Data Analytics in Power System

EE4750

Hybrid Deep LearningFramework for SolarPower Forecasting.

Index No.	Name
210426A	V.Nilesh
210666H	S.Vaishnavi

Department of Electrical Engineering

University of Moratuwa

Sri Lanka

25th October 2025

Table of Contents

1. Introduction

1.1. Background of Solar Power Forecasting

Solar power forecasting is the projection of solar plants' future power output based on historical generation data, meteorological parameters, and environmental conditions. Accurate solar power forecasting can minimize costs and improve energy management by providing grid reliable operation, less dependency on reserve generator capacity, and effective resource allocation.

1.2. Importance of Renewable energy Forecasting

Accurate forecasting is essential to infusing renewable energy into the grid. In contrast to traditional power plants, solar plants do not produce electricity whenever it is required; their output is reliant upon environmental factors. For this reason, grid operators need trustworthy forecasts not only to evaluate demand–supply balance, but also to dispatch backup power and manage their storage systems. Solar forecasting is beneficial because it lessens dependence on fossil fuel and promotes grid stability, as well as improving battery storage and microgrid performance efficiently.

1.3. Importance of Renewable energy Forecasting

The central challenge addressed in this project is the unpredictable nature of solar power generation. Traditional statistical models struggle to capture the non-linear interactions between meteorological variables and solar output. While physics-based models require detailed site-specific parameters, they often fail to generalize across varying conditions.

The objective of this study is to design and implement a hybrid deep learning framework combining Convolutional Neural Networks (CNN) and XGBoost to forecast short-term solar power generation. The specific goals are:

- To preprocess and engineer features from plant-level datasets, including environmental and generation variables.
- To use CNN for automatic feature extraction from time-series data, capturing temporal and spatial dependencies.
- To apply XGBoost as the regression model, enhancing predictive accuracy and robustness.
- To evaluate model performance using multiple error metrics (RMSE, MAE, R^2) and compare it against baseline approaches.
- To demonstrate the practical relevance of the model for solar plant operators through a deployment-ready inference pipeline.

2. Literature Review

Solar power forecasting has grown into an important area of research to enable more efficient utilization of renewable energy in power systems. Conventional approaches to predicting photovoltaic (PV) power, such as physical models and statistical regressions, are often based heavily on domain-specific equations and various meteorological assumptions. Although conventional power forecasting methods provide good interpretability, they cannot effectively account for the nonlinear, stochastic effects of solar radiation, weather conditions, and temperature. For this reason, researchers are exploring machine learning (ML) and hybrid deep learning approaches.

Recent research has also investigated hybrid models which combine different approaches for diversity, accuracy, and robustness. For example, Zhang et al. (2024) proposed a hybrid CNN– XGBoost framework for forecasting PV power. In their model, Convolutional Neural Networks (CNNs) were employed to automatically extract spatial and temporal features from meteorological data and a regression layer of the model, XGBoost, mapped the extracted features into accurate power output predictions. Their model filled two important gaps in existing work:

1. The existing regression methods could not effectively learn feature interactions, and
2. CNNs alone do not generalize well in regression tasks.

The hybrid framework exhibited robust effectiveness in capturing short-term variability and long-term dependencies across PV generation datasets. The CNN component converged nonlinearities of features, while the XGBoost layer offered improvement to generalizability and reduced overfitting. The authors also stressed the hybrid structure was effective using the time-series datasets (where historical context is important) and the tabular datasets (where features exist independently but with a nonlinear structure).

Yet, there exist research gaps as well. CNNs were mainly tested/used in shallow configurations for tabular datasets leaving further investigation with deeper structures including pooling and dense layers. Additionally, temporal dependencies were not always included because most models analysed from a row-by-row standpoint instead of using some sliding-window sequences. Lastly, K-means clustering strategies were suggested to address heterogeneous data situations but used predefined weather groups instead of fractal methods.

In summary, literature strongly suggests that hybrid deep learning models offer a promising path for solar forecasting. By combining CNN's feature extraction capabilities with XGBoost's robust regression mechanism, models can capture complex environmental dependencies and produce reliable predictions. Building on Zhang et al. (2024), our work extends this line of research by incorporating improved temporal modeling through sliding windows, deeper CNN feature extraction, and adaptive clustering strategies to further enhance accuracy and scalability.

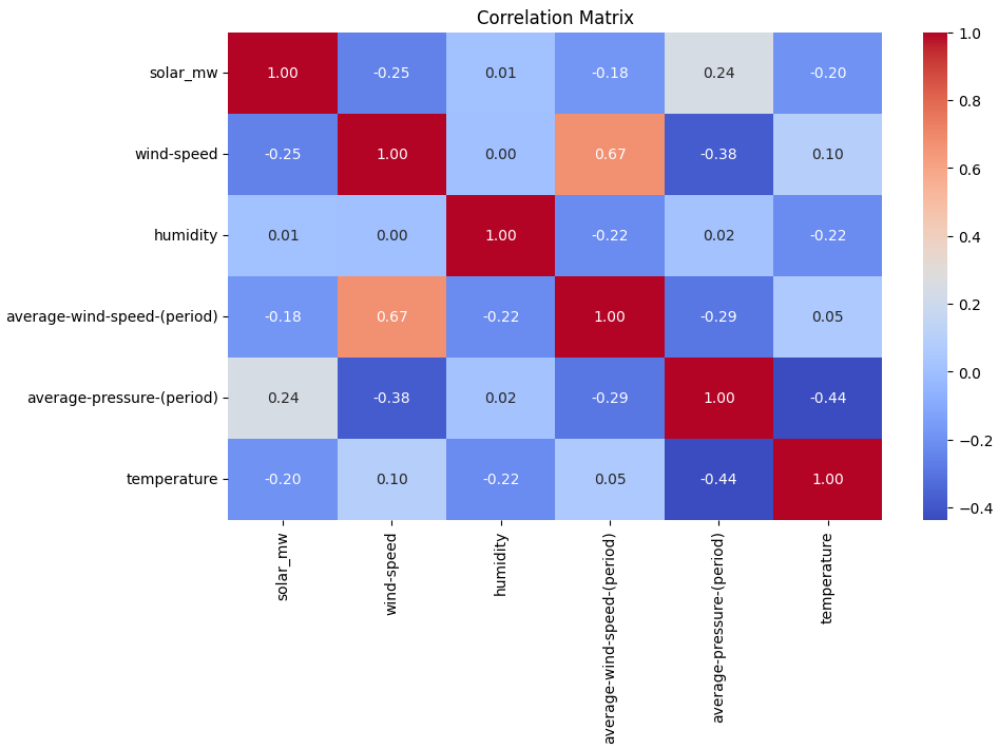
Reference:

https://www.frontiersin.org/journals/energy-research/articles/10.3389/fenrg.2024.1411461/full

3. DataSet Description

The dataset used in this project comprises historical records of solar power generation along with weather-related parameters such as temperature, humidity, wind speed, wind direction, cloud cover, solar irradiance, average wind speed (over a period), and average atmospheric pressure (over a period). These features serve as essential inputs for developing a predictive model capable of estimating solar power output under varying environmental conditions.

For the project *"AI Algorithm Used in Predicting the Solar Power Output"*, a structured dataset was provided containing attributes such as Datetime, Solar_MW, Wind Direction, Wind Speed, Humidity, Average Wind Speed (period), Average Pressure (period), and Temperature. A preliminary Explanatory Data Analysis step was carried out to examine the correlation between these features and the target output. The analysis revealed that most features demonstrated either a low positive or low negative correlation with the output variable.



	solar_mw	wind-speed	humidity	average-wind-speed-(period)	average-pressure-(period)	temperature
solar_mw	1.00	-0.25	0.01	-0.18	0.24	-0.20
wind-speed	-0.25	1.00	0.00	0.67	-0.38	0.10
humidity	0.01	0.00	1.00	-0.22	0.02	-0.22
average-wind-speed-(period)	-0.18	0.67	-0.22	1.00	-0.29	0.05
average-pressure-(period)	0.24	-0.38	0.02	-0.29	1.00	-0.44
temperature	-0.20	0.10	-0.22	0.05	-0.44	1.00

Figure 1.1: *Correlation matrix showing the relationships between weather parameters, inverter outputs, and solar power generation. The values indicate the strength and direction of linear correlations, where darker shades represent stronger positive or negative relationships.*

This observation led us to explore and develop our own dataset in order to improve feature relevance and build a more accurate model for predicting solar power output.

To address the limitations of the initial dataset, a new dataset was collected from Kaggle

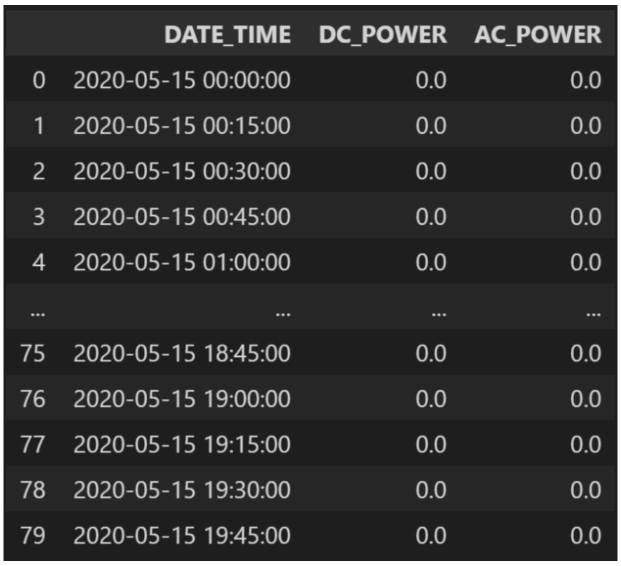
Solar Power Generation Data - https://www.kaggle.com/datasets/anikannal/solar-power-generation-data

This dataset was gathered from the solar power plant in India over a 34-day period. It consists of two pairs of files, where each pair includes one power generation dataset and one sensor readings dataset. The power generation data is recorded at the inverter level, with each inverter connected to multiple lines of solar panels, while the sensor data is collected at the plant level through an optimally placed array of sensors. This dataset provides a richer set of features and more reliable measurements, enabling the development of a more accurate model for predicting solar power output.

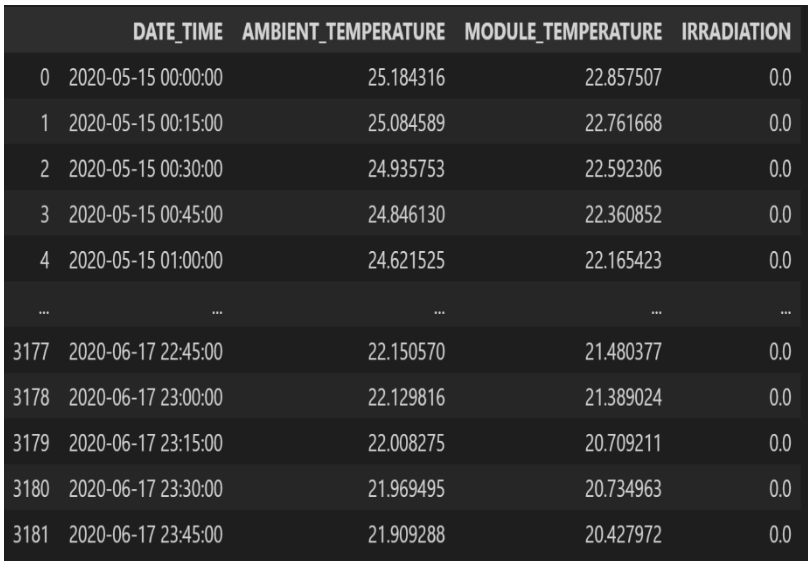
To effectively utilize this dataset for our study, several preprocessing techniques were applied in order to refine the data and construct a more reliable dataset, ultimately supporting the development of a better predictive model.

The dataset consists of data from a solar power plant collected over a 34-day period. For our study, the 80% data from the plant was used to train the predictive model, while the 20% data from the plant was used for validation and testing, ensuring the model's performance could be evaluated on independent, unseen data.

The Plant dataset has two sub datasets which are the Generation Dataset and the Weather Sensor Dataset. The Generation Dataset comprises of attributes like **date_time, plant_id, source_key, dc_power, ac_power, daily_yield, total_yield** while the weather sensor dataset includes **date_time, plant_id, source_key, ambient_temperature, module_temperature, irradiation.**



	DATE_TIME	DC_POWER	AC_POWER
0	2020-05-15 00:00:00	0.0	0.0
1	2020-05-15 00:15:00	0.0	0.0
2	2020-05-15 00:30:00	0.0	0.0
3	2020-05-15 00:45:00	0.0	0.0
4	2020-05-15 01:00:00	0.0	0.0
...	...	...	...
75	2020-05-15 18:45:00	0.0	0.0
76	2020-05-15 19:00:00	0.0	0.0
77	2020-05-15 19:15:00	0.0	0.0
78	2020-05-15 19:30:00	0.0	0.0
79	2020-05-15 19:45:00	0.0	0.0



	DATE_TIME	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION
0	2020-05-15 00:00:00	25.184316	22.857507	0.0
1	2020-05-15 00:15:00	25.084589	22.761668	0.0
2	2020-05-15 00:30:00	24.935753	22.592306	0.0
3	2020-05-15 00:45:00	24.846130	22.360852	0.0
4	2020-05-15 01:00:00	24.621525	22.165423	0.0
...	...	...	...	...
3177	2020-06-17 22:45:00	22.150570	21.480377	0.0
3178	2020-06-17 23:00:00	22.129816	21.389024	0.0
3179	2020-06-17 23:15:00	22.008275	20.709211	0.0
3180	2020-06-17 23:30:00	21.969495	20.734963	0.0
3181	2020-06-17 23:45:00	21.909288	20.427972	0.0

Figure 1.2: *Overview of the solar power dataset, illustrating the structure of the Generation and Weather Sensor sub-datasets for each plant, including key features such as power outputs and environmental parameters.*

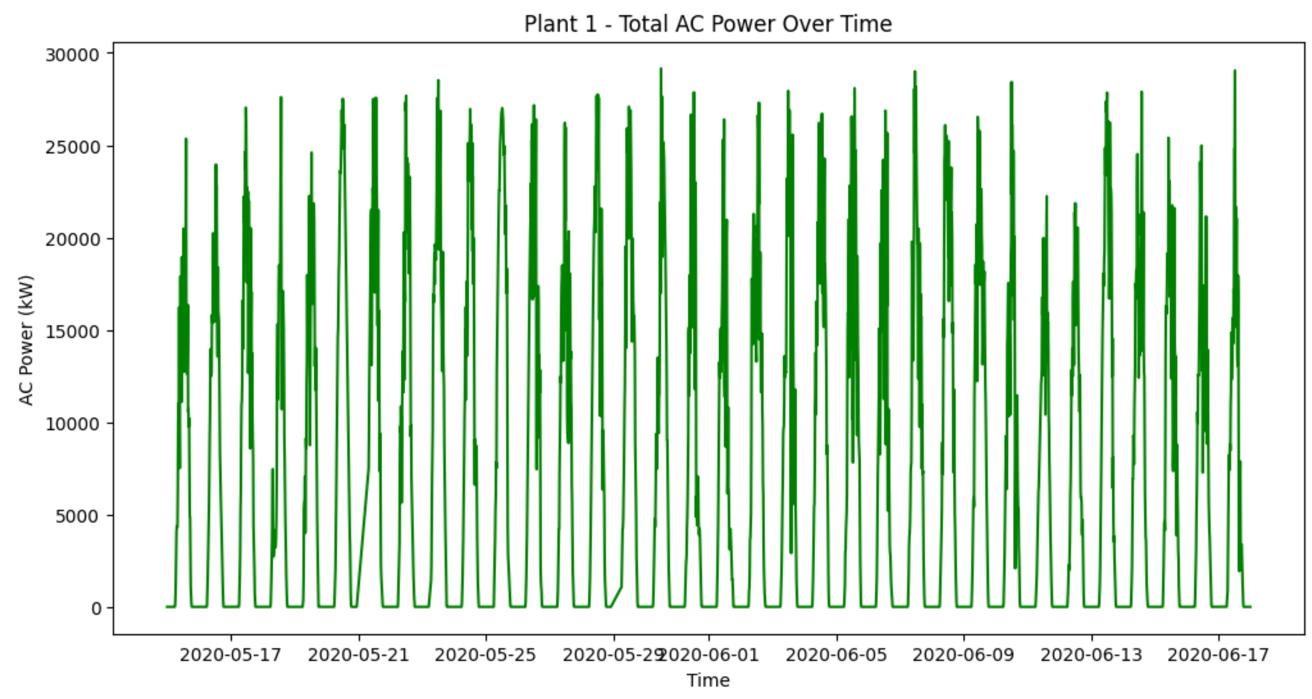


Figure 1.3: *AC Power Trend Over Time in the Plant.*

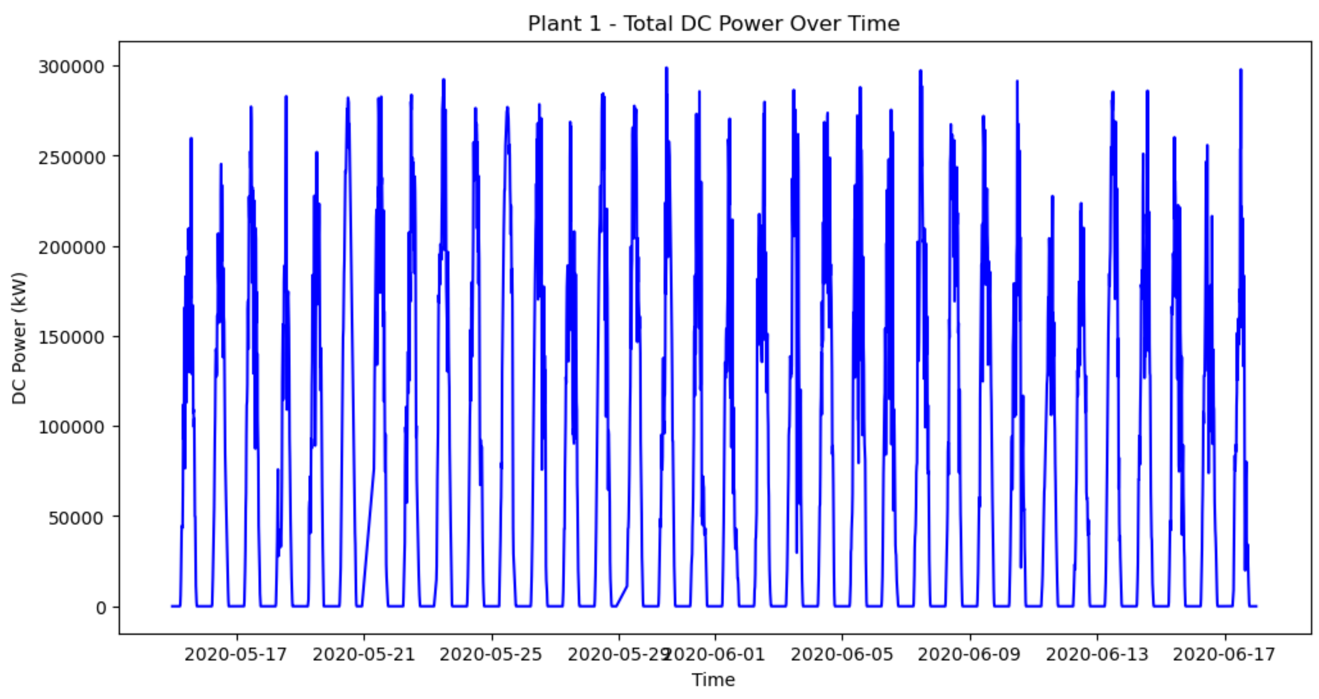
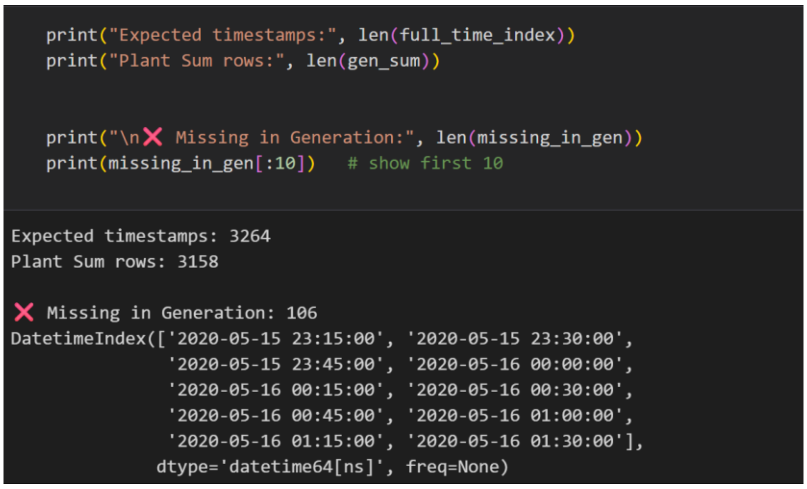


Figure 1.4: *DC Power Trend Over Time in the Plant.*

4. Data Preprocessing and Cleaning

Initially, the correlation between the features and the target output was analysed, revealing strong positive relationships that aligned well with the objectives of the study.

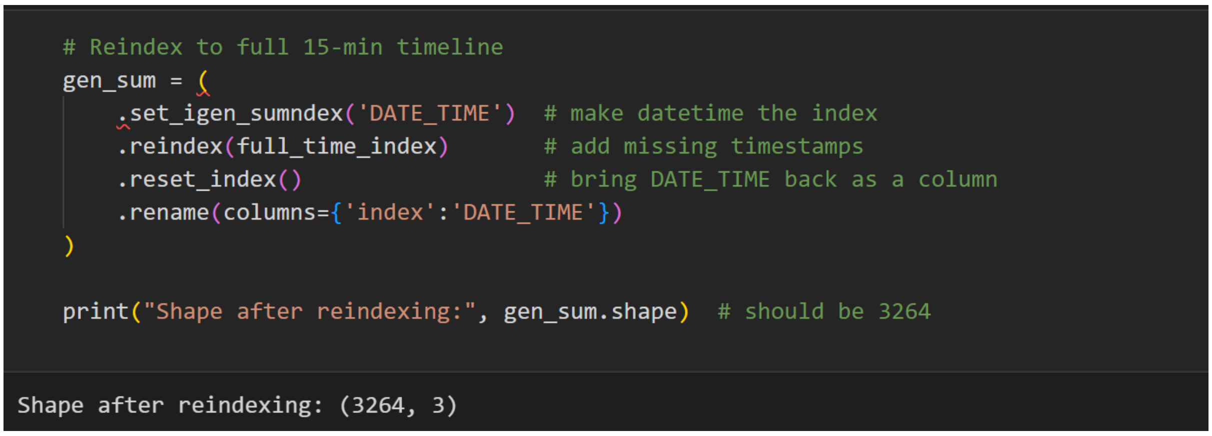
Following this, preprocessing was performed on the Generation Dataset. Certain features that were not relevant to the study, such as plant_id, source_key, daily_yield, and total_yield, were removed. A detailed examination of the dataset revealed that the data were recorded at 15-minute intervals. Ideally, this should have resulted in 3,264 data points over the 34-day period, but only 3,158 records were present, indicating 106 missing entries. These **missing timestamps were added manually, and linear interpolation** was applied to fill the missing values for dc_power and ac_power. This preprocessing resulted in a cleaned, complete, and reliable dataset suitable for model training.



```
print("Expected timestamps:", len(full_time_index))
print("Plant Sum rows:", len(gen_sum))


print("\n❌ Missing in Generation:", len(missing_in_gen))
print(missing_in_gen[:10])   # show first 10
```

```
Expected timestamps: 3264
Plant Sum rows: 3158

❌ Missing in Generation: 106
DatetimeIndex(['2020-05-15 23:15:00', '2020-05-15 23:30:00',
               '2020-05-15 23:45:00', '2020-05-16 00:00:00',
               '2020-05-16 00:15:00', '2020-05-16 00:30:00',
               '2020-05-16 00:45:00', '2020-05-16 01:00:00',
               '2020-05-16 01:15:00', '2020-05-16 01:30:00'],
              dtype='datetime64[ns]', freq=None)
```

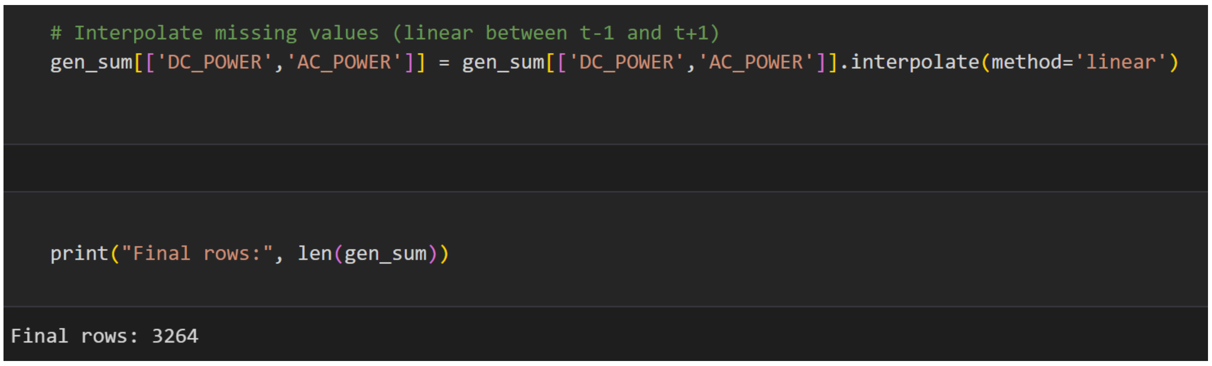
Figure 1.5: *Missing Time Stamps found.*



```
# Reindex to full 15-min timeline
gen_sum = (
    .set_igen_sumndex('DATE_TIME')  # make datetime the index
    .reindex(full_time_index)       # add missing timestamps
    .reset_index()                  # bring DATE_TIME back as a column
    .rename(columns={'index':'DATE_TIME'})
)

print("Shape after reindexing:", gen_sum.shape)  # should be 3264
```

```
Shape after reindexing: (3264, 3)
```

Figure 1.6: *Missing Time Stamps rectified and reshaped.*



```
# Interpolate missing values (linear between t-1 and t+1)
gen_sum[['DC_POWER','AC_POWER']] = gen_sum[['DC_POWER','AC_POWER']].interpolate(method='linear')


print("Final rows:", len(gen_sum))
```

```
Final rows: 3264
```

Figure 1.7: *Linear-Interpolation applied for the missing values.*

Similarly, the Weatl
methodology. This ensured a reliable and complete dataset, allowing meaningful correlations between weather parameters and power output to be captured for model training.

5. Exploratory Data Analysis (EDA)

After the preprocessing that was done separately on both the datasets of plant 1, then the datasets of Generation and Weather Data were merged and the new dataset was formed. This comprised of features such as **date_time, dc_power, ac_power, ambient_temperature, module_temperature, irradiation.**

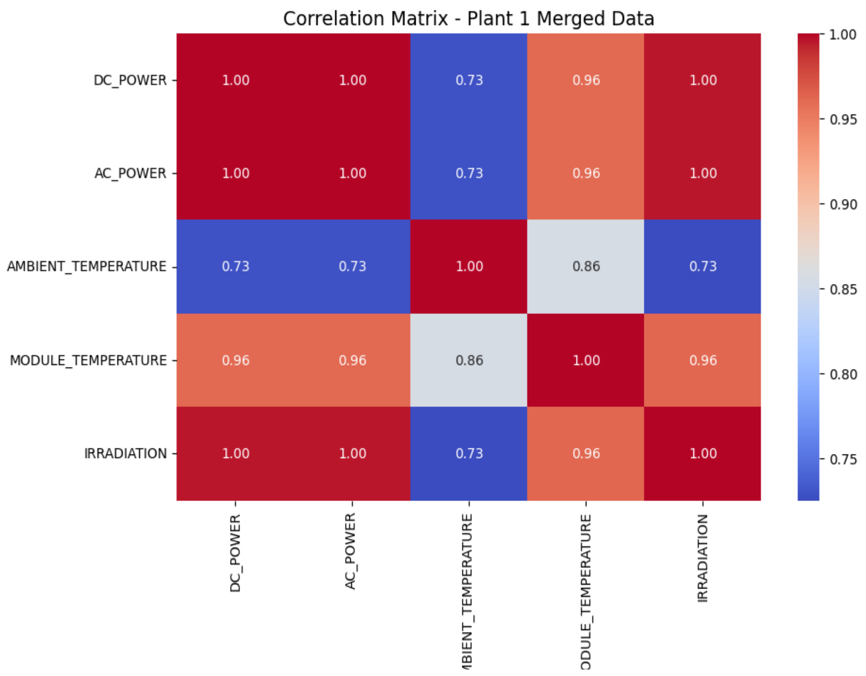
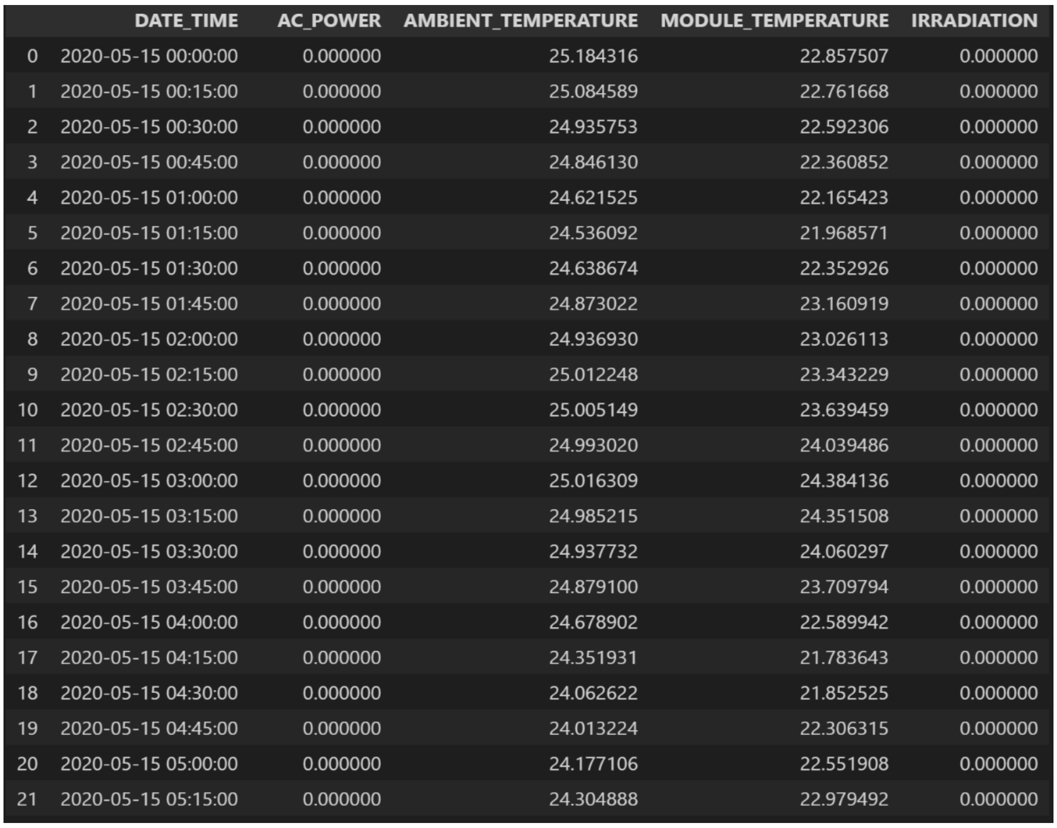


Figure 1.8: *Correlation matrix showing relationships between features and solar power output.*

After merging the Generation and Weather Sensor datasets, the correlation between the features and the target output was analyzed, revealing strong positive relationships that indicated the creation of a more reliable and meaningful dataset for model training.



	DATE_TIME	AC_POWER	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION
0	2020-05-15 00:00:00	0.000000	25.184316	22.857507	0.000000
1	2020-05-15 00:15:00	0.000000	25.084589	22.761668	0.000000
2	2020-05-15 00:30:00	0.000000	24.935753	22.592306	0.000000
3	2020-05-15 00:45:00	0.000000	24.846130	22.360852	0.000000
4	2020-05-15 01:00:00	0.000000	24.621525	22.165423	0.000000
5	2020-05-15 01:15:00	0.000000	24.536092	21.968571	0.000000
6	2020-05-15 01:30:00	0.000000	24.638674	22.352926	0.000000
7	2020-05-15 01:45:00	0.000000	24.873022	23.160919	0.000000
8	2020-05-15 02:00:00	0.000000	24.936930	23.026113	0.000000
9	2020-05-15 02:15:00	0.000000	25.012248	23.343229	0.000000
10	2020-05-15 02:30:00	0.000000	25.005149	23.639459	0.000000
11	2020-05-15 02:45:00	0.000000	24.993020	24.039486	0.000000
12	2020-05-15 03:00:00	0.000000	25.016309	24.384136	0.000000
13	2020-05-15 03:15:00	0.000000	24.985215	24.351508	0.000000
14	2020-05-15 03:30:00	0.000000	24.937732	24.060297	0.000000
15	2020-05-15 03:45:00	0.000000	24.879100	23.709794	0.000000
16	2020-05-15 04:00:00	0.000000	24.678902	22.589942	0.000000
17	2020-05-15 04:15:00	0.000000	24.351931	21.783643	0.000000
18	2020-05-15 04:30:00	0.000000	24.062622	21.852525	0.000000
19	2020-05-15 04:45:00	0.000000	24.013224	22.306315	0.000000
20	2020-05-15 05:00:00	0.000000	24.177106	22.551908	0.000000
21	2020-05-15 05:15:00	0.000000	24.304888	22.979492	0.000000

Figure 1.9: *Merged Final Dataset for Model Training*

6. Feature Engineering

Feature engineering is a crucial step in preparing data for solar power forecasting. Raw data collected from generation meters and weather sensors cannot be directly used for model training, as it lacks structure and contains variations that the model cannot interpret effectively. The goal of feature engineering in this project was to transform raw measurements into meaningful inputs that capture both environmental influence and temporal dynamics.

6.1. Time Stamp Feature Extraction

The dataset originally contained a DATE_TIME column with 15-minute interval records. Since machine learning models cannot directly interpret timestamps, we extracted time-based features:

- Hour of the Day
- Minute of the Hour
- Time Fraction

Time Fraction refers to the portion of the day represented as a decimal value, typically derived from the timestamp.

$$\mathbf{T_{frac} = hour + \frac{\mathit{minutes}}{60}}$$

6.2. Cyclic Encoding of time

Time is periodic in nature (00:00 follows 23:59). Using raw hour values would make 23:00 and 00:00 appear far apart, although they are only one hour apart. To preserve periodicity, cyclical encoding was applied:

$$\mathbf{\sin \mathit{time} = \sin\left(\frac{2\pi * \mathit{Tfrac}}{24}\right) \qquad \cos \mathit{time} = \cos\left(\frac{2\pi * \mathit{Tfrac}}{24}\right)}$$

This maps time onto a unit circle, ensuring the model recognizes the continuity of daily cycles such as sunrise and sunset.

6.3. Sliding Window Sequences

Solar power forecasting is inherently a time-series problem. The current output depends not only on present conditions but also on historical values. To capture this dependency, we used a sliding window approach with a sequence length of 6 intervals (90 minutes).

For Example: If $\mathbf{X_t}$ represents feature vector at time t then:

$$\mathbf{[X_{t-5}, X_{t-4}, X_{t-3}, X_{t-2}, X_{t-1}, X_t]}$$

With the Label list being the AC Power output of the next time stamp:

$$\mathbf{Y_{t+1} = \text{AC Power at time t+1}}$$

6.4. Normalization of Features

The dataset contained variables in different ranges (e.g., irradiation between 0–1, temperatures between 15–40, AC power up to 25,000 W). Without scaling, larger-magnitude variables would dominate training. Therefore, MinMax scaling was applied to input features:

$$X' = \frac{X - Xmin}{Xmax - Xmin}$$

7. Modeling Approach

In this project, we developed a hybrid deep learning and machine learning framework to forecast the solar power output. The high-level idea was to leverage the strength of two strong models:

- **Convolutional Neural Network (CNN):** used as a feature extractor that would learn beneficial patterns from the data automatically, including how temperature and irradiation impact power generation.

- **XGBoost:** used as the regressor (predictor) that takes these features extracted from the CNN above and predicts the next AC power value.

8. CNN Model Design

The Convolutional Neural Network (CNN) was employed as the primary feature extractor in the hybrid architecture. The CNN is particularly effective in handling sequential and structured data because it can learn local dependencies and extract higher-level features without manual engineering.

Input Sequence Structure

The model takes a sliding window approach with a sequence length of 6 which means the network is looking at six consecutive time steps (each a 15-minute period) to predict the AC power at the next time step. Each input sequence has dimensions (6 × number_of_features) with features including, ambient temperature, module temperature, and irradiation, plus time-based encoded features (sin_time and cos_time).

CNN Layers (Conv1D, Pooling, Dense)

- The CNN was constructed using 1D convolutional layers (Conv1D) with causal padding to ensure temporal order is preserved.
- **First Conv1D Layer**: 64 filters with kernel size 3, activation ReLU. This extracts coarse patterns in the time series data.
- **Second Conv1D Layer**: 32 filters with kernel size 3, activation ReLU. This refines feature maps and captures shorter dependencies.
- **Global Average Pooling Layer**: Converts the sequence of features into a single feature vector, reducing dimensionality and preventing overfitting.
- **Dense Layer**: A fully connected dense layer with 64 neurons and ReLU activation that transforms pooled features into a high-level representation.

Finally, the CNN output passes through a Dense(1) layer during standalone training to approximate AC power.

Feature Extraction Workflow

Instead of using CNN as the final predictor, the model was reconfigured to act simply as a feature extractor. The final dense layer was extracted before the output layer (64 features) as a compact feature vector. This feature vector encodes various non-linear patterns, interactions between environmental variables, and periodic changes in the vector of inputs.

Early Stopping and Overfitting Prevention

To minimize overfitting, early stopping with a patience of 5 epochs was implemented, which restored the best weights while validating. This was done in an effort to ensure that the CNN did not unnecessarily train on noise. Lastly, the use of a validation split (20%) allowed another unbiased validation during training.

9. XG Boost Model Design

The extracted CNN feature vectors were then passed into an XGBoost Regressor. XGBoost is a gradient boosting method that builds decision tree ensembles to minimize error.

Gradient Boosting Overview

Gradient boosting combines multiple weak learners (decision trees) into a strong model. Each subsequent tree corrects the errors of the previous ones by fitting the residuals. This makes XGBoost highly efficient for non-linear regression tasks such as solar power forecasting.

Training on Extracted Features

The XGBoost regressor was trained on the 64-dimensional CNN feature vectors rather than raw input data. This means that XGBoost could focus entirely on regression and CNN could focus entirely on feature representation.

Hyperparameters

The XGBoost model was configured with the following parameters:

- n_estimators = 1000: number of boosting rounds.
- learning_rate = 0.01: small step size for stable convergence.
- max_depth = 6: controls the depth of each tree.
- subsample = 0.8: prevents overfitting by using 80% of training data for each tree.
- colsample_bytree = 0.8: randomly selects 80% of features per tree.
- reg_lambda = 1.0: L2 regularization to penalize complexity.

10. Training Validation and Metrics

The training process followed a systematic workflow combining the CNN feature extractor with the XGBoost regressor:

- **Data Split:** The data was divided into 80% training and 20% testing. A full separation ensured that the data used for testing was not seen during training.

- **CNN Training:** The CNN model was trained on scaled sliding-window sequences to learn feature representations. Early stopping was set to a patience of 5 epochs in order to avoid overfitting and highly correlated weights being learned during the fitting process.

- **Feature Extraction:** The second-to-last dense layer of the CNN model was used as the feature representation for fitted data values. The feature vectors captured dependencies of an environmental and time-series nature.

- **XGBoost Training:** The feature vectors represented data from a fitted CNN to train an XGBoost regressor. The gradient boosting method also fits complex non-linear patterns as well as residual errors.

10.1. Evaluation Metrics

RMSE (Root Mean Square Error):

$$\boldsymbol{RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(yi - yi^\wedge)^2}}$$

RMSE provides an interpretable measure of prediction error in kW and is sensitive to larger deviations.

Mean Absolute Error:

$$\boldsymbol{MAE = \frac{1}{n}\sum_{i=1}^{n}|yi - yi^\wedge|}$$

MAE indicates the average prediction error in kW, offering an easy-to-understand measure of accuracy.

Co efficient Determination (R^2):

$$\boldsymbol{R^2 = 1 - \frac{\sum_{i=1}^{n}(yi - yi^\wedge)^2}{\sum_{i=1}^{n}(yi - yibar)^2}}$$

R^2 quantifies how much of the variation in AC power output is explained by the model, where 1 indicates perfect predictions.

11. Results and Discussion

The hybrid CNN–XGBoost model demonstrated strong predictive performance on the Plant 1 dataset. The results on the held-out test data were:

- RMSE: 2,585.90 kW
- R^2: 0.9063
- MAE: 1,376.32 kW

Interpretation of Results

- The low RMSE (2,585.90 kW) shows that the model maintains a small overall prediction error relative to the plant's generation capacity.

- The MAE (1,376.32 kW) indicates that, on average, the model deviates by just over 1 MW, which is acceptable given the variability of solar output.

- The R^2 score of 0.9063 confirms that the model explains over 90% of the variance in AC power, proving the effectiveness of the CNN+XGBoost hybrid framework

12. Conclusion

In this study, we provide empirical evidence of a hybrid deep learning framework for short-term solar power forecasting that combines the feature extraction capability of Convolutional Neural Networks (CNN) with the predictive capability of XGBoost. The model shown to achieve excellent performance using real plant level data including other environmental variables such as ambient temperature, module temperature, and irradiation with an R2 of 0.9063 and MAE of 1376.32 W. These results provide strong evidence that the hybrid method better captures temporal dependencies and the interactions of non-linear features than by using direct methods.

From a practical perspective, the forecasting paradigm also provides valuable benefits to operators of solar plants and to managers of power systems. Reliable short-term predictions can facilitate improved backup power scheduling, optimal utilization of energy storage, and smoother grid operation. Moreover, the deployment of the model with a simple interface also highlights its practicality for real world scenarios lastly.

Overall, this work contributes toward making renewable energy integration more reliable and efficient. Future improvements could involve incorporating external weather forecast data, extending the framework to multiple plants, and exploring advanced models such as Transformers for even higher accuracy. By bridging data science with renewable energy, this project provides a step forward in addressing one of the key challenges in the global transition to sustainable energy systems.

13. References

https://www.frontiersin.org/journals/energy-research/articles/10.3389/fenrg.2024.1411461/full

https://www.kaggle.com/datasets/anikannal/solar-power-generation-data

https://www.geeksforgeeks.org/machine-learning/introduction-convolution-neural-network/

https://www.geeksforgeeks.org/machine-learning/xgboost/

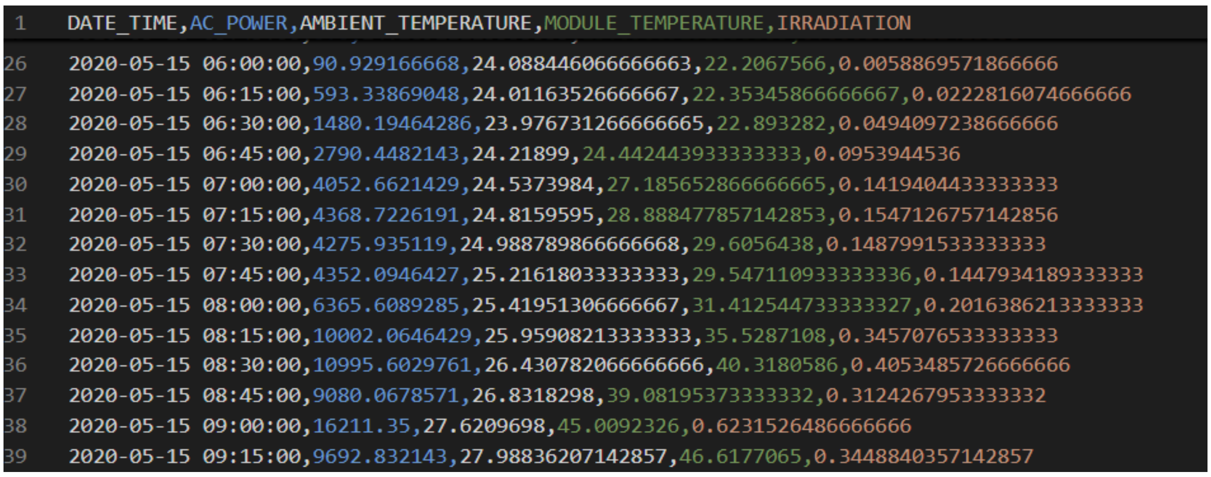
https://www.tensorflow.org/tutorials

https://shrmtmt.medium.com/understand-the-capabilities-of-cyclic-encoding-5b68f831387e

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

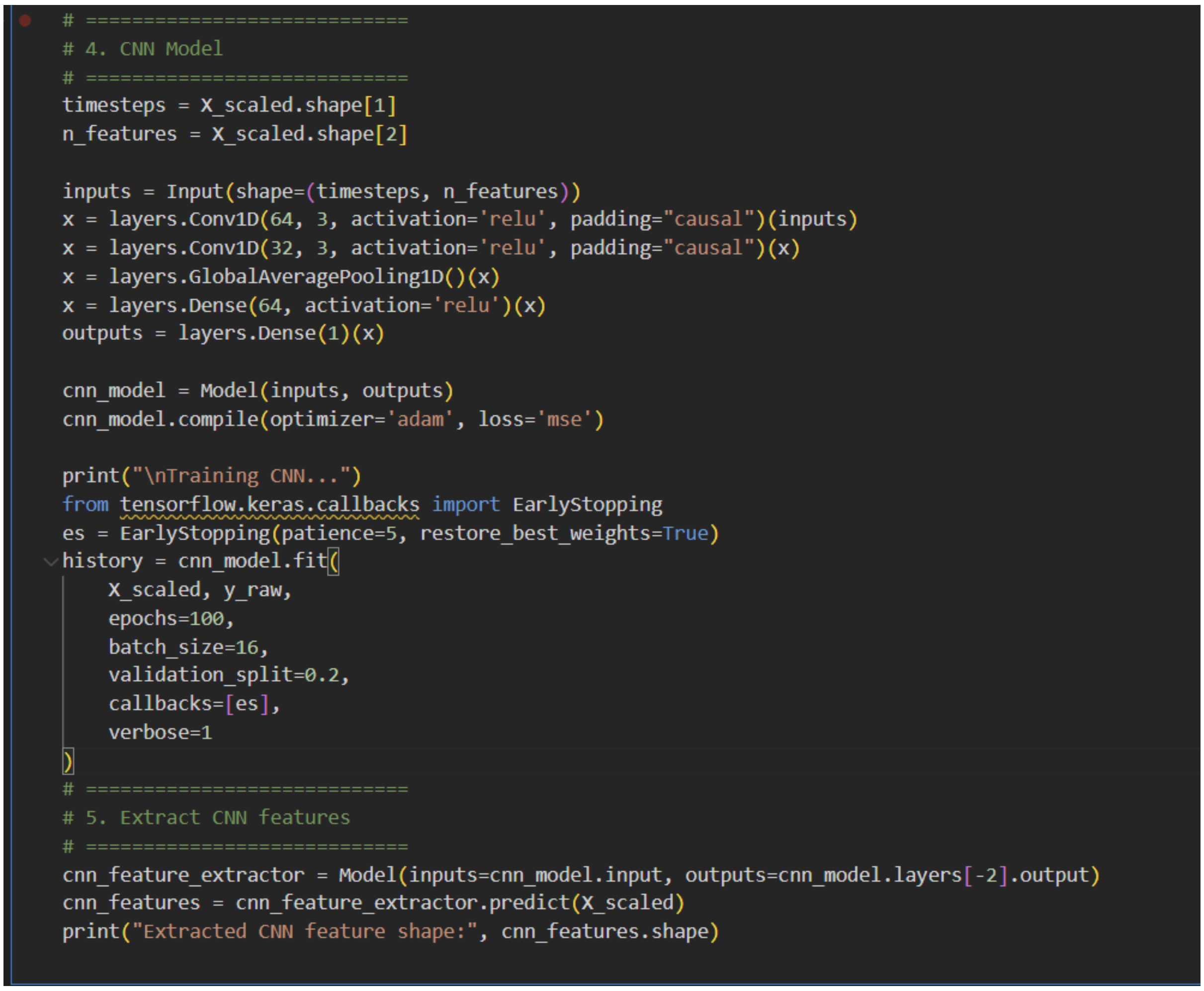
14.Appendix

1. Raw Dataset Samples.

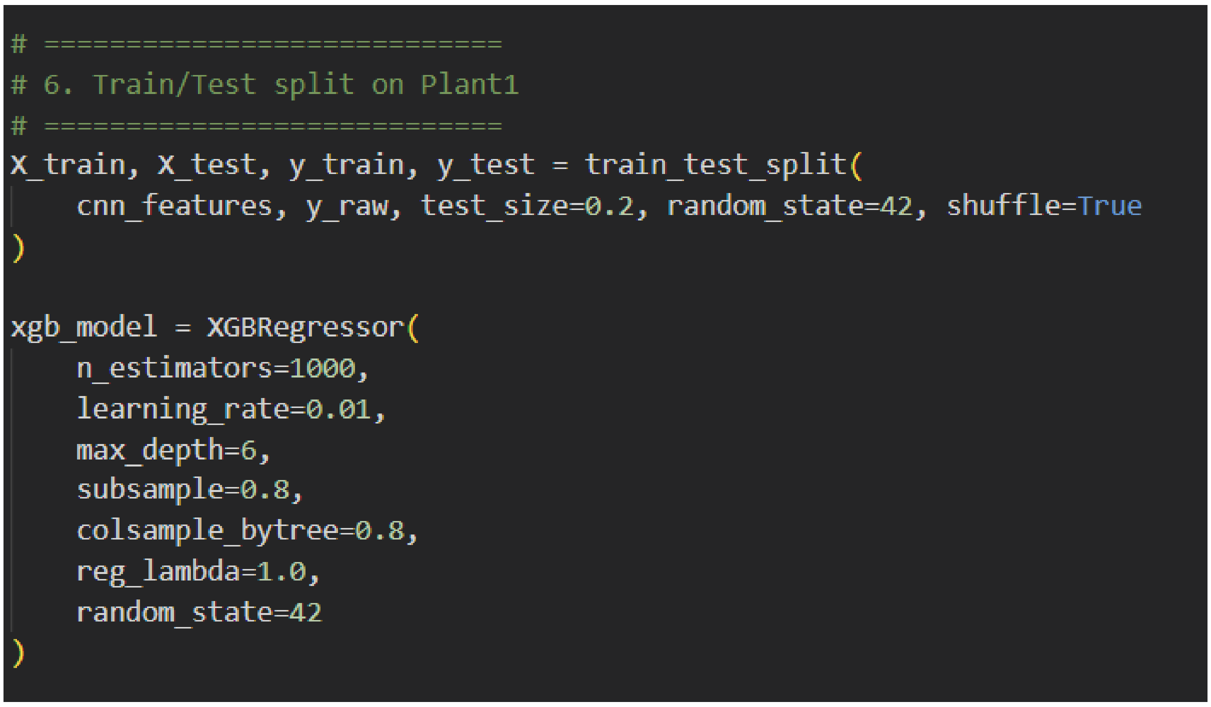


```
DATE_TIME,AC_POWER,AMBIENT_TEMPERATURE,MODULE_TEMPERATURE,IRRADIATION
2020-05-15 06:00:00,90.929166668,24.088446066666663,22.2067566,0.0058869571866666
2020-05-15 06:15:00,593.33869048,24.01163526666667,22.35345866666667,0.0222816074666666
2020-05-15 06:30:00,1480.19464286,23.976731266666665,22.893282,0.0494097238666666
2020-05-15 06:45:00,2790.4482143,24.21899,24.442443933333333,0.0953944536
2020-05-15 07:00:00,4052.6621429,24.5373984,27.185652866666665,0.1419404433333333
2020-05-15 07:15:00,4368.7226191,24.8159595,28.888477857142853,0.1547126757142856
2020-05-15 07:30:00,4275.935119,24.988789866666668,29.6056438,0.1487991533333333
2020-05-15 07:45:00,4352.0946427,25.21618033333333,29.547110933333336,0.1447934189333333
2020-05-15 08:00:00,6365.6089285,25.41951306666667,31.412544733333327,0.2016386213333333
2020-05-15 08:15:00,10002.0646429,25.95908213333333,35.5287108,0.3457076533333333
2020-05-15 08:30:00,10995.6029761,26.430782066666666,40.3180586,0.4053485726666666
2020-05-15 08:45:00,9080.0678571,26.8318298,39.08195373333332,0.3124267953333332
2020-05-15 09:00:00,16211.35,27.6209698,45.0092326,0.6231526486666666
2020-05-15 09:15:00,9692.832143,27.98836207142857,46.6177065,0.3448840357142857
```

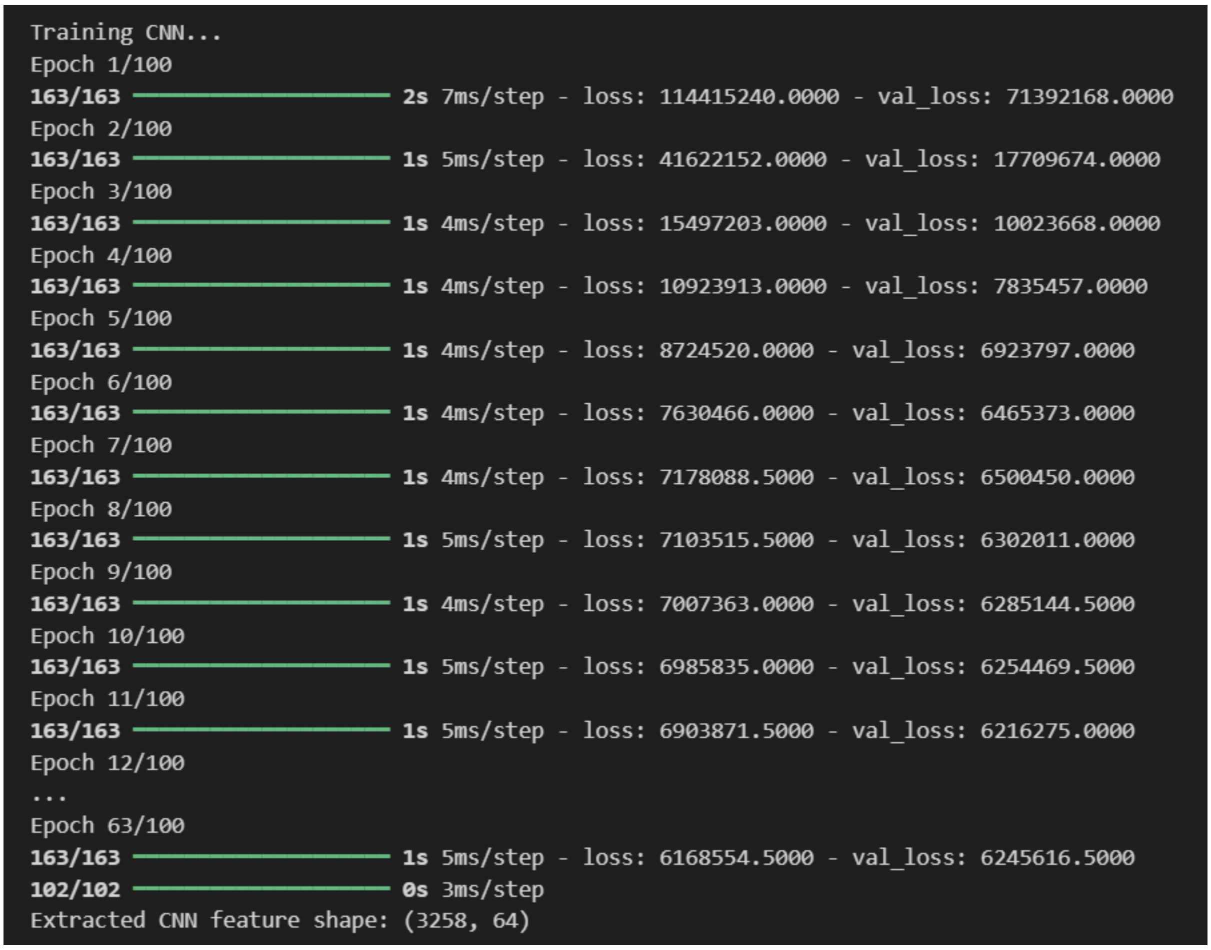
2. Code Snippets



```
# ============================
# 4. CNN Model
# ============================
timesteps = X_scaled.shape[1]
n_features = X_scaled.shape[2]

inputs = Input(shape=(timesteps, n_features))
x = layers.Conv1D(64, 3, activation='relu', padding="causal")(inputs)
x = layers.Conv1D(32, 3, activation='relu', padding="causal")(x)
x = layers.GlobalAveragePooling1D()(x)
x = layers.Dense(64, activation='relu')(x)
outputs = layers.Dense(1)(x)

cnn_model = Model(inputs, outputs)
cnn_model.compile(optimizer='adam', loss='mse')

print("\nTraining CNN...")
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(patience=5, restore_best_weights=True)
history = cnn_model.fit(
    X_scaled, y_raw,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[es],
    verbose=1
)
# ============================
# 5. Extract CNN features
# ============================
cnn_feature_extractor = Model(inputs=cnn_model.input, outputs=cnn_model.layers[-2].output)
cnn_features = cnn_feature_extractor.predict(X_scaled)
print("Extracted CNN feature shape:", cnn_features.shape)
```



```
# ============================
# 6. Train/Test split on Plant1
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    cnn_features, y_raw, test_size=0.2, random_state=42, shuffle=True
)

xgb_model = XGBRegressor(
    n_estimators=1000,
    learning_rate=0.01,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_lambda=1.0,
    random_state=42
)
```



```
Training CNN...
Epoch 1/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 114415240.0000 - val_loss: 71392168.0000
Epoch 2/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 41622152.0000 - val_loss: 17709674.0000
Epoch 3/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 15497203.0000 - val_loss: 10023668.0000
Epoch 4/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 10923913.0000 - val_loss: 7835457.0000
Epoch 5/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 8724520.0000 - val_loss: 6923797.0000
Epoch 6/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 7630466.0000 - val_loss: 6465373.0000
Epoch 7/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 7178088.5000 - val_loss: 6500450.0000
Epoch 8/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 7103515.5000 - val_loss: 6302011.0000
Epoch 9/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 7007363.0000 - val_loss: 6285144.5000
Epoch 10/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6985835.0000 - val_loss: 6254469.5000
Epoch 11/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6903871.5000 - val_loss: 6216275.0000
Epoch 12/100
...
Epoch 63/100
163/163 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - loss: 6168554.5000 - val_loss: 6245616.5000
102/102 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Extracted CNN feature shape: (3258, 64)
```