**VACATIONPY: Starter Code to Import Libraries and Load the Weather and Coordinates Data:**

**Step 1: Create a map that displays a point for every city in the `city\_data\_df` DataFrame. The size of the point should be the humidity in each city.**

1. **Explanation of changes and improvements:**

* **scale parameter:** The size parameter in hvplot controls the size of the points based on the data column. However, the raw humidity values might make the points too large or too small. The scale parameter allows you to adjust the visual scaling of the points to make them a more appropriate size. A value of 0.5 reduces the influence of the humidity value on the point size, making for a better visual representation. Experiment with different values to find what works best for your data.
* **hover\_cols for richer interaction:** By specifying hover\_cols, when you mouse over a point on the map, a tooltip will appear showing not just the city's location but also the additional data like Country, Max Temp, Humidity, Cloudiness, and Wind Speed. This greatly enhances the interactive nature of the visualization.
* **tiles="OSM":** Using OpenStreetMap tiles often provides a more visually appealing and detailed map compared to the default.

1. **Key improvements over simply using size="Humidity":**

* **Better visual clarity:** The scale parameter prevents points from being excessively large or small, leading to a more balanced and understandable map.
* **More information on hover:** hover\_cols provides valuable context and detail, making the map more informative and interactive for the user.
* **Enhanced map style:** OpenStreetMap tiles typically offer a more visually appealing map background.

**Step 2: Narrow down the `city\_data\_df` DataFrame to find your ideal weather condition**

1. **Explanation:**

* **Combined Filtering:** The conditions are combined using the logical AND operator (&) within the DataFrame indexing to filter in a single step. This is more efficient than separate filtering operations.
* **dropna() after filtering:** Although the filtering conditions should already exclude rows with null values in the relevant columns, it's good practice to include dropna() after filtering to be absolutely certain you have no missing data that could cause issues later.
* **Directly use filtered DataFrame:** The ideal\_city\_df is then used directly for the hvplot, eliminating the need for a separate data cleaning step and ensuring the map is created based on the filtered data.

1. **Improvements:**

* **Data Type Conversion and Cleaning:**
  + The crucial step is to use pd.to\_numeric(city\_data\_df["Humidity"], errors='coerce') to convert the "Humidity" column to a numeric type. The errors='coerce' argument handles any non-numeric values by converting them to NaN (Not a Number).
  + Then, city\_data\_df.dropna(subset=["Humidity"]) removes rows where "Humidity" is NaN. This ensures that hvplot correctly interprets the values for sizing and coloring.
* **Categorical Data Type:** The most important change is converting the "City" column to a categorical data type using astype("category"). This tells hvplot to treat "City" as a categorical variable, assigning a unique color to each city.
* **Categorical Colormap:** The cmap parameter is now set to "Category20" (or similar like Category10, Category20b, Category20c etc.) which are designed for categorical data. These colormaps provide distinct colors for a limited number of categories. If you have more than 20 cities, you might need to use a colormap with more categories or adapt your visualization strategy.
* **Titles added to plots:** For ease of identification I added a title to each of the plots.

**Step 3: Create a new DataFrame called `hotel\_df`.**

1. **hotel\_df = ideal\_city\_df[["City", "Country", "Lat", "Lng", "Humidity"]].copy():**
   * This line creates a new DataFrame called hotel\_df. It's crucial to use the .copy() method. If you omit .copy() and just use hotel\_df = ideal\_city\_df[...], then hotel\_df would be a *view* of ideal\_city\_df, not an independent copy. Any changes made to hotel\_df would also affect ideal\_city\_df, which is usually not what you want. The .copy() method creates a completely separate DataFrame.
   * ideal\_city\_df[["City", "Country", "Lat", "Lng", "Humidity"]] selects only the specified columns ("City", "Country", "Lat", "Lng", "Humidity") from the ideal\_city\_df DataFrame to include in the new hotel\_df DataFrame. This keeps the hotel\_df focused on the necessary data for hotel searches.
2. **hotel\_df["Hotel Name"] = "":**
   * This line adds a new column named "Hotel Name" to the hotel\_df DataFrame.
   * It initializes all the values in this new column to empty strings (""). This creates a placeholder where you can store the hotel names you find using the Geoapify API later in your code.
3. **display(hotel\_df.head()):**
   * This line uses the display() function (which is particularly useful in Jupyter notebooks or environments that support rich display outputs). It shows the first few rows (default is 5) of the newly created hotel\_df DataFrame. This allows you to quickly inspect the structure of the DataFrame, verify that the desired columns are present, and that the "Hotel Name" column has been correctly initialized with empty strings.

**Step 4: For each city, use the Geoapify API to find the first hotel located within 10,000 metres of your coordinates.**

1. **raise\_for\_status():** Added name\_address.raise\_for\_status() to handle potential HTTP errors during the API request. This will raise an exception if the API request returns a 4xx or 5xx status code, preventing your code from proceeding with potentially incorrect or missing data.
2. **Improved Exception Handling:** The try...except block now catches requests.exceptions.RequestException in addition to KeyError and IndexError. This broader exception handling covers various potential issues during the API request or JSON parsing. The error message is also printed for debugging purposes.
3. **Clearer Parameter Setting:** The params dictionary is set up with the necessary parameters for the Geoapify API, including categories, limit, and your API key. The filter and bias parameters are constructed within the loop to use the current city's coordinates.
4. **Proximity Bias:** The bias parameter is now set to f"proximity:{lng},{lat}" to bias the search results towards locations closest to the specified coordinates. This increases the likelihood of finding a hotel near the city center.
5. **Display with display():** The final hotel\_df is displayed using the display() function for better formatting in Jupyter notebooks or other compatible environments.

**Step 5: Add the hotel name and the country as additional information in the hover message for each city in the map.**

1. **hotel\_df.hvplot.points(...):** This initiates the creation of a scatter plot on a map using the data from the hotel\_df DataFrame. The .points() method indicates that we want to represent each data point as a marker on the map.
2. **Lng, Lat:** These arguments specify the columns in hotel\_df that contain the longitude and latitude coordinates for each city. These are essential for positioning the markers on the map.
3. **geo=True:** This parameter enables geographical plotting, which is crucial for creating a map visualization. It tells hvplot to interpret the Lng and Lat columns as geographical coordinates.
4. **tiles="OSM":** This specifies the type of background map tiles to use. "OSM" stands for OpenStreetMap, which is a popular choice for providing a detailed and visually appealing map background.
5. **frame\_width=800, frame\_height=600:** These parameters control the width and height of the plot in pixels.
6. **size="Humidity":** This is a key feature: it makes the size of each marker on the map proportional to the "Humidity" value for that city in the hotel\_df DataFrame. Higher humidity will result in larger markers.
7. **scale=1.0:** This parameter controls how the size parameter is scaled. A value of 1.0 means a direct proportional scaling (higher humidity = proportionally larger marker size). You can adjust this value to make the size differences more or less pronounced.
8. **color="City":** This determines the color of the markers. By setting color="City", each unique city will be assigned a different color, making it easy to visually distinguish cities on the map.
9. **cmap="Category20":** Because the color is set to "City," which is a categorical variable (not a continuous numerical value), you must use a categorical colormap. "Category20" provides 20 distinct colors for up to 20 different cities. If you have more than 20 cities, you might consider other categorical colormaps like Category10 or using a different visualization strategy.
10. **hover\_cols=[...]:** This is the parameter that defines what information is displayed when you hover the mouse pointer over a marker on the map. The list provided includes "City," "Country," "Hotel Name," "Max Temp," "Humidity," "Cloudiness," and "Wind Speed." This makes the map interactive and provides more detailed information about each city on demand.
11. **title="Ideal Cities Map with Hotels":** This sets the title of the plot, providing a descriptive label for the visualization.
12. **map\_plot\_filtered\_hotels:** This variable stores the resulting hvplot object.
13. **map\_plot\_filtered\_hotels:** Displaying the hvplot object will render the map in your output.