```
import numpy as np
import pandas as pd
from datetime import datetime, timedelta

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees
LONGITUDE = 77.23  # New Delhi longitude in degrees
SOLAR_CONSTANT = 1300  # W/m^2
ATMOSPHERIC_REDUCTION = 0.85  # 15% reduction for aerosols/clouds
STD_MERIDIAN = 82.5  # IST standard meridian (degrees)
DEG_PER_HOUR = 15  # Earth's rotation (degrees/hour)

# Function to calculate declination angle
def declination_angle(day_of_year):
    return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

# Function to calculate hour angle
def hour_angle(local_time, longitude, day):
    # Calculate solar noon adjustment
    longitude_diff = STD_MERIDIAN - longitude
    time_correction = longitude_diff / DEG_PER_HOUR * 60  # in minutes
    solar_noon = 12.0 + time_correction / 60  # in hours
    hours_from_noon = local_time - solar_noon
    return DEG_PER_HOUR * hours_from_noon

# Function to calculate solar altitude angle
def solar_altitude_angle(latitude, declination, hour_angle):
    lat_rad = np.radians(latitude)
    dec_rad = np.radians(declination)
    ha_rad = np.radians(hour_angle)
    sin_alpha = (np.sin(lat_rad) * np.sin(dec_rad) +
                 np.cos(lat_rad) * np.cos(dec_rad) * np.cos(ha_rad))
    return np.degrees(np.arcsin(sin_alpha))

# Function to calculate irradiance
def calculate_irradiance(alpha):
    cos_theta = np.sin(np.radians(alpha))  # For horizontal surface, cos(theta) = sin(alpha)
    I = SOLAR_CONSTANT * cos_theta * ATMOSPHERIC_REDUCTION
    return max(0, I)  # Ensure non-negative irradiance
```

```
    # Generate data for all days
    data = []
    start_date = datetime(2025, 1, 1)
    hours = list(range(9, 18))  # 9 AM to 5 PM

    for day in range(365):
        date = start_date + timedelta(days=day)
        day_of_year = day + 1
        delta = declination_angle(day_of_year)

        for hour in hours:
            h = hour_angle(hour, LONGITUDE, day_of_year)
            alpha = solar_altitude_angle(LATITUDE, delta, h)
            I = calculate_irradiance(alpha)
            data.append({
                'Date': date.strftime('%Y-%m-%d'),
                'Day': day_of_year,
                'Hour': hour,
                'Irradiance (W/m^2)': round(I, 2)
            })

    # Create DataFrame
    df = pd.DataFrame(data)

    # Save DataFrame to CSV for reference (optional in Colab)
    df.to_csv('solar_irradiance_2025_new_delhi.csv', index=False)

    # Function to query irradiance
    def get_irradiance():
        print("Enter date (e.g., '15 January'):")
        date_input = input().strip()
        try:
            date = datetime.strptime(f"{date_input} 2025", '%d %B %Y').strftime('%Y-%m-%d')
        except ValueError:
            print("Invalid date format. Use 'DD Month' (e.g., '15 January').")
            return

        print("Enter hour (9 to 17 for 9 AM to 5 PM):")
        try:
```

```
        hour = int(input().strip())
        if hour < 9 or hour > 17:
            print("Hour must be between 9 and 17.")
            return
    except ValueError:
        print("Invalid hour. Enter a number between 9 and 17.")
        return

    # Query DataFrame
    result = df[(df['Date'] == date) & (df['Hour'] == hour)]
    if not result.empty:
        irradiance = result['Irradiance (W/m^2)'].iloc[0]
        print(f"Solar Irradiance on {date_input} 2025 at {hour}:00 IST: {irradiance} W/m^2")
    else:
        print("Data not found for the specified date and time.")

# Run the query function
if __name__ == "__main__":
    get_irradiance()
```

```
Enter date (e.g., '15 January'):
12 march
Enter hour (9 to 17 for 9 AM to 5 PM):
14
Solar Irradiance on 12 march 2025 at 14:00 IST: 841.88 W/m^2
```

```
import numpy as np
import pandas as pd
from datetime import datetime, timedelta

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees
LONGITUDE = 77.23  # New Delhi longitude in degrees
SOLAR_CONSTANT = 1300  # W/m^2
DEFAULT_ATM_REDUCTION = 0.85  # Default 15% reduction for aerosols/clouds
STD_MERIDIAN = 82.5  # IST standard meridian (degrees)
DEG_PER_HOUR = 15  # Earth's rotation (degrees/hour)

# Function to calculate declination angle
def declination_angle(day_of_year):
```

```
        return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

    # Function to calculate hour angle
    def hour_angle(local_time, longitude, day):
        longitude_diff = STD_MERIDIAN - longitude
        time_correction = longitude_diff / DEG_PER_HOUR * 60  # in minutes
        solar_noon = 12.0 + time_correction / 60  # in hours
        hours_from_noon = local_time - solar_noon
        return DEG_PER_HOUR * hours_from_noon

    # Function to calculate solar altitude angle
    def solar_altitude_angle(latitude, declination, hour_angle):
        lat_rad = np.radians(latitude)
        dec_rad = np.radians(declination)
        ha_rad = np.radians(hour_angle)
        sin_alpha = (np.sin(lat_rad) * np.sin(dec_rad) +
                     np.cos(lat_rad) * np.cos(dec_rad) * np.cos(ha_rad))
        return np.degrees(np.arcsin(sin_alpha))

    # Function to calculate irradiance
    def calculate_irradiance(alpha, atm_reduction=DEFAULT_ATM_REDUCTION):
        cos_theta = np.sin(np.radians(alpha))  # For horizontal surface
        I = SOLAR_CONSTANT * cos_theta * atm_reduction
        return max(0, I)  # Ensure non-negative irradiance

    # Generate data for all days
    data = []
    start_date = datetime(2025, 1, 1)
    hours = list(range(9, 18))  # 9 AM to 5 PM

    for day in range(365):
        date = start_date + timedelta(days=day)
        day_of_year = day + 1
        delta = declination_angle(day_of_year)

        for hour in hours:
            h = hour_angle(hour, LONGITUDE, day_of_year)
            alpha = solar_altitude_angle(LATITUDE, delta, h)
            I = calculate_irradiance(alpha)
            data.append({
```

```
            'Date': date.strftime('%Y-%m-%d'),
            'Day': day_of_year,
            'Hour': hour,
            'Irradiance (W/m^2)': round(I, 2)
        })

# Create DataFrame
df = pd.DataFrame(data)

# Save DataFrame to CSV
df.to_csv('solar_irradiance_2025_new_delhi.csv', index=False)

# Function to query irradiance
def get_irradiance():
    print("Note: This script calculates clear-sky irradiance. Actual values depend on weather conditions.")
    print("Enter date (e.g., '18 November'):")
    date_input = input().strip()
    try:
        date = datetime.strptime(f"{date_input} 2025", '%d %B %Y').strftime('%Y-%m-%d')
    except ValueError:
        print("Invalid date format. Use 'DD Month' (e.g., '18 November').")
        return

    print("Enter hour (9 to 17 for 9 AM to 5 PM):")
    try:
        hour = int(input().strip())
        if hour < 9 or hour > 17:
            print("Hour must be between 9 and 17.")
            return
    except ValueError:
        print("Invalid hour. Enter a number between 9 and 17.")
        return

    print("Enter atmospheric reduction factor (0 to 1, default 0.85 for clear sky, e.g., 0.7 for cloudy):")
    try:
        atm_reduction = float(input().strip())
        if atm_reduction < 0 or atm_reduction > 1:
            print("Atmospheric reduction must be between 0 and 1.")
            return
    except ValueError:
```

```
        print("Using default atmospheric reduction (0.85).")
        atm_reduction = DEFAULT_ATM_REDUCTION

    # Query DataFrame
    result = df[(df['Date'] == date) & (df['Hour'] == hour)]
    if not result.empty:
        # Recalculate with custom atmospheric reduction
        day_of_year = result['Day'].iloc[0]
        delta = declination_angle(day_of_year)
        h = hour_angle(hour, LONGITUDE, day_of_year)
        alpha = solar_altitude_angle(LATITUDE, delta, h)
        irradiance = calculate_irradiance(alpha, atm_reduction)
        print(f"Solar Irradiance on {date_input} 2025 at {hour}:00 IST: {irradiance:.2f} W/m^2")
    else:
        print("Data not found for the specified date and time.")

# Run the query function
if __name__ == "__main__":
    get_irradiance()
```

```
Note: This script calculates clear-sky irradiance. Actual values depend on weather conditions.
Enter date (e.g., '18 November'):
20 december
Enter hour (9 to 17 for 9 AM to 5 PM):
13
Enter atmospheric reduction factor (0 to 1, default 0.85 for clear sky, e.g., 0.7 for cloudy):
0.8
Solar Irradiance on 20 december 2025 at 13:00 IST: 627.41 W/m^2
```

```
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
import requests
import os

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees
LONGITUDE = 77.23  # New Delhi longitude in degrees
SOLAR_CONSTANT = 1300  # W/m^2
DEFAULT_ATM_REDUCTION = 0.85  # Default 15% reduction for aerosols/clouds
```

```
STD_MERIDIAN = 82.5  # IST standard meridian (degrees)
DEG_PER_HOUR = 15  # Earth's rotation (degrees/hour)
CURRENT_DATE = datetime(2025, 7, 2)  # Current date (July 2, 2025)

# Function to calculate declination angle
def declination_angle(day_of_year):
    return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

# Function to calculate hour angle
def hour_angle(local_time, longitude, day):
    longitude_diff = STD_MERIDIAN - longitude
    time_correction = longitude_diff / DEG_PER_HOUR * 60  # in minutes
    solar_noon = 12.0 + time_correction / 60  # in hours
    hours_from_noon = local_time - solar_noon
    return DEG_PER_HOUR * hours_from_noon

# Function to calculate solar altitude angle
def solar_altitude_angle(latitude, declination, hour_angle):
    lat_rad = np.radians(latitude)
    dec_rad = np.radians(declination)
    ha_rad = np.radians(hour_angle)
    sin_alpha = (np.sin(lat_rad) * np.sin(dec_rad) +
                 np.cos(lat_rad) * np.cos(dec_rad) * np.cos(ha_rad))
    return np.degrees(np.arcsin(sin_alpha))

# Function to calculate clear-sky irradiance
def calculate_irradiance(alpha, atm_reduction=DEFAULT_ATM_REDUCTION):
    cos_theta = np.sin(np.radians(alpha))  # For horizontal surface
    I = SOLAR_CONSTANT * cos_theta * atm_reduction
    return max(0, I)  # Ensure non-negative irradiance

# Function to simulate fetching IMD data (placeholder for CSV or API)
def fetch_imd_data(date, hour):
    # Placeholder: Assume IMD data is in a CSV file 'imd_solar_data.csv'
    # Expected format: Date (YYYY-MM-DD), Hour (9-17), Irradiance (W/m^2)
    try:
        if os.path.exists('imd_solar_data.csv'):
            imd_df = pd.read_csv('imd_solar_data.csv')
            date_str = date.strftime('%Y-%m-%d')
            result = imd_df[(imd_df['Date'] == date_str) & (imd_df['Hour'] == hour)]
```

```
            if not result.empty:
                return result['Irradiance (W/m^2)'].iloc[0]
        return None  # No data found
    except Exception as e:
        print(f"Error accessing IMD data: {e}")
        return None

# Generate clear-sky data for all days (fallback)
data = []
start_date = datetime(2025, 1, 1)
hours = list(range(9, 18))  # 9 AM to 5 PM

for day in range(365):
    date = start_date + timedelta(days=day)
    day_of_year = day + 1
    delta = declination_angle(day_of_year)

    for hour in hours:
        h = hour_angle(hour, LONGITUDE, day_of_year)
        alpha = solar_altitude_angle(LATITUDE, delta, h)
        I = calculate_irradiance(alpha)
        data.append({
            'Date': date.strftime('%Y-%m-%d'),
            'Day': day_of_year,
            'Hour': hour,
            'Irradiance (W/m^2)': round(I, 2)
        })

# Create DataFrame for clear-sky data
clear_sky_df = pd.DataFrame(data)
clear_sky_df.to_csv('solar_irradiance_2025_new_delhi.csv', index=False)

# Function to query irradiance
def get_irradiance():
    print("Note: This script tries to use real IMD data for past dates. Future dates use clear-sky estimates.")
    print("Enter date (e.g., '18 November'):")
    date_input = input().strip()
    try:
        date = datetime.strptime(f"{date_input} 2025", '%d %B %Y')
    except ValueError:
```

```
            print("Invalid date format. Use 'DD Month' (e.g., '18 November').")
            return

        print("Enter hour (9 to 17 for 9 AM to 5 PM):")
        try:
            hour = int(input().strip())
            if hour < 9 or hour > 17:
                print("Hour must be between 9 and 17.")
                return
        except ValueError:
            print("Invalid hour. Enter a number between 9 and 17.")
            return

        # Check if date is in the future
        if date > CURRENT_DATE:
            print("Sorry, this data is not available for future dates.")
            return

        # Try to fetch IMD data for past dates
        imd_irradiance = fetch_imd_data(date, hour)
        if imd_irradiance is not None:
            print(f"Solar Irradiance on {date_input} 2025 at {hour}:00 IST (IMD data): {imd_irradiance:.2f} W/m^2")
            return

        # Fallback to clear-sky model
        print("IMD data not available. Using clear-sky estimate.")
        print("Enter atmospheric reduction factor (0 to 1, default 0.85 for clear sky, e.g., 0.7 for cloudy):")
        try:
            atm_reduction = float(input().strip())
            if atm_reduction < 0 or atm_reduction > 1:
                print("Atmospheric reduction must be between 0 and 1.")
                return
        except ValueError:
            print("Using default atmospheric reduction (0.85).")
            atm_reduction = DEFAULT_ATM_REDUCTION

        # Query clear-sky DataFrame
        date_str = date.strftime('%Y-%m-%d')
        result = clear_sky_df[(clear_sky_df['Date'] == date_str) & (clear_sky_df['Hour'] == hour)]
        if not result.empty:
```

```
        day_of_year = result['Day'].iloc[0]
        delta = declination_angle(day_of_year)
        h = hour_angle(hour, LONGITUDE, day_of_year)
        alpha = solar_altitude_angle(LATITUDE, delta, h)
        irradiance = calculate_irradiance(alpha, atm_reduction)
        print(f"Solar Irradiance on {date_input} 2025 at {hour}:00 IST (clear-sky estimate): {irradiance:.2f} W/m^2")
    else:
        print("Data not found for the specified date and time.")

# Run the query function
if __name__ == "__main__":
    get_irradiance()
```

```
Note: This script tries to use real IMD data for past dates. Future dates use clear-sky estimates.
Enter date (e.g., '18 November'):
10 january
Enter hour (9 to 17 for 9 AM to 5 PM):
15
IMD data not available. Using clear-sky estimate.
Enter atmospheric reduction factor (0 to 1, default 0.85 for clear sky, e.g., 0.7 for cloudy):
0.85
Solar Irradiance on 10 january 2025 at 15:00 IST (clear-sky estimate): 492.98 W/m^2
```

```
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
import os

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees
LONGITUDE = 77.23  # New Delhi longitude in degrees
SOLAR_CONSTANT = 1300  # W/m^2
STD_MERIDIAN = 82.5  # IST standard meridian (degrees)
DEG_PER_HOUR = 15  # Earth's rotation (degrees/hour)
CURRENT_DATE = datetime(2025, 7, 2)  # Current date (July 2, 2025)

# Cloud cover to atmospheric reduction mapping (oktas to factor)
CLOUD_REDUCTION = {
    range(0, 3): 0.9,  # Clear sky (0–2 oktas)
    range(3, 6): 0.7,  # Partly cloudy (3–5 oktas)
```

```
        range(6, 9): 0.4   # Overcast (6–8 oktas)
    }

    # Function to calculate declination angle
    def declination_angle(day_of_year):
        return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

    # Function to calculate hour angle
    def hour_angle(local_time, longitude, day):
        longitude_diff = STD_MERIDIAN - longitude
        time_correction = longitude_diff / DEG_PER_HOUR * 60  # in minutes
        solar_noon = 12.0 + time_correction / 60  # in hours
        hours_from_noon = local_time - solar_noon
        return DEG_PER_HOUR * hours_from_noon

    # Function to calculate solar altitude angle
    def solar_altitude_angle(latitude, declination, hour_angle):
        lat_rad = np.radians(latitude)
        dec_rad = np.radians(declination)
        ha_rad = np.radians(hour_angle)
        sin_alpha = (np.sin(lat_rad) * np.sin(dec_rad) +
                     np.cos(lat_rad) * np.cos(dec_rad) * np.cos(ha_rad))
        return np.degrees(np.arcsin(sin_alpha))

    # Function to calculate clear-sky irradiance
    def calculate_irradiance(alpha, atm_reduction=0.85):
        cos_theta = np.sin(np.radians(alpha))  # For horizontal surface
        I = SOLAR_CONSTANT * cos_theta * atm_reduction
        return max(0, I)

    # Function to get atmospheric reduction based on cloud cover
    def get_atm_reduction(cloud_cover):
        if pd.isna(cloud_cover):
            return 0.85  # Default for missing cloud data
        cloud_cover = int(cloud_cover)
        for okta_range, reduction in CLOUD_REDUCTION.items():
            if cloud_cover in okta_range:
                return reduction
        return 0.85
```

```
# Function to fetch IMD data from CSV
def fetch_imd_data(date, hour):
    try:
        if os.path.exists('imd_solar_data_2024.csv'):
            imd_df = pd.read_csv('imd_solar_data_2024.csv')
            date_str = date.strftime('%Y-%m-%d')
            result = imd_df[(imd_df['Date'] == date_str) & (imd_df['Hour'] == hour)]
            if not result.empty:
                ghi = result['GHI (W/m^2)'].iloc[0]
                cloud_cover = result.get('Cloud Cover (oktas)', pd.NA).iloc[0]
                return ghi, cloud_cover
        return None, None
    except Exception as e:
        print(f"Error accessing IMD data: {e}")
        return None, None

# Generate clear-sky data for 2024 (fallback)
data = []
start_date = datetime(2024, 1, 1)
hours = list(range(9, 18))  # 9 AM to 5 PM

for day in range(366):  # 2024 is a leap year
    date = start_date + timedelta(days=day)
    day_of_year = day + 1
    delta = declination_angle(day_of_year)

    for hour in hours:
        h = hour_angle(hour, LONGITUDE, day_of_year)
        alpha = solar_altitude_angle(LATITUDE, delta, h)
        I = calculate_irradiance(alpha)
        data.append({
            'Date': date.strftime('%Y-%m-%d'),
            'Day': day_of_year,
            'Hour': hour,
            'Irradiance (W/m^2)': round(I, 2)
        })

# Create DataFrame for clear-sky data
clear_sky_df = pd.DataFrame(data)
clear_sky_df.to_csv('solar_irradiance_2024_new_delhi.csv', index=False)
```

```
# Function to query irradiance
def get_irradiance():
    print("Enter date in 2024 (e.g., '15 January'):")
    date_input = input().strip()
    try:
        date = datetime.strptime(f"{date_input} 2024", '%d %B %Y')
    except ValueError:
        print("Invalid date format. Use 'DD Month' (e.g., '15 January').")
        return

    print("Enter hour (9 to 17 for 9 AM to 5 PM):")
    try:
        hour = int(input().strip())
        if hour < 9 or hour > 17:
            print("Hour must be between 9 and 17.")
            return
    except ValueError:
        print("Invalid hour. Enter a number between 9 and 17.")
        return

    # Check if date is in the future
    if date > CURRENT_DATE:
        print("Sorry, this data is not available for future dates.")
        return

    # Try to fetch IMD data
    ghi, cloud_cover = fetch_imd_data(date, hour)
    if ghi is not None:
        atm_reduction = get_atm_reduction(cloud_cover)
        # Adjust GHI based on cloud cover
        adjusted_ghi = ghi * atm_reduction
        print(f"Solar Irradiance on {date_input} 2024 at {hour}:00 IST (IMD data, cloud-adjusted): {adjusted_ghi:.2f} W/m^2")
        print(f"Cloud Cover: {cloud_cover if not pd.isna(cloud_cover) else 'Unknown'} oktas")
        return

    # Fallback to clear-sky model
    print("IMD data not available. Using clear-sky estimate.")
    print("Enter atmospheric reduction factor (0 to 1, default 0.85 for clear sky, e.g., 0.7 for cloudy):")
    try:
```

```
        atm_reduction = float(input().strip())
        if atm_reduction < 0 or atm_reduction > 1:
            print("Atmospheric reduction must be between 0 and 1.")
            return
    except ValueError:
        print("Using default atmospheric reduction (0.85).")
        atm_reduction = 0.85

    date_str = date.strftime('%Y-%m-%d')
    result = clear_sky_df[(clear_sky_df['Date'] == date_str) & (clear_sky_df['Hour'] == hour)]
    if not result.empty:
        day_of_year = result['Day'].iloc[0]
        delta = declination_angle(day_of_year)
        h = hour_angle(hour, LONGITUDE, day_of_year)
        alpha = solar_altitude_angle(LATITUDE, delta, h)
        irradiance = calculate_irradiance(alpha, atm_reduction)
        print(f"Solar Irradiance on {date_input} 2024 at {hour}:00 IST (clear-sky estimate): {irradiance:.2f} W/m^2")
    else:
        print("Data not found for the specified date and time.")

# Run the query function
if __name__ == "__main__":
    get_irradiance()
```

```
Enter date in 2024 (e.g., '15 January'):
20 march
Enter hour (9 to 17 for 9 AM to 5 PM):
12
IMD data not available. Using clear-sky estimate.
Enter atmospheric reduction factor (0 to 1, default 0.85 for clear sky, e.g., 0.7 for cloudy):
0.85
Solar Irradiance on 20 march 2024 at 12:00 IST (clear-sky estimate): 962.23 W/m^2
```

```
import numpy as np
from datetime import datetime, timedelta

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees
LONGITUDE = 77.23  # New Delhi longitude in degrees
SOLAR_CONSTANT = 1300  # W/m^2
```

```
STD_MERIDIAN = 82.5  # IST standard meridian (degrees)
DEG_PER_HOUR = 15  # Earth's rotation (degrees/hour)
CURRENT_DATE = datetime(2025, 7, 3)  # Current date (July 3, 2025)

# Cloud cover assumptions for specific dates
CLOUD_ASSUMPTIONS = {
    '2024-03-21': {'status': 'clear', 'oktas': 2, 'atm_reduction': 0.9},  # Spring, clear
    '2024-06-21': {'status': 'partly cloudy', 'oktas': 5, 'atm_reduction': 0.7},  # Monsoon start
    '2024-09-21': {'status': 'partly cloudy', 'oktas': 4, 'atm_reduction': 0.7},  # Monsoon end
    '2024-12-21': {'status': 'clear', 'oktas': 1, 'atm_reduction': 0.9}   # Winter, clear
}

# Day length assumptions from Report (1).pdf
DAY_LENGTHS = {
    '2024-03-21': 12.0,   # Equinox, ~12 hours
    '2024-06-21': 13.96,  # Summer solstice, max
    '2024-09-21': 12.0,   # Equinox, ~12 hours
    '2024-12-21': 10.32   # Winter solstice, min
}

# Function to calculate declination angle
def declination_angle(day_of_year):
    return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

# Function to calculate hour angle
def hour_angle(local_time, longitude, day):
    longitude_diff = STD_MERIDIAN - longitude
    time_correction = longitude_diff / DEG_PER_HOUR * 60  # in minutes
    solar_noon = 12.0 + time_correction / 60  # in hours
    hours_from_noon = local_time - solar_noon
    return DEG_PER_HOUR * hours_from_noon

# Function to calculate solar altitude angle
def solar_altitude_angle(latitude, declination, hour_angle):
    lat_rad = np.radians(latitude)
    dec_rad = np.radians(declination)
    ha_rad = np.radians(hour_angle)
    sin_alpha = (np.sin(lat_rad) * np.sin(dec_rad) +
                 np.cos(lat_rad) * np.cos(dec_rad) * np.cos(ha_rad))
    return np.degrees(np.arcsin(sin_alpha))
```

```
# Function to calculate solar azimuth angle
def solar_azimuth_angle(declination, hour_angle, alpha):
    dec_rad = np.radians(declination)
    ha_rad = np.radians(hour_angle)
    alpha_rad = np.radians(alpha)
    sin_as = np.cos(dec_rad) * np.sin(ha_rad) / np.cos(alpha_rad)
    sin_as = np.clip(sin_as, -1, 1)
    return np.degrees(np.arcsin(sin_as))

# Function to calculate hourly irradiance
def calculate_irradiance(alpha, atm_reduction):
    cos_theta = np.sin(np.radians(alpha))  # For horizontal surface
    I = SOLAR_CONSTANT * cos_theta * atm_reduction / 1000  # Convert to kW/m^2
    return max(0, I)

# Function to calculate daily GHI
def calculate_daily_ghi(date, day_length, atm_reduction):
    day_of_year = (date - datetime(2024, 1, 1)).days + 1
    delta = declination_angle(day_of_year)

    # Estimate sunrise and sunset times
    sunrise = 12 - day_length / 2  # Hours from midnight
    sunset = 12 + day_length / 2
    hours = np.arange(sunrise, sunset, 0.1)  # 6-minute intervals for integration

    ghi_sum = 0
    for hour in hours:
        h = hour_angle(hour, LONGITUDE, day_of_year)
        alpha = solar_altitude_angle(LATITUDE, delta, h)
        if alpha > 0:  # Only count when sun is above horizon
            ghi_sum += calculate_irradiance(alpha, atm_reduction) * 0.1  # kWh/m^2 for 6 minutes

    return ghi_sum

# Function to query daily irradiance
def get_daily_irradiance():
    print("Enter date in 2024 (e.g., '21 March') or press Enter for default dates (21 March, 21 June, 21 September, 21 December):")
    date_input = input().strip()
```

```
    if date_input:
        try:
            date = datetime.strptime(f"{date_input} 2024", '%d %B %Y')
            dates = [date]
        except ValueError:
            print("Invalid date format. Use 'DD Month' (e.g., '21 March'). Using default dates.")
            dates = [datetime(2024, 3, 21), datetime(2024, 6, 21),
                     datetime(2024, 9, 21), datetime(2024, 12, 21)]
    else:
        dates = [datetime(2024, 3, 21), datetime(2024, 6, 21),
                 datetime(2024, 9, 21), datetime(2024, 12, 21)]

    for date in dates:
        if date > CURRENT_DATE:
            print(f"Sorry, data for {date.strftime('%d %B %Y')} is not available (future date).")
            continue

        date_str = date.strftime('%Y-%m-%d')
        day_of_year = (date - datetime(2024, 1, 1)).days + 1
        delta = declination_angle(day_of_year)

        # Noon parameters (for display)
        h_noon = hour_angle(12, LONGITUDE, day_of_year)
        alpha_noon = solar_altitude_angle(LATITUDE, delta, h_noon)
        azimuth_noon = solar_azimuth_angle(delta, h_noon, alpha_noon)

        # Get cloud cover and day length
        cloud_info = CLOUD_ASSUMPTIONS.get(date_str, {'status': 'unknown', 'oktas': 4, 'atm_reduction': 0.7})
        day_length = DAY_LENGTHS.get(date_str, 12.0)  # Default 12 hours if not specified

        # Calculate daily GHI
        daily_ghi = calculate_daily_ghi(date, day_length, cloud_info['atm_reduction'])

        print(f"\nResults for {date.strftime('%d %B %Y')} (Day of Year: {day_of_year})")
        print(f"Latitude (L): {LATITUDE:.2f}°")
        print(f"Declination Angle (δ): {delta:.2f}°")
        print(f"Noon Hour Angle (h): {h_noon:.2f}°")
        print(f"Noon Solar Altitude Angle (α): {alpha_noon:.2f}°")
        print(f"Noon Solar Azimuth Angle (a_s): {azimuth_noon:.2f}°")
        print(f"Day Length: {day_length:.2f} hours")
```

```
        print(f"Assumed Cloud Cover: {cloud_info['status']} ({cloud_info['oktas']} oktas)")
        print(f"Atmospheric Reduction Factor: {cloud_info['atm_reduction']:.2f}")
        print(f"Daily GHI: {daily_ghi:.2f} kWh/m^2/day")
        print("-" * 50)

# Run the query function
if __name__ == "__main__":
    get_daily_irradiance()
```

```
Enter date in 2024 (e.g., '21 March') or press Enter for default dates (21 March, 21 June, 21 September, 21 December):
20 september

Results for 20 September 2024 (Day of Year: 264)
Latitude (L): 28.61°
Declination Angle (δ): -0.20°
Noon Hour Angle (h): -5.27°
Noon Solar Altitude Angle (α): 60.75°
Noon Solar Azimuth Angle (a_s): -10.83°
Day Length: 12.00 hours
Assumed Cloud Cover: unknown (4 oktas)
Atmospheric Reduction Factor: 0.70
Daily GHI: 6.07 kWh/m^2/day
--------------------------------------------------
```

```
import numpy as np
import pandas as pd
from datetime import datetime, timedelta

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees
LONGITUDE = 77.23  # New Delhi longitude in degrees
SOLAR_CONSTANT = 1300  # W/m^2
STD_MERIDIAN = 82.5  # IST standard meridian (degrees)
DEG_PER_HOUR = 15  # Earth's rotation (degrees/hour)

# Month-wise atmospheric reduction factors (based on typical New Delhi weather)
MONTH_ATM_REDUCTION = {
    1: 0.9,  # Jan: Winter, clear
    2: 0.9,  # Feb: Winter, clear
    3: 0.85, # Mar: Summer, clear to partly cloudy
```

```
        4: 0.85, # Apr: Summer, clear to partly cloudy
        5: 0.85, # May: Summer, clear to partly cloudy
        6: 0.7,  # Jun: Monsoon, partly cloudy to cloudy
        7: 0.7,  # Jul: Monsoon, cloudy
        8: 0.7,  # Aug: Monsoon, cloudy
        9: 0.7,  # Sep: Monsoon, partly cloudy
        10: 0.8, # Oct: Post-monsoon, clear to partly cloudy
        11: 0.9, # Nov: Winter, clear
        12: 0.9  # Dec: Winter, clear
    }

    # Function to calculate declination angle
    def declination_angle(day_of_year):
        return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

    # Function to calculate hour angle
    def hour_angle(local_time, longitude, day):
        longitude_diff = STD_MERIDIAN - longitude
        time_correction = longitude_diff / DEG_PER_HOUR * 60  # in minutes
        solar_noon = 12.0 + time_correction / 60  # in hours
        hours_from_noon = local_time - solar_noon
        return DEG_PER_HOUR * hours_from_noon

    # Function to calculate solar altitude angle
    def solar_altitude_angle(latitude, declination, hour_angle):
        lat_rad = np.radians(latitude)
        dec_rad = np.radians(declination)
        ha_rad = np.radians(hour_angle)
        sin_alpha = (np.sin(lat_rad) * np.sin(dec_rad) +
                     np.cos(lat_rad) * np.cos(dec_rad) * np.cos(ha_rad))
        return np.degrees(np.arcsin(sin_alpha))

    # Function to calculate solar azimuth angle
    def solar_azimuth_angle(declination, hour_angle, alpha):
        dec_rad = np.radians(declination)
        ha_rad = np.radians(hour_angle)
        alpha_rad = np.radians(alpha)
        sin_as = np.cos(dec_rad) * np.sin(ha_rad) / np.cos(alpha_rad)
        sin_as = np.clip(sin_as, -1, 1)
        return np.degrees(np.arcsin(sin_as))
```

```
# Function to calculate hourly irradiance
def calculate_irradiance(alpha, atm_reduction):
    cos_theta = np.sin(np.radians(alpha))  # For horizontal surface
    I = SOLAR_CONSTANT * cos_theta * atm_reduction / 1000  # Convert to kW/m^2
    return max(0, I)

# Function to estimate day length (interpolate between 10.32 and 13.96 hours)
def estimate_day_length(day_of_year):
    # Linear interpolation: 21 Jun (day 172) = 13.96 hours, 21 Dec (day 355) = 10.32 hours
    if day_of_year <= 172:
        fraction = day_of_year / 172
        return 10.32 + (13.96 - 10.32) * fraction
    else:
        fraction = (day_of_year - 172) / (355 - 172)
        return 13.96 - (13.96 - 10.32) * fraction

# Function to calculate daily GHI
def calculate_daily_ghi(date, day_length, atm_reduction):
    day_of_year = (date - datetime(2024, 1, 1)).days + 1
    delta = declination_angle(day_of_year)

    # Estimate sunrise and sunset times
    sunrise = 12 - day_length / 2  # Hours from midnight
    sunset = 12 + day_length / 2
    hours = np.arange(sunrise, sunset, 0.1)  # 6-minute intervals for integration

    ghi_sum = 0
    for hour in hours:
        h = hour_angle(hour, LONGITUDE, day_of_year)
        alpha = solar_altitude_angle(LATITUDE, delta, h)
        if alpha > 0:  # Only count when sun is above horizon
            ghi_sum += calculate_irradiance(alpha, atm_reduction) * 0.1  # kWh/m^2 for 6 minutes

    return ghi_sum

# Generate data for 2024
start_date = datetime(2024, 1, 1)
end_date = datetime(2024, 12, 31)
data = []
```

```
for day in range((end_date - start_date).days + 1):
    date = start_date + timedelta(days=day)
    day_of_year = day + 1
    month = date.month
    delta = declination_angle(day_of_year)

    # Noon parameters
    h_noon = hour_angle(12, LONGITUDE, day_of_year)
    alpha_noon = solar_altitude_angle(LATITUDE, delta, h_noon)
    azimuth_noon = solar_azimuth_angle(delta, h_noon, alpha_noon)

    # Day length and atmospheric reduction
    day_length = estimate_day_length(day_of_year)
    atm_reduction = MONTH_ATM_REDUCTION[month]

    # Calculate daily GHI
    daily_ghi = calculate_daily_ghi(date, day_length, atm_reduction)

    data.append({
        'Date': date.strftime('%Y-%m-%d'),
        'Declination Angle (δ, deg)': round(delta, 2),
        'Noon Solar Altitude Angle (α, deg)': round(alpha_noon, 2),
        'Noon Solar Azimuth Angle (γ, deg)': round(azimuth_noon, 2),
        'Latitude (L, deg)': LATITUDE,
        'Daily GHI (kWh/m^2/day)': round(daily_ghi, 2)
    })

# Create DataFrame and save to CSV
df = pd.DataFrame(data)
df.to_csv('solar_irradiance_new_delhi_2024.csv', index=False)
print("CSV file 'solar_irradiance_new_delhi_2024.csv' generated successfully!")
```

```
CSV file 'solar_irradiance_new_delhi_2024.csv' generated successfully!
```

```
import pandas as pd
import numpy as np
from datetime import datetime
from google.colab import files
```

```
# Upload the input CSV file to Colab
print("Please upload the input CSV file: solar_irradiance_2024_new_delhi.csv")
uploaded = files.upload()

# Read the input CSV
input_file = 'solar_irradiance_2024_new_delhi.csv'
df = pd.read_csv(input_file)

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees

# Cloud cover assumptions (monthly)
cloud_cover = {
    1: 0.9, 2: 0.9, 3: 0.85, 4: 0.85, 5: 0.85,
    6: 0.7, 7: 0.7, 8: 0.7, 9: 0.7, 10: 0.9, 11: 0.9, 12: 0.9
}

# Function to calculate declination angle (δ) in degrees
def calc_declination_angle(day_of_year):
    return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

# Function to calculate noon solar altitude angle (α) in degrees
def calc_noon_solar_altitude(latitude, declination):
    return 90 - abs(latitude - declination)

# Function to calculate noon solar azimuth angle (γ) in degrees
def calc_noon_solar_azimuth(latitude, declination, day_of_year):
    # Simplified: at noon, azimuth is ~0° (south), with slight offset
    return -5.0 * np.sin(np.radians(360 * day_of_year / 365))

# Function to calculate daily GHI from hourly irradiance
def calc_daily_ghi(group, month):
    hourly_sum = group['Irradiance (W/m^2)'].sum()
    ghi = (hourly_sum * 1 / 1000) * cloud_cover[month]
    return round(ghi, 2)

# Function to calculate daily mean irradiance
def calc_daily_mean_irradiance(group):
    mean_irradiance = group['Irradiance (W/m^2)'].mean()
```

```
        return round(mean_irradiance, 2)

    # Process the data
    # Convert Date to datetime and extract day of year and month
    df['Date'] = pd.to_datetime(df['Date'])
    df['Day_of_Year'] = df['Date'].dt.dayofyear
    df['Month'] = df['Date'].dt.month

    # Calculate angles
    df['Declination Angle (δ, deg)'] = df['Day_of_Year'].apply(calc_declination_angle).round(2)
    df['Noon Solar Altitude Angle (α, deg)'] = df.apply(
        lambda row: calc_noon_solar_altitude(LATITUDE, row['Declination Angle (δ, deg)']), axis=1
    ).round(2)
    df['Noon Solar Azimuth Angle (γ, deg)'] = df['Day_of_Year'].apply(
        lambda x: calc_noon_solar_azimuth(LATITUDE, calc_declination_angle(x), x)
    ).round(2)
    df['Latitude (L, deg)'] = LATITUDE

    # Calculate daily GHI and daily mean irradiance for each date
    daily_ghi = df.groupby(['Date', 'Month']).apply(
        lambda x: calc_daily_ghi(x, x['Month'].iloc[0])
    ).reset_index(name='Daily GHI (kWh/m^2/day)')
    daily_mean_irradiance = df.groupby('Date').apply(
        calc_daily_mean_irradiance
    ).reset_index(name='Daily I Mean (W/m^2)')

    # Merge daily GHI and mean irradiance back to the main dataframe
    df = df.merge(
        daily_ghi[['Date', 'Daily GHI (kWh/m^2/day)']],
        on='Date',
        how='left'
    ).merge(
        daily_mean_irradiance[['Date', 'Daily I Mean (W/m^2)']],
        on='Date',
        how='left'
    )

    # Drop unnecessary columns and reorder
    df = df[['Date', 'Hour', 'Irradiance (W/m^2)', 'Declination Angle (δ, deg)',
             'Noon Solar Altitude Angle (α, deg)', 'Noon Solar Azimuth Angle (γ, deg)',
```

```
        'Latitude (L, deg)', 'Daily GHI (kWh/m^2/day)', 'Daily I Mean (W/m^2)']]

# Save the output CSV
output_file = 'solar_irradiance_enhanced_2024_new_delhi.csv'
df.to_csv(output_file, index=False)
print(f"Enhanced CSV file '{output_file}' generated successfully!")

# Download the file
files.download(output_file)
```

```
Please upload the input CSV file: solar_irradiance_2024_new_delhi.csv
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-13-3252550848.py in <cell line: 0>()
      6 # Upload the input CSV file to Colab
      7 print("Please upload the input CSV file: solar_irradiance_2024_new_delhi.csv")
----> 8 uploaded = files.upload()
      9 
     10 # Read the input CSV
```

3 frames

```
/usr/local/lib/python3.11/dist-packages/google/colab/_message.py in read_reply_from_input(message_id, timeout_sec)
     94     reply = _read_next_input_message()
     95     if reply == _NOT_READY or not isinstance(reply, dict):
---> 96       time.sleep(0.025)
     97       continue
     98     if (

KeyboardInterrupt: 
```

```
import pandas as pd
import numpy as np
from google.colab import files

# Upload the input CSV file to Colab
print("Please upload the input CSV file: solar_irradiance_2024_new_delhi.csv")
uploaded = files.upload()
```

```
# Read the input CSV
input_file = 'solar_irradiance_2024_new_delhi.csv'
df = pd.read_csv(input_file)

# Constants
LATITUDE = 28.61  # New Delhi latitude in degrees

# Cloud cover assumptions (monthly)
cloud_cover = {
    1: 0.9, 2: 0.9, 3: 0.85, 4: 0.85, 5: 0.85,
    6: 0.7, 7: 0.7, 8: 0.7, 9: 0.7, 10: 0.9, 11: 0.9, 12: 0.9
}

# Function to calculate declination angle (δ) in degrees
def calc_declination_angle(day_of_year):
    return 23.45 * np.sin(np.radians(360 * (284 + day_of_year) / 365))

# Function to calculate noon solar altitude angle (α) in degrees
def calc_noon_solar_altitude(latitude, declination):
    return 90 - abs(latitude - declination)

# Function to calculate noon solar azimuth angle (γ) in degrees
def calc_noon_solar_azimuth(latitude, declination, day_of_year):
    # Simplified: at noon, azimuth is ~0° (south), with slight offset
    return -5.0 * np.sin(np.radians(360 * day_of_year / 365))

# Function to calculate daily GHI from hourly irradiance
def calc_daily_ghi(group, month):
    hourly_sum = group['Irradiance (W/m^2)'].sum()
    ghi = (hourly_sum * 1 / 1000) * cloud_cover[month]
    return round(ghi, 2)

# Function to calculate daily mean irradiance
def calc_daily_mean_irradiance(group):
    mean_irradiance = group['Irradiance (W/m^2)'].mean()
    return round(mean_irradiance, 2)

# Process the data
# Extract month from Date for cloud cover (temporary, will drop Date later)
df['Date'] = pd.to_datetime(df['Date'])
```

```
    df['Month'] = df['Date'].dt.month

    # Use Day column as day_of_year
    df['Day_of_Year'] = df['Day']

    # Calculate angles
    df['Declination Angle (δ, deg)'] = df['Day_of_Year'].apply(calc_declination_angle).round(2)
    df['Noon Solar Altitude Angle (α, deg)'] = df.apply(
        lambda row: calc_noon_solar_altitude(LATITUDE, row['Declination Angle (δ, deg)']), axis=1
    ).round(2)
    df['Noon Solar Azimuth Angle (γ, deg)'] = df['Day_of_Year'].apply(
        lambda x: calc_noon_solar_azimuth(LATITUDE, calc_declination_angle(x), x)
    ).round(2)
    df['Latitude (L, deg)'] = LATITUDE

    # Calculate daily GHI and daily mean irradiance for each day
    daily_ghi = df.groupby(['Day', 'Month']).apply(
        lambda x: calc_daily_ghi(x, x['Month'].iloc[0])
    ).reset_index(name='Daily GHI (kWh/m^2/day)')
    daily_mean_irradiance = df.groupby('Day').apply(
        calc_daily_mean_irradiance
    ).reset_index(name='Daily I Mean (W/m^2)')

    # Merge daily GHI and mean irradiance back to the main dataframe
    df = df.merge(
        daily_ghi[['Day', 'Daily GHI (kWh/m^2/day)']],
        on='Day',
        how='left'
    ).merge(
        daily_mean_irradiance[['Day', 'Daily I Mean (W/m^2)']],
        on='Day',
        how='left'
    )

    # Drop unnecessary columns (Date, Day_of_Year, Month) and reorder
    df = df[['Day', 'Hour', 'Irradiance (W/m^2)', 'Declination Angle (δ, deg)',
             'Noon Solar Altitude Angle (α, deg)', 'Noon Solar Azimuth Angle (γ, deg)',
             'Latitude (L, deg)', 'Daily GHI (kWh/m^2/day)', 'Daily I Mean (W/m^2)']]

    # Save the output CSV
```

```
output_file = 'solar_irradiance_enhanced_2024_new_delhi.csv'
df.to_csv(output_file, index=False)
print(f"Enhanced CSV file '{output_file}' generated successfully!")

# Download the file
files.download(output_file)
```

```
Please upload the input CSV file: solar_irradiance_2024_new_delhi.csv
```
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Enhanced CSV file 'solar_irradiance_enhanced_2024_new_delhi.csv' generated successfully!
/tmp/ipython-input-9-1914268466.py:65: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is dep
  daily_ghi = df.groupby(['Day', 'Month']).apply(
/tmp/ipython-input-9-1914268466.py:68: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is dep
  daily_mean_irradiance = df.groupby('Day').apply(
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime, timedelta
import matplotlib.dates as mdates

# Data from your file
data = """485.27 26.8
486.37 23.7
487.57 25.4
488.86 28.5
490.24 28.5
491.7 28.3
493.26 25.7
494.9 26.5
496.63 27.4
498.45 27.5
500.35 29.35
502.34 30.35
504.41 31.2
506.56 30.35
508.79 26.9
511.1 30
```

```
513.49 30.5
515.96 31
518.5 31.75
521.12 31.35
523.81 30.6
526.58 29.9
529.41 30.1
532.31 27.9
535.28 28.75
538.32 30.5
541.42 31.6
544.58 30.75
547.81 30.5
551.09 30.25
554.43 17.5
557.83 17.3
561.28 17.1
564.78 16.9
568.33 16.7
571.93 16.5
575.58 16.3
579.28 16.1
583.01 15.9
586.79 15.7
590.6 15.5
594.45 15.3
598.34 15.1
602.26 14.9
606.21 14.7
610.19 14.5
614.19 14.3
618.22 14.1
622.28 13.9
626.35 13.7
630.44 13.5
634.55 13.3
638.67 13.1
642.81 16.9
646.95 16.55
651.11 16.45
```

```
655.27 18.45
659.43 18.65
663.6 18.05
667.76 18.45
671.93 19.2
676.09 22.3
680.24 19.45
684.39 17.85
688.53 16.95
692.65 16.2
696.76 16.2
700.86 18
704.94 18.7
709 18.45
713.04 20.85
717.06 22.8
721.05 21.95
725.02 23
728.96 22.15
732.87 19.4
736.76 21.1
740.61 21.5
744.42 21.4
748.2 22.5
751.95 25
755.66 24.05
759.33 24.4
762.96 25.5
766.55 22.8
770.1 25.45
773.6 27
777.06 29.1
780.48 28.75
783.85 29.8
787.17 28.25
790.44 26.8
793.67 23.7
796.84 25.4
799.97 28.5
803.05 28.5
```

```
806.07 28.3
809.05 25.7
811.97 26.5
814.83 27.4
817.65 27.5
820.41 29.35
823.12 30.35
825.77 31.2
828.38 30.35
830.93 26.9
833.42 30
835.86 31.75
838.24 31.35
840.58 30.6
842.85 29.9
845.08 30.1
847.25 27.9
849.37 28.75
851.43 30.5
853.44 31.6
855.4 30.75
857.31 30.5
859.17 30.25
860.97 28.75
862.73 26.5
864.43 26.9
866.09 30.65
867.7 31.5
869.25 32.75
870.76 32.45
872.23 34.75
873.64 32.1
875.01 33.25
876.34 31.95
877.62 32.25
878.86 33.65
880.05 32.05
881.2 31.75
882.31 33
883.39 33.95
```

```
884.42 35.35
885.41 35.95
886.36 36.8
887.27 37.4
888.15 36.65
888.99 37
889.8 35.9
890.56 36.15
891.3 35.2
892 37.3
892.67 36.25
893.31 37.6
893.92 38.6
894.49 37.3
895.04 38
895.56 37.3
896.04 36.5
896.5 37.55
896.93 37.4
897.33 34.3
897.71 34.6
898.06 36.5
898.39 34.2
898.68 35.55
898.96 35.7
899.21 36.15
899.43 37.05
899.63 39.05
899.81 38.2
899.97 38.9
900.1 38.95
900.2 39.5
900.29 39.4
900.35 39
900.39 38.5
900.41 38
900.4 37.5
900.37 37
900.32 36.5
900.25 36
```

```
900.15 35.5
900.04 35
899.89 34.5
899.73 34
899.54 33.5
899.32 33
899.09 32.5
898.83 32
898.54 31.5
898.23 31
897.89 30.5
897.53 30
897.14 29.5
896.72 29
896.27 28.5
895.8 28
895.3 27.5
894.77 27
894.21 27.8
893.62 27.8
893 27.8
892.34 27.8
891.66 27.8
890.94 27.8
890.18 27.8
889.4 27.8
888.57 27.8
887.71 27.8
886.82 27.8
885.89 27.8
884.91 27.8
883.9 27.8
882.86 27.8
881.77 27.8
880.63 27.8
879.46 27.8
878.24 27.8
876.99 27.8
875.68 27.8
874.33 27.8
```

```
872.94 27.8
871.5 27.8
870.01 27.8
868.48 27.8
866.9 27.8
865.26 27.8
863.59 27.8
861.86 27.8
860.07 27.8
858.25 27.8
856.36 27.8
854.43 27.8
852.44 27.8
850.4 27.8
848.31 27.8
846.17 27.8
843.97 27.8
841.72 27.8
839.42 27.8
837.06 27.8
834.64 27.8
832.18 27.8
829.66 27.8
827.08 27.8
824.45 27.8
821.77 27.8
819.04 27.8
816.25 27.8
813.41 27.8
810.51 27.8
807.57 27.8
804.57 27.8
801.52 27.8
798.41 27.8
795.26 27.8
792.06 27.8
788.81 27.8
785.51 27.8
782.17 27.8
778.78 27.8
```

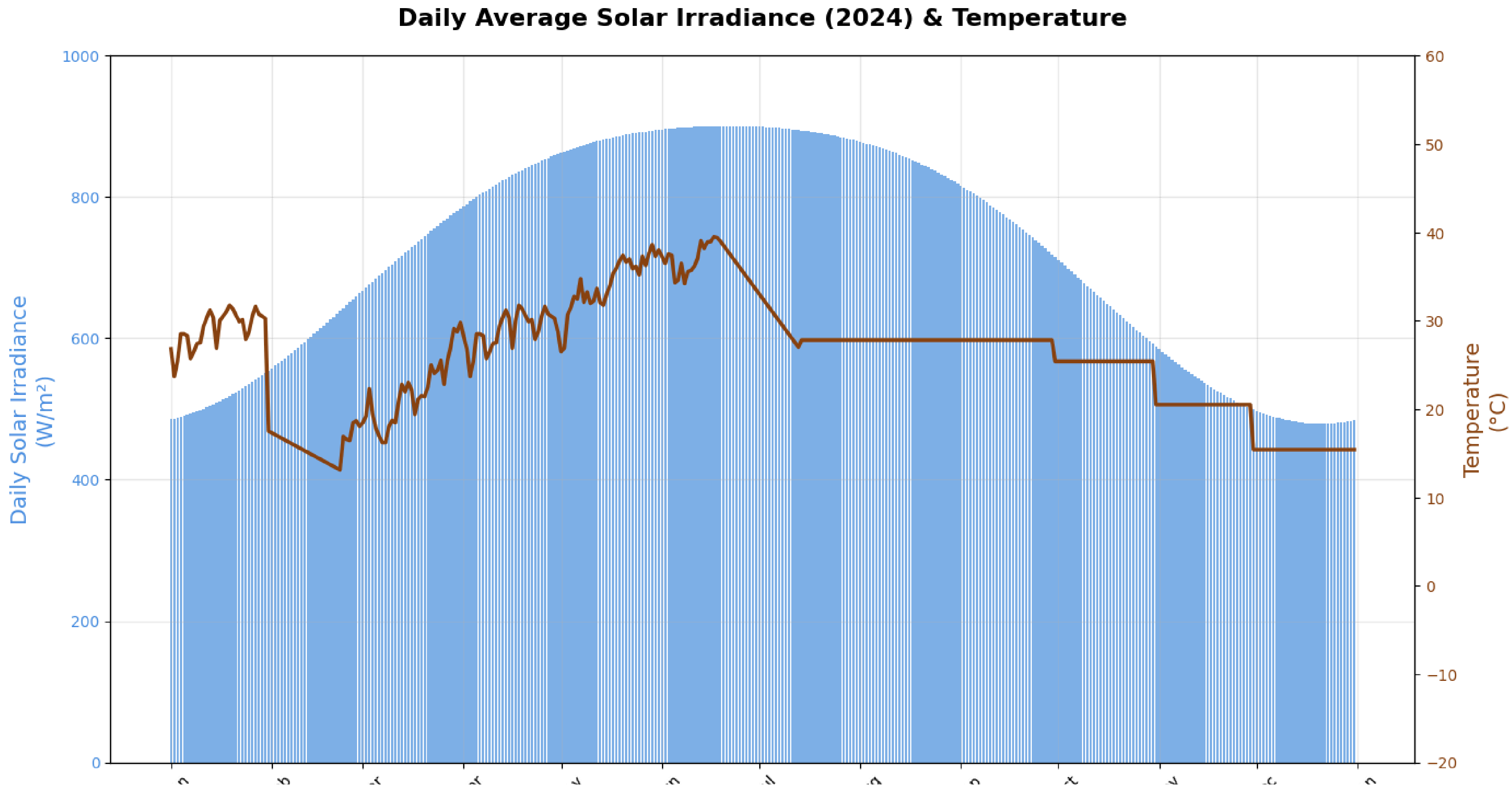
```
775.34 27.8
771.86 27.8
768.33 27.8
764.76 27.8
761.15 27.8
757.5 27.8
753.81 27.8
750.08 27.8
746.32 27.8
742.52 27.8
738.69 27.8
734.82 27.8
730.92 27.8
726.99 27.8
723.04 27.8
719.06 27.8
715.05 25.4
711.02 25.4
706.97 25.4
702.9 25.4
698.81 25.4
694.71 25.4
690.59 25.4
686.46 25.4
682.31 25.4
678.17 25.4
674.01 25.4
669.85 25.4
665.68 25.4
661.51 25.4
657.35 25.4
653.19 25.4
649.03 25.4
644.88 25.4
640.74 25.4
636.61 25.4
632.49 25.4
628.39 25.4
624.31 25.4
620.25 25.4
```

```
616.21 25.4
612.19 25.4
608.2 25.4
604.23 25.4
600.3 25.4
596.39 25.4
592.52 25.4
588.69 20.5
584.89 20.5
581.14 20.5
577.42 20.5
573.75 20.5
570.13 20.5
566.55 20.5
563.02 20.5
559.55 20.5
556.12 20.5
552.75 20.5
549.44 20.5
546.19 20.5
543 20.5
539.86 20.5
536.79 20.5
533.79 20.5
530.85 20.5
527.98 20.5
525.18 20.5
522.46 20.5
519.8 20.5
517.22 20.5
514.72 20.5
512.29 20.5
509.94 20.5
507.66 20.5
505.47 20.5
503.36 20.5
501.33 20.5
499.39 15.4
497.53 15.4
495.76 15.4
```

```
494.07 15.4
492.47 15.4
490.96 15.4
489.54 15.4
488.21 15.4
486.96 15.4
485.81 15.4
484.75 15.4
483.79 15.4
482.91 15.4
482.13 15.4
481.45 15.4
480.85 15.4
480.35 15.4
479.95 15.4
479.64 15.4
479.43 15.4
479.31 15.4
479.29 15.4
479.36 15.4
479.52 15.4
479.78 15.4
480.14 15.4
480.59 15.4
481.14 15.4
481.78 15.4
482.51 15.4
483.34 15.4
484.26 15.4"""

# Parse the data
lines = data.strip().split('\n')
solar_insolation = []
temperature = []

for line in lines:
    parts = line.split()
    solar_insolation.append(float(parts[0]))
    temperature.append(float(parts[1]))
```

```
    # Create day numbers (assuming this is a full year of data)
    days = list(range(1, len(solar_insolation) + 1))

    # Create dates for x-axis (assuming starting from January 1st)
    start_date = datetime(2014, 1, 1)
    dates = [start_date + timedelta(days=i) for i in range(len(solar_insolation))]

    # Keep solar insolation in W/m² (no conversion needed)
    solar_insolation_wm2 = solar_insolation

    # Create the plot
    fig, ax1 = plt.subplots(figsize=(14, 8))

    # Plot solar insolation as individual bars (daily basis) - more separated
    bars = ax1.bar(dates, solar_insolation_wm2, color='#4A90E2', alpha=0.7, width=0.8, label='Solar Irradiance')

    # Set up the first y-axis (left side) for solar insolation
    ax1.set_xlabel('Day', fontsize=14, color='gray')
    ax1.set_ylabel('Daily Solar Irradiance\n(W/m²)', fontsize=14, color='#4A90E2')
    ax1.tick_params(axis='y', labelcolor='#4A90E2')
    ax1.set_ylim(0, 1000)
    ax1.grid(True, alpha=0.3)

    # Create second y-axis (right side) for temperature
    ax2 = ax1.twinx()
    ax2.plot(dates, temperature, color='#8B4513', linewidth=2.5, label='Temperature (Min-Max)')
    ax2.set_ylabel('Temperature\n(°C)', fontsize=14, color='#8B4513')
    ax2.tick_params(axis='y', labelcolor='#8B4513')
    ax2.set_ylim(-20, 60)

    # Format x-axis
    ax1.xaxis.set_major_formatter(mdates.DateFormatter('%d %b'))
    ax1.xaxis.set_major_locator(mdates.MonthLocator())
    plt.setp(ax1.xaxis.get_majorticklabels(), rotation=45)

    # Add title (without source)
    plt.title('Daily Average Solar Irradiance (2024) & Temperature', fontsize=16, pad=20, fontweight='bold')

    # Add legend
    lines1, labels1 = ax1.get_legend_handles_labels()
```

```
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines1 + lines2, ['Solar Irradiance', 'Temperature (Min-Max)'],
           loc='lower center', bbox_to_anchor=(0.5, -0.15), ncol=2, fontsize=12)

# Adjust layout
plt.tight_layout()
plt.subplots_adjust(bottom=0.12)

# Show the plot
plt.show()

# Print some statistics
print(f"Total data points: {len(solar_insolation)}")
print(f"Solar Insolation range: {min(solar_insolation):.2f} - {max(solar_insolation):.2f} W/m²")
print(f"Temperature range: {min(temperature):.2f} - {max(temperature):.2f} °C")
print(f"Average Solar Insolation: {np.mean(solar_insolation):.2f} W/m²")
print(f"Average Temperature: {np.mean(temperature):.2f} °C")
```


Daily Average Solar Irradiance (2024) & Temperature
Daily Solar Irradiance (W/m²)
Temperature (°C)
0
200
400
600
800
1000
−20
−10
0
10
20
30
40
50
60