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.

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)

coordinates = list(lat\_lngs)

* + Use the citipy module to list the nearest city to the latitudes and longitudes.

from citipy import citipy

# Create a list for holding the cities.

cities = []

# Identify the nearest city for each latitude and longitude combination.

for coordinate in coordinates:

city = citipy.nearest\_city(coordinate[0], coordinate[1]).city\_name

# If the city is unique, then we will add it to the cities list.

if city not in cities:

cities.append(city)

# Print the city count to confirm sufficient count.

len(cities)

* + Use the OpenWeatherMap API to request the current weather data from each unique city in your list.

for i, city in enumerate(cities):

# Create endpoint URL with each city.

city\_url = url + "&q=" + city.replace(" ","+")

* + Parse the JSON data from the API request.

# Parse the JSON and retrieve data.

city\_weather = requests.get(city\_url).json()

* + 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

city\_lat = city\_weather["coord"]["lat"]

city\_lng = city\_weather["coord"]["lon"]

city\_max\_temp = city\_weather["main"]["temp"]

city\_humidity = city\_weather["main"]["humidity"]

city\_clouds = city\_weather["clouds"]["all"]

city\_wind = city\_weather["wind"]["speed"]

city\_country = city\_weather["sys"]["country"]

# Append the city information into city\_data list.

city\_data.append({"City": city.title(),

"Lat": city\_lat,

"Lng": city\_lng,

"Max Temp": city\_temp,

"Humidity": city\_humidity,

"Cloudiness": city\_clouds,

"Wind Speed": city\_wind,

"Country": city\_country,

"Date": city\_date})

# Convert the array of dictionaries to a Pandas DataFrame.

city\_data\_df = pd.DataFrame(city\_data)

1. **Exploratory Analysis with Visualization**
   * Create scatter plots of the weather data for the following comparisons:

plt.scatter(lats, max\_temps, edgecolor="black", linewidths=1, marker="o", alpha=0.8, label="Cities")

* + - Latitude versus temperature
    - Latitude versus humidity
    - Latitude versus cloudiness
    - Latitude versus wind speed
  + Determine the correlations for the following weather data:

# Import linear regression from the SciPy stats module.

from scipy.stats import linregress

* + - 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:

$ conda install -c conda-forge gmaps

$ jupyter nbextension enable --py --sys-prefix widgetsnbextension

$ pip install gmaps

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

* + - Latitude and temperature
    - Latitude and humidity
    - Latitude and cloudiness
    - Latitude and wind speed

1. **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:

* + Filter the Pandas DataFrame based on user inputs for a minimum and maximum temperature.
  + Create a heatmap for the new DataFrame.
  + Find a hotel from the cities' coordinates using Google's Maps and Places API, and Search Nearby feature.
  + Store the name of the first hotel in the DataFrame.
  + Add pop-up markers to the heatmap that display information about the city, current maximum temperature, and a hotel in the city.

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

sing 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

**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.

## The Python Requests Library

To request JSON data over the internet, we use the Requests Library in Python.

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.

My version is: '2.27.1'