```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


# Create Latitude and Longitude Combinations
# Import the dependencies.
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


# In[2]:


# Create a set of random latitude and longitude combinations, size 1500.
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


# In[3]:


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


# In[4]:


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


# In[5]:


# Use the citipy module to determine city based on latitude and longitude.
from citipy import citipy


# In[6]:
```

```python
# Use the fifteen hundred  pairs of latitudes and longitudes we used from our zip practice to get a city and country code from the citipy module
# Use the tuple() function to display the latitude and longitude combinations.
for coordinate in coordinates:
    print(citipy.nearest_city(coordinate[0], coordinate[1]).city_name,
          citipy.nearest_city(coordinate[0], coordinate[1]).country_code)


# In[7]:


# 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
    print(coordinate[0], coordinate[1])
    # 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)


# In[8]:


city = citipy.nearest_city(25.12903645, -67.59741259)
city.city_name


# In[9]:


# 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)


# In[11]:


# Import the requests library.
import import_ipynb
import requests
import json
# Import the API key.
from config import weather_api_key


# In[12]:


# Starting URL for Weather Map API Call.
url = "http://api.openweathermap.org/data/2.5/weather?units=Imperial&APPID=" +
weather_api_key
type(url)


# In[13]:


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


# In[14]:


print(requests.get(city_url))


# In[15]:


# Using json()
requests.get(city_url).json()


# In[16]:


# Make a 'Get' request for the city weather.
```

```
city_weather = requests.get(city_url)
city_weather


# In[17]:


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


# In[18]:


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


# In[19]:


# Handle Request Errors
# 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.")


# In[20]:


# Mining the JSON file to retrieve specific weather data for each city and add it
to a DataFrame
# 1. Let's look at Boston city first: correct the spelling for the city of Boston
# Create an endpoint URL for a city.
city_url = url + "&q=" + "Boston"
city_weather = requests.get(city_url)
city_weather.json()


# In[21]:


# Get the JSON data.
```

```
boston_data = city_weather.json()
# Get the corresponding value using sys
boston_data["sys"]


# In[22]:


# Get the corresponding value using sys & country
boston_data["sys"]["country"]


# In[23]:


# Retrieve the date in the weather data using dt
boston_data["dt"]


# In[24]:


boston_data["main"]["temp_max"]


# In[25]:


# Get the time of day, the latitude, longitude, maximum temperature, humidity,
percent cloudiness, and wind speed
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)


# In[26]:


# Convert the Date Timestamp
# Import the datetime module from the datetime library.
from datetime import datetime
# Get the date from the JSON file.
date = boston_data["dt"]
```

```python
# Convert the UTC date to a date format with year, month, day, hours, minutes,
and seconds.
datetime.utcfromtimestamp(date)


# In[27]:


# Convert this datetime format to 2019-10-21 17:24:35 using the Python string
format method
# strftime() and adding how we want the string to look inside the parentheses
datetime.utcfromtimestamp(date).strftime('%Y-%m-%d %H:%M:%S')


# In[28]:


# 1. Get the 500 City Weather Data
# 1.1. Import Dependencies, and Initialize an Empty List and Counters
# Import the time library and the datetime module from the datetime library
import time
from datetime import datetime


# In[29]:


# 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[31]:


# Loop through all the cities in the list.But, 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.
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
        time.sleep(60)

    # Create endpoint URL with each city.
    city_url = url + "&q=" + city.replace(" ","+")

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


# In[32]:


# Handle API Request Errors with try-except Blocks
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
        time.sleep(60)

    # Create endpoint URL with each city.
    city_url = url + "&q=" + city.replace(" ","+")

    # Log the URL, record, and set numbers and the city.
    print(f"Processing Record {record_count} of Set {set_count} | {city}")
    # Add 1 to the record count.
    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"]).strftime('%Y-
%m-%d %H:%M:%S')
        # 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("-----------------------------")


# In[33]:


# Create a DataFrame of City Weather Data
# Convert the array of dictionaries to a Pandas DataFrame.
import pandas as pd
city_data_df = pd.DataFrame(city_data)
city_data_df.head(10)


# In[34]:


new_column_order = ["City", "Country", "Date","Lat", "Lng", "Max Temp",
"Humidity", "Cloudiness", "Wind Speed"]


# In[35]:
```

```
city_data_df = city_data_df[new_column_order]
city_data_df.head(10)


# In[38]:


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


# In[39]:


# 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"]


# In[40]:


# Import the time module.
import time
# Get today's date in seconds.
today = time.time()
today


# In[41]:


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


# In[43]:


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

```python
import time
from matplotlib import pyplot as plt

# 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("%x"))
plt.ylabel("Max Temperature (F)")
plt.xlabel("Latitude")
plt.grid(True)

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

# Show plot.
plt.show()


# In[44]:


# Build the scatter plot 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("Fig2.png")

# Show plot.
plt.show()


# In[45]:
```

```python
# Build the scatter plot 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("Fig3.png")

# Show plot.
plt.show()


# In[46]:


# Build the scatter plot for latitude vs. cloudiness.
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("Fig4.png")

# Show plot.
plt.show()


# In[47]:
```

```
# Import necessary libraries
# Import linregress
from scipy.stats import linregress
# Import matplotlib
import matplotlib.pyplot as plt


# In[48]:


# Create a Linear Regression Function
# Create a function to create perform linear regression on the weather data
# and plot a regression line and the equation with the data.
def plot_linear_regression(x_values, y_values, title, y_label, text_coordinates):

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

    # Calculate the regression line "y values" from the slope and intercept.
    regress_values = x_values * slope + intercept
    # Get the equation of the line.
    line_eq = "y = " + str(round(slope,2)) + "x + " + str(round(intercept,2))
    # 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="red")
    plt.title(title)
    plt.xlabel('Latitude')
    plt.ylabel(y_label)
    plt.show()
# For this code there will be no output until we call the function with five
parameters.


# In[49]:


# Create the Hemisphere DataFrames
# 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)]


# In[50]:
```

```
# Perform Linear Regression on the Maximum Temperature for the Northern
Hemisphere
# 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 Hemisphere \
                       for Maximum Temperature', 'Max Temp',(30,90))

r = np.corrcoef(x_values, y_values)
r


# In[51]:


# Perform Linear Regression on the Maximum Temperature for the Southern
Hemisphere
# 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 Hemisphere \
                       for Maximum Temperature', 'Max Temp',(-30,40))

r = np.corrcoef(x_values, y_values)
r


# In[52]:


# Create the linear equation and scatter plot of the latitude and percent
humidity for the Northern and Southern Hemispheres.
# Perform Linear Regression on the percent humidity for the Northern Hemisphere
# 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 Hemisphere \
                       for percent humidity', 'Humidity',(40,10))
```

```
r = np.corrcoef(x_values, y_values)
r


# In[53]:


# Perform Linear Regression on the Percent Humidity for the Southern Hemisphere
# Perform Linear Regression on the percent humidity for the Southern Hemisphere
# 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 Hemisphere \
                       for percent humidity', 'Humidity',(-25,20))

r = np.corrcoef(x_values, y_values)
r


# In[54]:


# Perform Linear Regression on the Percent Cloudiness for the Northern Hemisphere
# Perform Linear Regression on the percent Cloudiness for the Northern Hemisphere
# Linear regression on the Northern Hemisphere
x_values = northern_hemi_df["Lat"]
y_values = northern_hemi_df["Cloudiness"]
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Northern Hemisphere \
                       for percent cloudiness', 'Cloudiness',(10,60))

r = np.corrcoef(x_values, y_values)
r


# In[55]:


# Perform Linear Regression on the percent Cloudiness for the Southern Hemisphere
# Linear regression on the Southern Hemisphere
x_values = southern_hemi_df["Lat"]
y_values = southern_hemi_df["Cloudiness"]
```

```python
# Call the function.
plot_linear_regression(x_values, y_values,
                       'Linear Regression on the Southern Hemisphere \
                        for % Cloudiness', '% Cloudiness',(-50,60))

r = np.corrcoef(x_values, y_values)
r


# In[60]:


# Perform Linear Regression on the Wind Speed for the Northern Hemisphere
# 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 Hemisphere \
                        for Wind Speed', 'Wind Speed',(10,40))

r = np.corrcoef(x_values, y_values)
r


# In[57]:


# Perform Linear Regression on the Wind Speed for the Southern Hemisphere
# 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 Hemisphere \
                        for Wind Speed', 'Wind Speed',(-40,30))

r = np.corrcoef(x_values, y_values)
r


# In[ ]:
```