```
import pandas as pd

# Load the datasets
weather_data = pd.read_csv('/content/weather_data.csv')
solar_panel_data = pd.read_csv('/content/solar_panel_data.csv')
maintenance_logs = pd.read_csv('/content/maintenance_logs.csv')
geographic_data = pd.read_csv('/content/geographic_data.csv')
future_weather_data = pd.read_csv('/content/future_weather_data.csv')
```

```
weather_data.head()
```

	timestamp	temperature	sunlight_hours	humidity	wind_speed
0	2023-07-01 00:00:00	25.3	0.0	60	5.2
1	2023-07-01 01:00:00	24.8	0.0	61	5.0
2	2023-07-01 02:00:00	24.5	0.0	62	4.8
3	2023-07-01 03:00:00	24.2	0.0	63	4.7
4	2023-07-01 04:00:00	23.8	0.5	64	4.5

Next steps: Generate code with weather_data | View recommended plots | New interactive sheet

```
solar_panel_data.head()
```

	timestamp	energy_output	efficiency	degradation_rate	location_id
0	2023-07-01 00:00:00	0.0	98.5	0.1	1
1	2023-07-01 01:00:00	0.0	98.5	0.1	1
2	2023-07-01 02:00:00	0.0	98.5	0.1	1
	2023-07-01				

Next steps: Generate code with solar_panel_data | View recommended plots | New interactive sheet

```
maintenance_logs.head()
```

	maintenance_id	timestamp	maintenance_type	downtime_hours	l
0	1	2023-06-25 08:00:00	Cleaning	1	

Next steps: Generate code with maintenance_logs | View recommended plots | New interactive sheet

```
geographic_data
```

	location_id	latitude	longitude	altitude
0	1	35.6895	139.6917	44

```
future_weather_data.head()
```

	timestamp	temperature	sunlight_hours	humidity	wind_speed	
0	2023-08-01 00:00:00	26.0	0.0	65	5.0	
1	2023-08-01	25.5	0.0	66	4.8	

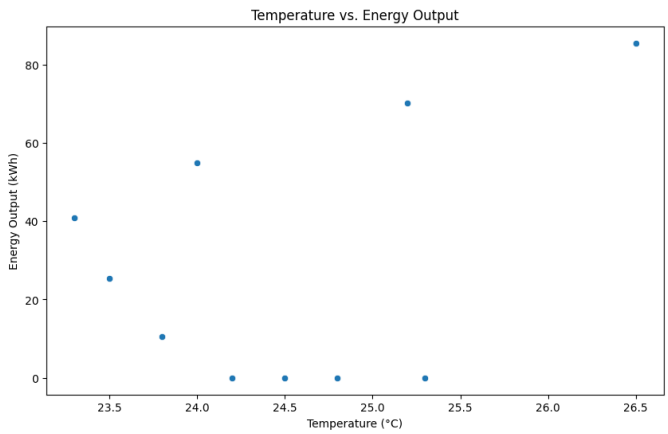
Next steps: Generate code with future_weather_data | View recommended plots | New interactive sheet

```
# Convert timestamp columns to datetime
weather_data['timestamp'] = pd.to_datetime(weather_data['timestamp'])
solar_panel_data['timestamp'] = pd.to_datetime(solar_panel_data['timestamp'])
maintenance_logs['timestamp'] = pd.to_datetime(maintenance_logs['timestamp'])
future_weather_data['timestamp'] = pd.to_datetime(future_weather_data['timestamp'])


# Merge dataframes on common columns
data = weather_data.merge(solar_panel_data, on='timestamp')
data = data.merge(geographic_data, on='location_id')


data.head()
```

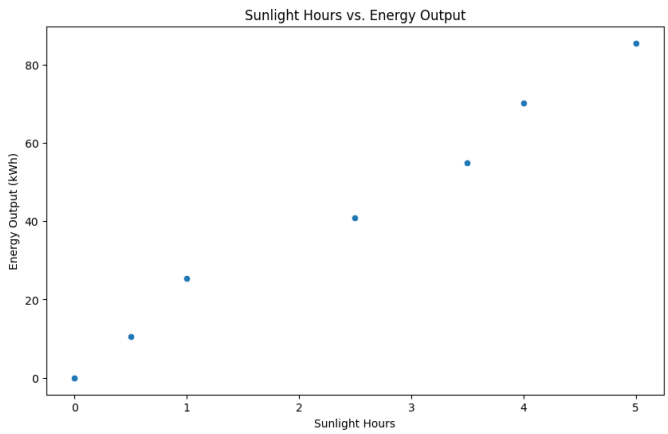
	timestamp	temperature	sunlight_hours	humidity	wind_speed
0	2023-07-01 00:00:00	25.3	0.0	60	5.2
1	2023-07-01	24.8	0.0	61	5.0

Next steps: Generate code with `data` View recommended plots New interactive sheet

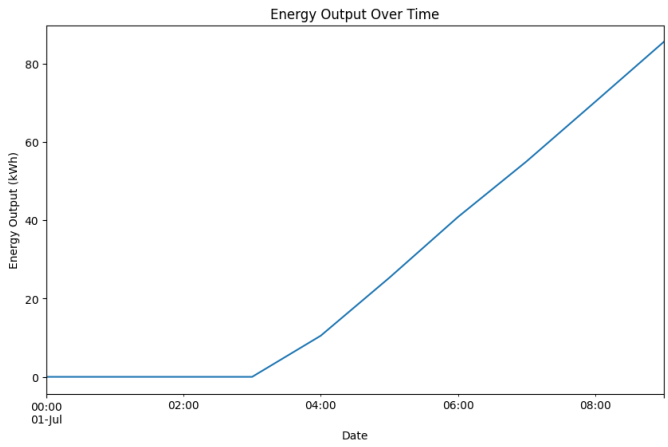
```
import seaborn as sns
import matplotlib.pyplot as plt


# Plot temperature vs. energy output
plt.figure(figsize=(10, 6))
sns.scatterplot(x='temperature', y='energy_output', data=data)
plt.title('Temperature vs. Energy Output')
plt.xlabel('Temperature (°C)')
plt.ylabel('Energy Output (kWh)')
plt.show()
```



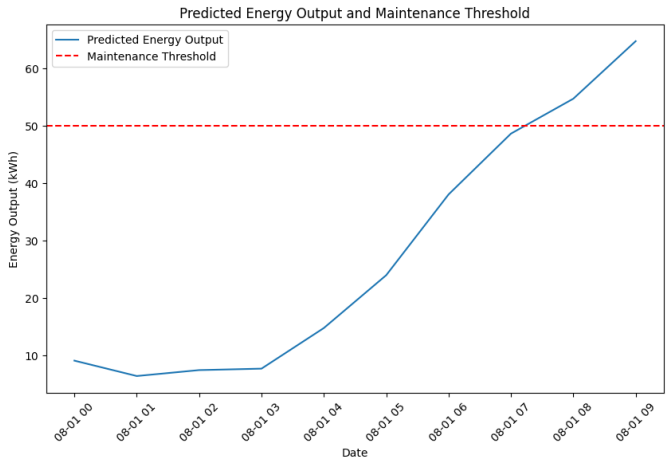

```
# Plot sunlight hours vs. energy output
plt.figure(figsize=(10, 6))
sns.scatterplot(x='sunlight_hours', y='energy_output', data=data)
plt.title('Sunlight Hours vs. Energy Output')
plt.xlabel('Sunlight Hours')
plt.ylabel('Energy Output (kWh)')
plt.show()
```




```
plt.figure(figsize=(10, 6))
data.set_index('timestamp')['energy_output'].plot()
plt.title('Energy Output Over Time')
plt.xlabel('Date')
plt.ylabel('Energy Output (kWh)')
plt.show()
```




```
plt.figure(figsize=(10, 6))
plt.plot(future_weather_data['timestamp'], future_weather_data['predicted_energy_output'], label='Predicted Energy Output')
plt.axhline(y=maintenance_threshold, color='r', linestyle='--', label='Maintenance Threshold')
plt.xlabel('Date')
plt.ylabel('Energy Output (kWh)')
plt.title('Predicted Energy Output and Maintenance Threshold')
plt.legend()
plt.xticks(rotation=45)
plt.show()
```




```
# Prepare the dataset for modeling
X = data[['temperature', 'sunlight_hours', 'humidity', 'wind_speed', 'latitude', 'longitude', 'altitude']]
y = data['energy_output']
```

```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Train a Random Forest model
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
from sklearn.metrics import mean_absolute_error
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')
```

```
Mean Absolute Error: 7.930499999999998
```

```
# Add missing columns to future weather data
future_weather_data['latitude'] = 35.6895  # Example latitude
future_weather_data['longitude'] = 139.6917  # Example longitude
future_weather_data['altitude'] = 44  # Example altitude
```

```
# Prepare future weather data for prediction
X_future = future_weather_data[['temperature', 'sunlight_hours', 'humidity', 'wind_speed', 'latitude', 'longitude', 'altitude']]
future_predictions = model.predict(X_future)
```

```
# Threshold for maintenance
maintenance_threshold = 50  # Example threshold in kWh
future_weather_data['predicted_energy_output'] = future_predictions
maintenance_schedule = future_weather_data[future_predictions < maintenance_threshold]
```

```
# Save maintenance schedule to a CSV file
maintenance_schedule.to_csv('maintenance_schedule.csv', index=False)
```

```
'''This analysis demonstrates the potential for using data science techniques to improve the performance and maintenance of solar PV sys
```

```
'This analysis demonstrates the potential for using data science techniques to improve the performance and maintenance of solar PV
systems'
```