Task 1: Git & Environment Setup Challenges

1. **Repository Initialization and Local Setup:**
    a. Create the `solar-challenge-week1` repository on GitHub.
    b. Clone it to your local machine.
    c. Set up a Python 3 virtual environment using either `venv` or Conda. Activate it.
    d. Install the necessary data science libraries (like pandas, numpy, matplotlib, seaborn) within your virtual environment. Create a `requirements.txt` file reflecting these.
2. **Branching and Initial Commits:**
    a. Create a branch named `setup-task`.
    b. Create a `.gitignore` file that ignores the `data/` directory and any `*.csv` files, as well as `.ipynb_checkpoints/`. Commit this with the message "init: add .gitignore".
    c. Install the libraries from your `requirements.txt` using `pip install -r requirements.txt` within your activated virtual environment. Commit this with the message "chore: venv setup".
    d. Create the suggested folder structure (`.vscode`, `.github`, `src`, `notebooks`, `tests`, `scripts`). Add empty `__init__.py` files to the `src`, `notebooks`, `tests`, and `scripts` directories. Commit this with the message "feat: create directory structure".
3. **Basic Continuous Integration (CI):**
    a. Create the directory `.github/workflows`.
    b. Inside, create a file named `ci.yml`.
    c. Configure this GitHub Actions workflow to:
        i. Checkout your code.
        ii. Set up a Python environment.
        iii. Install the dependencies listed in `requirements.txt`.
        iv. *(Challenge)* Add a step to simply print the Python version to the CI logs using `python --version`.
4. **README Documentation and Pull Request:**
    a. In your `README.md` file, clearly document the steps someone else would need to take to reproduce your development environment. This should include instructions on cloning the repo, setting up the virtual environment, and installing dependencies.
    b. Commit any final changes to your `setup-task` branch.
    c. Push the `setup-task` branch to your GitHub repository.

   d. Create a Pull Request (PR) from `setup-task` to `main`.
   e. Merge the Pull Request into your `main` branch.

Task 2: Data Profiling, Cleaning & EDA Challenges (for one country initially, e.g., Benin)

1. **Branching and Notebook Creation:**
   a. Create a new branch named `eda-benin`.
   b. Create a Jupyter Notebook named `benin_eda.ipynb` in the `notebooks/` directory.
2. **Initial Data Loading and Exploration:**
   a. *(Assuming you have a `benin.csv` file in a `data/` directory - remember this should be ignored by Git)* Load the Benin dataset into a pandas DataFrame within your notebook.
   b. Generate summary statistics for all numeric columns using `df.describe()`.
   c. Calculate and display the number of missing values in each column using `df.isna().sum()`. Identify and list any columns with more than 5% missing values.
3. **Outlier Detection and Basic Cleaning:**
   a. Focus on the 'GHI', 'DNI', 'DHI', 'ModA', 'ModB', 'WS', and 'WSgust' columns.
   b. For these columns, compute the Z-score for each value.
   c. Create a boolean flag column (e.g., 'outlier_flag') that is `True` if the absolute Z-score for any of these key columns is greater than 3, and `False` otherwise.
   d. Decide on a strategy for handling missing values in the key columns (GHI, DNI, DHI, ModA, ModB, WS, WSgust'). Implement either dropping rows with any missing values in these columns OR imputing missing values using the median. Document your choice.
4. **Time Series Analysis:**
   a. Assuming your dataset has a 'Timestamp' column, convert it to a datetime object if it isn't already.
   b. Create line plots (or bar charts if appropriate for the frequency of your data) showing the trends of 'GHI', 'DNI', 'DHI', and 'Tamb' against the 'Timestamp'.
   c. *(Challenge)* Try to extract month and hour information from the 'Timestamp' and create a visualization (e.g., boxplot or line plot with error bars) to observe potential monthly patterns in 'GHI' or daily patterns in 'Tamb'.
5. **Cleaning Impact Analysis:**
   a. If you implemented an outlier flagging mechanism, group your DataFrame by the 'outlier_flag' column.

    b. Calculate and visualize the average 'ModA' and 'ModB' values for the flagged and non-flagged groups (e.g., using bar plots). This will help show the impact of your cleaning.

6. **Correlation and Relationship Analysis:**
    a. Calculate the correlation matrix for 'GHI', 'DNI', 'DHI', 'TModA', and 'TModB' and visualize it using a heatmap.
    b. Create scatter plots to explore the relationships between:
        i. 'WS', 'WSgust', and 'WD' (on separate plots) against 'GHI'.
        ii. 'RH' against 'Tamb'.
        iii. 'RH' against 'GHI'.
7. **Wind and Distribution Analysis:**
    a. *(Challenge)* Explore creating a wind rose plot using libraries like `windrose` or by manually binning wind direction and speed. If that's too complex initially, a radial bar plot showing the frequency of different wind directions could be a good start.
    b. Create histograms for 'GHI' and one other variable of interest (e.g., 'WS') to visualize their distributions.
8. **Temperature Analysis:**
    a. Write a brief observation (in a markdown cell) about any potential influence of 'RH' on 'Tamb' or 'GHI' based on your visualizations and understanding.
9. **Bubble Chart:**
    a. Create a bubble chart with 'GHI' on one axis and 'Tamb' on the other. Use the 'RH' column to determine the size of the bubbles. If 'BP' (Barometric Pressure) is available and seems relevant, you could use that instead of or in addition to 'RH'.
10. **Export Cleaned Data:**
    a. Save the cleaned DataFrame to `data/benin_clean.csv`. **Ensure that the `data/` directory is still in your `.gitignore` and you do not commit this CSV file.**

Task 3: Cross-Country Comparison Challenges

1. **Branching and Notebook Creation:**
    a. Create a new branch named `compare-countries`.
    b. Create a Jupyter Notebook named `compare_countries.ipynb` in the `notebooks/` directory.
2. **Load Cleaned Data:**

    a. Load the cleaned CSV files for Benin (`benin_clean.csv`), Sierra Leone (`sierra_leone_clean.csv`), and Togo (`togo_clean.csv`) from the `data/` directory into separate pandas DataFrames.

3. **Metric Comparison (Boxplots):**
    a. Create three separate box plots, one for each of 'GHI', 'DNI', and 'DHI'. Each box plot should display the distribution of the metric for all three countries side-by-side, with different colors representing each country.
4. **Summary Table:**
    a. Calculate the mean, median, and standard deviation for 'GHI', 'DNI', and 'DHI' for each of the three countries.
    b. Present these statistics in a clear summary table (e.g., using pandas `groupby()` and aggregation).
5. **Statistical Testing (Optional but Recommended):**
    a. Perform a one-way ANOVA test (or the non-parametric Kruskal-Wallis test if the assumptions of ANOVA are not met) on the 'GHI' values across the three countries to determine if the differences in means are statistically significant.
    b. Report the calculated p-value.
    c. *(Challenge)* Briefly explain what the p-value signifies in the context of your comparison.
6. **Key Observations:**
    a. In a markdown cell, write down at least three key observations based on your analysis. These should highlight the relative solar potential and any significant differences you've noticed between Benin, Sierra Leone, and Togo. Aim for actionable insights.
7. **(Bonus) Visual Summary:**
    a. Create a bar chart that ranks the three countries based on their average 'GHI'.

Bonus Task: Interactive Dashboard Challenges

1. **Branching and App Structure:**
    a. Create a new branch named `dashboard-dev`.
    b. Create the app/ directory with an empty `__init__`.py file and a `main.py` file.
2. **Basic Streamlit UI (app/`main.py`):**
    a. Import the `streamlit` library.
    b. Add a title to your app.

   c. Implement a Streamlit widget that allows the user to select one or more countries (Benin, Sierra Leone, Togo).
   d. *(Challenge)* Based on the selected countries, load the corresponding cleaned CSV files (assume they are in the `data/` directory). *Remember the app should read local CSVs, not have the data embedded*.
   e. If at least one country is selected, display a box plot of 'GHI' for the selected countries using `streamlit.pyplot()`.
3. **Top Regions Table (Conceptual):**
   a. *(Conceptual Challenge - you might not have region-specific data yet, so focus on the structure)* Outline how you would display a table of "top regions" based on some metric (e.g., average GHI) if your data contained regional information. You don't need to implement the data loading or calculation for this yet, just the Streamlit table structure.
4. **Git Hygiene and Commits:**
   a. Ensure the `data/` directory is still ignored in your `.gitignore`.
   b. Make one commit with the message "feat: basic Streamlit UI".
5. **README Documentation:**
   a. Update your main `README.md` to include instructions on how to run your Streamlit application (e.g., navigating to the app/ directory and running `streamlit run main.py`).

Important Reminders:

- **Git Hygiene:** Keep the `data/` directory and any raw or intermediate CSV files ignored by Git.
- **Documentation:** Document your code and analysis clearly in your notebooks and reports.
- **Proactivity:** Don't hesitate to look up resources and learn new techniques as you go. The "Proactivity to self-learn - sharing references" KPI is important!
- **Collaboration (Future):** While these are individual tasks, keep in mind the importance of collaboration using GitHub issues and projects in a real-world scenario.