Source code for predicting solar power output using linear regression

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# 1. Load the data (replace 'solar_data.csv' with your actual file)
try:
    data = pd.read_csv('solar_data.csv')  # Assuming CSV format.  Adjust if needed.
except FileNotFoundError:
    print("Error: 'solar_data.csv' not found. Please provide the correct file.")
    exit()  # Exit the script if the file isn't found

# 2. Data Exploration and Preprocessing (Crucial!)
print(data.head())  # Display first few rows
print(data.info())  # Get data types and check for missing values
print(data.describe())  # Get summary statistics

# Handle missing values (if any).  Common methods:
#   - Drop rows with missing values: data.dropna()
#   - Impute missing values (e.g., with mean, median): data.fillna(data.mean())
#   - More advanced imputation techniques (KNN imputation, etc.) if needed.
# Example (dropping rows with NaNs):
data = data.dropna() # Or another method as appropriate

# Feature Engineering (if needed). Examples:
#   - Create new features from existing ones (e.g., time of day, day of the week).
#   - Transform features (e.g., log transformation for skewed data).

# 3. Define features (X) and target (y)
#   - 'SolarPower' is assumed to be your target variable.
#   - Adjust the feature names ('Temperature', 'Irradiance', etc.) based on your data.
X = data[['Temperature', 'Irradiance']]  # Features (independent variables)
y = data['SolarPower']  # Target (dependent variable)

# 4. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  #
80% train, 20% test

# 5. Train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# 6. Make predictions on the test set
y_pred = model.predict(X_test)

# 7. Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# 8. Visualize the results (optional but highly recommended)
plt.scatter(X_test['Irradiance'], y_test, label='Actual') # Example: Plot against one feature
plt.scatter(X_test['Irradiance'], y_pred, label='Predicted', color='red')
plt.xlabel('Irradiance')
plt.ylabel('Solar Power')
plt.title('Actual vs. Predicted Solar Power')
plt.legend()
plt.show()

# If you have multiple features, you might want to create separate plots
# for each feature vs. the predicted/actual values.

# 9. Make predictions on new data (after training and evaluation)
# new_data = pd.DataFrame({'Temperature': [25, 30], 'Irradiance': [800, 1000]}) # Example
# predictions = model.predict(new_data)
# print(f"Predictions for new data: {predictions}")


# 10. Save the trained model (optional)
# import joblib
# joblib.dump(model, 'solar_model.pkl')  # Save the model
# loaded_model = joblib.load('solar_model.pkl') # Load the model later
```