Solar Power Generation Forecasting

MALSAWMZUALA KHIANGTE
IIT BHUBANESWAR

Problem Statement:

The irregular nature of solar energy generation creates big challenges for keeping the power grid stable and managing energy efficiently. Sudden changes in generation can cause energy waste, higher operating costs, and greater dependence on less eco-friendly backup power sources.

This project builds a machine learning model tailored to a solar park in Mizoram to accurately predict its power output. This helps operators plan better, manage resources efficiently, and integrate solar energy smoothly into the grid — reducing energy waste and supporting Mizoram's shift toward clean energy.

Project Workflow

DATA COLLECTION
&
DATA CLEANING

FEATURE
ENGINEERING

MODEL TRAINING
&
EVALUATION

DEPLOYMENT

Data Acquisition

Raw Data

Source.Name	20 M	Column	Column	Column4	Column5	Column6	Colum	Colum
Vankal 1.4.2023.xls				ACTUAL GENERATION (in MW)			Date : 01/04/2023	
Vankal 1.4.2023.xls	Block	From	To	SUNFREE - 10 MW	SUNFREE - 5 MW	ATA - 5 MW	Total MW	Total MWh
Vankal 1.4.2023.xlsx				ACTIVE POWER (M	ACTIVE POWER (M	ACTIVE POWER (MW)		
Vankal 1.4.2023.xls	1	00:00:00	00:15:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	2	00:15:00	00:30:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	3	00:30:00	00:45:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	4	00:45:00	01:00:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	5	01:00:00	01:15:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	6	01:15:00	01:30:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	7	01:30:00	01:45:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	8	01:45:00	02:00:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	9	02:00:00	02:15:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	10	02:15:00	02:30:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	11	02:30:00	02:45:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	12	02:45:00	03:00:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	13	03:00:00	03:15:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	14	03:15:00	03:30:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	15	03:30:00	03:45:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	16	03:45:00	04:00:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	17	04:00:00	04:15:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	18	04:15:00	04:30:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	19	04:30:00	04:45:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	20	04:45:00	05:00:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	21	05:00:00	05:15:00	0.00	0.00	0.00	0.00	0.00
Vankal 1.4.2023.xls	22	05:15:00	05:30:00	0.07	0.00	0.00	0.08	0.02
Vankal 1.4.2023.xls	23	05:30:00	05:45:00	0.19	0.07	0.09	0.35	0.08
Vankal 1.4.2023.xls	24	05:45:00	06:00:00	0.49	0.19	0.24	0.93	0.26
Vankal 1.4.2023.xls	25	06:00:00	06:15:00	0.63	0.25	0.31	1.19	0.53
Vankal 1.4.2023.xls	26	06:15:00	06:30:00	0.99	0.39	0.49	1.87	0.96
Vankal 1.4.2023.xls	27	06:30:00	06:45:00	0.76	0.30	0.38	1.43	1.34
Vankal 1.4.2023.xls	28	06:45:00	07:00:00	0.64	0.25	0.32	1.21	1.63
Vankal 1.4.2023.xls	29	07:00:00	07:15:00	0.56	0.22	0.28	1.05	1.97
Vankal 1.4.2023.xls	30	07:15:00	07:30:00	0.38	0.15	0.19	0.72	2.15
Vankal 1.4.2023.xls	31	07:30:00	07:45:00	0.18	0.06	0.09	0.32	2.29
Vankal 1.4.2023.xls	32	07:45:00	08:00:00	0.12	0.00	0.06	0.19	2.35
Vankal 1.4.2023.xls	33	08:00:00	08:15:00	0.00	0.00	0.04	0.05	2.40
Vankal 1.4.2023.xls	34	08:15:00	08:30:00	0.20	0.08	0.10	0.39	2.48
Vankal 1.4.2023.xls	35	08:30:00	08:45:00	0.10	0.04	0.05	0.19	2.55
Vankal 1.4.2023.xls	36	08:45:00	09:00:00	0.01	0.01	0.04	0.06	2.60
Vankal 1.4.2023.xls	37	09:00:00	09:15:00	0.16	0.06	0.09	0.30	2.66
Vankal 1.4.2023.xls	38	09:15:00	09:30:00	0.27	0.11	0.14	0.52	2.75
Vankal 1.4.2023.xls	39	09:30:00	09:45:00	0.69	0.22	0.24	1.15	2.93
Vankal 1.4.2023.xls	40	09:45:00	10:00:00	1.69	0.67	0.85	3.21	3.46
Vankal 1.4.2023.xls	41	10:00:00	10:15:00	2.40	0.97	1.18	4.55	4.40
Vankal 1.4.2023.xls	42	10:15:00	10:30:00	2.27	1.17	1.52	4.96	5.90
Vankal 1.4.2023.xls	43	10:30:00	10:45:00	2.36	1.02	1.27	4.65	7.17
Vankal 1.4.2023.xls	44	10:45:00	11:00:00	2.11	0.91	1.21	4.23	8.16
Vankal 1.4.2023.xls	45	11:00:00	11:15:00	8.01	3.56	3.64	15.20	9.39
Vankal 1.4.2023.xls	46	11:15:00	11:30:00	3.21	1.54	1.97	6.73	12.28
Vankal 1.4.2023.xls	47	11:30:00	11:45:00	3.23	1.31	1.63	6.17	13.72
Vankal 1.4.2023.xls	48	11:45:00	12:00:00	6.26	2.46	3.22	11.93	15.89
Vankal 1.4.2023.xls	49	12:00:00	12:15:00	7.89	4.30	4.34	16.52	19.02

Cleaning data

```python
time_pattern = re.compile(r'(\d{2}:\d{2})')      # matches times like 06:15
date_pattern = re.compile(r'Date\s*:\s*(\d{2}/\d{2}/\d{4})', re.IGNORECASE)
number_pattern = re.compile(r'-?\d+\.\d+|-?\d+')

for idx, row in df.iterrows():
    # join row into a single string for searching
    row_str = ' '.join(row.dropna().astype(str).values)
    # detect date line like "Date : 01/04/2023"
    date_match = date_pattern.search(row_str)
    if date_match:
        current_date = date_match.group(1)
        continue
    if current_date is None:
        # skip lines before the first date
        continue
    # try to extract the first time (prefer 'From' time)
    time_match = time_pattern.search(row_str)
    if not time_match:
        # no time on this row — skip
        continue
    time_str = time_match.group(1)  # e.g. '06:15'
    # Extract numeric tokens from the row string
    nums = [float(x) for x in number_pattern.findall(row_str)]
    # Heuristic to find Total MW
    mw_value = None
    if len(nums) >= 2:
        mw_value = nums[-2]
    if mw_value is None or np.isnan(mw_value):
        continue
    # compute energy for this 15-min block (MWh)
    energy_block_mwh = mw_value * 0.25
    # compose full timestamp
    try:
        ts = pd.to_datetime(f"{current_date} {time_str}", dayfirst=True, format="%d/%m/%Y %H:%
    except Exception:
        ts = pd.to_datetime(f"{current_date} {time_str}", dayfirst=True)

    records.append({
        'timestamp': ts,
        'Total_MW': mw_value,
        'Block_Energy_MWh': energy_block_mwh
    })
```

Cleaning data

```python
    # create DataFrame of blocks
    blocks_df = pd.DataFrame(records)
    blocks_df = blocks_df.set_index('timestamp').sort_index()

    # resample to hourly
    hourly = blocks_df.resample('h').agg({
        'Block_Energy_MWh': 'sum',    # total energy produced in that hour (MWh)
        'Total_MW': ['mean', 'max']   # average MW and max MW during that hour
    })

    # flatten MultiIndex columns
    hourly.columns = ['MWh', 'Avg_MW', 'Max_MW']
    hourly = hourly.reset_index()

    # Save individual monthly results
    #hourly.to_csv(output_filename, index=False)
    print(f"Successfully processed and saved '{output_filename}'")

    return hourly

# --- Main script execution ---

# List of months to process
months_to_process = [
    'April', 'May', 'June', 'July', 'August',
    'September', 'October', 'November', 'December'
    ]

all_dataframes = []
for month in months_to_process:
    monthly_df = process_energy_file(month)
    if not monthly_df.empty:
        all_dataframes.append(monthly_df)

# Combine all the monthly data into a single DataFrame
if all_dataframes:
    final_combined_df = pd.concat(all_dataframes, ignore_index=True)

    print("\n--- All months combined successfully! ---")
    final_combined_df.to_csv('all_months_hourly_energy.csv', index=False)
    print("Final combined data saved to 'all_months_hourly_energy.csv'")
    print("\nFinal DataFrame Info:")
    print(final_combined_df.info())
else:
    print("\nNo data was processed. The final DataFrame is empty.")
```

Data Cleaning

- Aggregated 9 months (Apr-Dec 2023) of raw 15-minute solar production logs into a single dataset.
- Standardized park's IST logs and weather's UTC data to a common timezone.
- Upscaled 15-minute production data to hourly averages to match the weather data's granularity.
- Merged the two clean, time-synced datasets (Production + Weather) on the new hourly timestamp.
- Complete Sanity Check. Checked for physically impossible values (e.g., humidity > 100, negative ghi).
- Confirmed all feature data (.describe() table) was within realistic physical bounds.

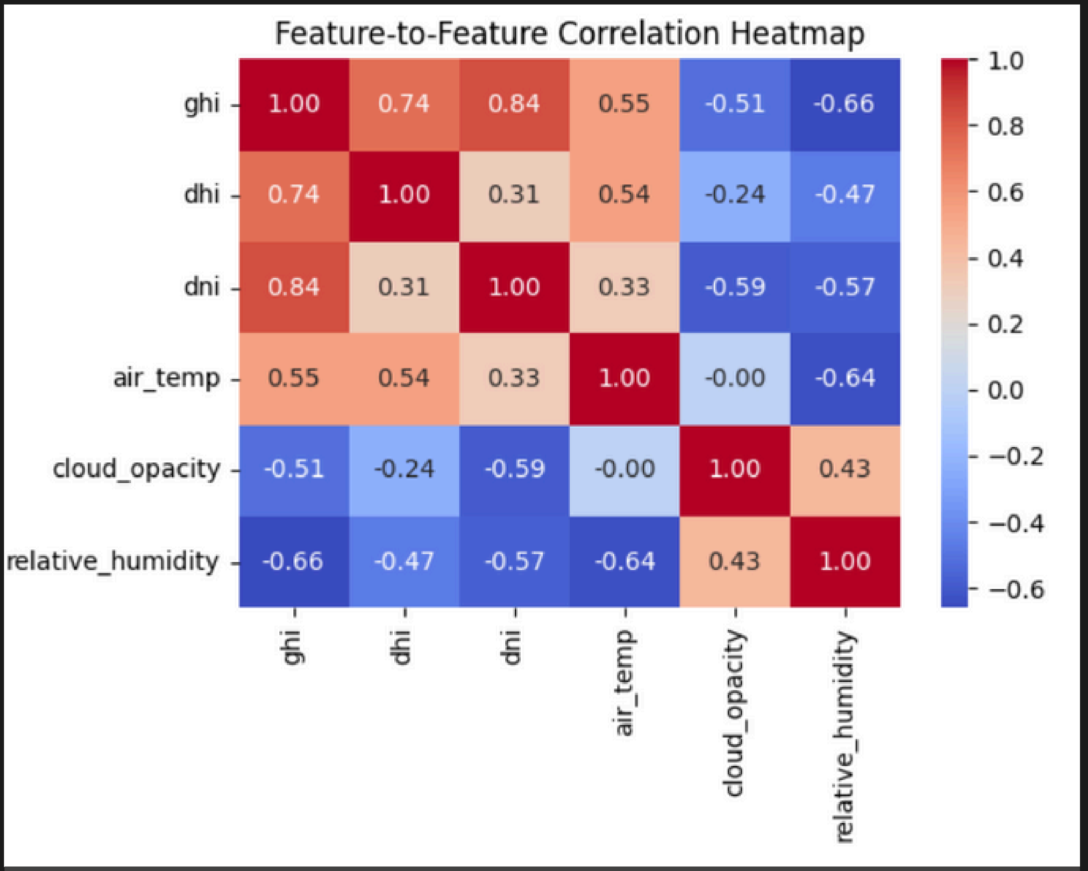
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
air_temp	albedo	clearsky_	clearsky_	clearsky_	clearsky_	cloud_op	dhi	dni	ghi	relative_	min_air_t	max_air_	date	month	time	MWh	Avg_MW	Max_MW
16	0.13	0	0	0	0	23	0	0	0	87.9	14.8	25.1	02-04-2023	4	00:00:00	0	0	0
15	0.13	0	0	0	0	13.7	0	0	0	87.9	14.8	25.1	02-04-2023	4	01:00:00	0	0	0
15	0.13	0	0	0	0	17.9	0	0	0	87	14.8	25.1	02-04-2023	4	02:00:00	0	0	0
15	0.13	0	0	0	0	27.9	0	0	0	83.4	14.8	25.1	02-04-2023	4	03:00:00	0	0	0
15	0.13	0	0	0	0	26.5	0	0	0	80.1	14.8	25.1	02-04-2023	4	04:00:00	0	0	0
16	0.13	3	1	3	3	39.6	2	0	2	77.6	14.8	25.1	02-04-2023	4	05:00:00	0.278	0.278	0.762
16	0.13	66	222	114	203	59.4	43	0	43	78.4	14.8	25.1	02-04-2023	4	06:00:00	2.1345	2.1345	4.286
17	0.13	122	532	344	677	63.6	117	0	117	80.9	14.8	25.1	02-04-2023	4	07:00:00	2.41775	2.41775	3.109
18	0.13	149	702	579	880	76.2	141	0	141	78.2	14.8	25.1	02-04-2023	4	08:00:00	5.4335	5.4335	8.272
17	0.13	172	775	771	945	10.2	261	553	699	74.2	14.8	25.1	02-04-2023	4	09:00:00	16.8163	16.8163	17.319
19	0.13	161	855	917	985	14.8	353	485	782	63.5	14.8	25.1	02-04-2023	4	10:00:00	16.156	16.156	17.71
22	0.13	155	889	988	996	21.5	438	360	776	53.9	14.8	25.1	02-04-2023	4	11:00:00	10.4162	10.4162	16.711
22	0.13	165	874	976	994	29.9	502	196	685	50.6	14.8	25.1	02-04-2023	4	12:00:00	5.323	5.323	8.84
24	0.13	162	850	892	989	23.7	385	344	680	46.4	14.8	25.1	02-04-2023	4	13:00:00	5.3925	5.3925	7.359
25	0.13	151	803	740	965	61.9	287	0	287	44.4	14.8	25.1	02-04-2023	4	14:00:00	3.4775	3.4775	4.019
24	0.13	132	715	534	896	46.5	249	53	276	47.1	14.8	25.1	02-04-2023	4	15:00:00	5.80725	5.80725	10.333
23	0.13	100	547	295	633	4.3	115	467	284	51.6	14.8	25.1	02-04-2023	4	16:00:00	4.0195	4.0195	5.979
22	0.13	41	197	74	134	24	44	37	48	56.7	14.8	25.1	02-04-2023	4	17:00:00	0.42175	0.42175	1.246
21	0.13	0	0	0	0	40.4	0	0	0	61.3	14.8	25.1	02-04-2023	4	18:00:00	0	0	0
21	0.13	0	0	0	0	38.2	0	0	0	66.1	14.8	25.1	02-04-2023	4	19:00:00	0	0	0
19	0.13	0	0	0	0	17.9	0	0	0	70.6	14.8	25.1	02-04-2023	4	20:00:00	0	0	0
18	0.13	0	0	0	0	31.4	0	0	0	72	14.8	25.1	02-04-2023	4	21:00:00	0	0	0
17	0.13	0	0	0	0	47.1	0	0	0	75.9	14.8	25.1	02-04-2023	4	22:00:00	0	0	0
16	0.13	0	0	0	0	69.2	0	0	0	83	14.8	25.1	02-04-2023	4	23:00:00	0	0	0
16	0.13	0	0	0	0	69.1	0	0	0	88.8	14.1	24.1	03-04-2023	4	00:00:00	0	0	0
15	0.13	0	0	0	0	29.8	0	0	0	92.8	14.1	24.1	03-04-2023	4	01:00:00	0	0	0
14	0.13	0	0	0	0	50.1	0	0	0	95	14.1	24.1	03-04-2023	4	02:00:00	0	0	0
15	0.13	0	0	0	0	65.7	0	0	0	93.4	14.1	24.1	03-04-2023	4	03:00:00	0	0	0
15	0.13	0	0	0	0	16.2	0	0	0	90.4	14.1	24.1	03-04-2023	4	04:00:00	0	0	0
15	0.13	3	5	4	4	16.7	3	0	3	88.7	14.1	24.1	03-04-2023	4	05:00:00	1.084	1.084	2.094
15	0.13	63	294	125	240	52.8	51	8	52	87.3	14.1	24.1	03-04-2023	4	06:00:00	1.55525	1.55525	3.488
17	0.13	119	560	353	703	88	43	0	43	81.5	14.1	24.1	03-04-2023	4	07:00:00	3.23575	3.23575	4.379
19	0.13	153	694	581	877	79.5	116	0	116	72.3	14.1	24.1	03-04-2023	4	08:00:00	5.50025	5.50025	11.613
20	0.13	169	778	774	947	77.9	167	0	167	66	14.1	24.1	03-04-2023	4	09:00:00	5.22725	5.22725	11.119
22	0.13	179	824	910	975	44.8	250	289	514	56.1	14.1	24.1	03-04-2023	4	10:00:00	16.5345	16.5345	17.469

Final dataset

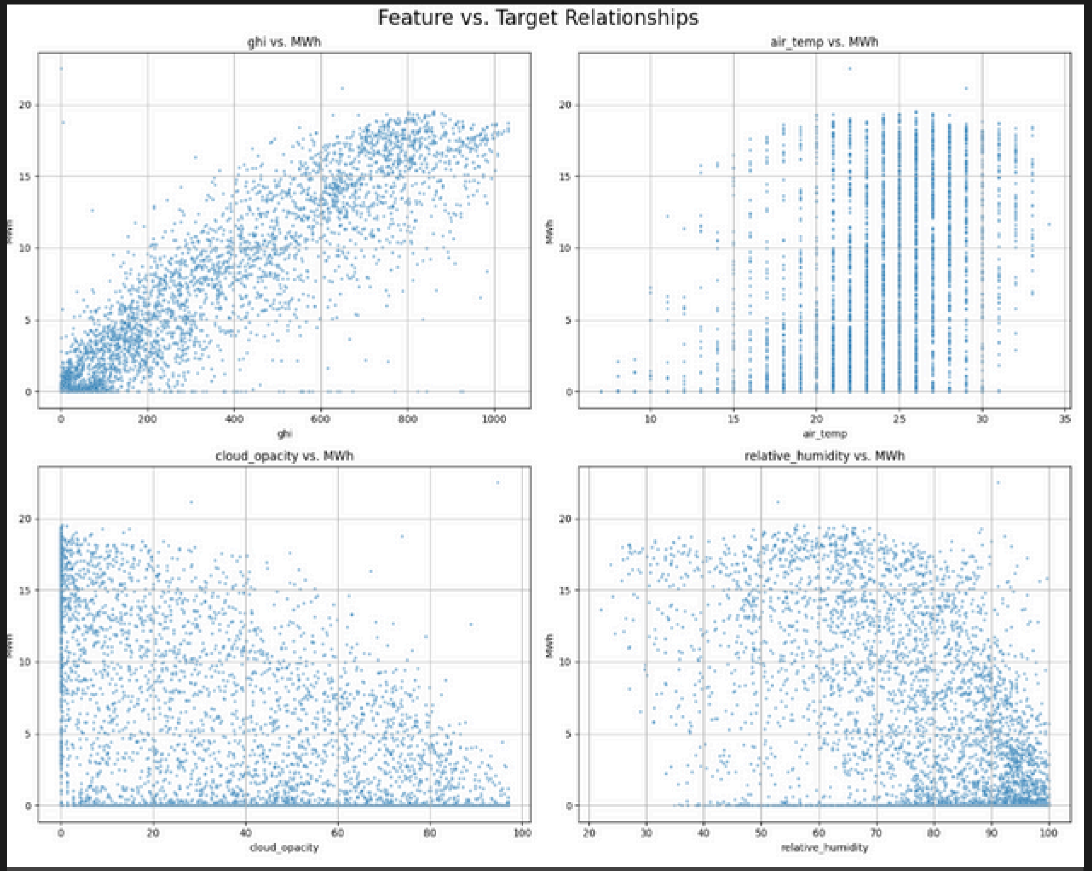
Feature Engineering

```
MWh                  1.000000
Avg_MW               0.998994
Max_MW               0.983705
ghi                  0.933388
clearsky_ghi         0.857391
clearsky_dni         0.829833
clearsky_gti         0.826177
dni                  0.804911
clearsky_dhi         0.717035
dhi                  0.695938
air_temp             0.449341
max_air_temp         0.106725
month               -0.012404
albedo              -0.025858
min_air_temp        -0.044655
cloud_opacity       -0.473979
relative_humidity   -0.582096
Name: MWh, dtype: float64
```

TARGET CORRELATION ANALYSIS.

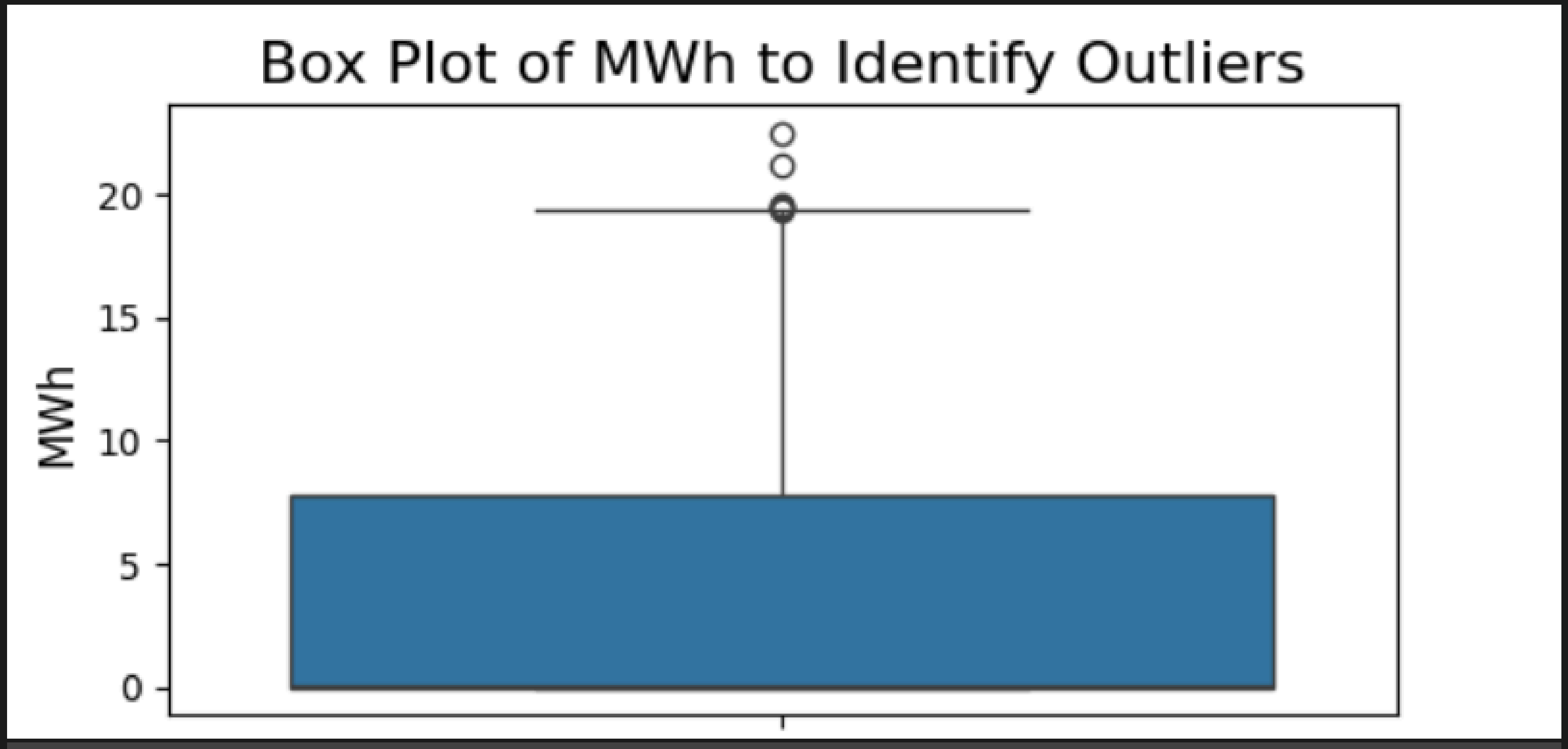


MULTICOLLINEARITY ANALYSIS



SCATTERPLOT OF FINAL FEATURES WITH TARGET VARIABLE

Feature Engineering



COUNT	6558.000000
MEAN	4.130625
STD	5.883984
MIN	0.000000
25%	0.000000
50%	0.082150
75%	7.743325
MAX	22.500000

THE OUTLIER BOUNDARY IS: 19.3583 MWH
THERE ARE 6 OUTLIERS OUT OF 6558 TOTAL ROWS.

- **Finding**: Statistical analysis (1.5 * IQR rule) identified 6 data points as "upper outliers" (values > 19.36 MWh).
- **Investigation:** These high values were cross-referenced with their corresponding weather data.
- **Conclusion:** These outliers are NOT errors or bad data. They are real, valid, peak-production events—representing the sunniest, clearest, and most productive hours in the dataset.
- **Decision: KEEP ALL 6 OUTLIERS.**
- **Reasons:** The model must learn what conditions lead to peak output. Removing these points would prevent the model from ever predicting a high-production day.

Model Training And Evaluation

- First, the feature matrix (X) was created by dropping the target variable (MWh) and all non-numeric date/time columns. The target vector (y) was then set as the MWh column.
- This dataset was then split into an 80% training set and a 20% test set to prepare for modeling.
- StandardScaler was fit only on the training dataset to avoid data leakage and to normalize the feature scales before model training
- Finally, a range of models (Linear Regression, Decision Tree, etc.) were trained on this scaled data and evaluated using R^2 score, MAE, and RMSE to measure their performance.

```
X = merged.drop(['MWh', 'date', 'time', 'month'], axis=1)
y=merged['MWh']

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)

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)
from sklearn.linear_model import LinearRegression
lr_model=LinearRegression()
lr_model.fit(X_train_scaled,y_train)
y_pred_lr=lr_model.predict(X_test_scaled)


from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

r2=r2_score(y_test,y_pred_lr)
mae=mean_absolute_error(y_test,y_pred_lr)
mse=mean_squared_error(y_test,y_pred_lr)
rmse=np.sqrt(mse)
```

- A comparative evaluation of all models was conducted. The Random Forest, once optimized with GridSearchCV, was identified as the champion model, yielding the highest predictive accuracy by achieving the top R^2 score and the lowest MAE.
This final, optimized model was then saved as a .pkl file for future use in live predictions.

```
from sklearn.ensemble import RandomForestRegressor
rf_model=RandomForestRegressor(random_state=42)
rf_model.fit(X_train_scaled,y_train)
y_pred_rf=rf_model.predict(X_test_scaled)

from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100, 200],          # How many trees in the forest
    'max_depth': [None, 10, 20],         # How deep each tree can go
    'min_samples_leaf': [1, 2, 4]        # Min samples required at a leaf node
}
grid_search=GridSearchCV(estimator=RandomForestRegressor(random_state=42),param_grid=param_grid,
                         cv=10,n_jobs=-1,scoring='neg_mean_squared_error',verbose=1)
grid_search.fit(X_train_scaled,y_train)
print(f'Best Params found : {grid_search.best_params_}')
```

```
Best Params found : {'max_depth': 10, 'min_samples_leaf': 4, 'n_estimators': 100}
```

```
best_rf_model=grid_search.best_estimator_
y_pred_best_rf=best_rf_model.predict(X_test_scaled)

r2=r2_score(y_test,y_pred_best_rf)
mae=mean_absolute_error(y_test,y_pred_best_rf)
mse=mean_squared_error(y_test,y_pred_best_rf)
rmse=np.sqrt(mse)
print("\n--- Tuned Random Forest Performance ---")
print(f'R2 Score : {r2:.4f}')
print(f"Mean Absolute Error : {mae:.4f}")
print(f"Root Mean Squared Error : {rmse:.4f} MWh")
```

```
--- Tuned Random Forest Performance ---
R2 Score : 0.9072
Mean Absolute Error : 0.8808
Root Mean Squared Error : 1.7879 MWh
```

Model Training And Evaluation

Model deployment

- **Load Tools:** The system first loads the three essential "production-ready" files:
  1. The saved solar_prediction_model.pkl
  2. The saved scaler.pkl
  3. The model_columns.json (to ensure column order)

- **Fetch Live Data:** A script calls the Solcast API in real-time to get the next 24 hours of weather forecast data for the Mizoram park's exact coordinates.

- **Prepare Features:** The raw JSON data from the API is processed:
  1. Timestamps are converted from UTC to the local timezone (IST).
  2. Data is formatted to exactly match the feature columns the model was trained on.

- **Scale & Predict:**
  1. The loaded scaler is used to .transform() the new weather data.
  2. The scaled data is fed into the model.predict() function to get the MWh forecasts.

- Clean & Display: The result is a clean, 24-hour forecast table, which can be saved to a CSV

	Local_Time	Predicted_MWh
0	2025-10-30 16:00	1.8527
1	2025-10-30 17:00	0.0155
2	2025-10-30 18:00	0.0107
3	2025-10-30 19:00	0.0101
4	2025-10-30 20:00	0.0093
5	2025-10-30 21:00	0.0361
6	2025-10-30 22:00	0.0289
7	2025-10-30 23:00	0.0040
8	2025-10-31 00:00	0.0182
9	2025-10-31 01:00	0.0042
10	2025-10-31 02:00	0.0044
11	2025-10-31 03:00	0.0044
12	2025-10-31 04:00	0.0044
13	2025-10-31 05:00	0.0044
14	2025-10-31 06:00	2.5741
15	2025-10-31 07:00	5.4134
16	2025-10-31 08:00	10.6889
17	2025-10-31 09:00	13.3560
18	2025-10-31 10:00	14.3939
19	2025-10-31 11:00	13.9482
20	2025-10-31 12:00	12.6150
21	2025-10-31 13:00	10.5143
22	2025-10-31 14:00	7.9699

Conclusion

In conclusion, this project built a successful forecasting tool for the Mizoram P&ED that turns unpredictable solar power into a reliable asset. This model directly saves money by reducing costly emergency power purchases. The Tuned Random Forest is a strong baseline, and the future roadmap is clear.

The immediate next step is to re-validate all models using a TimeSeriesSplit to get a true, robust performance score that prevents any data leakage. Following that, engineering cyclical (sin/cos) features for time of day will further enhance the model's accuracy, delivering even greater financial savings.