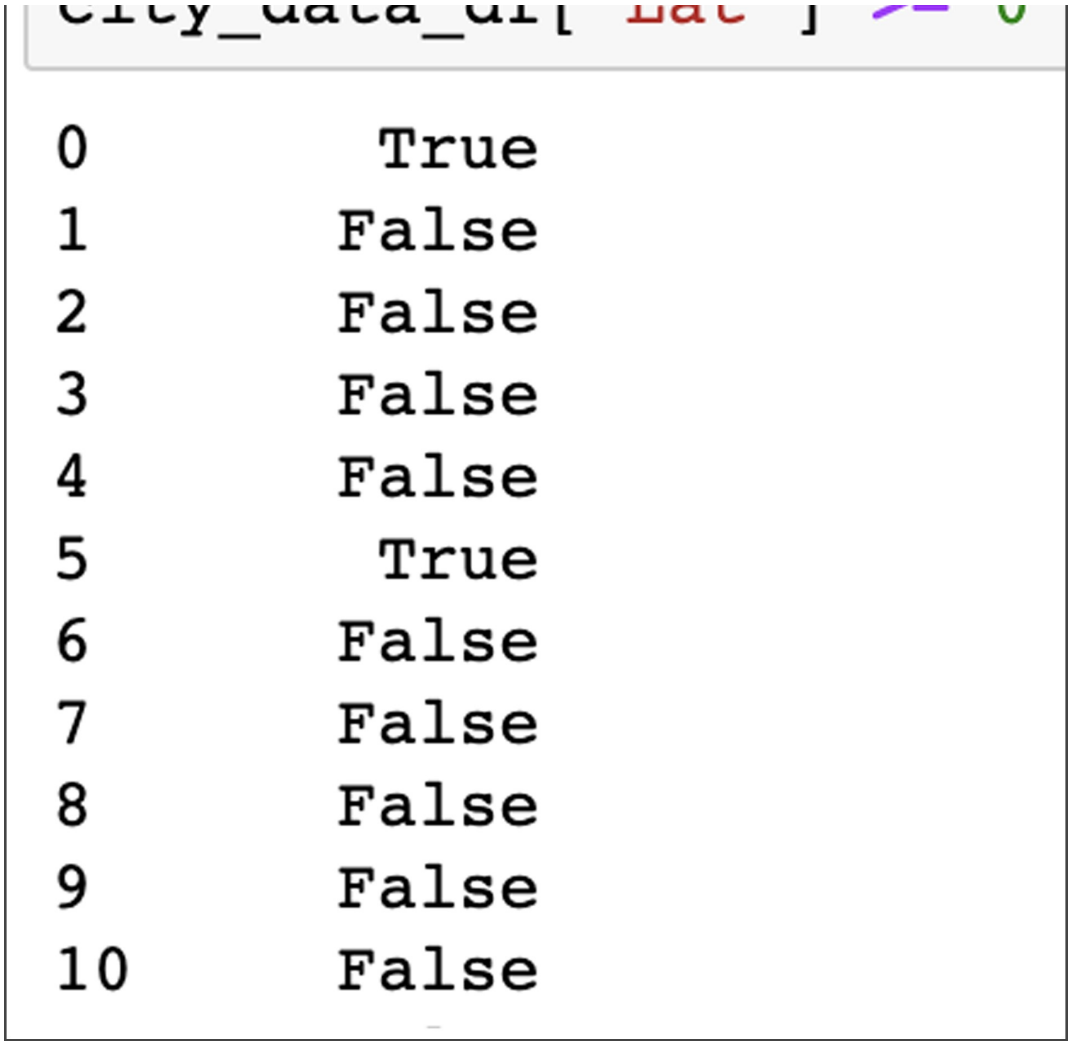
Let's apply this method to our `city_data_df` DataFrame by adding the code `index13 = df.loc[13]` in a cell and running the cell. The output will present all the information at index 13 of the `city_data_df` DataFrame. Note that you may see a different city in your output cell than the one shown in the following image.

```
index13 = city_data_df.loc[13]
index13
```

```
City_ID                      13
City                      Vaini
Country                      IN
Date          2019-08-08 22:26:24
Lat                        15.34
Lng                        74.49
Max Temp                    77.3
Humidity                      91
Cloudiness                   100
Wind Speed                  8.88
Name: 13, dtype: object
```

We can also filter a DataFrame based on a value of a row. For instance, if we wanted to get all Northern Hemisphere latitudes, for latitudes greater than or equal to 0, we can filter the `city_data_df` DataFrame using the code `city_data_df["Lat"] >= 0`. Executing this code will return either "True" or "False" for all the rows that meet these criteria.

```
city_data_df["Lat"] >= 0
```



If we want to return a DataFrame with all data fitting the criteria, for latitudes greater than or equal to 0, we can use the `loc` method on the `city_data_df` DataFrame. Inside the brackets, we would add the conditional filter `city_data_df["Lat"] >= 0` so that our statement would appear as:

```
city_data_df.loc[(city_data_df["Lat"] >= 0)]
```

Also, since this is a DataFrame, we can add the `head()` method at the end to get the first five rows, not counting the row of column headings.

If we execute this code, our output will be the first five rows of a DataFrame.

	City_ID	City	Country	Date	Lat	Lng	Max Temp	Humidity	Cloudiness	Wind Speed
0	0	Tuktoyaktuk	CA	2019-08-08 22:26:20	69.44	-133.03	42.80	87	75	5.82
5	5	Agadez	NE	2019-08-08 22:26:21	16.97	7.99	98.24	25	0	4.79
13	13	Vaini	IN	2019-08-08 22:26:24	15.34	74.49	77.30	91	100	8.88
15	15	Tasiilaq	GL	2019-08-08 22:26:24	65.61	-37.64	46.40	75	1	4.70
16	16	Dingle	PH	2019-08-08 22:26:24	11.00	122.67	75.43	90	99	11.18

Now assign this DataFrame to the variable `northern_hemi_df` to access the data to perform linear regression.

We can take the same approach to get the cities for the Southern Hemisphere by filtering the `city_data_df` DataFrame for latitudes less than 0.

To create DataFrames for the Northern and Southern Hemispheres' data, add the code to a new cell and run the code.

```
# Create Northern and Southern Hemisphere DataFrames.
northern_hemi_df = city_data_df.loc[(city_data_df["Lat"] >= 0)]
southern_hemi_df = city_data_df.loc[(city_data_df["Lat"] < 0)]
```

Now we can perform linear regression on latitude and maximum temperature from each hemisphere DataFrame.

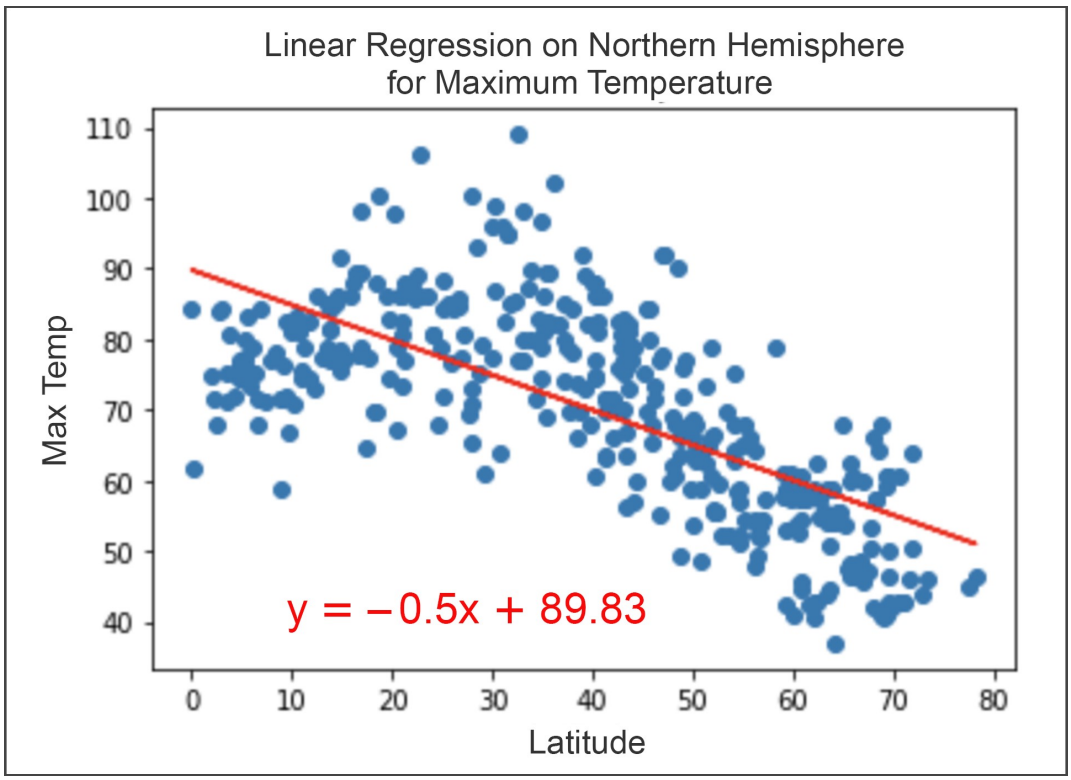
Perform Linear Regression on the Maximum Temperature for the Northern Hemisphere

To generate the linear regression on the maximum temperature for the Northern Hemisphere, we'll need *x* and *y* values. Set the *x* values equal to the latitude column and the *y* values equal to the maximum temperature column from the `northern_hemi_df` DataFrame.

Call the `plot_linear_regression` function with the *x* and *y* values, and edit the `title`, `y_label`, and `text_coordinates` for the maximum temperature scatter plot. Add the code to a new cell and run it to generate the linear regression and plot the data.

```
# Linear regression on the Northern Hemisphere
x_values = northern_hemi_df["Lat"]
y_values = northern_hemi_df["Max Temp"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Northern Hemisph
                        for Maximum Temperature', 'Max Temp',(10,
```

The scatter plot with the regression line and equation should look like the following.



Note

> If the equation for the regression line doesn't show up on your graph, you can change the `text_coordinates` until you see the equation.

Perform Linear Regression on the Maximum Temperature for the Southern Hemisphere

Now we can generate linear regression on the maximum temperature for the Southern Hemisphere.

Complete the code to create the x- and y-values that will be used for the linear regression line and plot for the Southern Hemisphere.

```
x_values =
```

```
y_values =
```

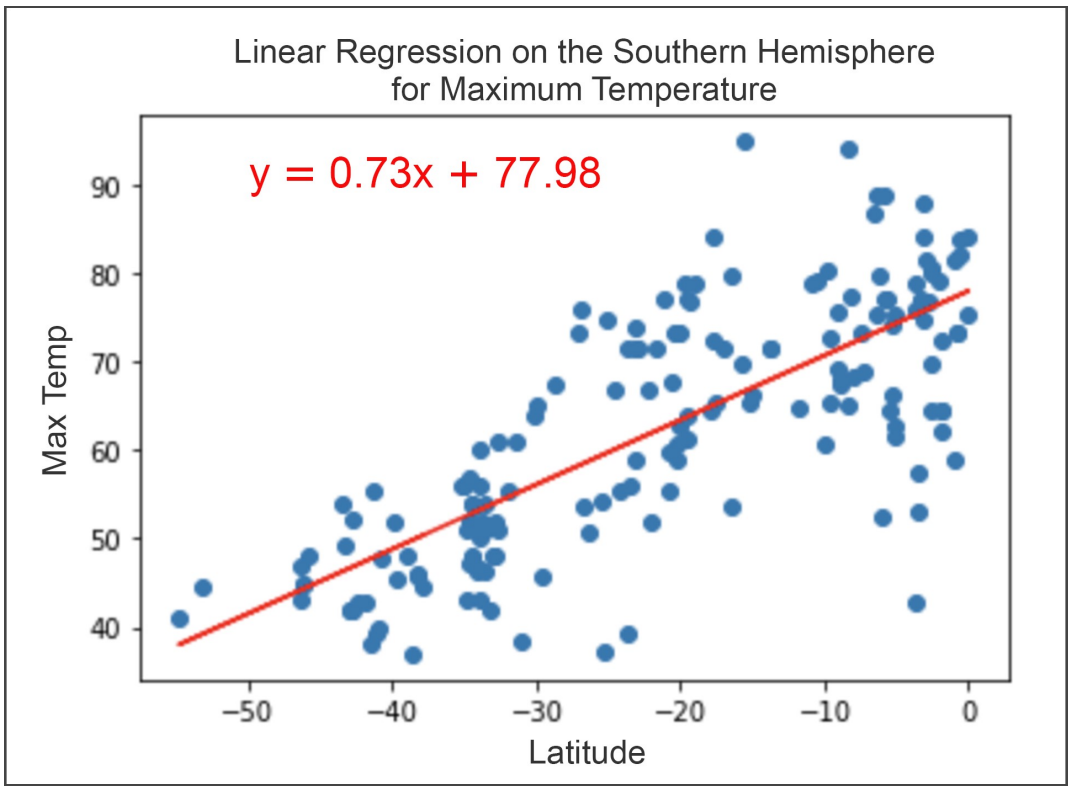
Check Answer

To generate the linear regression on the maximum temperature for the Southern Hemisphere, reuse the code for the Northern Hemisphere and replace the `northern_hemi_df` DataFrame with the `southern_hemi_df` DataFrame to get the x- and y-values.

Call the `plot_linear_regression` function with the x- and y-values, and edit the `title`, `y_label`, and `text_coordinates` for the maximum temperature scatter plot. Add the code to a new cell and run it to generate the linear regression and plot the data.

```
# Linear regression on the Southern Hemisphere
x_values = southern_hemi_df["Lat"]
y_values = southern_hemi_df["Max Temp"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Southern Hemisph
                        for Maximum Temperature', 'Max Temp',(-50
```

The scatter plot with the regression line and equation should look like the following.



Congratulations! You have plotted the regression line and equation for latitude and maximum temperature for your Northern and Southern Hemispheres.

> **Finding**
>
> The correlation between the latitude and the maximum temperature is strong to very strong because the r-value is less than –0.7 for the Northern Hemisphere and greater than 0.7 for the Southern Hemisphere, as shown by the plots here. This means that as we approach the equator, 0° latitude, the temperatures become warmer. And when we are further from the equator the temperatures become cooler. Check the r-values for your plots.

6.4.3: Find the Correlation Between Latitude and Percent Humidity

You have a few more plots with regression lines to create. It's easier to make a mistake when you are copying and pasting code over and over, so stay focused! Now you'll create the linear equation and scatter plot of the latitude and percent humidity for the Northern and Southern Hemispheres.

Using the `plot_linear_regression` function, we can generate the regression lines on the percent humidity for the Northern and Southern Hemispheres.

Perform Linear Regression on the Percent Humidity for the Northern Hemisphere

Complete the code to create the x- and y-values and call the `plot_linear_regression` function with the arguments that will be used for the linear regression line and plot on the Northern Hemisphere.

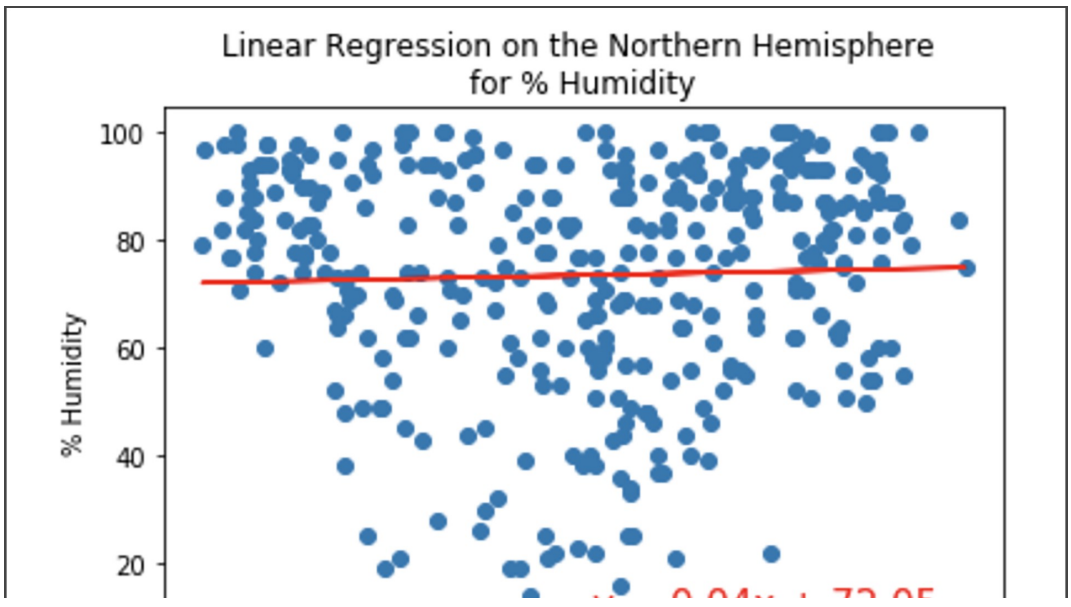
To perform the linear regression on the percent humidity for the Northern Hemisphere, set the x-value equal to the latitude column and y-value equal to the Humidity column from the `northern_hemi_df` DataFrame.

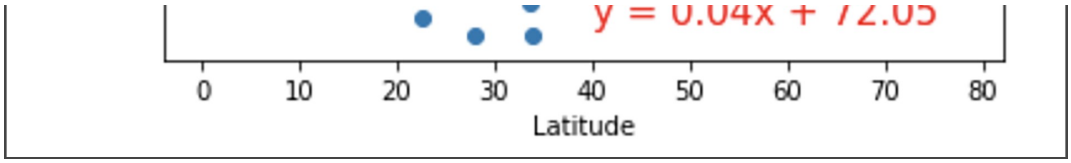
Call the `plot_linear_regression` function with the x- and y-values, and edit the `title`, `y_label`, and `text_coordinates` for the percent humidity scatter plot.

Add the code to a new cell and run it to generate the linear regression and plot the data.

```
# Linear regression on the Northern Hemisphere
x_values = northern_hemi_df["Lat"]
y_values = northern_hemi_df["Humidity"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Northern Hemisph
                        for % Humidity', '% Humidity',(40,10))
```

The scatter plot with the regression line and equation should look like the following.





Perform Linear Regression on the Percent Humidity for the Southern Hemisphere

Next, we will perform linear regression on the percent humidity and latitudes for the Southern Hemisphere.

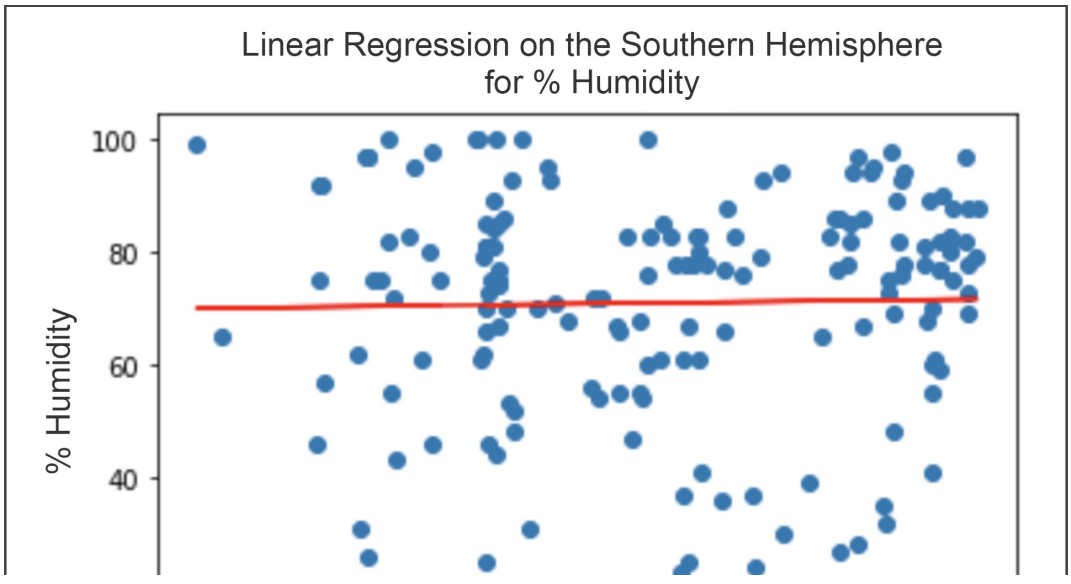
For our linear regression line and plot of the percent humidity and latitudes for the Southern Hemisphere, set the x-value equal to the latitude column and y-value equal to the humidity column from the `southern_hemi_df` DataFrame.

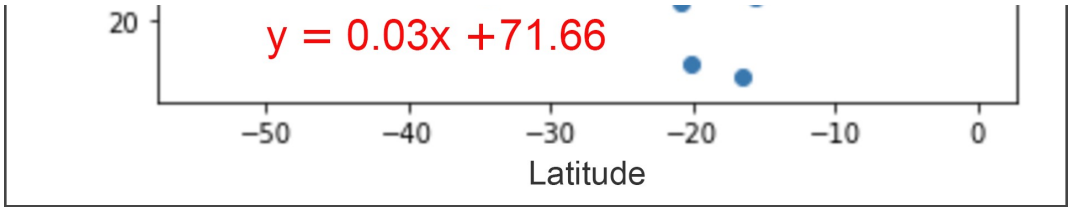
Call the **`plot_linear_regression`** function, with the x-and y-values, and edit the `title`, `y_label`, and `text_coordinates` for the percent humidity scatter plot.

Add the code to a new cell and run it to generate the linear regression and plot the data.

```
# Linear regression on the Southern Hemisphere
x_values = southern_hemi_df["Lat"]
y_values = southern_hemi_df["Humidity"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Southern Hemisph
                        for % Humidity', '% Humidity',(-50,15))
```

The scatter plot with the regression line and equation should look like the following.





Finding

The correlation between the latitude and percent humidity is very low because the r-value is less than 0.04 for the Northern and Southern Hemispheres for the plots shown here. This means that percent humidity is unpredictable due to changing weather patterns that can increase or decrease percent humidity. Check the r-values for your plots.

6.4.4: Find the Correlation Between Latitude and Percent Cloudiness

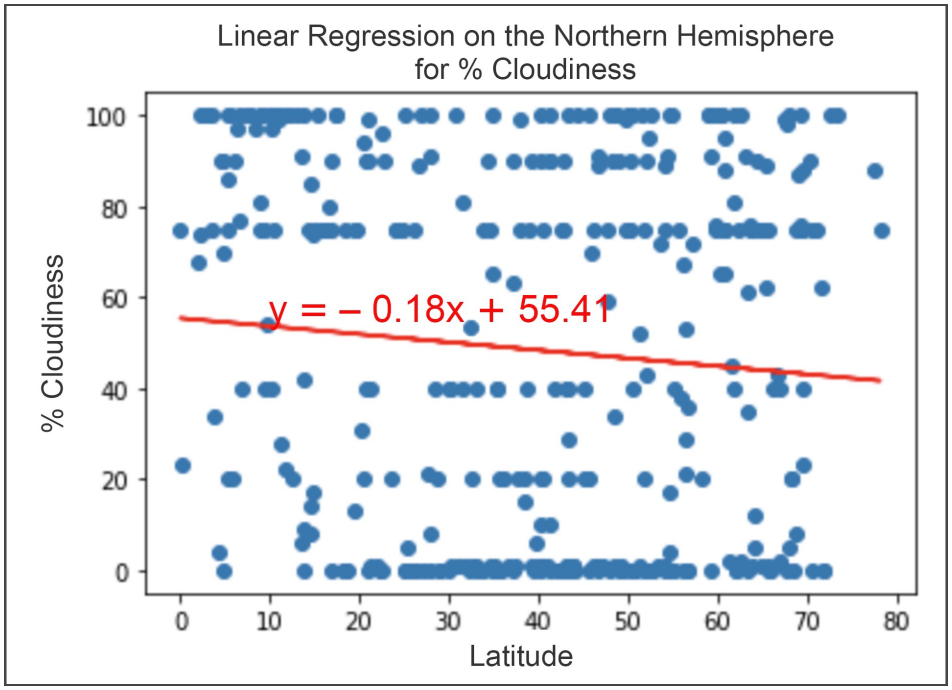
You're almost done creating the linear regression line and plots. This time, create the linear equation plot on the latitude and percent humidity for the Northern and Southern Hemispheres. Stay focused so you don’t make any mistakes when you refactor the code.

Being able to write a function and call it with the appropriate data is a valuable skill. Let’s knock out another regression line and plot. This time we’ll get the data for the percent cloudiness for the Northern and Southern Hemispheres.

Perform Linear Regression on the Percent Cloudiness for the Northern Hemisphere

> **Skill Drill**

1. Refactor the code we have been using for linear regression lines and plots to create the x- and y-values for the percent cloudiness and latitudes on the Northern Hemisphere DataFrame.
2. Call the `plot_linear_regression` function with the correct arguments to create the linear regression line and plot for percent cloudiness in the Northern Hemisphere. It should look like the following plot.



Were you successful in creating the linear regression line and plot for percent cloudiness in the Northern Hemisphere?

- ○ Yes
- ○ No

Check Answer

Finish ▸

Perform Linear Regression on the Percent Cloudiness for the Southern Hemisphere

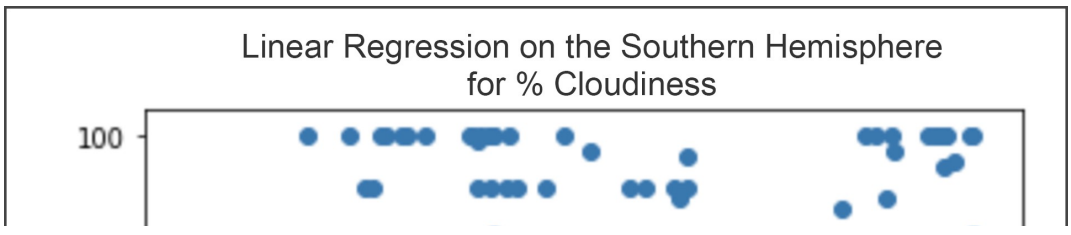
Now we'll create the linear regression line and plot for percent cloudiness in the Southern Hemisphere.

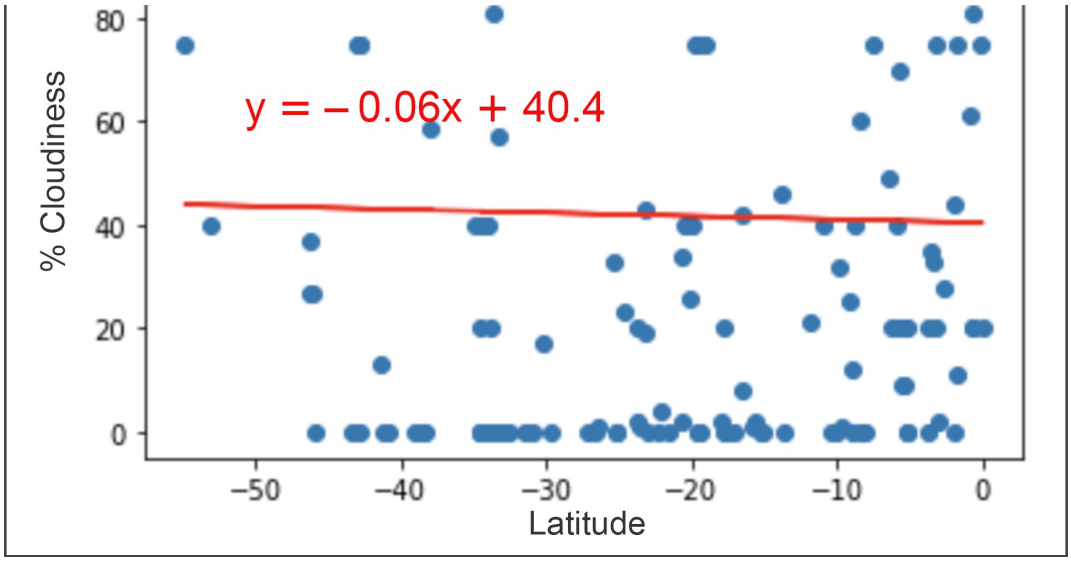
Generate the x-value equal to the latitude column and the y-value equal to the cloudiness column from the `southern_hemi_df` DataFrame. Call the `plot_linear_regression` function. Be sure to edit the `title`, `y_label`, and `text_coordinates` for the percent cloudiness scatter plot.

Add the code to a new cell and run it to generate the linear regression and plot the data.

```
# Linear regression on the Southern Hemisphere
x_values = southern_hemi_df["Lat"]
y_values = southern_hemi_df["Cloudiness"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Southern Hemisph
                        for % Cloudiness', '% Cloudiness',(-50,60
```

The scatter plot with the regression line and equation should look like the following.





Finding

The correlation between the latitude and percent cloudiness is very low because the r-value is less than −0.09 for the Northern Hemisphere and less than −0.02 for the Southern Hemisphere for the plots shown here. This means that cloudiness is unpredictable due to changing weather patterns that can increase or decrease percent cloudiness. Check the r-values for your plots.

6.4.5: Find the Correlation Between Latitude and Wind Speed

One more linear regression line and plot and you'll be done! This time you need to create the linear equation plot on the latitude and wind speed for the Northern and Southern Hemispheres. Remember to stay focused so you don't make any mistakes when you refactor the code.

By now you're considering how to automate this process. You could write an algorithm to execute the scripts to generate the x- and y-values from the Northern and Southern Hemispheres DataFrames and call the `plot_linear_regression` function. Your manager and CEO would be impressed!

Seeing that we're nearly finished, let's stay focused and knock out the last regression lines and equations for the wind speed for the Northern and Southern Hemispheres.

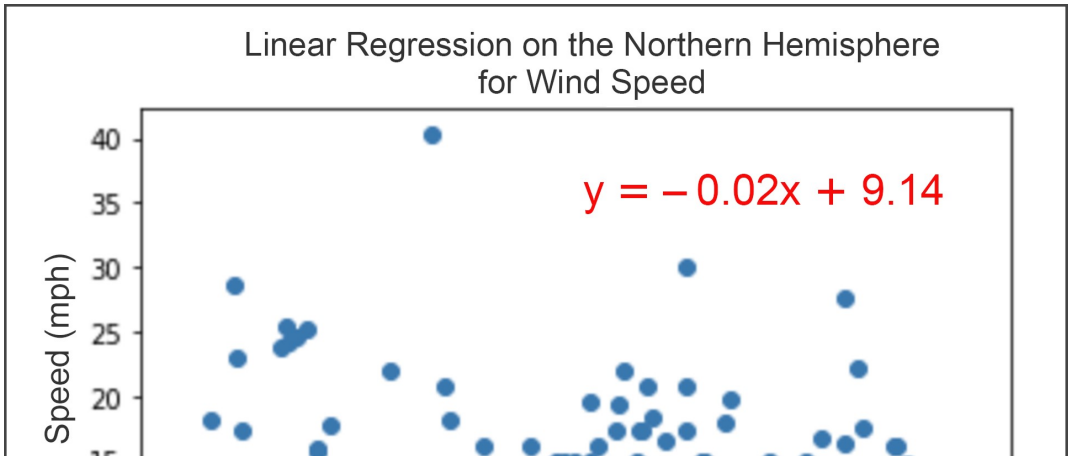
Perform Linear Regression on the Wind Speed for the Northern Hemisphere

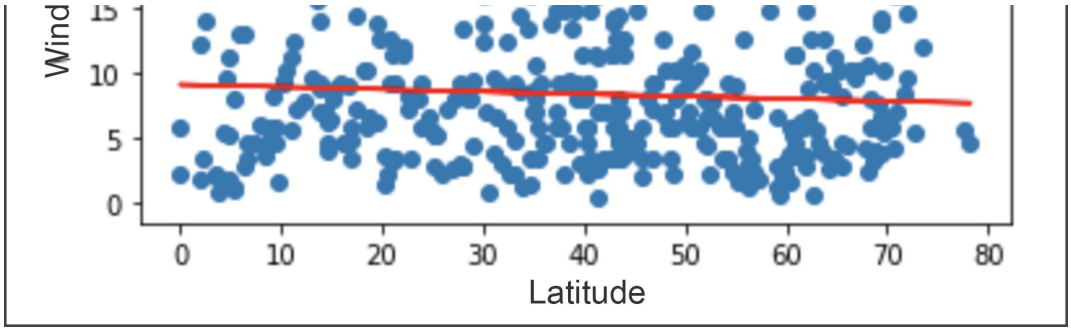
Call the `plot_linear_regression` function, with the x-value equal to the Latitude column and the y-value equal to the Wind Speed column from the `northern_hemi_df` DataFrame. Edit the `title`, `y_label`, and `text_coordinates` for the wind speed scatter plot.

Add the code to a new cell and run it to generate the linear regression and plot the data.

```
# Linear regression on the Northern Hemisphere
x_values = northern_hemi_df["Lat"]
y_values = northern_hemi_df["Wind Speed"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Northern Hemisph
                        for Wind Speed', 'Wind Speed',(40,35))
```

The scatter plot with the regression line and equation should look like the following.





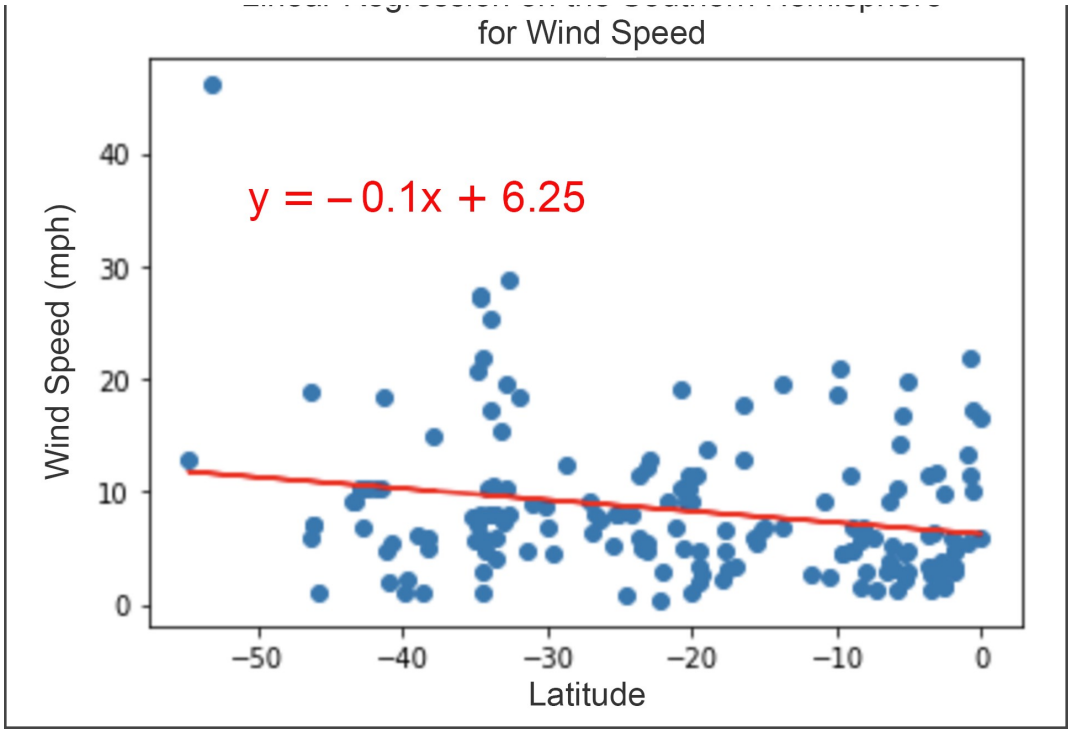
Perform Linear Regression on the Wind Speed for the Southern Hemisphere

Finally, let's call the `plot_linear_regression` function, with the x-value equal to the latitude column and the y-value equal to wind speed column from the `southern_hemi_df` DataFrame. Edit the `title`, `y_label`, and `text_coordinates` for the wind speed scatter plot.

Add the code to a new cell and run it to generate the linear regression and plot the data.

```
# Linear regression on the Southern Hemisphere
x_values = southern_hemi_df["Lat"]
y_values = southern_hemi_df["Wind Speed"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Southern Hemisph
                        for Wind Speed', 'Wind Speed',(-50,35))
```

The scatter plot with the regression line and equation should look like the following.



Finding

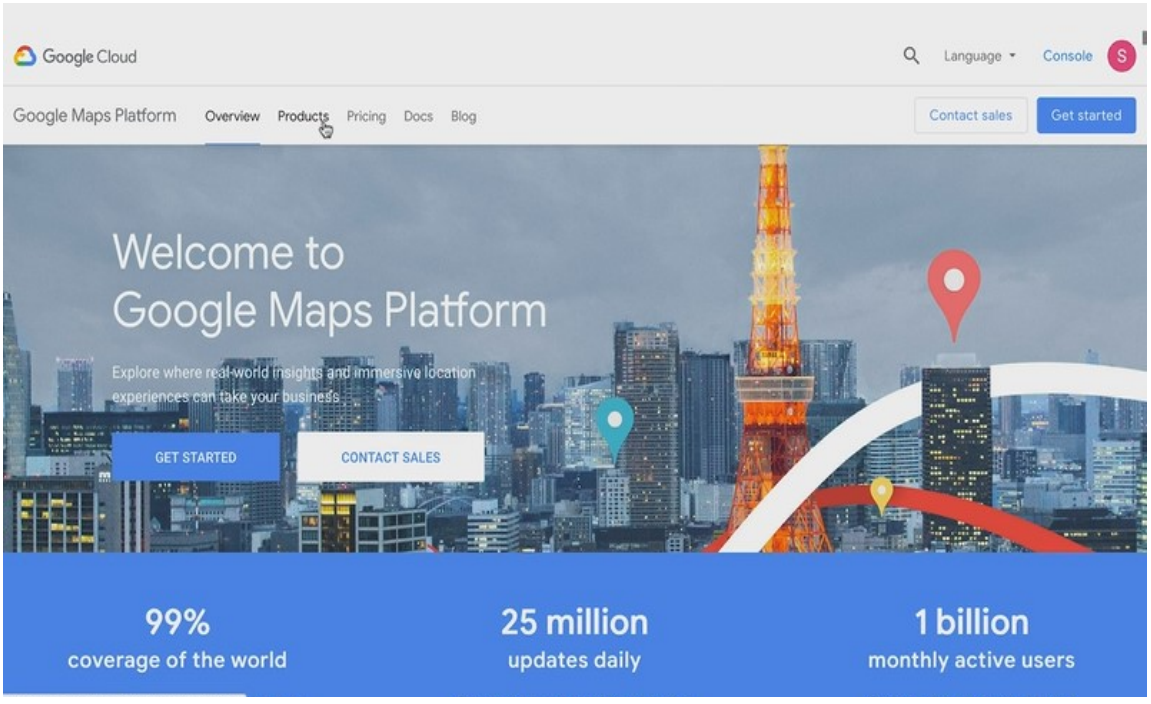
The correlation between the latitude and wind speed is very low because the r-value is less than −0.07 for the Northern Hemisphere and less than −0.3 for the Southern Hemisphere for the plots shown here . This means that wind speed is unpredictable due to changing weather patterns that can increase or decrease wind speed. Check the r-values for your plots.

Add, Commit, Push

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

6.5.1: Set Up Google Maps and Places API

Now that you have finished helping with your company's STEM project, it's time to get back to working on the travel app for customers. You and Jack want this to feel like a really cool, interactive experience, so you decide to create a variety of heatmaps for the weather data on the website, with some interactive dropdowns for additional information. You will need to write the code that uses the Google Maps and Places API that will create each heatmap.



Google makes available some of the vast sets of tools that power Google Maps, so that any developer, such as yourself, can use the same technologies and datasets in their own applications.

These APIs help developers perform the following tasks:

- Convert latitudinal and longitudinal coordinates into locations on a map.
- Create a heatmap based on the density or weight of a feature, such as an earthquake.
- Identify the hotels or restaurants closest to a given location.
- Determine the distance between two points.

To create custom maps with Google, install the gmaps dependency, which is a Jupyter plugin for embedding Google Maps in your notebooks.

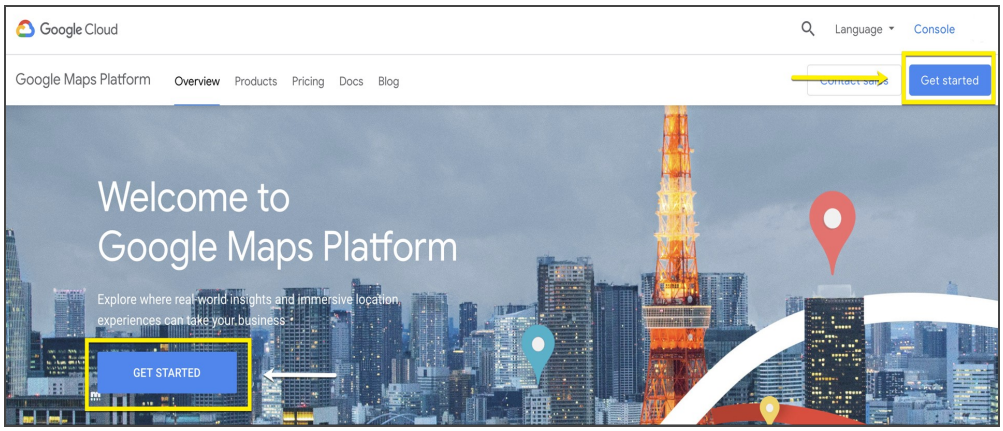
Using gmaps, we'll create heatmaps and location markers for hotels within a certain radius of the cities where our customers travel. Below is an example of the type of heatmap we'll create, covering the now-familiar weather parameters: maximum temperature, percent humidity, percent cloudiness, and wind speed.

To create this heatmap, we'll need to register for a Google Maps and Places API key on the Google Developer site.

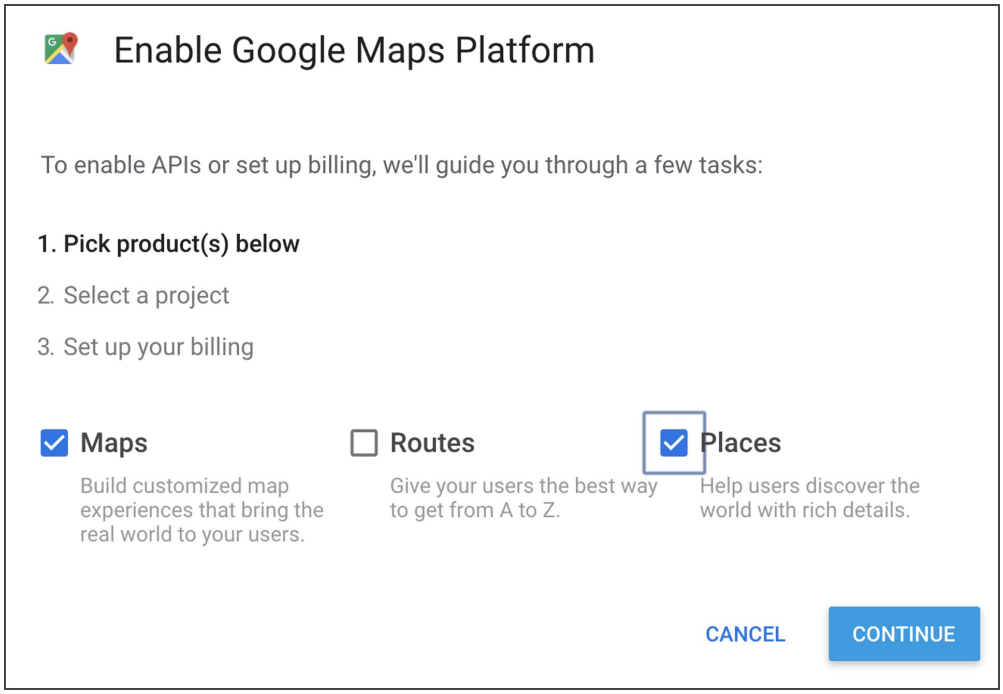
Register for a Google API Key

Follow these steps to register for an API key:

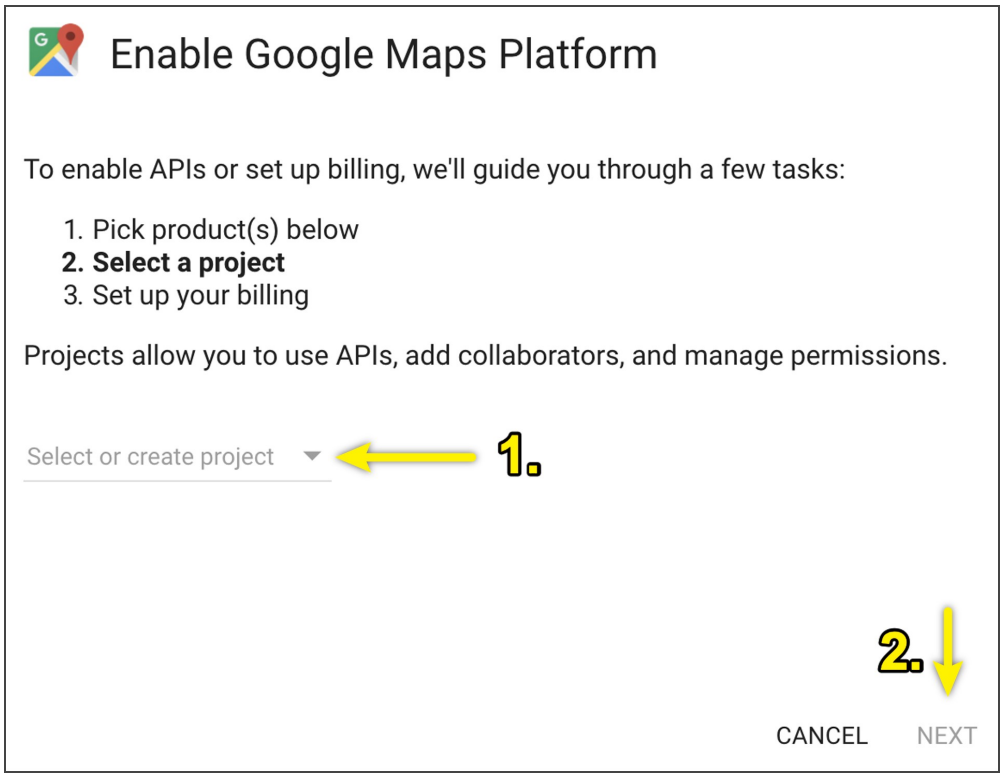
1. Log in to Google with your credentials.
2. Navigate to the [Google Maps Platform page (https://cloud.google.com/maps-platform/)](https://cloud.google.com/maps-platform/) .
3. Click on either “Get Started” box.



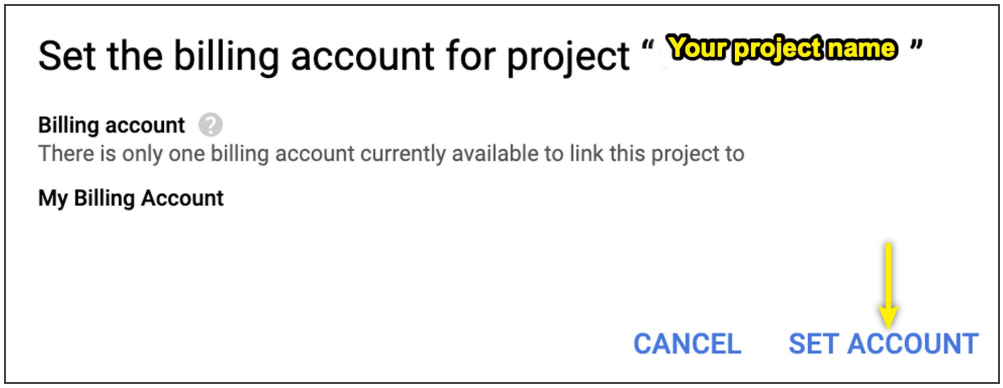
4. Check the “Maps” and “Places” boxes, and then click “Continue.”



5. Create a project name, or select one if you have a project, and then click “Next.”



6. You will have to add a credit card to use the Maps and Places API. Click “Set Account” and follow the instructions.

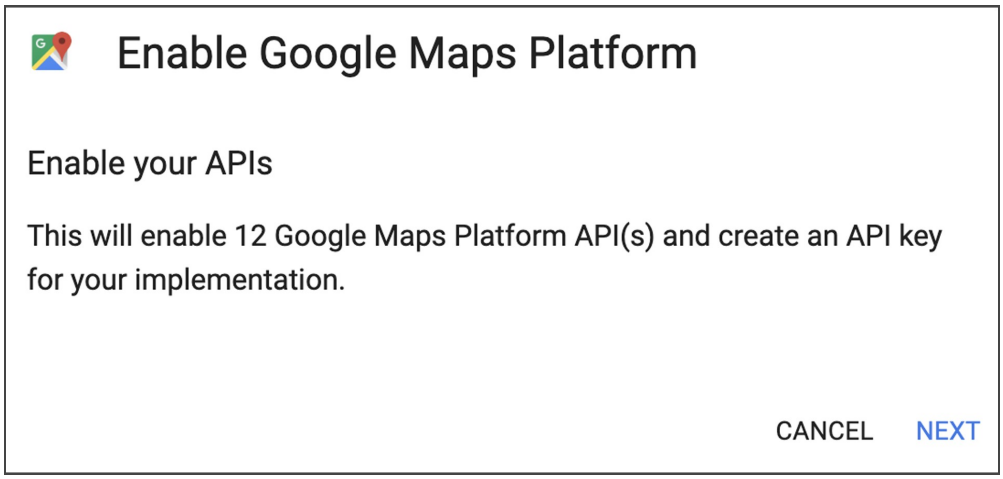


IMPORTANT!

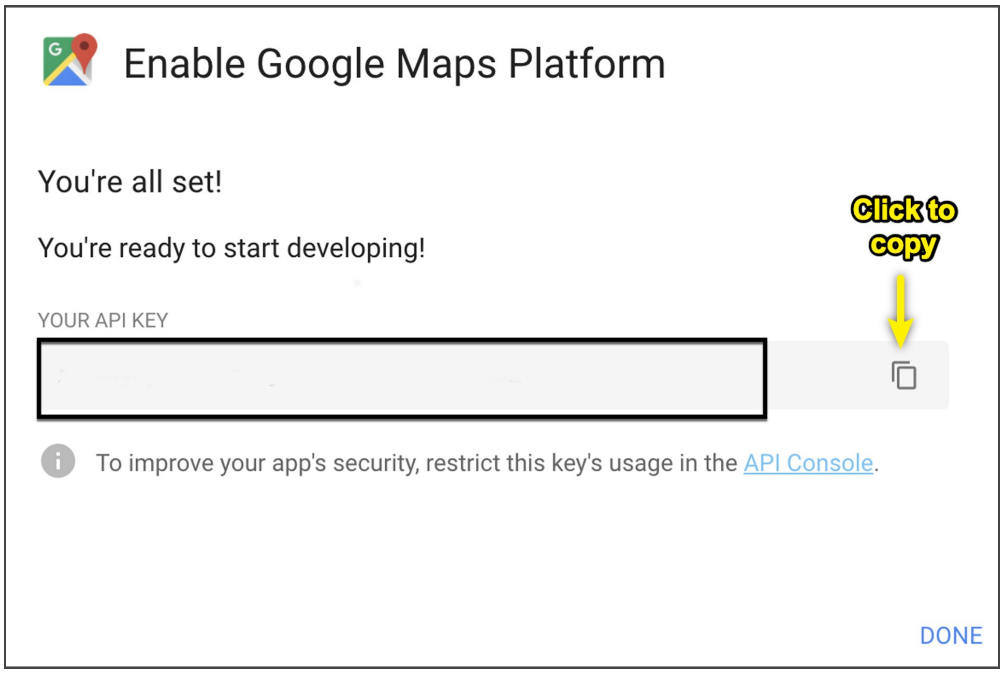
Any API usage beyond the $200 credit will be charged to your account. You must add a credit card to use your Google API key. For more information, read about the [Places API Usage

> **and Billing** (https://developers.google.com/places/web-service/usage-and-billing) .

7. Once you add the billing information, you'll see the following message. Click "Next."



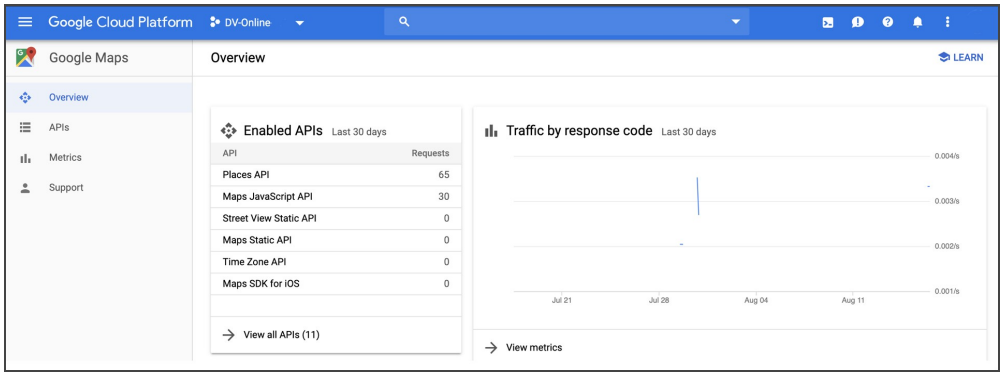
8. You will see your API key. Copy your API key and place it in a safe place, and then click "Done."



9. Add the key to your `config.py` file as `g_key`.

```
# Google API Key
g_key = "your Google API key goes here"
```

10. After you click “Done,” you’ll see your Google Cloud Platform Overview page, where you can track your usage using the menu on the left-hand side.



Next, install the `gmaps` dependency in our PythonData development environment.

Install the gmaps Dependency

To install gmaps, make sure you have activated your Python development environment. Next, add the following into your command line prompt and press Enter.

```
$ conda install -c conda-forge gmaps
```

Alternatively, you can install gmaps using `pip`. Follow these steps:

1. First, make sure you have enabled “ipywidgets” widgets extension with the following command.

```
$ jupyter nbextension enable --py --sys-prefix widgetsnbexter
```

2. Install gmaps with the following command:

```
$ pip install gmaps
```

3. Tell Jupyter to load the `widgetsnbextension` by running the following command:

```
$ jupyter nbextension enable --py --sys-prefix gmaps
```

Now you are ready to use your Google Maps and Places API key!

> **Note**
>
> For more information, see the **documentation on the gmaps dependency (https://jupyter-gmaps.readthedocs.io/en/latest/tutorial.html)**.

6.5.2: Create Heatmaps for Weather Parameters

To implement the heatmap feature on your company's website, you'll need to build out the code for the heatmap and test the feature using the weather data with an API call. Time to write some code.

Creating heatmaps is going to be easy and fun. We'll feel like software developers when we add information to geological maps and can visual the data based on density. The amount of coding needed to create a heatmap is small. For a basic heatmap, all we need to do is provide the following:

- Latitudes and longitudes for the locations
- A measurement value for each latitude and longitude in the form of arrays that have the same number of items in each array

Begin by importing our dependencies and Google API key, and then add our `cities.csv` file to a DataFrame. Create a new Jupyter Notebook file named `VacationPy.ipynb`. In the first cell, add our dependencies and API key.

```
# Import the dependencies.
import pandas as pd
import gmaps
```

```
import requests
# Import the API key.
from config import g_key
```

Let's review how we will use our dependencies. We'll use Pandas to read our CSV file and create the locations and measurements from the DataFrame. We'll use gmaps and the API key to create heatmaps and the locations map, and we'll use the requests dependency to make a request to the Google Places JSON file. This will allow us to get hotel locations from the latitude and longitude of the city.

Next, we'll read our `cities.csv` file into a DataFrame. Add the following code and run it to create the DataFrame.

```
# Store the CSV you saved created in part one into a DataFrame.
city_data_df = pd.read_csv("weather_data/cities.csv")
city_data_df.head()
```

One caveat to using gmaps: The data we use for any mapping must be either an integer or a floating-point decimal number. Let's check the data types for the columns of our DataFrame.

> **Rewind**
> Recall that you use the `dtypes` method to get the data types of a DataFrame.

Confirm the data types for the data columns are integers or floating-point decimal numbers.

```
city_data_df.dtypes
```

```
City_ID          int64
City            object
Country         object
Date            object
Lat            float64
Lng            float64
Max Temp       float64
Humidity         int64
Cloudiness       int64
Wind Speed     float64
dtype: object
```

Create a Maximum Temperature Heatmap

First, tell gmaps to use your API key. You only need to configure gmaps to use your API key once.

Add the following code to a new cell and run the cell.

```
# Configure gmaps to use your Google API key.
gmaps.configure(api_key=g_key)
```

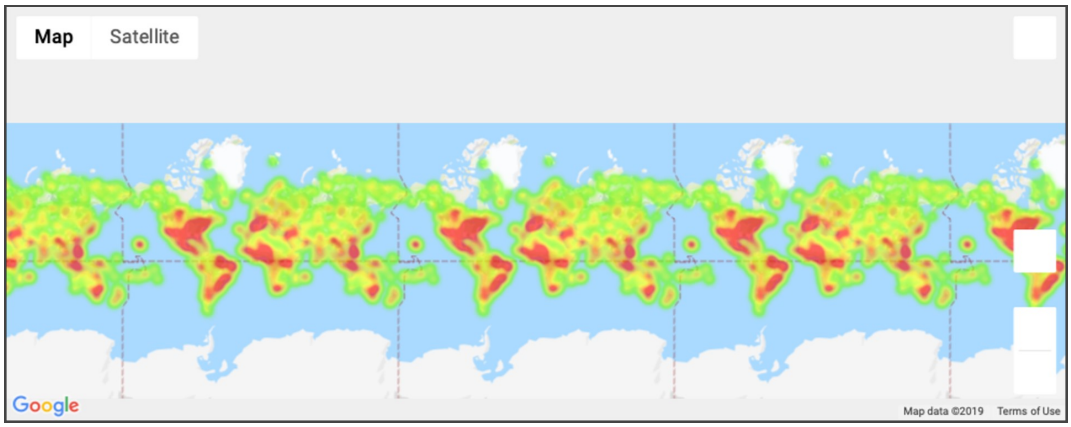
Next, create the heatmap for the maximum temperature. The general syntax for creating a heatmap is as follows.

```
# 1. Assign the locations to an array of latitude and longitude p
locations = [latitude, longitude]
# 2. Assign the weights variable to some values.
temperatures = # an array of equal length of the locations array.
# 3. Assign the figure variable to the gmaps.figure() attribute.
fig = gmaps.figure()
# 4. Assign the heatmap_layer variable to the heatmap_layer attri
heatmap_layer = gmaps.heatmap_layer(locations, weights=temperatur

# 5. Add the heatmap layer.
fig.add_layer(heatmap_layer)
# 6. Call the figure to plot the data.
fig
```

Add the locations array and the values from the maximum temperature from our `city_data_df` DataFrame for Steps 1 and 2 and the rest of the code above.

```
# Heatmap of temperature
# Get the latitude and longitude.
locations = city_data_df[["Lat", "Lng"]]
# Get the maximum temperature.
max_temp = city_data_df["Max Temp"]
# Assign the figure variable.
fig = gmaps.figure()
# Assign the heatmap variable.
heat_layer = gmaps.heatmap_layer(locations, weights=max_temp)
# Add the heatmap layer.
fig.add_layer(heat_layer)
# Call the figure to plot the data.
fig
```

When we run this cell, we get the following large landscape map in the output window.



Google heatmaps do not plot negative numbers. If you have a maximum temperature that is less than 0 °F, then you will get an `InvalidWeightException` error for this line of code:

```
heat_layer = gmaps.heatmap_layer(locations, weights=max_temp)
```

To remove the negative temperatures we can use a `for` loop to iterate through the `max_temp` and add the temperatures that are greater than 0 °F to a new list.

Add a new cell above our previous code block, and then add the following code in the cell and run the cell.

```
# Get the maximum temperature.
max_temp = city_data_df["Max Temp"]
temps = []
for temp in max_temp:
    temps.append(max(temp, 0))
```

In the `for` loop, we're using the `max()` function to get the largest value between the `temp` and 0. If the `temp` is less than 0, then 0 will be added to the list in its place. Otherwise, the `temp` is added to the list.

Now change the following code in the `heat_layer`:

```
heat_layer = gmaps.heatmap_layer(locations, weights=max_temp)
```

To look like the following:

```
heat_layer = gmaps.heatmap_layer(locations, weights=temps)
```

Rerun the cell. You should see a heatmap in the output window.

How would you convert the following for loop using list comprehension?

```
for temp in max_temp:
    temps.append(max(temp, 0))
```

- ○ `[for temp in max_temp max(temp, 0)]`
- ○ `[for temp in max_temp(temps.append(temp)]`
- ○ `[max(temp, 0) for temp in max_temp]`

Check Answer

Finish ▸

Instead of using the `for` loop, we can perform a list comprehension within the `heatmap_layer()` function.

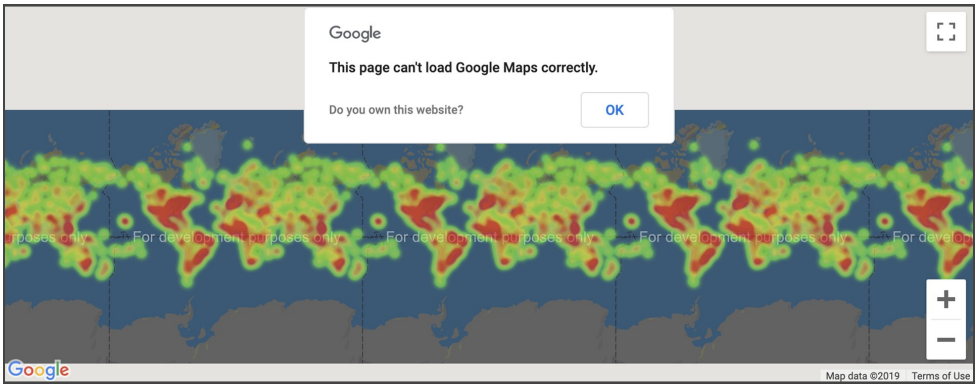
Replace `temps` with our code for the list comprehension so our `heat_layer` code looks like the following:

```
heat_layer = gmaps.heatmap_layer(locations, weights=[max(temp, 0) for temp in max_temp])
```

Rerun the cell.

IMPORTANT!

Does your map looks like the following image with a pop-up message saying “The page can’t load Google Maps correctly”?



The common errors for this are:

- Not configuring gmaps to use your API key with `gmaps.configure(api_key=g_key)`
- Not importing your API key from the correct folder
- Not having or using the correct API key for gmaps
- Not adding a credit card to your account for billing

Other options can be found by clicking on “Do you own this website?” in the pop-up message window.

For more information, see the [documentation on error messages (https://developers.google.com/maps/documentation/javascript/error-messages?utm_source=maps_js&utm_medium=degraded&utm_campaign=keyless#api-key-and-billing-errors)](https://developers.google.com/maps/documentation/javascript/error-messages?utm_source=maps_js&utm_medium=degraded&utm_campaign=keyless#api-key-and-billing-errors) .

This map is too large. We will need to make some adjustments to the `gmaps.figure()` attribute.

Adjust Heatmap Zoom, Intensity, and Point Radius

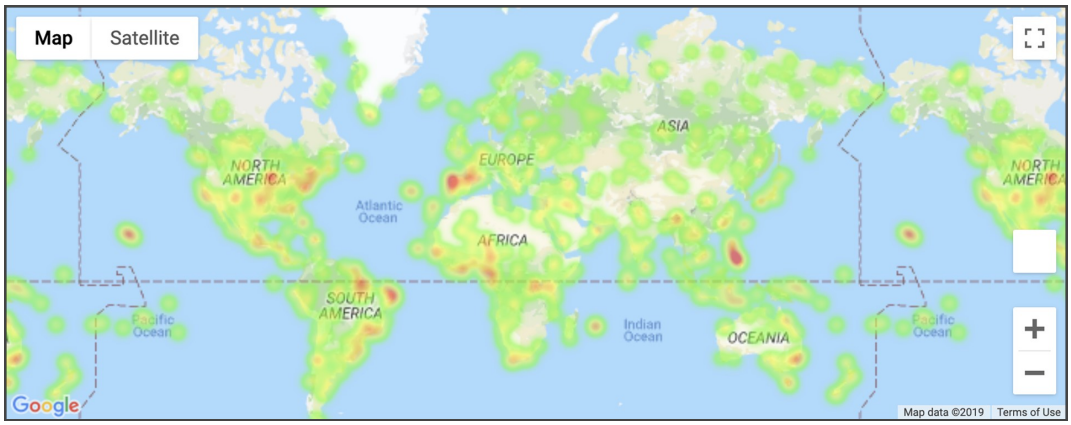
First, add the geographic center of Earth in the form of latitude and longitude (30.0° N and 31.0° E). Also, add a zoom level so that only one map of Earth is shown.

When we add a center and zoom level to the `gmaps.figure()` attribute, it will look like this:

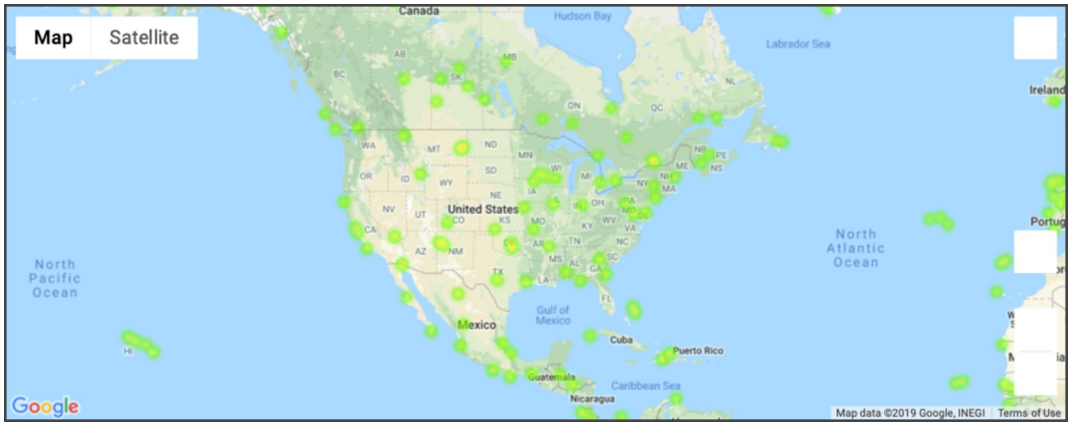
```
fig = gmaps.figure(center=(30.0, 31.0), zoom_level=1.5)
```

You might have to play with the zoom level to get things right.

When we run the cell, our map is centered and shows all the inhabitable places on Earth.



If we scroll and zoom in on North America, we can see that the circles for the temperatures need to be modified so that they are larger and show temperature gradient differences.



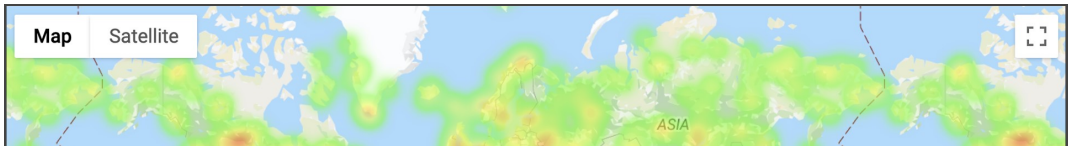
If you review the gmaps documentation, you may notice there is a dissipation option for creating heatmaps that can be added to the `gmaps.heat_layer()` attribute.

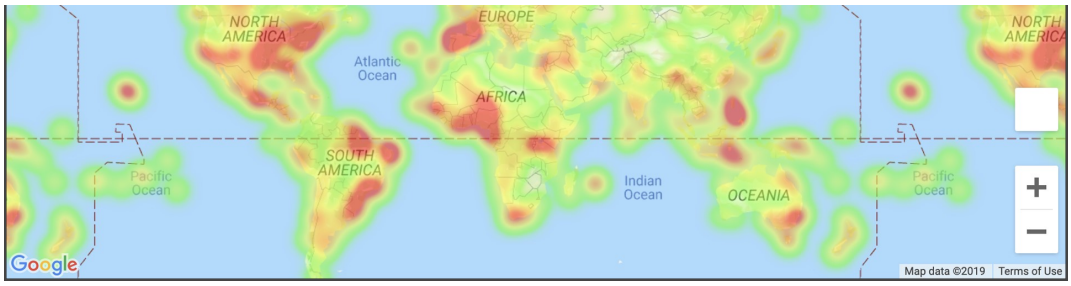
The options for this are:

1. The default option for the dissipation is “True,” so we need to set our “dissipation” to “False.”
2. We can add `max_intensity` to make each measurement have a better gradient variance.
3. We can add `point_radius` to make each measurement radius larger.

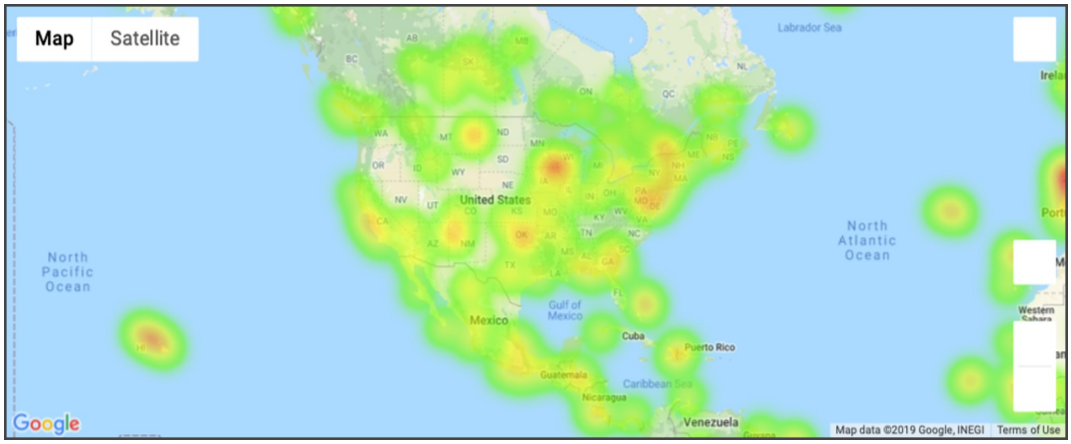
Also, you’ll have to tweak these values until you are satisfied with the final map.

Let’s modify our `gmaps.heat_layer()` attribute to `heat_layer = gmaps.heatmap_layer(locations, weights=temps, dissipating=False, max_intensity=300, point_radius=4)` and rerun our code.





If we scroll and zoom in on North America, we can see that the circles for the temperature are larger and the gradients show differences.



Now our maximum temperature heat map looks a lot better!

> ## Note
>
> If you see the following message in the output window below your code block, when you open your Jupyter Notebook file that had the Google map, you will have rerun the cell to create the map.

```
# Add the heatmap layer
fig.add_layer(heat_layer)

fig
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

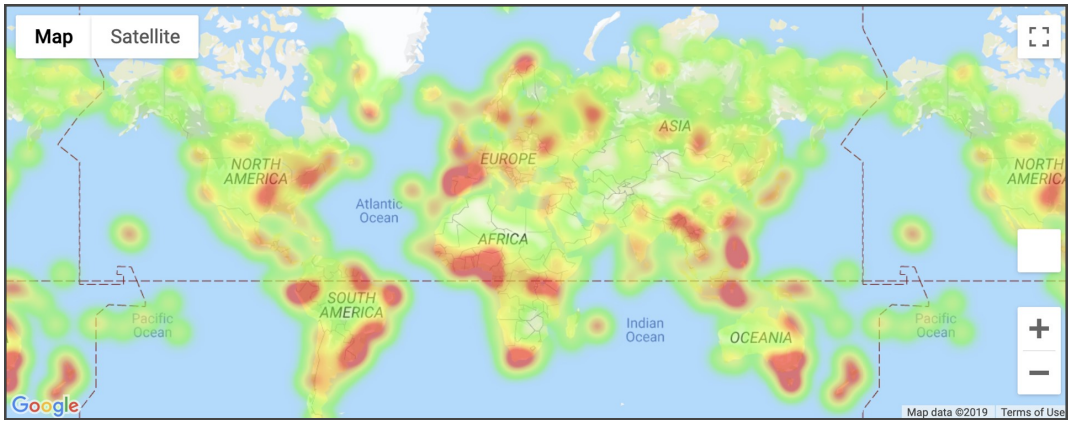
Create a Percent Humidity Heatmap

Now that we have created our maximum temperature heatmap, let's create the heatmap for humidity. We can reuse the code and use the humidity values for the measurements.

Add the following code to a new cell and run the cell.

```
# Heatmap of percent humidity
locations = city_data_df[["Lat", "Lng"]]
humidity = city_data_df["Humidity"]
fig = gmaps.figure(center=(30.0, 31.0), zoom_level=1.5)
heat_layer = gmaps.heatmap_layer(locations, weights=humidity, dis

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

Our humidity heatmap should look like the following.



Create a Percent Cloudiness Heatmap

Next, we will create the heatmap for percent cloudiness.

Complete the code to create the heatmap for percent cloudiness.

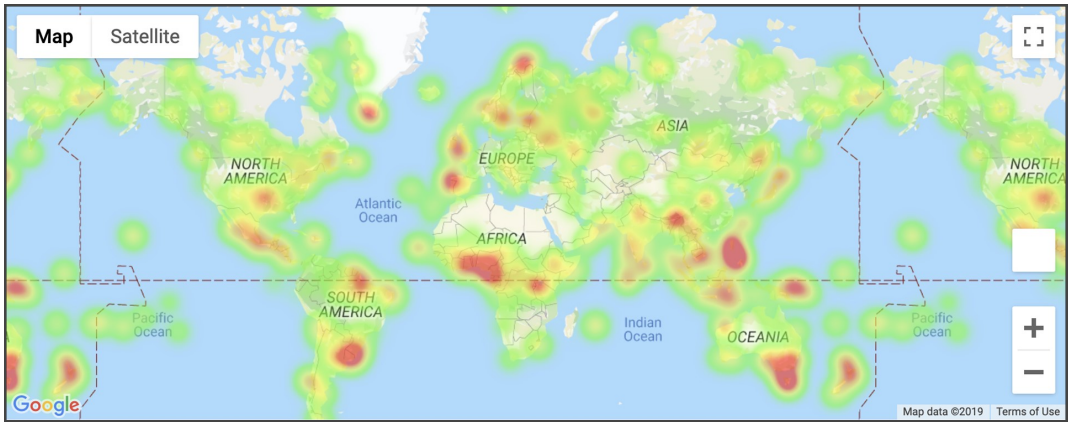
```
# Heatmap of percent cloudiness

locations = city_data_df[["Lat", "Lng"]]



 
= 
fig = gmaps.figure(center=(30.0, 31.0), zoom_level=1.5)

heat_layer = gmaps.heatmap_layer(locations, 
dissipating=False, max_intensity=300, point_radius=4)


fig.add_layer(heat_layer)

# Call the figure to plot the data.


fig
```

Check Answer

To create the heatmap for percent cloudiness, replace the measurement from the humidity values to cloudiness values. Copy the code that created the previous heatmap and edit your code with the line `clouds =`

`city_data_df["Cloudiness"]` so that the percent cloudiness replaces the percent humidity, and set the variable `weights=clouds`, and run your cell.

Your percent cloudiness heatmap should look like the following.



Create a Wind Speed Heatmap

Now we can create the final heatmap. Copy the code that created the previous heatmap and edit your code with the line `wind = city_data_df["Wind Speed"]` so that the wind speed replaces the percent cloudiness, and set the variable `weights=wind`, and run your cell.

Your wind speed heatmap should look similar to the following.

Congratulations on creating the heatmaps for each weather parameter!

6.5.3: Get Vacation Criteria

Now, the cherry on top of the project: a feature on the app that allows customers to search for locations they want to travel based on their temperature preferences.

Your clients will enter their preferences. Your code will tell them where to travel. It's beautiful, simple, and just needs a few more lines of code to actually work.

For the app we are creating, we need to prompt the user to enter the minimum and maximum temperature ranges as floating-point decimal numbers to filter the `city_data_df` DataFrame.

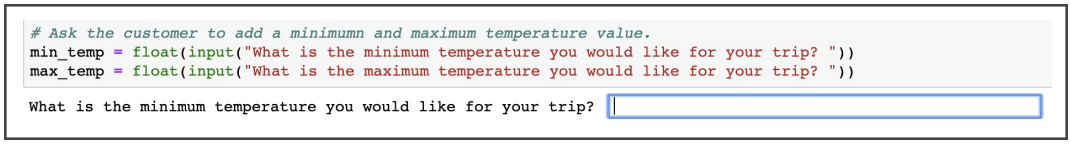
> **Rewind**
> Recall that you use the `input()` statement to prompt a user to enter information. To convert the input to a floating-point decimal number, wrap the `input()` statement with the `float()` method, like this: `float(input())`.

In a new cell, we'll write two input statements for the app that will prompt the customer to add minimum and maximum temperature values and convert these values to floating-point decimals.

The statement can be whatever you want it to be, but for our app, we'll prompt the customer with the following two input statements.

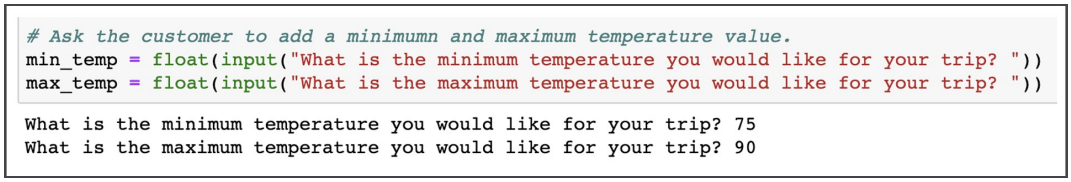
```
# Ask the customer to add a minimum and maximum temperature value
min_temp = float(input("What is the minimum temperature you would
max_temp = float(input("What is the maximum temperature you would
```

Add these two input statements to a new cell and run the cell. When we run this cell, the customer will be prompted to enter the minimum temperature.



Add 75 as the minimum temperature. After the customer enters the minimum temperature and presses Enter, he or she will be prompted to enter a maximum temperature.

For the maximum temperature, add 90 and press Enter.



Next, filter the maximum temperature column in the `city_data_df` DataFrame using logical operators to create a new DataFrame with the cities that meet the customer's criteria.

> **Rewind**
>
> Recall that you use the `loc[]` method on the current DataFrame to create a new DataFrame from a current DataFrame. Inside the brackets, add the conditional filter.

The conditional filter will be `city_data_df["Max Temp"] <= max_temp) & (city_data_df["Max Temp"]>= min_temp`. Also, filter the `city_data_df` DataFrame using the following statement.

```
# Filter the dataset to find the cities that fit the criteria.
preferred_cities_df = city_data_df.loc[(city_data_df["Max Temp"]
                                        (city_data_df["Max Temp"]
preferred_cities_df.head(10)
```

The `preferred_cities_df` DataFrame will contain all the cities that meet the temperature criteria.

	City_ID	City	Country	Date	Lat	Lng	Max Temp	Humidity	Cloudiness	Wind Speed
7	7	Avarua	CK	2019-08-08 22:26:22	-21.21	-159.78	77.00	78	90	6.93
13	13	Vaini	IN	2019-08-08 22:26:24	15.34	74.49	77.30	91	100	8.88
16	16	Dingle	PH	2019-08-08 22:26:24	11.00	122.67	75.43	90	99	11.18
21	21	Vinh Yen	VN	2019-08-08 22:26:26	21.31	105.60	77.00	100	40	3.36
28	28	Mazagao	BR	2019-08-08 22:26:28	-0.12	-51.29	84.20	79	75	5.82
29	29	Marsaxlokk	MT	2019-08-08 22:26:28	35.84	14.54	82.00	88	0	4.70
30	30	Portland	US	2019-08-08 22:24:00	43.66	-70.25	82.00	78	1	6.93
32	32	Camacha	PT	2019-08-08 22:26:29	33.08	-16.33	77.00	88	40	2.24
34	34	Prata	BR	2019-08-08 22:26:29	-19.31	-48.92	76.75	41	0	2.64
45	45	Albany	US	2019-08-08 22:21:44	42.65	-73.75	75.99	68	75	11.41

Before moving on, determine if the `preferred_cities_df` DataFrame has any null values for any of the rows.

How would you determine if there are any null values for the rows in the `preferred_cities_df` DataFrame? (Select all that apply)

- [ ] `preferred_cities_df.count()`
- [ ] `preferred_cities_df.counts()`
- [ ] `preferred_cities_df.isnull().sum()`
- [ ] `preferred_cities_df.notnull().sum()`

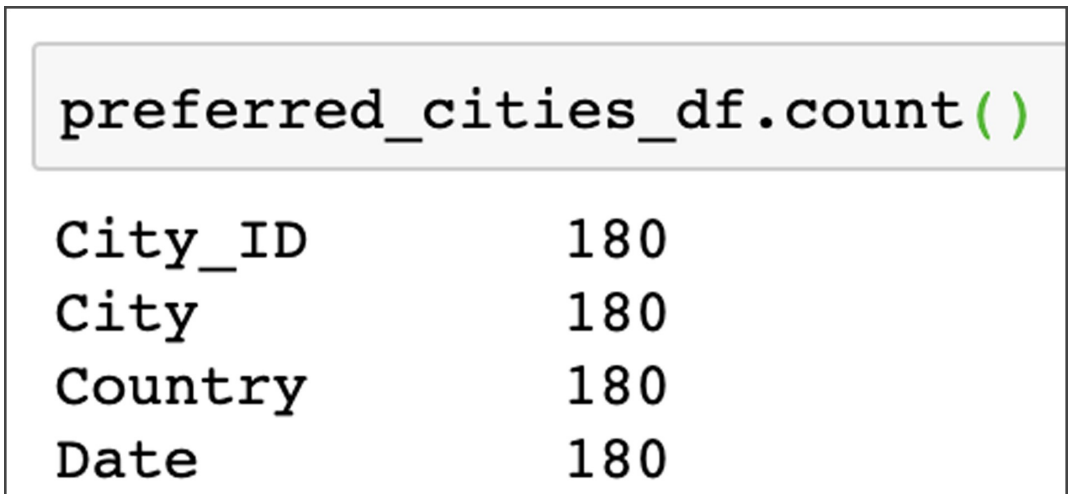
Check Answer

Finish ▸

Add the following code to a new cell and run the cell to determine if there are any null values.

```
preferred_cities_df.count()
```

For the above DataFrame, there are 180 cities and no null values.



```
Lat               180
Lng               180
Max Temp          180
Humidity          180
Cloudiness        180
Wind Speed        180
dtype: int64
```

IMPORTANT!

Consider the following guidance:

1. Depending on the time of year and the seasons, you might have to adjust the minimum and maximum temperature to get enough cities.
2. It is a good idea to keep the number of cities to fewer than 200 to make it easier to plot the markers on the heatmap.
3. If you have some rows with null values, you'll need to drop them using the `dropna()` method at the end of your filtering statement when you are creating the new DataFrame.

Now that we have all the cities the customer wants to travel to, they will need to find a hotel to stay in the city.

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 wit
hotel_df = preferred_cities_df[["City", "Country", "Max Temp", "L
hotel_df["Hotel Name"] = ""
hotel_df(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.

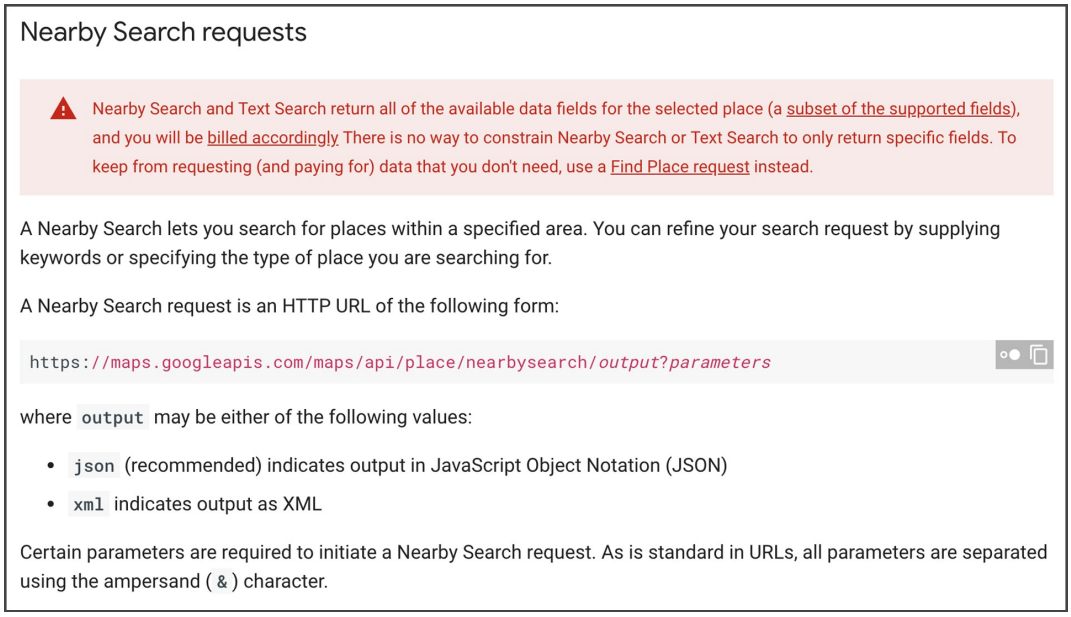
Retrieve Hotels from a Nearby Search

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

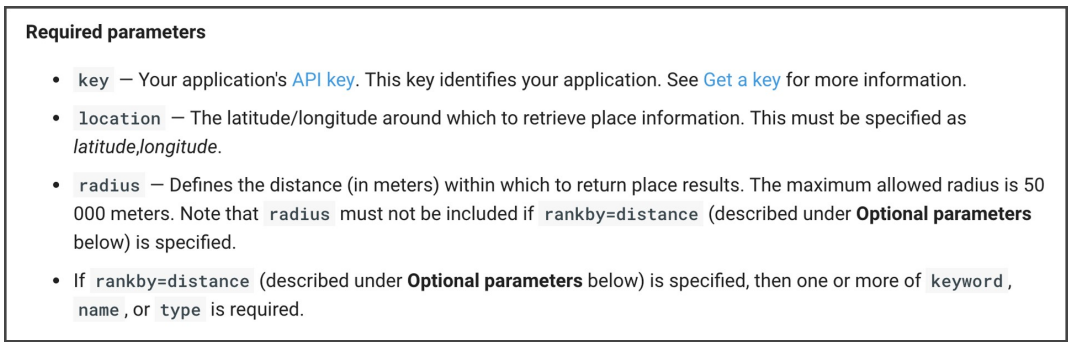
Certain parameters are required to initiate a Nearby Search request. As is standard in URLs, all parameters are separated using the ampersand (`&`) character.

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.



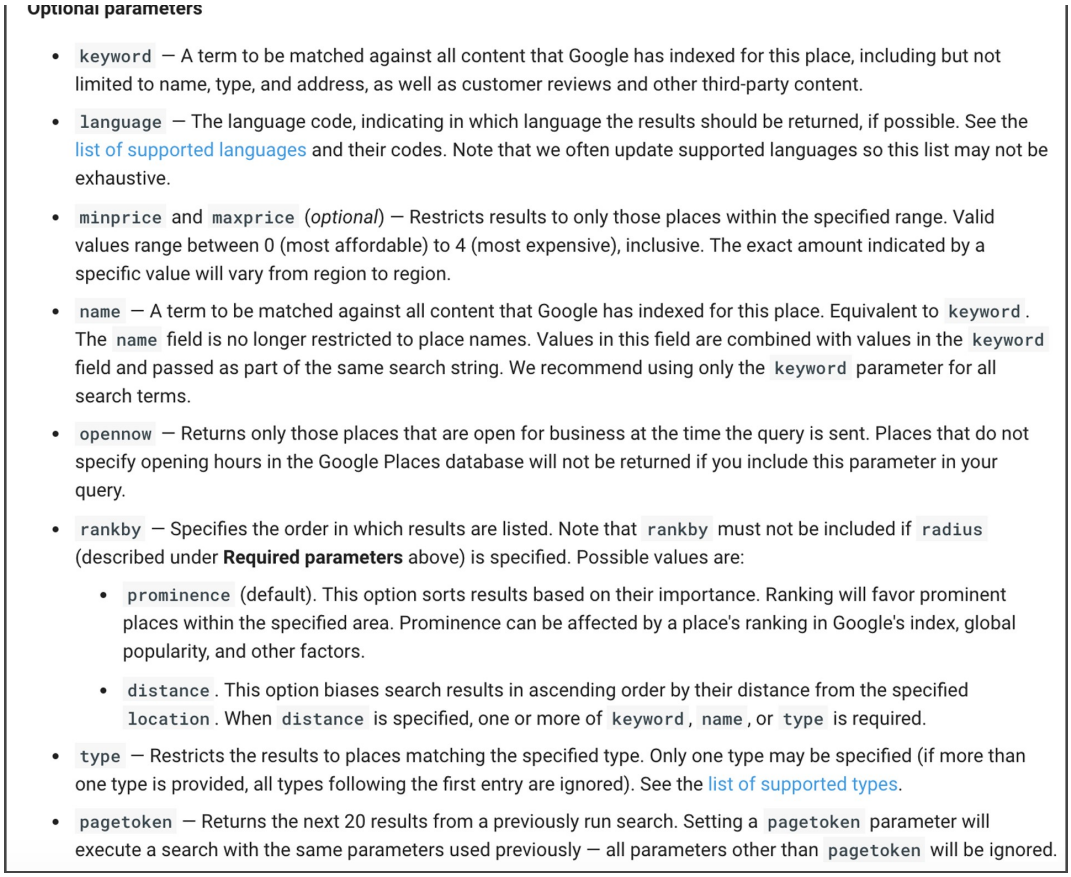
Required parameters

- `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.

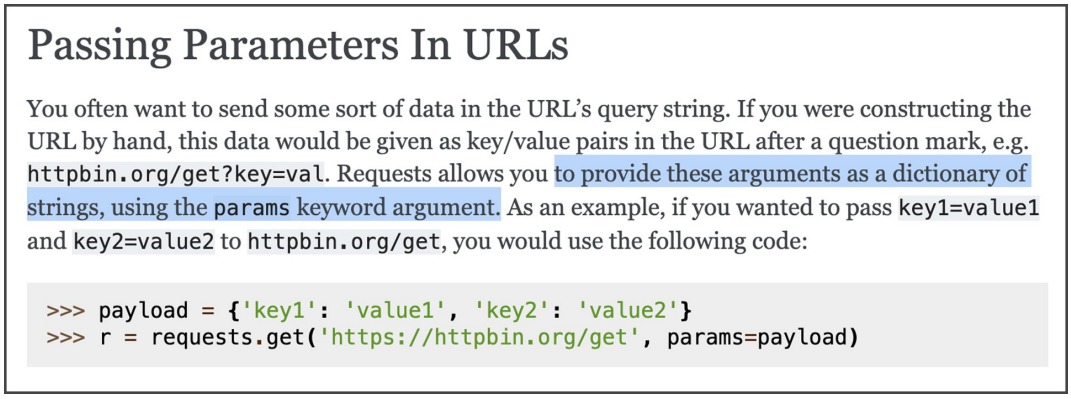
Next, we need to know how to add the other parameters in the search.

> **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.

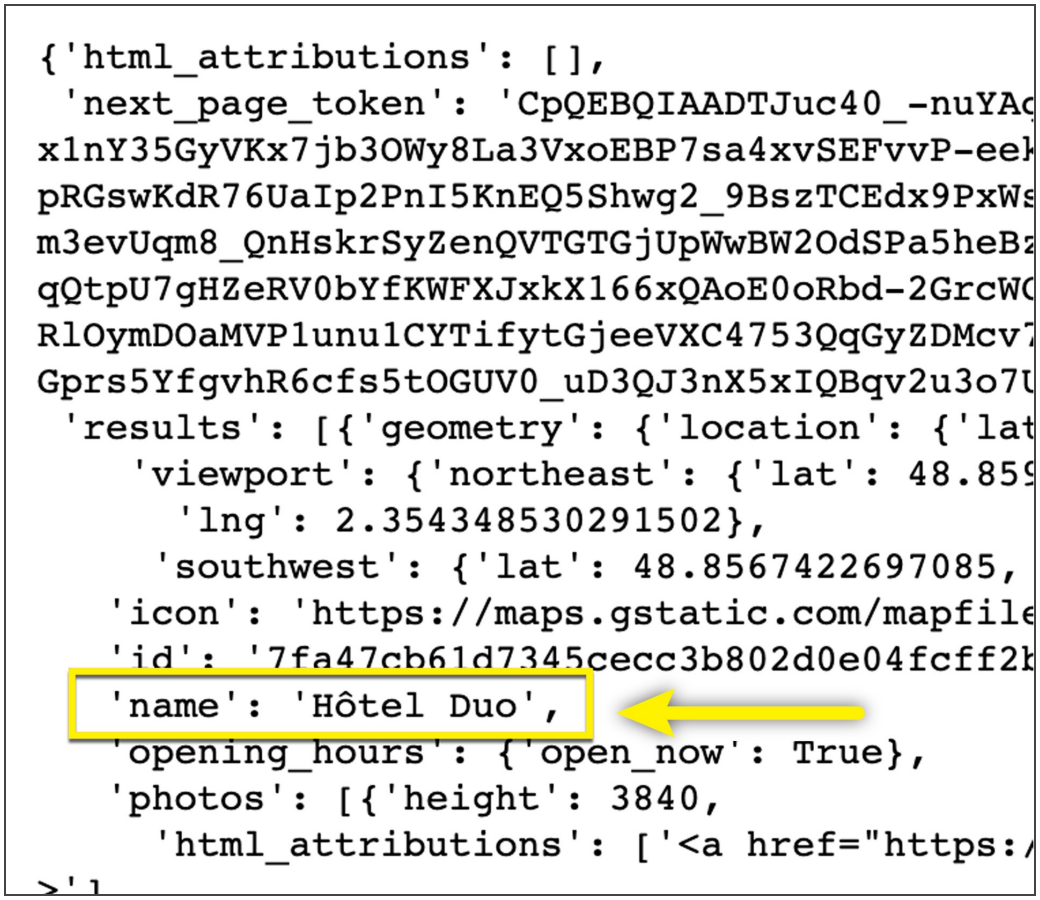
```
# Set the parameters to search for a hotel in Paris.
params = {
    "radius": 5000,
    "types": "lodging",
    "key": g_key,
    "location": "48.8566, 2.3522"}
# Use base URL to search for hotels in Paris.
base_url = "https://maps.googleapis.com/maps/api/place/nearbysear
# Make request and get the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

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"])`.

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`.



How would you retrieve the name of the first hotel in the `results` dictionary?

- ○ `hotels["results"][0]`
- ○ `hotels["results"][0]["name"]`
- ○ `hotels["results"]["name"]`

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 para
    params["location"] = f"{lat},{lng}"

    # Use the search term: "lodging" and our latitude and longitu
    base_url = "https://maps.googleapis.com/maps/api/place/nearby
    # 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]["nam
```

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
```

How would you handle the `IndexError` to get the code to run?

- ○ `try-except` block
- ○ `if-else` statement
- ○ `for` loop

Check Answer

Finish ▸

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]["nam
  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.

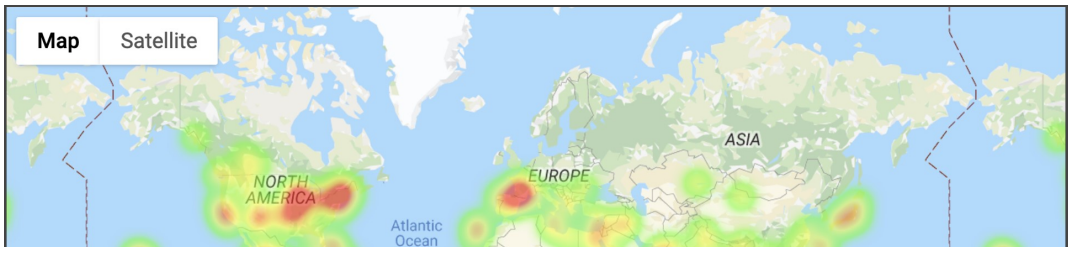
> **Note**
>
> For more information, see the [Google Developers' Nearby Search documentation (https://developers.google.com/places/web-service/search#PlaceSearchRequests)](https://developers.google.com/places/web-service/search#PlaceSearchRequests) .

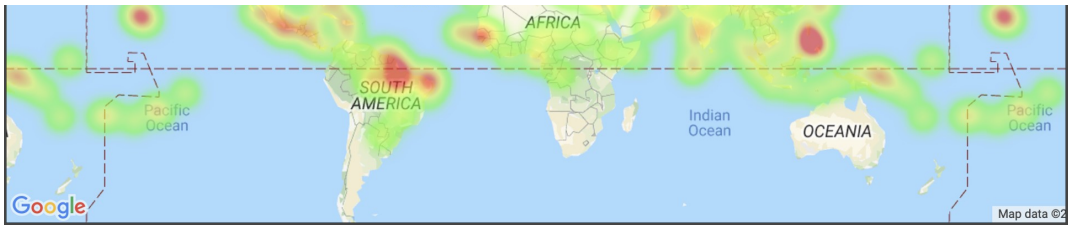
Create a Maximum Temperature Heatmap from a Hotel DataFrame

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, dis
             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.



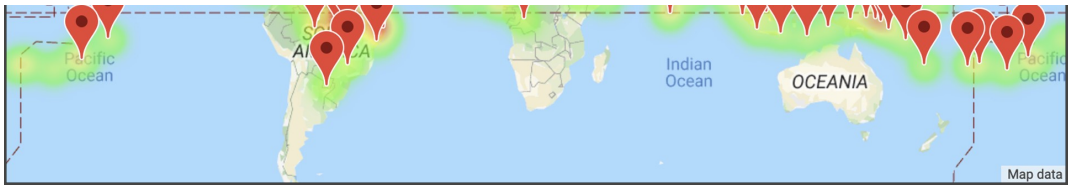


If we refer to the [gmaps documentation on how to add markers (https://jupyter-gmaps.readthedocs.io/en/latest/tutorial.html#markers-and-symbols)](), the syntax is `markers = gmaps.marker_layer(marker_locations)`, where the `marker_locations` are latitudes and longitudes.

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
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 descrtiption list (dl).

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

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

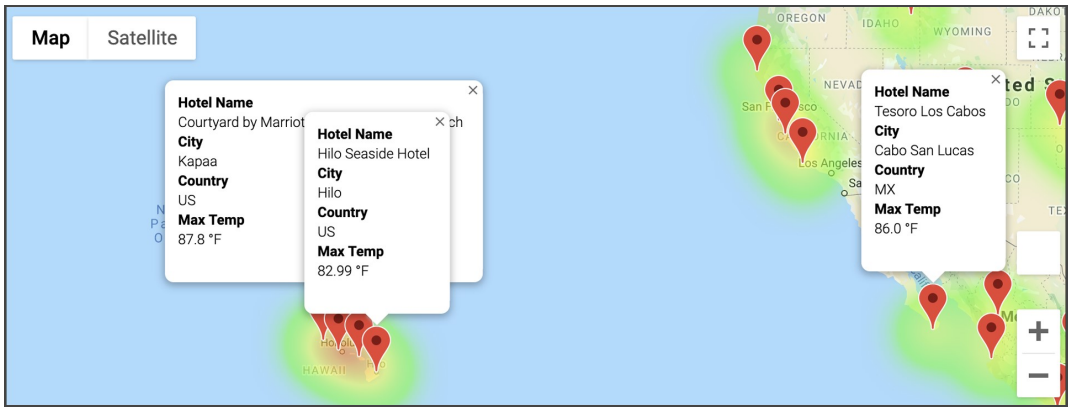
1. We set the `hotel_info` equal to the info_box_content.
2. In the list comprehension, `info_box_template.format(**row) for index, row in hotel_df.iterrows()`, we iterate through each “row” of the `hotel_df` DataFrame and then format the `info_box_template` with the data we set to populate the from 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-u
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,diss
             max_intensity=300, point_radius=4)
marker_layer = gmaps.marker_layer(locations, info_box_content=hot
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.

Module 6 Challenge

Submit Assignment

Due Feb 23 by 11:59pm **Points** 100
Submitting a text entry box or a website url

> Jack loves the app. Beta testers love the app. And, with any new product, there are some changes that could take it from a nice app to an awesome app. Your mission, should you choose to accept it, is to implement the feedback from the beta testers, which is listed below.
>
> 1. A weather description to the pop-up markers for customers so that they know what the weather is as they are traveling
> 2. A notation in the search criteria to indicate if it is raining or snowing for customers who are making travel decisions in real-time
> 3. A map that shows the directions for customers' travel itinerary

In this challenge, you will continue to build on your Python programming skills using `try-except` blocks, input statements and nested decision statements, logical expressions, and Pandas methods and attributes to make an API request and build a new DataFrame. You will then filter the DataFrame to create a marker layer map, and a directions layer Google map that shows directions to travel between multiple cities.

Background

For the new modifications to the PlanMyTrip app, you are asked to add more data to the database, or cities DataFrame, so that customers know the weather in the cities when they click on a pop-up marker. You'll also need to add the amount of rainfall or snowfall within the last three hours so that customers can filter the DataFrame using input statements based on the temperature range and whether or not it is raining or snowing. Finally, you'll need to create a directions layer Google map that shows the directions between multiple cities for travel.

Objectives

The goals for this challenge are for you to:

- Use nested `try-except` blocks.
- Use Pandas methods and attributes on a DataFrame or Series.
- Create a new DataFrame from a new API search with new weather parameters.
- Filter DataFrames based on input and nested decision statements, and logical expressions.
- Create pop-up markers on a Google map from a filtered DataFrame.
- Add a directions layer on a Google map between cities in the filtered DataFrame.

Part 1 Instructions

Get the Weather Description and Amount of Precipitation for Each City

To complete this task, follow these steps:

1. Create a new Jupyter Notebook file and name it `Weather_Database.ipynb`.
2. Generate a new set of 1,500 random latitudes and longitudes.
3. Get the nearest city using the citipy module.
4. Perform an API call with the OpenWeatherMap.
5. Retrieve the following information from the API call:
    - Latitude and longitude
    - Maximum temperature
    - Percent humidity
    - Percent cloudiness
    - Wind speed
    - Weather description (e.g., clouds, fog, light rain, clear sky)
    - Using a `try-except` block, if it is raining, get the amount of rainfall in inches for the last three hours. If it is not raining, add 0 inches for the city.
    - Using a `try-except` block, if it is snowing, get the amount of snow in inches for the last three hours. If it is not snowing, add 0 inches for the city.
6. Add the data to a new DataFrame.
7. Save the new DataFrame as a CSV file to be used for Part 2.

8. Upload the CSV file as part of your submission as `WeatherPy_challenge.csv`.
9. **Answer this question using Pandas methods**: **How many cities have recorded rainfall or snow?**

Your new DataFrame should look similar to the following image:

	City	Country	Date	Lat	Lng	Max Temp	Humidity	Cloudiness	Wind Speed	Current Description	Rain inches (last 3 hrs)	Snow inches (last 3 hrs)
0	Castro	CL	2019-08-26 17:25:49	-42.48	-73.76	48.20	61	40	14.99	scattered clouds	0.000	0
1	Lebu	ET	2019-08-26 17:25:49	8.96	38.73	58.69	83	72	1.45	light rain	2.187	0
2	Sitka	US	2019-08-26 17:25:49	37.17	-99.65	90.00	46	6	21.00	clear sky	0.000	0
3	Torbay	CA	2019-08-26 17:25:49	47.66	-52.73	75.00	43	20	10.29	few clouds	0.000	0
4	Grandview	US	2019-08-26 17:25:50	38.89	-94.53	69.80	93	40	4.70	heavy intensity rain	0.000	0
5	Mataura	NZ	2019-08-26 17:25:50	-46.19	168.86	36.91	86	87	8.93	light rain	0.124	0
6	Cidreira	BR	2019-08-26 17:25:50	-30.17	-50.22	72.01	50	48	3.44	scattered clouds	0.000	0
7	Hilo	US	2019-08-26 17:25:50	19.71	-155.08	75.20	88	90	4.70	heavy intensity rain	0.000	0
8	Great Falls	US	2019-08-26 17:25:51	47.50	-111.29	63.00	58	1	2.39	clear sky	0.000	0
9	Lata	IN	2019-08-26 17:25:51	30.78	78.62	41.77	93	87	1.66	light rain	0.375	0

Part 2 Instructions

Have Customers Narrow Their Travel Searches Based on Temperature and Precipitation

To complete this task, follow these steps:

1. Create a new Jupyter Notebook file and name it `Vacation_Search.ipynb`.
2. Import the `WeatherPy_vacation.csv` file from Part 1 as a new DataFrame.
3. Filter the DataFrame for minimum and maximum temperature preferences, and if the rain or snow accumulation is 0 inches or not using conditional statements. Do the following:

- Prompt the customer for the minimum temperature preference.
- Prompt the customer for the maximum temperature preference.
- Prompt the customer to answer if he or she would like it to be raining or not, using `input("Do you want it to be raining? (yes/no) ")`.
- Prompt the customer to answer if he or she would like it to be snowing or not, using `input("Do you want it to be snowing? (yes/no) ")`.

4. Add the cities to a marker layer map with a pop-up marker for each city that includes:
   - Hotel name
   - City
   - Country
   - Current weather description with the maximum temperature
5. Save and upload the new DataFrame as `WeatherPy_vacation.csv`.
6. Save and upload the new marker layer map as `WeatherPy_vacation_map.png`.

Your new hotel DataFrame should look similar to the following image:

	City	Country	Max Temp	Current Description	Lat	Lng	Hotel Name
2	Sitka	US	90.00	clear sky	37.17	-99.65	
15	Avera	US	86.00	broken clouds	33.19	-82.53	
17	Kapaa	US	82.40	broken clouds	22.08	-159.32	Courtyard by Marriott Kaua'i at Coconut Beach
19	Rock Sound	BS	84.43	overcast clouds	24.90	-76.20	Rock Sound Club
23	Fazilka	IN	86.59	broken clouds	30.41	74.03	Hotel Gulmarg
34	Souillac	FR	91.40	clear sky	45.60	-0.60	Jolysable
35	Yatou	CM	82.40	broken clouds	3.63	9.81	Village Belart
37	Gazanjyk	TM	84.97	clear sky	39.24	55.52	
41	Dodge City	US	90.00	clear sky	37.75	-100.02	Stay Suites of America-Dodge City
45	Villa Maria	CO	80.60	broken clouds	4.50	-74.09	Rumbo al llano "www.musicallanera.co"

Your marker layer map should look similar to the following image:

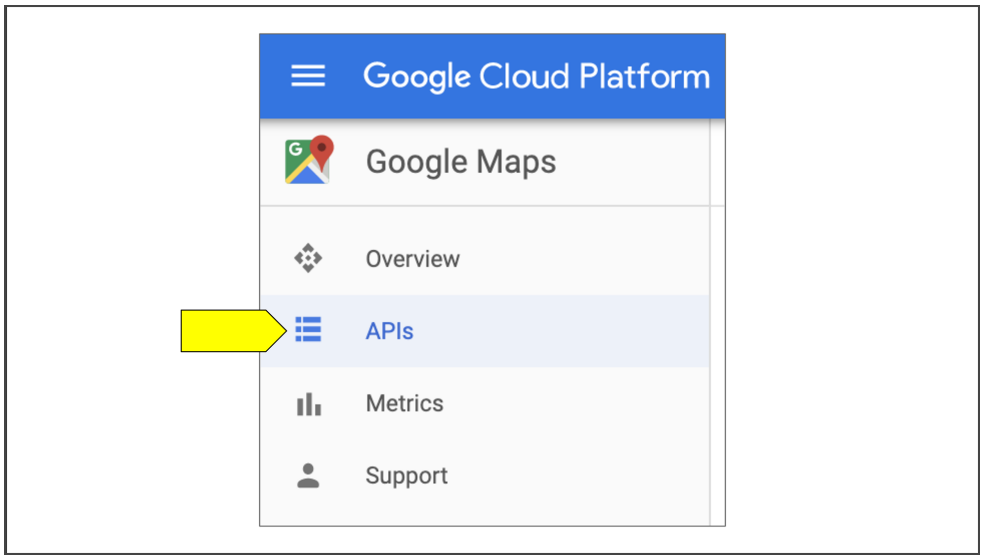
A pop-up marker for each city should look similar to the following image:

Part 3 Instructions

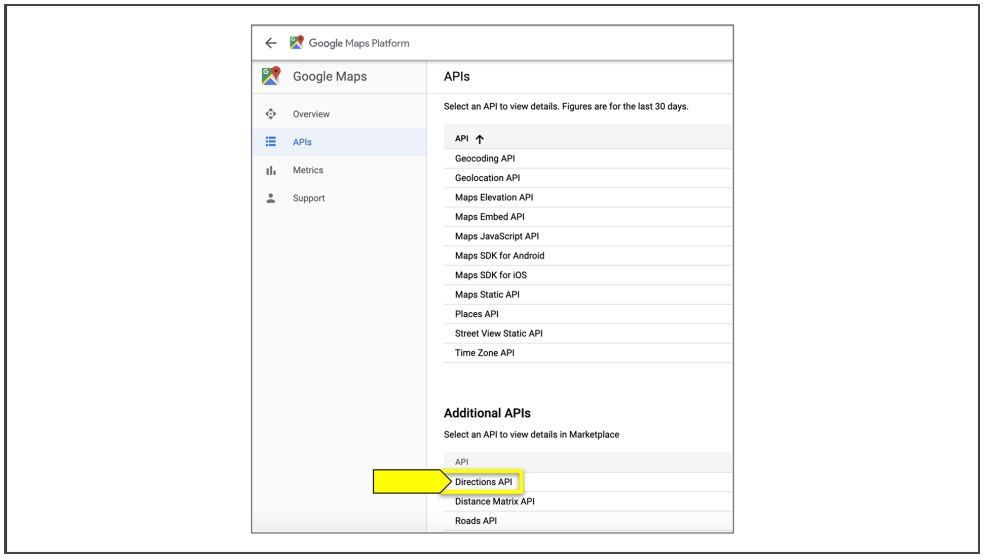
Create a Travel Itinerary with a Corresponding Map

Finally, you will create a map (travel itinerary) that shows the route between four cities from the customer's possible travel destinations, and then create a map with pop-up markers for the four cities. To complete these tasks, follow these steps:

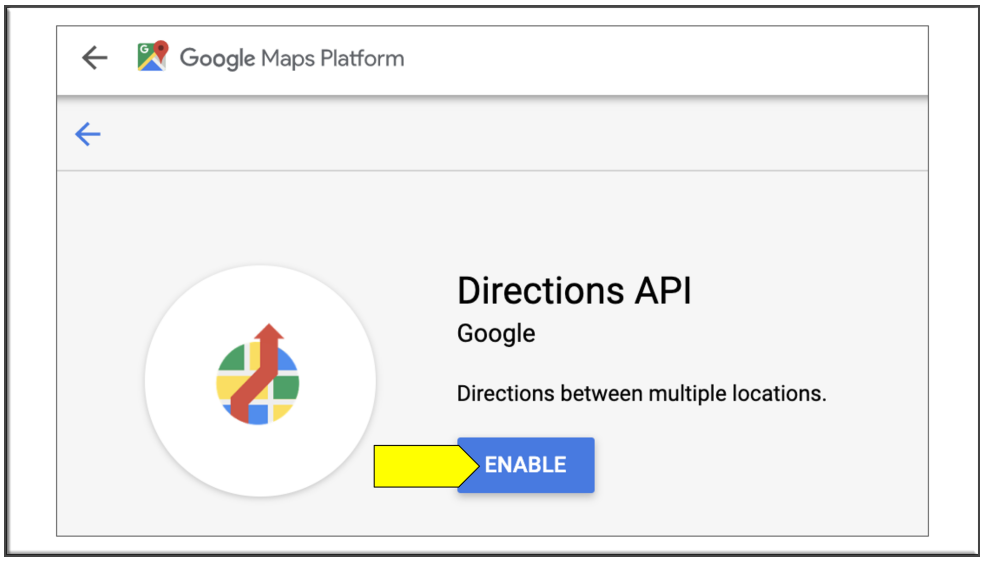
1. Enable the "Directions API" in your Google account for your API key.
    - On the Google Cloud Platform, select APIs from the left-hand side.



- Then, select "Directions API."



- Click “Enable” on the Directions API.



2. Create a new Jupyter Notebook file and label it `Vacation_Itinerary.ipynb`.
3. Import the `WeatherPy_vacation.csv` file as a new DataFrame.
4. From the vacation search map, ***choose at least four cities*** in close proximity on your map that are on the same continent that a customer might travel to, and then create a directions layer map.

Hints:

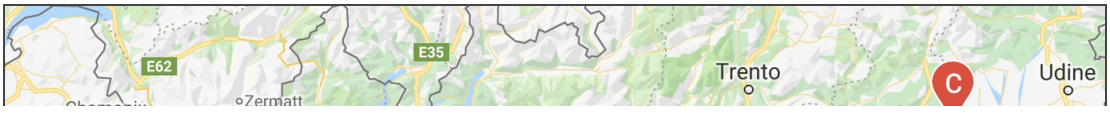
- Filter the DataFrame for each city you want to go to and create separate DataFrames for each city.
- Use the **directions Layer instructions from the gmaps documentation (https://jupyter-gmaps.readthedocs.io/en/latest/tutorial.html#directions-layer)**.
- Use the list indexing and Pandas methods to get the latitude-longitude pairs for each city DataFrame as tuples.

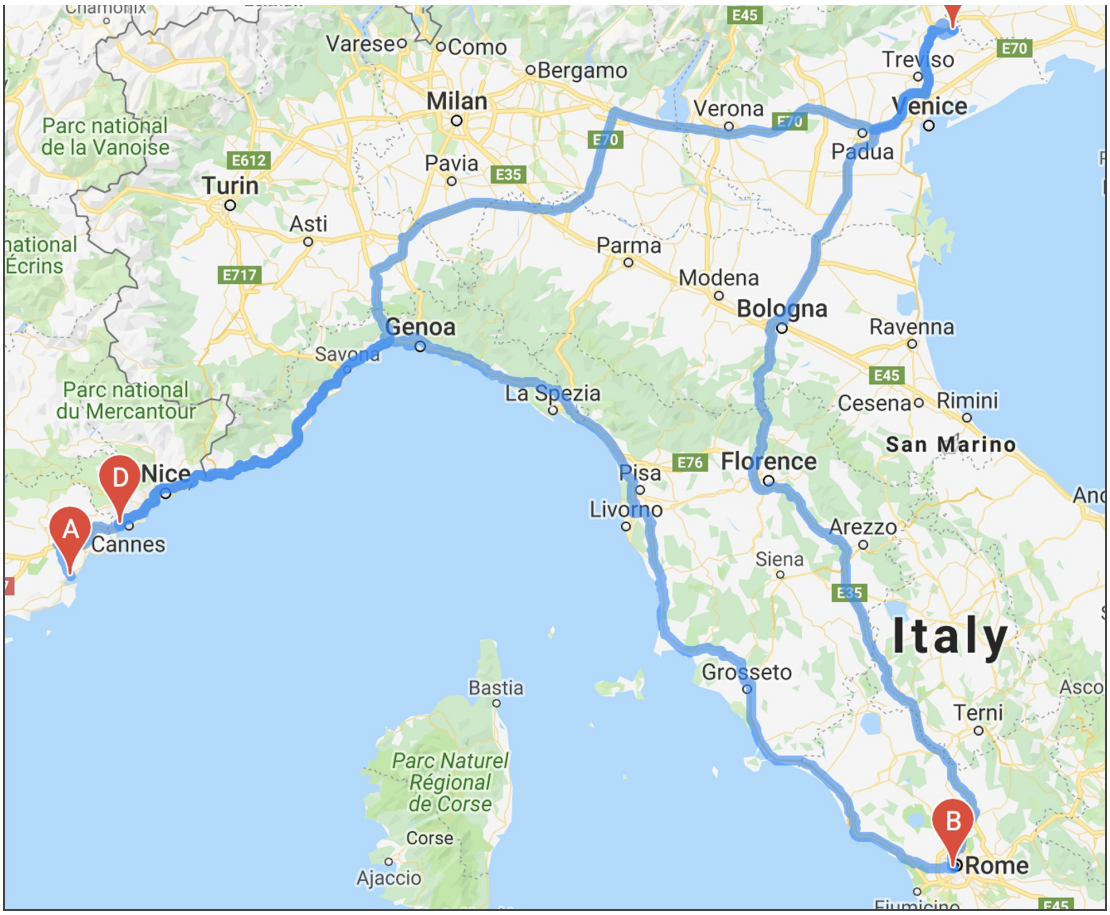
5. For the `travel_mode`, use either DRIVING, BICYCLING, or WALKING.
   Hint: If the cities are too far apart, some travel modes will not be available.
6. Take a screenshot of the route and save it as `WeatherPy_travel_map.png`.
7. Create a marker layer map for the four cities.

   Hint: Create a new DataFrame that has all the individual city DataFrames you created.

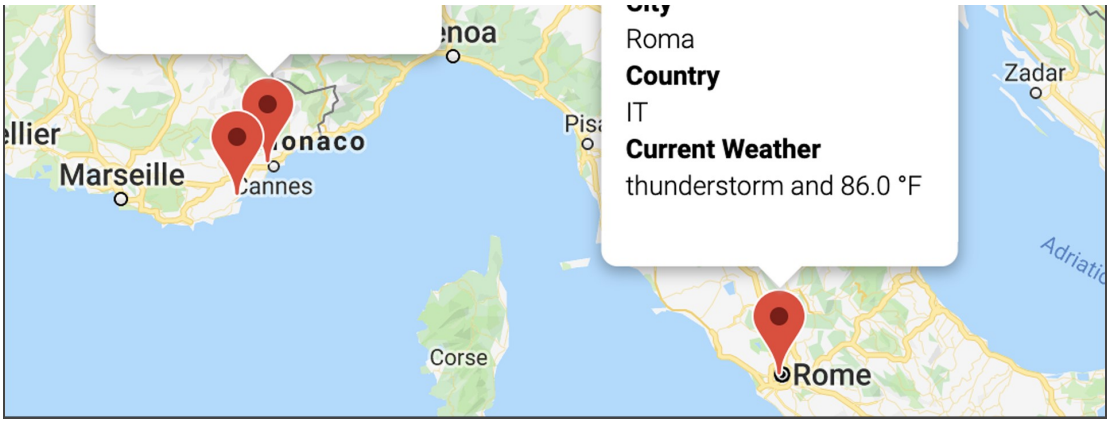
8. On the marker layer map, make sure each city has a pop-up marker that contains the following:
   - Hotel name
   - City
   - Country
   - Current weather description with the maximum temperature
9. Take a screenshot of the marker layer map for the route and save it as `WeatherPy_travel_map_markers.png`.

The directions layer map should look similar to the following image:





The pop-up marker for each city in the vacation itinerary look similar to the following image:



Add, Commit, Push When you are done. Commit the following to your WeatherPy GitHub repository:

1. Your code for Parts 1, 2, and 3 of the challenge.
    1. `Weather_Database.ipynb`.
    2. `Vacation_Search.ipynb`
    3. `Vacation_Itinerary.ipynb`
2. A “data” folder containing the following CSV files.
    1. `WeatherPy_challenge.csv`
    2. `WeatherPy_vacation.csv`
3. An “image” folder containing the images of your maps for Parts 2 and 3.
    1. `WeatherPy_vacation_map.png`
    2. `WeatherPy_travel_map.png`
    3. `WeatherPy_travel_map_markers.png`

Submission

To submit your challenge assignment, click Submit, then provide the URL of your WeatherPy GitHub repository for grading.

[Module 6 Challenge Rubric]

Note: You are allowed to miss up to two Challenge assignments and still earn your certificate. If you complete all Challenge assignments, your lowest two grades will be dropped. If you wish to skip this assignment, click Submit then indicate you are skipping by typing “I choose to skip this assignment” in the text box.

Some Rubric

Criteria	Ratings					Pts
Retrieve basic information with API call Please see complete description, including description of Mastery levels, hyperlinked in assignment description.	**10.0 pts Mastery**	**7.0 pts Approaching Mastery**	**4.0 pts Progressing**	**1.0 pts Emerging**	**0.0 pts Incomplete**	10.0 pts
Try-except block Please see complete description, including description of Mastery levels, hyperlinked in assignment description.	**20.0 pts Mastery**	**15.0 pts Approaching Mastery**	**10.0 pts Progressing**	**5.0 pts Emerging**	**0.0 pts Incomplete**	20.0 pts
Have Customers Narrow Their Travel Searches Based on Temperature and Precipitation using if/elif/else statements Please see complete description, including description of Mastery levels, hyperlinked in assignment description.	**20.0 pts Mastery**	**15.0 pts Approaching Mastery**	**10.0 pts Progressing**	**5.0 pts Emerging**	**0.0 pts Incomplete**	20.0 pts

Criteria	Ratings					Pts
Create a New Dataframe with Hotel Information Using Google API Places and try/except Please see complete description, including description of Mastery levels, hyperlinked in assignment description.	**10.0 pts** **Mastery**	**7.0 pts** **Approaching Mastery**	**4.0 pts** **Progressing**	**1.0 pts** **Emerging**	**0.0 pts** **Incomplete**	10.0 pts
Create a pop-up marker with city and weather data and hotel name Please see complete description, including description of Mastery levels, hyperlinked in assignment description.	**10.0 pts** **Mastery**	**7.0 pts** **Approaching Mastery**	**4.0 pts** **Progressing**	**1.0 pts** **Emerging**	**0.0 pts** **Incomplete**	10.0 pts
Create a directions layer map to travel between cities. Please see complete description, including description of Mastery levels, hyperlinked in	**20.0 pts** **Mastery**	**15.0 pts** **Approaching Mastery**	**10.0 pts** **Progressing**	**5.0 pts** **Emerging**	**0.0 pts** **Incomplete**	20.0 pts

Criteria	Ratings					Pts
assignment description.						
Create a pop-up marker for each city on the itinerary. Please see complete description, including description of Mastery levels, hyperlinked in assignment description.	**10.0 pts** **Mastery**	**7.0 pts** **Approaching Mastery**	**4.0 pts** **Progressing**	**1.0 pts** **Emerging**	**0.0 pts** **Incomplete**	10.0 pts
						Total Points: 100.0

Module 6 Career Connection

Get the Most Out of Working with Your Profile Coach

Did you know that your profile will offer unlimited feedback on your career materials to support you in becoming Employer Competitive. If you've already submitted your brand statement to a Profile Coach for review, take a moment to implement the feedback now so you can stay on track towards your career goals. You're also welcome to re-submit for another round of review. If you haven't yet submitted your statement for review, **click here to do that now. (https://courses.bootcampspot.com/courses/138/pages/milestone-2-creating-your-professional-brand-statement)**

Unit Assessment: Python

You are about to complete your first Unit Assessment! This Unit Assessment allows you to check your knowledge, as well as demonstrate your competency in key concepts from Modules 3 through 6.

After submitting the assessment, you will see a summary of your performance. While you will not be able to see your performance on individual questions, you are allowed unlimited attempts to complete the assessment.

Some of the questions on this assessment require specific resources. Download the following resources before you get started.

cereal.csv

donors2008.csv

avg_rain_state.csv

youtube_response.json

Data-Unit-Assessment-2-Python

Please click **Start** when you are ready to begin the activity.

Start

Data-Unit-Assessment-2-Python

Open and read the file, *avg_rain_state.csv* into a Pandas DataFrame and answer the following questions

1. What is the mean rainfall for the entire DataFrame? ,

2. What is the median rainfall for the entire DataFrame?

3. How many modes are there for rainfall?

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

Complete the following code below so that it will do the following tasks:

1. Open and read the *cereal.csv* file.
2. Skip the header row.
3. Read through each row.
4. Print out the cereals the following cereals that contain 5 or more grams of fiber in this order:

```
100% Bran
All-Bran
All-Bran with Extra Fiber
Bran Flakes
Fruit & Fibre Dates; Walnuts; and Oats
Fruitful Bran
Post Nat. Raisin Bran
Raisin Bran
```


◄ Next ►

```
import os
import csv
cereal_csv = os.path.join("..",
"cereal.csv")

with open(                  , ne
wline="") as csvfile:
    csvreader = csv.reader(
                  , delimiter
=",")

    csv_header = next(csvfile)

    for row in
                  :
        if float(
                  ) >= 5:
```

Data-Unit-Assessment-2-Python

Download *avg_rain_state.csv* and read it into a pandas DataFrame as `rain_df` .

What is the code that will return the following output.

	State	Inches	Millimetres	Rank
8	Florida	54.5	1385	5
0	Alabama	58.3	1480	4
23	Mississippi	59.0	1499	3
17	Louisiana	60.1	1528	2
10	Hawaii	63.7	1618	1

```
rain_df.[          ]
(['Rank'],
[          ]).
[          ]
```

- sort
- ascending=False
- sort=True
- tail()

◄
Next ►

Data-Unit-Assessment-2-Python

What Python method is used to get all the keys from a dictionary?

- ○ `get()`
- ○ `keys()`
- ○ `get_keys()`
- ○ `key()`

Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

1. Open and read the donors2008.csv file into a pandas DataFrame.
2. Using groupby, create a Series with the average “Amount” for each state.
3. How many states have averages equal to or greater than $750.00?

- ○ 8
- ○ 9
- ○ 11
- ○ 10

◄ Next ►

Data-Unit-Assessment-2-Python

From the following information that creates a pie chart. Drag the best answer for each of the following questions.

```
pies = ["Apple", "Pumpkin", "Chocolat
e Creme", "Cherry", "Apple Crumb", "P
ecan", "Lemon Meringue", "Blueberry",
"Key Lime", "Peach"]
pie_votes = [47,37,32,27,25,24,24,21,
18,16]
colors = ["yellow","green","lightblu
e","orange","red","purple","pink","ye
llowgreen","lightskyblue","lightcora
l"]
offset = (0,0,0,0.2,0,0,0,0,0,0)
```

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10

◄ Next ►

What type of pie will be offset in the pie chart?	What is the color of the pie wedge that is offset?	What is the percentage of votes for the pie wedge that is offset?

Pies

Apple Cherry

Peach Pumpkin

Chocolate Creme

Data-Unit-Assessment-2-Python

What piece of code defines the beginning of a Python function?

- ○ def():
- ○ define():
- ○ def
- ○ function()

◄ Next ►

Data-Unit-Assessment-2-Python

Using the following code.

```
import request
url = "https://api.spacexdata.com/v2/
launchpads"
```

Using a for loop, what is the order of the "full name" of each launch site in the JSON file when printed to the output cell?

- ≡ Kwajalein Atoll Omelek Island
- ≡ Vandenberg Air Force Base Space Launch Complex 3W
- ≡ Kennedy Space Center Historic Launch Complex 39A
- ≡ Vandenberg Air Force Base Space Launch Complex 4E
- ≡ SpaceX South Texas Launch Site

◄ Next ►

Data-Unit-Assessment-2-Python

You are trying to find which classroom “Dan” is in, i.e., `name = "Dan"`. You have access to a list of students first names only from three separate classrooms, `classroom_1`, `classroom_2`, and `classroom_3`. Dan is only in one of the classrooms.

What is the order of the following code fragments that would check all three classrooms and print out which classroom “Dan” is in.

1.

:

2.

3.

:

4.

Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

Assuming you have a folder named “output” and a file named “employees.txt”. What is the correct code that needs to be used to write the data to the file?

```
import os
import csv

file_to_save = os.path.join("out
put", "employees.txt")

with open(file_to_save, "w") as
new_file:
    employees = (
        f"First Name', 'Last Nam
e', 'SSN\n"
        f"Caleb', 'Frost', '505-
80-2901\n")

    [                    ].write(e
mployees)
```

- file
- employees

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

What is the output when the following code is run?

```
students = ("Frank", "Mary", "Jasmin
e", "Ivana", "Ahmed")
students.append("Serena")
print(students)
```

- ○ The code won't compile.
- ○ `['Frank', 'Mary', 'Jasmine', 'Ivana', 'Ahmed', 'Serena']`
- ○ `AttributeError: 'tuple' object has no attribute 'append'`
- ○ `('Frank', 'Mary', 'Jasmine', 'Ivana', 'Ahmed', 'Serena')`
- ○ `('Frank', 'Mary', 'Jasmine', 'Ivana', 'Ahmed')`

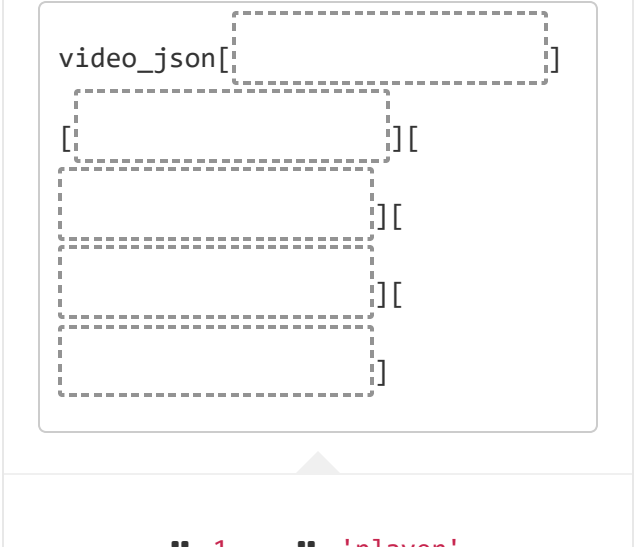
◄ Next ►

Data-Unit-Assessment-2-Python

Download youtube_response.json and use the following code to open and read the JSON file. Then answer the following question.

```
import json
import os

filepath = os.path.join("youtube_response.json")
with open(filepath) as jsonfile:
    video_json = json.load(jsonfile)
```

Complete the code that returns the link to the video's thumbnail jpg from the video_json variable.



◄
Next ►

Data-Unit-Assessment-2-Python

The following code has a syntax error. Assuming you have a folder named “output” and a file named “employees.txt”. Highlight the line with the syntax error.

```
import os
import csv

file_to_save = os.path.join("outpu
t", "employees.txt")

with open(file_to_save, "w") as new
_file:
    employees = (
        f"First Name', 'Last Name',
'SSN\n"
        f"Caleb', 'Frost', '505-80-2
901\n")

    txt_file.write(employees)
```

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

You have converted a CSV file to a DataFrame,“df”. How would you get the names of the columns from the DataFrame?

- ○ `df.headers()`
- ○ `df.head()`
- ○ `df.columns`
- ○ `df.columns()`

◄ Next ►

Data-Unit-Assessment-2-Python

Using the following code.

```
import requests
url = "https://api.spacexdata.com/v2/
launchpads"
```

What is the length of the JSON file after you make the request?

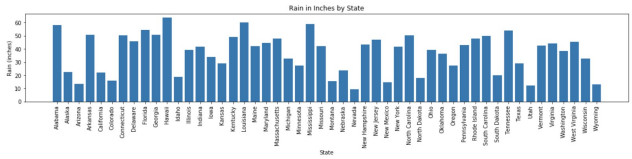
- ○ 3
- ○ 6
- ○ `TypeError: object of type 'resp onse' has no len()`
- ○ 1

Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

Download *avg_rain_state.csv*. Open and read the file the CSV file into a pandas DataFrame, called “df”. Using the MATLAB matplotlib method, complete the code to create the following graph.




```
x_axis =
                                    

tick_locations = [value for valu
e in                               ]

plt.figure(figsize=(20,3))
plt.bar(
                                    ,
                                    )
plt.xticks(tick_locations,
                                    , rotat
ion=                                )
plt.title("Rain in Inches by Sta
te")
plt.xlabel("State")
plt.ylabel("Rain (inches)")
plt.show()
```

- `df["Inches"]`
- `len(df)`
- `np.arange(len(df))`

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

What is the matplotlib function that is used to create box and whisker plots?

- ○ box()
- ○ boxplots()
- ○ boxwhisker()
- ○ boxplot()

Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

Using pandas, how would you export a DataFrame, “df” as the file, “data_file.csv” into a folder, “Output” without the index, but with the header? (Select all that apply)

- [ ] `df.to_csv("Output/data_file.csv", index=False)`
- [ ] `df.to_csv("Output/data_file.csv")`
- [ ] `df.to_csv("Output/data_file.csv", header=True)`
- [ ] `df.to_csv("Output/data_file.csv", index=False, header=False)`
- [ ] `df.to_csv("Output/data_file.csv", header=False)`
- [ ] `df.to_csv("Output/data_file.csv", index=False, header=True)`

Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8
Item 9
Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

For the following data.

```
test_grades = {
    'Class': ['Oct', 'Oct', 'Jan', 'J
an', 'Oct', 'Jan'],
    'Name': ["Cyndy", "Logan", "Lac
i", "Elmer", "Crystle", "Emmie"],
    'Test Score': [90, 59, 72, 88, 9
8, 60]}
```

1. Create a DataFrame from the “test_grades” dictionary.
2. Create bins, 0, 59, 69, 79, 89, 100, to hold the data.
3. Assign a letter grade, as “letter_grades”, for the four bins as the labels.

Complete the code that will create a new column “Letter Grade” based on the four bins and labels

```
df[                    ] = pd.
[                    ](df
[                    ], bins, l
abels=[                    ])
```

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10

◄ Next ►

Data-Unit-Assessment-2-Python

Add the correct exception error so that when the code is run you get the last line to print to the terminal.

```
try:
    print("Infinity looks like + " + str(10 / 0) + ".")
except [          ]:
    try:
        print("I think her name was + " + name + "?")
    except
[          ]:
        try:
            print("Your name is a nonsense number. Look: " + int("Gabriel"))
        except
[          ]:
            print("I made it through the gauntlet. The message survived!")
```

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5
- Item 6
- Item 7
- Item 8
- Item 9
- Item 10

◄ Finish ►