Overview

Objective

To process the given data and create a visualization using the pre-processed dataset.

Dataset

The data is organized into folders by year-month and parameter (PR or GHI).

- ***PR (Performance Ratio):*** This parameter is used to track the daily performance of the PV plant. A high value indicates that the plant is performing well and there are no issues.
- ***GHI (Global Horizontal Irradiance):*** This parameter tracks the total irradiation for a particular day. A high value indicates a sunny day.

For example,

PR/
  2023-01/
    2023-01-01_PR.csv
    2023-01-06_PR.csv

GHI/
  2023-01/
    2023-01-01_GHI.csv
    2023-01-06_GHI.csv

Import Necessary Modules

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import os
import numpy as np
from datetime import datetime, timedelta
import matplotlib.dates as mdates     # Import for date formatting
from matplotlib.lines import Line2D    # Import Line2D for custom legend handles
```

In [2]:

```
# Path to directory
data_directory_path = "D:\Assignment\data"
```

Data Preprocessing

For this combine PR (Performance Ratio) and GHI (Global Horizontal Irradiance) data from various CSV files into a single CSV file with three columns: Date, GHI, and PR. The final CSV should contain 982 rows. It's required to create a single, readable function for this preprocessing.

Steps for the Preprocessing Function:

- Traverse through the PR and GHI directories.
- For each day, read both the PR and GHI CSV files.
- From each CSV, extract the relevant daily value. The date can be extracted from the filename.
- Create a structure a Pandas DataFrame to hold the Date, GHI, and PR for each day.
- Handle Missing Data (if any)
- Delete duplicated data (if any)
- Once all data is collected, save it into a single CSV file.

In [3]:

```python
def data_preprocess(data_directory):
    """
    Processes PR and GHI data from the specified directory structure and generates
    Each CSV file contains data for multiple days.
    """

    print(f"\nStarting data preprocessing from: {data_directory}")

    pr_root_dir = os.path.join(data_directory, 'PR')
    ghi_root_dir = os.path.join(data_directory, 'GHI')

    # Validate existence of PR and GHI root directories
    print(f"\nChecking PR directory: {pr_root_dir}")
    if not os.path.isdir(pr_root_dir):
        print(f"\nError: PR directory not found or is not a directory at {pr_root_d
        return None

    print(f"\nChecking GHI directory: {ghi_root_dir}")
    if not os.path.isdir(ghi_root_dir):
        print(f"\nError: GHI directory not found or is not a directory at {ghi_root
        return None

    list_of_dfs = []    # List to store DataFrames from each multi-day CSV file

    # --- Collect PR file paths ---
    # Map PR file paths by their starting date (e.g., '2019-07-01' -> 'path/to/2019
    pr_files_map = {}
    total_pr_files_found = 0

    # Traverse PR root directory
    for year_month in sorted(os.listdir(pr_root_dir)):
        current_pr_month_path = os.path.join(pr_root_dir, year_month)
        if os.path.isdir(current_pr_month_path):    # Ensure it's a directory like
            for pr_filename in sorted(os.listdir(current_pr_month_path)):
                if pr_filename.endswith('.csv'):
                    date_key = pr_filename.replace('.csv', '')    # Extract 'YYYY-M
                    try:
                        pd.to_datetime(date_key)    # Validate if it's a valid date
                    pr_files_map[date_key] = os.path.join(current_pr_month_path
                    total_pr_files_found += 1
                except ValueError:
                    print(f"\nSkipping non-date or malformed PR file: {pr_filen
            else:
                print(f"\nSkipping non-CSV PR file: {pr_filename} in {current_p

# --- Process GHI files and merge with corresponding PR data ---
# This loop processes CSV files containing data for multiple days.
files_processed_count = 0     # Counts the number of successfully processed mul
total_ghi_files_found = 0

# Traverse GHI root directory
for year_month in sorted(os.listdir(ghi_root_dir)):
    current_ghi_month_path = os.path.join(ghi_root_dir, year_month)

    if not os.path.isdir(current_ghi_month_path):
        print(f"\nSkipping non-directory item in GHI folder: {current_ghi_month
        continue    # Skip if not a directory

    # Iterate through GHI files in the current month folder
    for ghi_filename in sorted(os.listdir(current_ghi_month_path)):
        if ghi_filename.endswith('.csv'):
            date_key = ghi_filename.replace('.csv', '')    # Extract 'YYYY-MM-D
            try:
                pd.to_datetime(date_key)    # Validate date string
                total_ghi_files_found += 1

                # Check if a corresponding PR file exists for this date_key
                if date_key in pr_files_map:
                    pr_file_path = pr_files_map[date_key]
                    ghi_file_path = os.path.join(current_ghi_month_path, ghi_fi

                    #print(f"\nProcessing data from CSVs starting: {date_key}")

                    try:
                        # Read entire CSV files for the current data block into
                        pr_df = pd.read_csv(pr_file_path)
                        ghi_df = pd.read_csv(ghi_file_path)

                        # Convert 'Date' columns to datetime objects for accura
                        pr_df['Date'] = pd.to_datetime(pr_df['Date'])
                        ghi_df['Date'] = pd.to_datetime(ghi_df['Date'])

                        # Merge the PR and GHI dataframes for this data block o
                        # 'outer' merge ensures all dates are kept even if one
                        # No suffixes needed as 'PR' and 'GHI' are distinct col
                        ghi_pr_df = pd.merge(pr_df, ghi_df, on='Date', how='out

                        # Ensure columns are in the required order: Date, GHI,
                        ghi_pr_df = ghi_pr_df[['Date', 'GHI', 'PR']]

                        list_of_dfs.append(ghi_pr_df)
                        files_processed_count += 1
                    except Exception as e:
                        print(f"\nWarning: Could not read or merge data from fi
                            print(f"\nPlease check the internal CSV format of: {pr_
                    else:
                        print(f"\nWarning: GHI file for {date_key} found, but no ma
                except ValueError:
                    print(f"\nSkipping non-date or malformed GHI file: {ghi_filenam
            else:
                print(f"\nSkipping non-CSV GHI file: {ghi_filename} in {current_ghi

    # --- Final Consolidation and Reporting ---
    if not list_of_dfs:
        print("\nNo data blocks found or processed. Please verify your data directo
        return None

    # Concatenate all individual data block DataFrames into one final DataFrame
    # drop_duplicates ensures no duplicate dates if blocks overlapped or individual
    # sort_values and reset_index ensure chronological order and clean index
    df = pd.concat(list_of_dfs).drop_duplicates(subset=['Date']).sort_values(by='Da

    # Save the processed data to a CSV file as 'Date,GHI,PR' as specified
    output_csv_path = "processed_data.csv"
    df.to_csv(output_csv_path, index=False)
    print(f"\nProcessed data saved to '{output_csv_path}' with {len(df)} rows.")

    # Explicitly report the count discrepancy
    print(f"\n--- Data Count Report for the Data ---")
    print(f"\nNumber of paired PR and GHI data files processed: {files_processed_co
    print(f"Total individual PR files found in directories: {total_pr_files_found}"
    print(f"Total individual GHI files found in directories: {total_ghi_files_found
    print(f"Total rows in final collated data (after combining all blocks): {len(df

    if len(df) == 982:
        print("\nThis matches the expected 982 rows. ")
    else:
        print(f"\nThe final file should contain 982 rows, but {len(df)} were formed
    return df
```

In [4]:

```
# Read the actual files, generate a 'processed_data.csv' file, and return a DataFra
processed_dataframe = data_preprocess(data_directory_path)
```

```
Starting data preprocessing from: D:\Assignment\data

Checking PR directory: D:\Assignment\data\PR

Checking GHI directory: D:\Assignment\data\GHI

Processed data saved to 'processed_data.csv' with 982 rows.

--- Data Count Report for the Data ---

Number of paired PR and GHI data files processed: 197
Total individual PR files found in directories: 197
Total individual GHI files found in directories: 197
Total rows in final collated data (after combining all blocks): 982

This matches the expected 982 rows.
```

Data Visualization

Generate graph using the processed data. This also requires a single, readable function.

Graph Components -

- Scatter Plot: This will show daily PR values. The color shade of the scatter points should represent the GHI value. (GHI < 2: Navy blue, 2-4: Light blue, 4-6: Orange, 6: Brown)

- 30-day Moving Average of PR: A red line representing the 30-day moving average of the PR.

- Budget Line (Dark Green): This line starts at 73.9% for the first year (July 2019 to June 2020) and reduces by 0.8% annually. This reduction should be dynamic, not hardcoded.

- "Points above Target Budget PR": This should display the number of PR points above the Budget PR for that particular year and the percentage.
  Average PR for last 7-d, 30-d, 60-d, 90-d, 365-d, and Lifetime.

- Dynamic Budget Line: Implement logic to calculate the budget line value based on the year of the data point.

- Color Mapping: Create a function or use conditional logic to assign colors to scatter points based on GHI ranges.

- Annotations: Use plt.text() or similar functions to add the required text annotations.

- Labels and Title: Ensure the graph has appropriate x-axis (Date), y-axis (Performance Ratio [%]), and a descriptive title.

- Legend: Include a legend for the scatter plot GHI ranges, the 30-d moving average, and the budget line.

- Date Range Arguments: Script accept start_date and end_date as arguments. This means the visualization function should filter the processed data based on these dates before plotting.

In [5]:

```
def generate_performance_graph(df, start_date=None, end_date=None):
    """
    Generates the Performance Ratio (PR) evolution graph, including the 30-day movi
    The generated graph is also saved as a PNG file.
    """

    if df is None or df.empty:
        print("\nNo data to generate graph. Please check data preprocessing.")
        return

    # Ensure 'Date' column is datetime
    df['Date'] = pd.to_datetime(df['Date'])
```

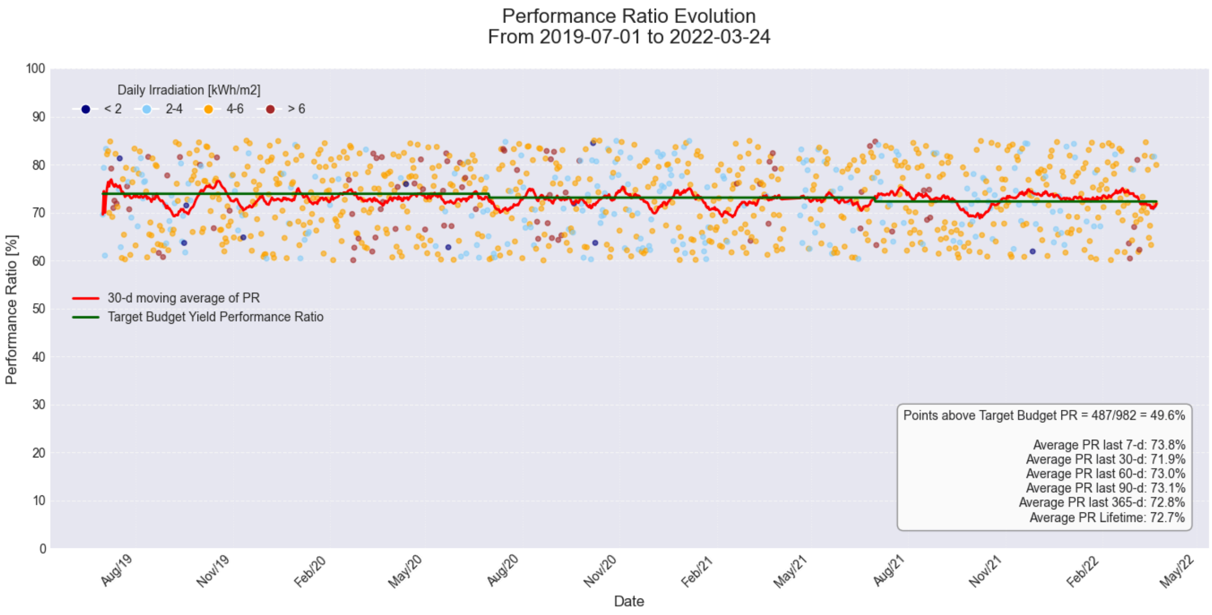
```python
# Apply date range filter
filtered_df = df.copy()
if start_date:
    filtered_df = filtered_df[filtered_df['Date'] >= pd.to_datetime(start_date)
if end_date:
    filtered_df = filtered_df[filtered_df['Date'] <= pd.to_datetime(end_date)]

if filtered_df.empty:
    print(f"No data found for the specified date range: {start_date} to {end_da
    return

# Calculate 30-day moving average of PR
filtered_df['PR_30d_MA'] = filtered_df['PR'].rolling(window=30, min_periods=1).

# Dynamic Budget Line Calculation
budget_start_date = datetime(2019, 7, 1)
initial_budget_pr = 73.9
annual_reduction = 0.8

# Logic for calculating Budget_PR
filtered_df['Budget_PR'] = filtered_df['Date'].apply(
    lambda date: (
        initial_budget_pr - (
            max(0, (date.year - budget_start_date.year if date.month >= budget_
            * annual_reduction
        )
    ) if date >= budget_start_date else np.nan
)

# GHI Color Mapping for Scatter Plot
conditions = [
    filtered_df['GHI'] < 2,
    (filtered_df['GHI'] >= 2) & (filtered_df['GHI'] < 4),
    (filtered_df['GHI'] >= 4) & (filtered_df['GHI'] < 6),
    filtered_df['GHI'] >= 6
]
colors = ['navy', 'lightskyblue', 'orange', 'brown']
ghi_colors = np.select(conditions, colors, default='grey')

# Plotting
plt.style.use('seaborn-v0_8-darkgrid')
fig, ax = plt.subplots(figsize=(14, 7))

# Scatter plot for daily PR, colored by GHI
scatter = ax.scatter(filtered_df['Date'], filtered_df['PR'], c=ghi_colors, s=15

# 30-day moving average of PR
ax.plot(filtered_df['Date'], filtered_df['PR_30d_MA'], color='red', linewidth=2

# Dynamic Budget Line
ax.plot(filtered_df['Date'], filtered_df['Budget_PR'], color='darkgreen', linew
        label=f'Target Budget Yield Performance Ratio [1Y-{initial_budget_pr}%,

# Annotations and Labels
# Dynamic Title based on date range
```

```
    if start_date and end_date:
        title_text = f"Performance Ratio Evolution\nFrom {start_date} to {end_date}
    else:
        min_date_str = filtered_df['Date'].min().strftime('%Y-%m-%d')
        max_date_str = filtered_df['Date'].max().strftime('%Y-%m-%d')
        title_text = f"Performance Ratio Evolution\nFrom {min_date_str} to {max_dat

    ax.set_title(title_text, fontsize=16, pad=20)
    ax.set_xlabel('Date', fontsize=12)
    ax.set_ylabel('Performance Ratio [%]', fontsize=12)
    ax.set_ylim(0, 100)

    # Horizontal grid lines, fainter
    ax.grid(axis='y', linestyle='--', alpha=0.6)
    ax.grid(axis='x', linestyle=':', alpha=0.3)

    # Set Y-axis ticks to match original (every 10 units)
    ax.set_yticks(np.arange(0, 101, 10))

    # X-axis formatting to match original (e.g., Jul/19, Oct/19)
    ax.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
    ax.xaxis.set_minor_locator(mdates.MonthLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%b/%y'))
    plt.setp(ax.get_xticklabels(), rotation=45, ha='right')

    # Legend Placement
    # GHI legend (Daily Irradiation [kWh/m2]) - positioned at the top inside the pl
    legend_elements_ghi = [
        Line2D([0], [0], marker='o', color='w', label='< 2', markerfacecolor='navy'
        Line2D([0], [0], marker='o', color='w', label='2-4', markerfacecolor='light
        Line2D([0], [0], marker='o', color='w', label='4-6', markerfacecolor='orang
        Line2D([0], [0], marker='o', color='w', label='> 6', markerfacecolor='brown
    ]
    first_legend = ax.legend(handles=legend_elements_ghi, loc='upper left', bbox_to
                             ncol=4, title="Daily Irradiation [kWh/m2]", title_font
                             frameon=False, columnspacing=1.0, handletextpad=0.5)
    ax.add_artist(first_legend)

    # Main legend for lines (30-d moving average and Budget PR) - positioned lower
    legend_elements_lines = [
        Line2D([0], [0], color='red', lw=2, label='30-d moving average of PR'),
        Line2D([0], [0], color='darkgreen', lw=2, label='Target Budget Yield Perfor
    ]
    ax.legend(handles=legend_elements_lines, loc='center left', bbox_to_anchor=(0.0
              frameon=False, fancybox=True, shadow=False)

    # Calculate and display average PRs
    avg_pr_7d = filtered_df['PR'].tail(7).mean()
    avg_pr_30d = filtered_df['PR'].tail(30).mean()
    avg_pr_60d = filtered_df['PR'].tail(60).mean()
    avg_pr_90d = filtered_df['PR'].tail(90).mean()
    avg_pr_365d = filtered_df['PR'].tail(365).mean()
    avg_pr_lifetime = filtered_df['PR'].mean()

    # Calculate points above Target Budget PR
    above_budget_df = filtered_df[filtered_df['PR'] > filtered_df['Budget_PR']]
    total_points = len(filtered_df)
    points_above_budget = len(above_budget_df)
    percentage_above_budget = (points_above_budget / total_points) * 100 if total_p

    # Text box for average PRs and points above budget - positioned at bottom right
    text_box_content = (
        f"Points above Target Budget PR = {points_above_budget}/{total_points} = {p
        f"Average PR last 7-d: {avg_pr_7d:.1f}%\n"
        f"Average PR last 30-d: {avg_pr_30d:.1f}%\n"
        f"Average PR last 60-d: {avg_pr_60d:.1f}%\n"
        f"Average PR last 90-d: {avg_pr_90d:.1f}%\n"
        f"Average PR last 365-d: {avg_pr_365d:.1f}%\n"
        f"Average PR Lifetime: {avg_pr_lifetime:.1f}%"
    )
    ax.text(0.98, 0.05, text_box_content, transform=ax.transAxes, fontsize=10,
            verticalalignment='bottom', horizontalalignment='right',
            bbox=dict(boxstyle='round,pad=0.5', fc='white', alpha=0.8, ec='gray'))

    plt.tight_layout(rect=[0, 0, 0.98, 1])

    # Save the plot
    if start_date and end_date:
        output_filename = f"performance_ratio_evolution_{start_date}_to_{end_date}.
    else:
        output_filename = "performance_ratio_evolution_full_dataset.png"

    plt.savefig(output_filename, bbox_inches='tight', dpi=300)
    print(f"Graph saved as '{output_filename}'")
    plt.show()
```

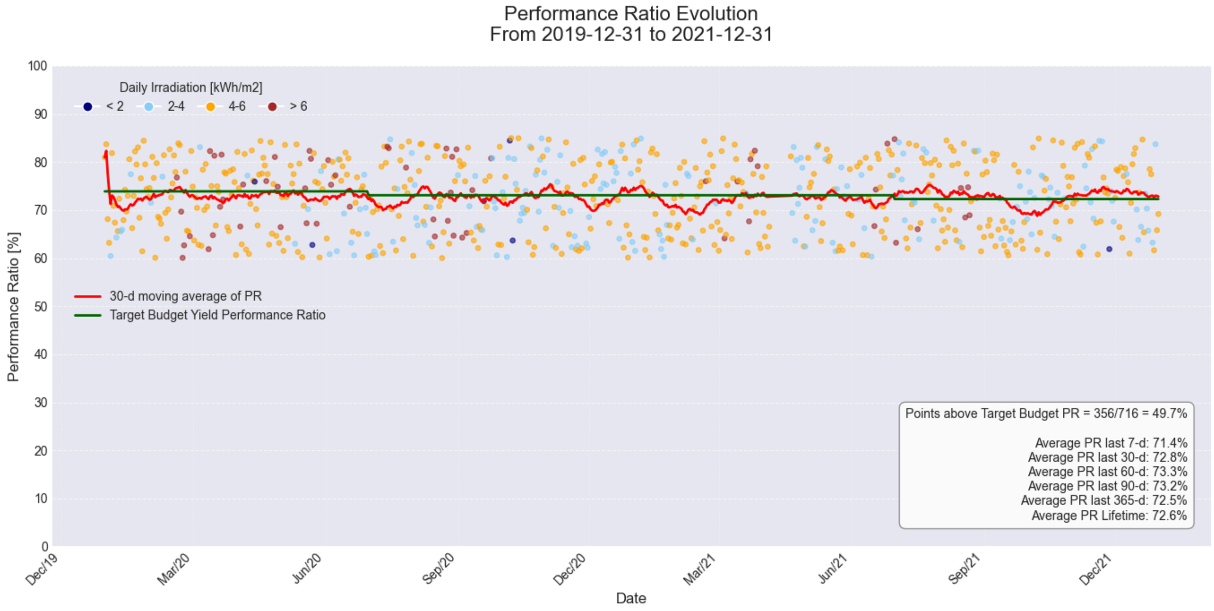
In [6]:

```
# Generate the PR evolution graph with the processed data
if processed_dataframe is not None:
    print("\nGenerating graph for the full dataset...")
    generate_performance_graph(processed_dataframe)

    # Generate graph for a specific date range (example dates)
    print("\nGenerating graph for a specific date range (e.g., 2019-12-31 to 2021-1
    generate_performance_graph(processed_dataframe, start_date="2019-12-31", end_da

    print("\nGraphs generated successfully (check your plot window).")
else:
    print("\nFailed to preprocess data. Cannot generate graph.")
```

```
Generating graph for the full dataset...
Graph saved as 'performance_ratio_evolution_full_dataset.png'
```



```
Generating graph for a specific date range (e.g., 2019-12-31 to 2021-12-31)...
Graph saved as 'performance_ratio_evolution_2019-12-31_to_2021-12-31.png'
```



```
Graphs generated successfully (check your plot window).
```