**REQUIREMENT 1: CREATE PLOTS TO SHOWCASE THE RELATIONSHIP BETWEEN WEATHER VARIABLES AND LATITUDE**

**Use the OpenWeatherMap API to retrieve weather data from the cities list generated in the started code:**

**1. Dependencies and Setup:**

* import matplotlib.pyplot as plt, import pandas as pd, import numpy as np, import requests, import time, from scipy.stats import linregress: Imports necessary libraries for plotting, data manipulation, numerical operations, making HTTP requests, pausing execution, and linear regression (although not used in this specific code).
* from api\_keys import weather\_api\_key: Imports the OpenWeatherMap API key from a separate api\_keys.py file. This is good practice for security, keeping your API key out of your main code.
* from citipy import citipy: Imports the citipy library, which is used to find the nearest city to a given latitude and longitude.

**2. Generate City List:**

* lat\_lngs = [], cities = []: Initializes empty lists to store latitude/longitude pairs and city names.
* lat\_range = (-90, 90), lng\_range = (-180, 180): Defines the range of latitudes and longitudes.
* lats = np.random.uniform(lat\_range[0], lat\_range[1], size=1500), lngs = np.random.uniform(lng\_range[0], lng\_range[1], size=1500): Generates 1500 random latitude and longitude values within the specified ranges using NumPy's uniform distribution.
* lat\_lngs = zip(lats, lngs): Combines the latitude and longitude lists into a list of pairs.
* The for lat\_lng in lat\_lngs: loop iterates through the latitude/longitude pairs:
  + city = citipy.nearest\_city(lat\_lng[0], lat\_lng[1]).city\_name: Uses citipy to find the nearest city to the current latitude/longitude pair.
  + if city not in cities:: Checks if the city is already in the cities list. If not, it adds the city to the list, ensuring uniqueness.
* print(f"Number of cities in the list: {len(cities)}"): Prints the number of unique cities found.

**3. API Calls and Data Retrieval:**

* url = "http://api.openweathermap.org/data/2.5/weather?": Sets the base URL for the OpenWeatherMap API.
* city\_data = []: Initializes an empty list to store the weather data for each city.
* print("Beginning Data Retrieval..."), print("-----------------------------"): Prints messages to the console to indicate the start of the data retrieval process.
* record\_count = 1, set\_count = 1: Initializes counters for tracking the progress of the API calls.
* The for i, city in enumerate(cities): loop iterates through the list of cities:
  + if (i % 50 == 0 and i >= 50):: Checks if 50 cities have been processed. If so, it increments the set\_count and resets the record\_count. This is for logging and organization purposes.
  + city\_url = f"{url}q={city}&appid={weather\_api\_key}&units=metric": Constructs the complete API URL for the current city, including the API key and specifying metric units for temperature.
  + print(f"Processing Record {record\_count} of Set {set\_count} | {city}"): Prints the current city being processed.
  + record\_count += 1: Increments the record count.
  + The try...except block handles potential errors during the API call:
    - response = requests.get(city\_url).json(): Makes the API call using requests.get() and converts the response to a JSON object.
    - city\_lat = response["coord"]["lat"], etc.: Extracts the relevant weather data from the JSON response.
    - city\_data.append(...): Appends the extracted data as a dictionary to the city\_data list.
  + except KeyError:: Catches KeyError exceptions, which occur if a city is not found or the API response is incomplete. Prints a message and skips the city.
  + time.sleep(1): Pauses for 1 second to avoid exceeding the API's rate limit.
* print("-----------------------------"), print("Data Retrieval Complete "), print("-----------------------------"): Prints messages to indicate the completion of the data retrieval process.

**4. Create DataFrame:**

* city\_data\_df = pd.DataFrame(city\_data): Creates a Pandas DataFrame from the collected city\_data.
* city\_data\_df.count(): Displays the number of records retrieved for each column, providing a quick check of the data.

**5. Key improvements:**

* **units=metric added to the URL:** This ensures the temperature data is retrieved in Celsius.
* **record\_count reset to 1:** The record\_count should reset to 1, not 0, at the start of each new set.
* **f-strings for cleaner print statements:** Using f-strings makes the print statements more readable.
* **More specific exception handling:** Using KeyError is better practice than a bare except because it targets the specific issue of a missing key in the JSON response, which is the most likely error you'll encounter. This prevents accidentally masking other potential errors.
* **Added data frame creation at the end:** This creates the city\_data\_df DataFrame as requested by the original prompt's implicit requirements. Also added a count to check the retrieved data.

**Create a Scatter Plot for Latitude versus Temperature:**

1. **plt.scatter(...):** This creates the scatter plot.

* city\_data\_df["Lat"]: The x-values (latitude).
* city\_data\_df["Max Temp"]: The y-values (maximum temperature).
* edgecolors="black": Adds a black outline to the markers for better visibility.
* linewidths=1: Sets the width of the marker outlines.
* marker="o": Uses circles as markers.
* alpha=0.8: Adds a slight transparency to the markers.
* label="Cities": Provides a label for the legend (if you add one later).

1. **plt.title(...):** Sets the title of the plot. The f-string and time.strftime() are used to include the current date in the title, making it more informative.
2. **plt.ylabel(...), plt.xlabel(...):** Labels the y and x axes.
3. **plt.grid(True):** Adds a grid to the plot for better readability.
4. **plt.savefig(...):** Saves the plot to a file named "Fig1.png" in the "output\_data" directory. Make sure this directory exists before running the code.
5. **plt.show():** Displays the plot.

**Create a Scatter Plot for Latitude versus Humidity:**

1. The changes are minimal compared to the latitude vs. temperature plot:

* **Y-axis data:** The plt.scatter function now uses city\_data\_df["Humidity"] for the y-values, representing humidity.
* **Y-axis label:** The plt.ylabel is updated to "Humidity (%)".
* **Title:** The plt.title is updated to reflect that this is a humidity plot.
* **Filename:** The plt.savefig function now saves the plot to "Fig2.png".

1. The rest of the code (marker styling, grid, etc.) remains the same, providing consistency across visualizations.

**Create a Scatter Plot for Latitude versus Cloudiness:**

1. The changes from the previous plots are:

* **Y-axis Data:** city\_data\_df["Cloudiness"] is used for the y-values in the plt.scatter function.
* **Y-axis Label:** plt.ylabel is changed to "Cloudiness (%)".
* **Title:** plt.title is updated to reflect the cloudiness plot.
* **Filename:** The saved filename in plt.savefig is now "Fig3.png".

1. Everything else (marker styling, grid, etc.) remains consistent with the other scatter plots.

**Create a Scatter Plot for Latitude versus Wind Speed:**

1. Here's what's changed for this plot:

* **Y-axis data:** The plt.scatter function now uses city\_data\_df["Wind Speed"] for the y-values.
* **Y-axis label:** The plt.ylabel is updated to "Wind Speed (m/s)". The units are meters per second because we specified units=metric in the API call.
* **Title:** The plot title is updated to "City Latitude vs. Wind Speed".
* **Filename:** The saved filename is "Fig4.png".

1. The rest of the styling and code structure remain the same as the other scatter plots.

**REQUIREMENT 2: COMPUTE LINEAR REGRESSION FOR EACH RELATIONSHIP**

**Define a function to create Linear Regression plots:**

1. **Function Definition:** The code is now organized within a function plot\_linear\_regression for reusability. It takes the x-values, y-values, title, y-axis label, and text coordinates for the equation as arguments.
2. **Linear Regression Calculation:** The linregress function from scipy.stats is used to perform linear regression. The slope, intercept, r-value, p-value, and standard error are calculated.
3. **Regression Line and Equation:** regress\_values calculates the y-values for the regression line. line\_eq creates the string representation of the equation.
4. **Scatter Plot and Regression Line:** The plt.scatter function plots the data points, and plt.plot plots the regression line in red.
5. **Equation Annotation with** **Dynamic Placement:** Inside the function, the x and y limits of the plot are calculated (x\_min, x\_range, y\_min, y\_range). The annotation is then placed at 10% of the x-range and 90% of the y-range, ensuring it's within the visible plot area, regardless of the data values.
6. **Labels and Title:** The x-axis label, y-axis label (taken as an argument), and title (including the current date) are set.
7. **R-squared Value:** The r-squared value, a measure of how well the regression line fits the data, is printed to the console.
8. **Gridlines:** plt.grid(True) adds gridlines to the plot (optional, but often helpful).
9. **Example Usage:** The code demonstrates how to call the function for each of the weather variables (Max Temp, Humidity, Cloudiness, Wind Speed). You'll need to adjust the text\_coordinates argument if necessary to prevent the equation from overlapping data points.

**Create a DataFrame with the Northern Hemisphere data (Latitude >= 0):**

1. **northern\_hemi\_df = city\_data\_df[city\_data\_df["Lat"] >= 0]:** This is the core of the filtering operation. Let's break it down:

* city\_data\_df["Lat"]: This accesses the "Lat" column (latitude values) of the city\_data\_df DataFrame.
* city\_data\_df["Lat"] >= 0: This creates a boolean Series (a sequence of True/False values). For each row in the DataFrame, it checks if the latitude value is greater than or equal to 0. True indicates the city is in the Northern Hemisphere, and False indicates it's in the Southern Hemisphere.
* city\_data\_df[...]: This part uses boolean indexing. The boolean Series created in the previous step is used to select only the rows from city\_data\_df where the corresponding value in the boolean Series is True. In other words, it selects only the rows where the latitude is greater than or equal to 0.
* northern\_hemi\_df = ...: The result of this boolean indexing (a filtered DataFrame) is assigned to the new variable northern\_hemi\_df.

1. **northern\_hemi\_df.head():** This line displays the first 5 rows of the newly created northern\_hemi\_df DataFrame. This is a common way to quickly inspect the contents of a DataFrame and verify that the filtering operation worked correctly.

**Create a DataFrame with the Southern Hemisphere data (Latitude < 0):**

1. The code is almost identical to the Northern Hemisphere version, with one key difference:

* **southern\_hemi\_df = city\_data\_df[city\_data\_df["Lat"] < 0]:** This line filters the city\_data\_df DataFrame to select only the rows where the "Lat" column (latitude) is *less than* 0. This corresponds to cities in the Southern Hemisphere. The resulting filtered DataFrame is assigned to the southern\_hemi\_df variable.

1. The rest of the code remains the same: southern\_hemi\_df.head() displays the first 5 rows of the new Southern Hemisphere DataFrame for inspection.

**Create a Temperature vs. Latitude Linear Regression Plot for the Northern Hemisphere:**

1. **Data Selection:** Uses the northern\_hemi\_df (created earlier) to get the latitude and maximum temperature data for the Northern Hemisphere.
2. **Linear Regression:** Calculates the linear regression using linregress.
3. **Scatter Plot:** Creates a scatter plot of latitude vs. maximum temperature.
4. **Regression Line:** Plots the regression line on the scatter plot.
5. **Annotation:** Adds the linear regression equation to the plot. The position is dynamically calculated to avoid overlapping data points. Adjust the multipliers (0.1 and 0.1 in the example) if necessary.
6. **Labels and Title:** Sets the plot title, x-axis label, and y-axis label, including the date.
7. **Grid:** Adds a grid to the plot.
8. **R-squared:** Prints the r-squared value.
9. **Show Plot:** Displays the plot.

**Create a Temperature vs. Latitude Linear Regression Plot for the Southern Hemisphere:**

1. The code is almost identical to the Northern Hemisphere version, with these key changes:

* **Data Source:** It uses southern\_hemi\_df instead of northern\_hemi\_df to get the latitude and temperature data.
* **Plot Title:** The plot title is updated to "Southern Hemisphere - Max Temp vs. Latitude Linear Regression".

1. The rest of the code (linear regression calculation, plotting, annotation, etc.) remains the same.

**Create a Humidity vs. Latitude Linear Regression Plot for the Northern Hemisphere:**

1. Key changes from the Max Temp vs. Latitude plot:

* **Y-axis data:** Now uses northern\_hemi\_df["Humidity"] for the y-values.
* **Y-axis label:** Updated to "Humidity (%)".
* **Plot title:** Updated to reflect the humidity relationship.

1. The rest of the code (linear regression, plotting, annotation, etc.) remains the same.

**Create a Humidity vs. Latitude Linear Regression Plot for the Southern Hemisphere:**

1. The changes from the Northern Hemisphere version are:

* **Data Source:** Uses southern\_hemi\_df instead of northern\_hemi\_df.
* **Plot Title:** The title is updated to reflect the Southern Hemisphere.

1. Everything else (linear regression calculation, plotting, annotation, etc.) remains the same.

**Create a Cloudiness vs. Latitude Linear Regression Plot for the Northern Hemisphere:**

1. The changes from the previous plots are:

* **Y-axis Data:** Uses northern\_hemi\_df["Cloudiness"] for the y-values.
* **Y-axis Label:** Changed to "Cloudiness (%)".
* **Plot Title:** Updated to "Northern Hemisphere - Cloudiness (%) vs. Latitude Linear Regression".

1. The rest of the code (linear regression, plotting, annotation, etc.) is the same.

**Create a Cloudiness vs. Latitude Linear Regression Plot for the Northern Hemisphere:**

1. The key changes from the Northern Hemisphere version are:

* **Data Source:** It uses southern\_hemi\_df instead of northern\_hemi\_df.
* **Plot Title:** The title is updated to "Southern Hemisphere - Cloudiness (%) vs. Latitude Linear Regression".

1. All other aspects of the code (linear regression, plotting, annotation, grid, etc.) remain the same.

**Create a Wind Speed vs. Latitude Linear Regression Plot for the Northern Hemisphere:**

1. Here's what's different for this plot:

* **Y-axis Data:** Uses northern\_hemi\_df["Wind Speed"] for the y-axis values.
* **Y-axis Label:** Changed to "Wind Speed (m/s)".
* **Plot Title:** Updated to reflect wind speed vs. latitude.

1. The core code structure (linear regression, plotting, annotation, etc.) remains consistent.

**Create a Wind Speed vs. Latitude Linear Regression Plot for the Northern Hemisphere:**

1. Here's what's changed for the Southern Hemisphere version:

* **Data Source:** Uses southern\_hemi\_df instead of northern\_hemi\_df.
* **Plot Title:** Updated to "Southern Hemisphere - Wind Speed (m/s) vs. Latitude Linear Regression".

1. The rest of the code (linear regression, plotting, annotation, etc.) remains identical.

**Discussion about the linear relationships:**

**1. Maximum Temperature vs. Latitude:**

* **Strong Negative Correlation (Northern Hemisphere):** As expected, the Northern Hemisphere plot demonstrates a strong negative correlation between latitude and maximum temperature. Cities closer to the equator (lower latitudes) experience higher maximum temperatures. The r-squared value from our analysis confirms this strong negative relationship, indicating that latitude explains a significant portion of the variability in maximum temperatures.
* **Strong Positive Correlation (Southern Hemisphere):** Mirroring the Northern Hemisphere, the Southern Hemisphere plot shows a strong positive correlation. As you move closer to the equator (from higher negative latitudes to lower negative latitudes), maximum temperatures increase. The calculated r-squared value supports this strong positive correlation. This reinforces the idea that solar radiation, which is most intense at the equator, plays a dominant role in determining temperature.

**2. Humidity vs. Latitude:**

* **Weak or No Relationship (Both Hemispheres):** The plots for humidity versus latitude reveal a much weaker relationship compared to temperature. In both hemispheres, the scattered data points and the relatively flat regression lines indicate that latitude is not a strong predictor of humidity. The low r-squared values we obtained support this observation. Other factors like proximity to water bodies, wind patterns, and local geography likely play a more significant role in determining humidity levels. While a slight positive or negative slope might be present in the regression lines, the low r-squared values suggest these trends are weak and not statistically significant.

**3. Cloudiness vs. Latitude:**

* **No Consistent Linear Relationship:** The cloudiness versus latitude plots exhibit no consistent linear relationship in either hemisphere. The scattered data points and low r-squared values confirm this. The plots may show clusters of cities with similar cloudiness at certain latitudes, hinting at regional weather patterns. For example, you might observe a band of higher cloudiness around mid-latitudes, potentially associated with prevailing weather systems. However, a global linear trend is not apparent.

**4. Wind Speed vs. Latitude:**

* **Complex Relationship:** Wind speed's relationship with latitude is complex and not strongly linear. The r-squared values obtained are relatively low, confirming a weak linear relationship. The plots might show a slight increase in wind speeds at higher latitudes, potentially related to the jet stream or larger pressure gradients. However, this trend is not pronounced, and other local factors likely exert a greater influence on wind speeds. Differences in topography, temperature gradients, and proximity to coastlines can create localized wind patterns that override any broad latitudinal trend.

1. **Overall Conclusions:**

* **Latitude and Temperature:** The strongest relationships observed are between latitude and maximum temperature, with clear negative (Northern Hemisphere) and positive (Southern Hemisphere) correlations.
* **Other Factors at Play:** For humidity, cloudiness, and wind speed, latitude alone is not a strong predictor. Other geographic and meteorological factors play a more significant role.
* **Data Limitations:** It's crucial to remember these observations are based on a snapshot in time. Analyzing data across different seasons and longer timeframes would be essential to draw more robust conclusions about climate patterns.

1. **To strengthen the analysis, one could:**

* **Increase sample size:** Include more cities in your dataset.
* **Analyze data over time:** Track these relationships over different seasons or years to see if the patterns change.
* **Consider other factors:** Incorporate other relevant variables into your analysis, such as altitude, proximity to water, or prevailing wind patterns.