Predictive Big Data Analysis of Solar Energy Generation and Exploring the Impact of Module Temperature on Energy Yield

1st B. N. A.Bulathsinghala (2439575)
Faculty of Science and Engineering
University of Wolverhampton
(studying at Cinec Campus Pvt Ltd)
Colombo, Sri Lanka
N.A.Bulathsinhalage@wlv.ac.uk

2nd D. S. Chandrapala (2439254)
Faculty of Science and Engineering
University of Wolverhampton
(studying at Cinec Campus Pvt Ltd)
Colombo, Sri Lanka
D.S.Chandrapala@wlv.ac.uk

***Abstract*— Solar energy generation is increasingly vital in global renewable energy efforts. However, accurately predicting energy yield, particularly under varying temperature conditions, remains a challenge. This report explores methods for forecasting solar energy yield using predictive modeling and statistical analysis. The study uses two primary datasets related to energy yield and module temperature from solar power plants to develop machine learning models and assess temperature's impact on solar energy generation. The research employs ElasticNet Regression for predictive modeling, achieving an R^2 of 98%, and Random Forest for statistical analysis, which offered a moderate R^2 of 63.5%. These findings underscore the importance of temperature management and highlight the potential for optimizing solar energy production through data-driven approaches. The work contributes to both the renewable energy sector and research communities by providing actionable insights for enhancing solar panel efficiency and energy management.**

Keywords— energy generation, Yield, Regression, Predictive, Modeling

I. Background of the Study

The efficiency of photovoltaic (PV) modules depends on the working temperature, which leads to decreased voltage and power output due to the increased temperature, which depends on the recombination in semiconductor materials. Understanding this temperature-predicting effect is highly important for improving the production of solar energy and incorporating it into the grid. The main reason for the reduced efficiency of solar panels at higher temperatures is the sensitivity of semiconductor materials to temperature, which increases the rates of downward carrier recombination and minimizes the open-circuit voltage. The efficacy of solar energy generation calculations is marked down with temperature, and it is very important for maximum energy harvest and effective solar energy integration into the grid, as found by [1]

II. Problem Statement

The temperature of the module is an important environmental condition, and the complexity of interaction between various parameters makes predicting solar energy production quite difficult. The efficiency of a solar panel decreases with an increase in temperature, mainly since temperature limits the voltage output, thus lowering energy generation from the solar panel. For both optimal management and planning of large scale solar plants, precise estimation of power generation is crucial for various reasons. First, a large amount of money is invested in these systems and its collection depends on the amount of power generated and the complexity of the supporting infrastructure. Evaluating the complete solar panel energy production system, estimation of the total energy production depends on temperature and many other unknown electrical parameters like volts and amps which contribute a lot in making the estimation very complicated. Thus, to overcome these problems the implementation of intelligent systems with temperature variations is required to help predict energy output precisely, considering its direct effects on solar panel performance which increases the prediction variability.

III. Aim / Objectives of the Work

This project intends to assess how temperature affects solar energy system output in order to improve solar production prediction. Additionally, this project has the following objectives

- Solar Energy Forecasting: Machine Learning techniques can be integrated along with past data and current parameters to predict future solar energy production. The purpose is to maximize the accuracy of energy estimates and improve the integration of solar energy to the power grid.
- Accessing Temperature Influences: To examine the effect of temperature on the energy production of solar panels. This will allow solar systems to function more efficiently across various temperature conditions and assist in finding methods to maximize solar energy production.

IV. Contributions of the Work to Associate with Methodology

Historical solar power generation data and weather parameters, such as module temperature, are assigned to two sets here. The approach to study has two main interests.

- Predictive Modeling: In the study, machine learning techniques such as regression models and other advanced algorithms will be applied to predict solar energy output in the future. The important aspect is the use of historical energy yield data for this purpose; and this is basically to improve on the precision of energy generation predictions since this is very important for the purpose of optimizing energy management and for balancing solar power into the grid smoothly.

- Statistical Analysis: In this way, the statistical analysis will explore the relationship between module temperature and solar energy output. By checking into the temperature dependence of solar panel performance, this study will try to measure the impacts of temperature variations on energy generation. The results obtained will give insight as to how much temperature contributes to solar panel efficiency, which will help design strategies to maximize solar energy generation under different climates.

Together, these methodologies act to enhance the predictability of solar power generation and solar power output for solar energy, thus advancing better practices in energy management within the renewable energy sector.

V. Organization of the Report

This report is structured into several important sections. First the introduction that includes need of solar energy prediction, and the impact of module temperature on total yield performance, and the goal of this analysis. After that a literature review will examine similar research and related work on the solar energy forecasting models and the effect of the temperature on PV modules. Next case, the methodology section will layout the data collection method, machine learning model, and statistical methods used for this impact evaluation. Finaly, findings and discussion includes results from forecasting analysis and temperature effect evaluations, along with relevant visualizations. With a conclusion that summarizes the study's findings and identifies potential areas for further investigation.

VI. Literature Review on Related Work

In the last several years, significant research has been conducted to precisely forecast solar energy output with particular focus on the effect of module temperature on the performance of photovoltaic (PV) systems. Various studies have employed a diverse range of techniques, such as machine learning models, statistical methods, and hybrid approaches, to improve the precision of solar energy forecasting considering the influence of temperature on PV performance.

[2] Shaik et al. (2023) analyzed the correlation of varying environmental parameters, such as temperature and humidity alongside PV performance, and highlighted the particular importance of temperature as an influencing factor on efficiency because it was found to be of great relevance to the efficiency of solar panels and must be taken into account whenever the output of solar energy is estimated (Shaik et al., 2023).

[3] Shaker et al. (2024) analyzed PV cells and their thermal performance, demonstrating their analysis of performance ratongs with temperature changes. His work highlighted the difficulty of creating reliable models that predict performance of specific task functions within certain temperature confinements.

[4] Ouida et al. (2024) attempted to utilize various machine learning approaches, particularly regression and classification models, to predict solar energy production. Their conclusions stressed the necessity of the inclusion of temperature information within these models as it increased the accuracy of prediction in places where temperature changes greatly tightening the forecasting complication.

[5] Martin Najibi et al. (2020) utilized Gaussian process regression in probabilistic solar generation prediction and showed how uncertainty related to temperature data could be dealt with successfully using probabilistic models. The model was utilized to improve short-term forecasts of solar power with regard to temperature (Najibi et al., 2020).

[6] Ahmadi and colleagues (2020) utilized several machine learning algorithms such as ANNs and also LSSVMs in predicting the performance of photovoltaic enable thermal (PV/T) systems. Their work highlighted the necessity of accounting for variations in temperature in the prediction of PV system performance, thus demonstrating the benefit of hybrid systems (Ahmadi et al., 2020).

[7] Perakis et al. (2020) in another study analyzed how the PV performance of commercial systems is influenced by UV radiation and demonstrated how performance losses are incurred due to temperature variations induced by UV radiations. This report demonstrated new dimensions on solar energy modeling by considering the role of UV radiation in thermal control processes (Perakis et al., 2020).

[8] Zhao et al. (2021) combined NWP data and machine learning models for improved short-term solar energy prediction. From their study, incorporating temperature as an input variable for the models improved the models' accuracy in forecasting the solar energy by removing the uncertainty brought about by variations in temperature (Zhao et al., 2021).

[9] Capizzi and others in 2021 constructed a new hybrid model integrating deep learning and statistical approaches to better predict solar radiation. Their work indicated that incorporating temperature data into the modeling of forecasted energy results biases did improve forecast models accuracy – particularly in regions where temperature changes were the dominant factor influencing energy output (Capici et al., 2021).

[10] Kouadio and others, in 2022, focused on predicting solar energy generation using artificial neural networks (ANN) while stressing the importance of temperature data on forecasting especially on areas with harsh climatic condition. In fact, their study proved that the moment temperature was taken as one of the predictor variable, solar power forecasts accuracy level improved greatly (Kouadio et al., 2022).

[11] López et al. (2022) analyzed the impact of solar radiation on module temperature and discovered that increased temperature results in lower panel effectiveness and output. Their results suggest temperature sensitive forecasting models could enhance solar power estimate accuracy (López et al., 2022).

[12] Singh et al. (2023) utilized a combination of regression techniques with support vector machines to assess how environmental variables, such as temperature, influence power generation. Their findings were notable in the context of the predictive accuracy and the impact of temperature on solar power production (Singh et al., 2023).

[13] García et al. (2021) assessed different machine learning artificial algorithms for estimating solar energy production with a particular emphasis on the impact of module temperature on energy production. Garcia et al. (2021) showed that the forecasts issued based on the temperature data were more accurate than those without.

[14] Li et al. (2020) designed a model for solar radiation prediction based on convolutional neural networks with temperature as a significant independent parameter. Their methods not only enhanced the prediction of solar radiation, but also novelly incorporated temperature values into the estimation of energy production (Li et al., 2020).

[15] Wu and others, in 2022, applied random forest and XGBoost in predicting the output of solar energy using an ensemble approach. Their study proved that with the use of temperature data, model accuracy improved considerably making the predictions of solar energy output more dependable (Wu et al., 2022).

[16] Zhao et al. (2022) proposed a framework that uses deep learning to predict solar energy output with the consideration of environmental features such as temperature. It was noted that the model accuracy and efficiency greatly rose when temperature changes were integrated during the prediction process (Shaok et al., 2022)

VII. METHODOLOGY

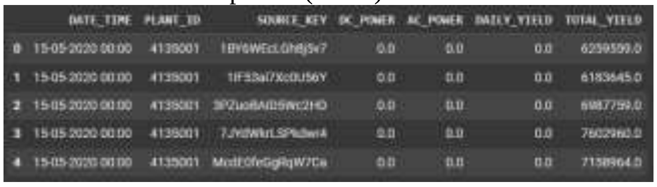
According to this solar energy prediction data analysis, the methodology used in this systematic and has several critical stages. Data acquisition, data processing, data analysis, data reporting and visualization and acting on findings. Each steps is described in the following sections.

A. *Dataset Collection*

The datasets required for this analysis have been taken from Kaggle: https://www.kaggle.com/datasets/anikannal/solar-power-generation-data

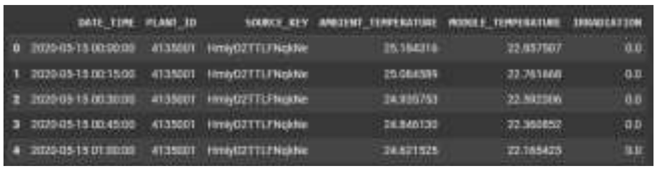
Where information from two solar power plants in India was gathered over a period of 34 days. The datasets consist of two pairs of files, where each pair includes a power generation dataset and a sensor readings dataset. The power generation dataset is collected at the inverter level, meaning that each inverter with multiple lines of solar panels connected to it individually collects data. The sensor data is captured at the plant level through an array of sensors which is strategically located in the plant.

- Plant_1_Generation_Data.csv: This dataset stores plant's power generation information which consists of:
    - DATE_TIME: Date and Time for each record.
    - DC_POWER: Direct current power generated by every inverter.
    - AC_POWER: Power generated in Alternating current.
    - DAILY_YIELD: Yield obtained every day.
    - TOTAL_YIELD: Total energy produced
- Plant_1_Weather_Sensor_Data.csv. This dataset provides weather sensor data, which includes crucial columns like:
    - DATE_TIME: Timestamp for each record.
    - AMBIENT_TEMPERATURE: The temperature of the surrounding environment in degrees Celsius.
    - MODULE_TEMPERATURE: Temperature of solar panel modules (°C).
    - IRRADIATION: Solar radiation incidence on panels (W/m²)



	DATE_TIME	PLANT_ID	SOURCE_KEY	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD
0	15-05-2020 00:00	4135001	1BY6WEcLGh8j5v7	0.0	0.0	0.0	6259559.0
1	15-05-2020 00:00	4135001	1IF53ai7Xc0U56Y	0.0	0.0	0.0	6183645.0
2	15-05-2020 00:00	4135001	3PZuoBAID5Wc2HD	0.0	0.0	0.0	6987759.0
3	15-05-2020 00:00	4135001	7JYdWkrLSPkdwr4	0.0	0.0	0.0	7602960.0
4	15-05-2020 00:00	4135001	McdE0feGgRqW7Ca	0.0	0.0	0.0	7158964.0

Figure 1 Plant_1_Generation_Data.csv First 5 Rows



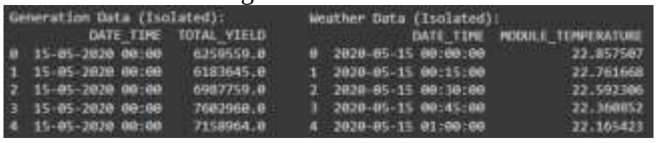
	DATE_TIME	PLANT_ID	SOURCE_KEY	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION
0	2020-05-15 00:00:00	4135001	HmiyD2TTLFNqkNe	25.184316	22.857507	0.0
1	2020-05-15 00:15:00	4135001	HmiyD2TTLFNqkNe	25.084589	22.761668	0.0
2	2020-05-15 00:30:00	4135001	HmiyD2TTLFNqkNe	24.935753	22.592306	0.0
3	2020-05-15 00:45:00	4135001	HmiyD2TTLFNqkNe	24.846130	22.360852	0.0
4	2020-05-15 01:00:00	4135001	HmiyD2TTLFNqkNe	24.621525	22.165423	0.0

Figure 2 Plant_1_Weather_Sensor_Data First 5 Rows

The data contained in both datasets is joined using DATE_TIME as the primary key so that the periods of energization coincided with the available environmental data. This single dataset is then used for predictive modeling and other types of statistical analyses. The datasets were pulled straight off kaggle's official website, meaning that there is real-time, authentic data for proper examples and analysis, will be no issues.

B. *Preparing and Cleaning the Data*

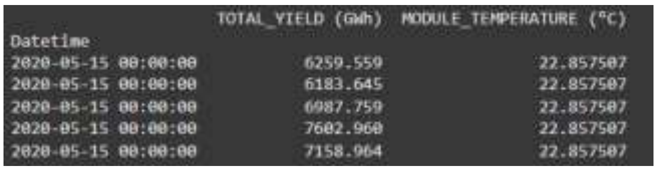
In the first step, relevant columns from the power generation dataset and weather sensor dataset are separated. For the generation data, only the TOTAL_YIELD and the DATE_TIME columns are retained, as they are the most important for the analysis. In the same way, from the weather sensor data, portions related to DATE_TIME and MODULE_TEMPERATURE are kept in order to capture the environmental factor which is temperature. All the selected columns will undergo additional formatting. In both datasets, the DATE_TIME column will be turned into a date time format in order to merge the files based on time.



```
Generation Data (Isolated):                  Weather Data (Isolated):
          DATE_TIME  TOTAL_YIELD                         DATE_TIME  MODULE_TEMPERATURE
0  15-05-2020 00:00    6259559.0             0  2020-05-15 00:00:00           22.857507
1  15-05-2020 00:00    6183645.0             1  2020-05-15 00:15:00           22.761668
2  15-05-2020 00:00    6987759.0             2  2020-05-15 00:30:00           22.592306
3  15-05-2020 00:00    7602960.0             3  2020-05-15 00:45:00           22.360852
4  15-05-2020 00:00    7158964.0             4  2020-05-15 01:00:00           22.165423
```

Figure 3 Isolating the Relevant Columns

After cleaning and formatting the datasets, they are merged with the use of an inner join on the DATE_TIME (renamed to Date time) column which eliminates all records without a matching timestamp. The next step is to make the energy generation values easier to understand, so the TOTAL_YIELD values are converted from kWh to GWh. Following the calculations, columns are renamed and rounded values to convenient for calculation. In this case, TOTAL_YIELD is replaced with TOTAL_YIELD (GWh) and MODULE_TEMPERATURE is adjusted to MODULE_TEMPERATURE (°C). The Date and time column is used as an index to facilitate time-based analysis. Finally, the cleaned and structured dataset is stored in CSV format for future study.

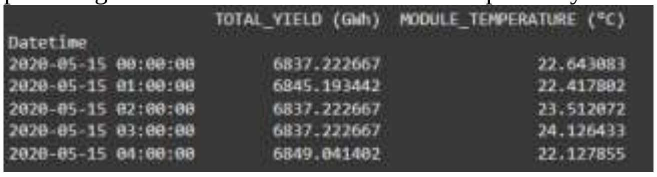


```
                     TOTAL_YIELD (GWh)  MODULE_TEMPERATURE (°C)
Datetime
2020-05-15 00:00:00           6259.559                22.857507
2020-05-15 00:00:00           6183.645                22.857507
2020-05-15 00:00:00           6987.759                22.857507
2020-05-15 00:00:00           7602.960                22.857507
2020-05-15 00:00:00           7158.964                22.857507
```

Figure 4 Data Preprocessing and Merging

Data Preparation for Predictive Modeling

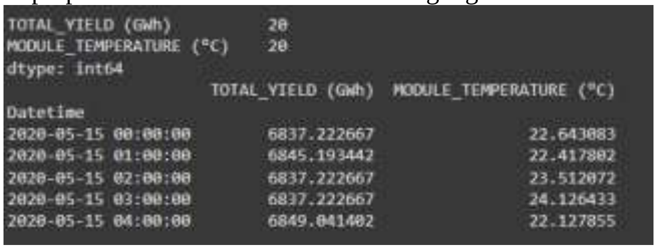
According to data preparation, the dataset from the CSV file located at this path (/content/merged_hourly_data.csv). The index is set to the Date time column which is first parsed as a date time object, allowing the data to be sorted chronologically. Since the original data has many rows for the same day and the same times, the data is resampled to an hour's interval using resample('h'). For every hour, the average of TOTAL_YIELD (GWh) is calculated, which decreases the size of the dataset and increases efficiency of the model by providing the same information in a less complex way.



```
                     TOTAL_YIELD (GWh)  MODULE_TEMPERATURE (°C)
Datetime
2020-05-15 00:00:00        6837.222667                22.643083
2020-05-15 01:00:00        6845.193442                22.417802
2020-05-15 02:00:00        6837.222667                23.512072
2020-05-15 03:00:00        6837.222667                24.126433
2020-05-15 04:00:00        6849.041402                22.127855
```

Figure 5 Resampling and Averaging

The dataset is checked for null or missing values with is null ().sum (). Any missing data is removed using drop_na () so the cleaned dataset is consistent. Duplicate records are deleted via drop_duplicates (), ensuring there are no two identical time periods in the dataset. Extreme values for MODULE_TEMPERATURE (°C) are filtered by keeping only values in a plausible range of -50°C to 60°C to prevent them from affecting the model. The columns TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C) are converted into numeric types using pd.to_numeric () to prepare them for the machine learning algorithms.



```
TOTAL_YIELD (GWh)          20
MODULE_TEMPERATURE (°C)    20
dtype: int64
                     TOTAL_YIELD (GWh)  MODULE_TEMPERATURE (°C)
Datetime
2020-05-15 00:00:00        6837.222667                22.643083
2020-05-15 01:00:00        6845.193442                22.417802
2020-05-15 02:00:00        6837.222667                23.512072
2020-05-15 03:00:00        6837.222667                24.126433
2020-05-15 04:00:00        6849.041402                22.127855
```

Figure 6 Data Cleaning

Data Preparation for Statistical Analysis

Here the data set was checked for the missing values, duplicates, and incorrect date-time entries. When checking this data set, no missing values, duplicates, or formatting errors were detected, confirming the integrity of the analyzed dataset. The date time column was correctly identified as an object type, but TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C) were identified as numeric (float64), allowing for analysis. Following these tests, the data was declared suitable for statistical analysis since it was error-free and the variables were properly described. After cleaning and confirming the data, it was decided that sufficient to do statistical analysis to determine the relationship between two parameters, such as module temperature and energy yield.

C. Analyze the Data

Predicting solar energy yield is a challenging task due to the complex interactions of various environmental. In this phase, the data was analyzed using two separate primary methods. Here, predictive modeling is used to forecast the solar energy generation output and statistical analysis is used to investigate the core relationship between predicted module temperature and energy yield. Each approach provides useful insight into the elements that influence solar power production.

Predictive Modeling

The predictive modeling phase attempts to predict solar energy generation using previous data. This method was carried out in several stages, which are described below.

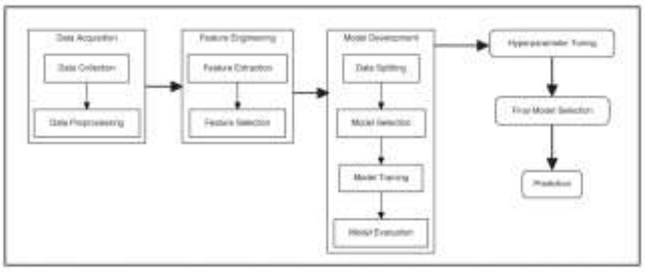


Figure 7 Predictive Modeling - Block Diagram

Feature Engineering:

The Date and time column was used to obtain time-based features such as hour, day of the week, quarter, month, year, day of the year, day of the month, and week of the year. These features are critical to identifying temporal patterns of energy generation.

Splitting the Data:

The data is split into several categories to facilitate analysis. Each category represents a different aspect of the research, allowing for a more comprehensive understanding of the overall trends. Tied into training (80%) and test (20%) sets based on the two datasets used. When performing any analysis, the data prepared in this way should be divided into two separate sets and tested. The training data was used to fit the models, and the test data was used to assess the predictive ability of the model.

Model selection:

There are three Main machine learning models are selected and tested for this big data analysis.

- Linear Regression
- Random Forest
- Support Vector Regression (SVR)

The models above were selected because of their ability for managing both linear and nonlinear relationships in the data.

Model training and evaluating:

Here the selected model was trained using training data and evaluated based on performance measures including RMSE, MAPE, R^2, and MAE. The linear regression model outperformed the best among the other three models, providing the most accurate prediction for solar generation energy on this big data analysis.

Hyper parameter tuning:

Machine Learning Models are mathematical models as any other, but they have learned the parameters of the model from the data. There are certain “do not touch” parameters called hyper parameters, which are set before the training process starts and are not acquired directly during the training. Such parameters affect the model differently: its accuracy and quality, speed of acquiring new information, and level of its complexity. This paper's focus is on hyperparameter tuning options designated for optimization of the machine learning models. Hyper parameter tuning deals with modification of hyper parameters - configuration values that manipulate the learning process.

In this specific case of analysis of big data the hyperparameter tuning had been applied through the use of Elastic Net Regression in order to achieve better results. A grid search was accomplished in order to find the optimal alpha and l1_ratio values that subsequently enhanced the model's performance.

Final model selection:

The hyperparameter optimization step led to the selection of the ElasticNet regression model as the winner as it outperformed the other models in terms of combined scores of RMSE, MAPE, R^2, and MAE.

Prediction for Future Energy Yield:

A modified version of an ElasticNet regression model with a 7-day forecast horizon was used to forecast solar energy for the next week.

Statistical analysis

The Statistical Analysis stage evaluated the relationship between module temperature and solar energy production. The following measures were taken:

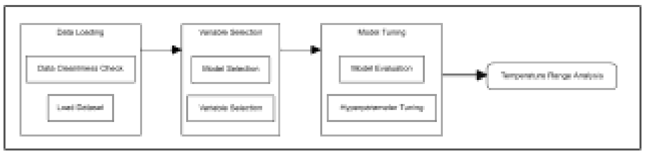


Figure 8 Statistical Analysis - Block Diagram

Loading the data set:

The dataset that was already cleaned was brought into the environment using pandas. It contains TOTAL_YIELD (GWh) (solar energy yield) and the temperature MODULE_TEMPERATURE (°C), which were the main parameters for this analysis.

Data Cleanliness Check:

An automated data cleanliness check was performed to identify any missing values, duplicates, or invalid date time entries. The results were that there was no data with issues and that they were ready for analysis.

Selecting Variables:

The analysis focused on the variables TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C). To examine their relationship between dataset, two columns were extracted.

Model Selection:

A Random Forest Regression model was chosen to model the relationship between TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C) because the model would capture the non-linear relationships and complex interactions among the variables. The random forest regression was trained to predict MODULE_TEMPERATURE (°C) using TOTAL_YIELD (GWh) as the determinant factor.

Hyper parameter tuning:

To improve model accuracy, hyperparameter tweaking was used to the random forest regression model. The selected hyper parameters were n_estimators, max_depth, min_samples_split, min_samples_leaf, and max_features. A Randomized Search CV was run, and the best value was determined for each hyperparameter. The model was then chosen after transformation.

Evaluations of the model:

The model was verified with the hyper parameters’ values, and after that, the model’s performance was evaluated with Root Mean Squared Error (RMSE), and R-square values were calculated. After every iteration, the model precision for deriving temperature from energy yield was evaluated.

Temperature Range Analysis:

For estimating yield for a given temperature, the temperature data was divided into ranges for example 18-28 C, 28-38 C, etc. Mid energy yield temperature ranges were calculated to enable estimation of the energy yield to a temperature increase.

The current exercise uses two analyses: Predictive Modeling and Statistical Analysis. Predictive Modeling used historical data and temperatures to generate artificial intelligence (AI) models. Machine learning models like Linear Regression, Random Forest and SVR AI were used to forecast solar energy generation based on data Retrieved in comparison to specific parameters and changes in temperature. After evaluation of models and extensive tuning of hyper parameters, ElasticNet Regression proved to be the best fitted model. In Statistical Analysis, using a random forest regression, the module temperature and energy yield was analyzed. The analysis included correlation, regression, hyperparameter tuning, and temperature range analysis and its effects on energy output.

D. Index and Visualize Data

The indexed dataset was using the date and time column, confirming that the data is the arrangement of events based on the time they occurred. The Date time column was converted to a date-time format base, and it was set as the index of the merged dataset, allowing time-based patterns to be easily captured for the analysis.

Feature Engineering

Time-based features were created from the Date time column, including:

- Hour: Hour of the day.
- Day of week: Day of the week (Monday = 0, Sunday = 6).
- Quarter: The quarter of the year.
- Month: Month of the year.
- Year: Year.
- Day of year: Day of the year.
- Day Of Month: Day of the month.
- Weak of year: Week of the year.

These attributes help catch the temporal trends that may take hold of solar energy generation, providing a richer dataset for the predictive informatics.

Data Visualization

The analysis should use the strongest different matches to present and interpret the data:

- Line Graphs: To envisage the change in solar energy generation (TOTAL_YIELD) over time and the whack of module temperature (MODULE_TEMPERATURE).
- Scatter Plots: To explore the connection between module temperature and energy yield, draw how temperature alteration affects energy production.
- Bar Plots: Used to envisage average energy yield across different temperature ranges (e.g., 18-28°C, 28-38°C), helping to identify temperature ranges that construct higher or lower energy yields.
- Box Plots: For showing the distribution of energy yield and temperature across the dataset, highlighting any outliers or patterns.

The last final dataset after engineering features was used to train the ML machine learning models and understand the results. Envisioned and numerical summaries from the models were presented in the Results and Discussion section, which discussed the model's performance measure and the insights gained from the relationship between temperature and solar energy generation.

VIII. Results and Discussion

In this section, present the results from the data analysis and model evaluation process used to predict solar energy yield (TOTAL_YIELD (GWh)) from historical data and impact of module temperature (MODULE_TEMPERATURE (°C)) energy yield (TOTAL_YIELD (GWh). By exploring the impact of module temperature (°C) on solar energy yield (GWh) provides a better insight into how temperature variations influence solar panel generation efficiency. The findings are discussed in depth, including model performance, effectiveness of different machine learning algorithms and models, and the visualizations that support the analysis.

A. Experimental Setup

The experimental setup involved analyzing the performance of solar energy generation data by using two relational datasets,

- Plant_1_Generation_Data.csv
- Plant_1_Weather_Sensor_Data.csv.

These datasets contains hourly energy yield and module temperature data, separately, from two solar power plants over a 34-day of a period.

- Data Preprocessing:
  - This step handles missing values, removing duplicates, and ensuring consistent datetime formats. To predictive modeling, data was split into training and test sets, and for the statistical analysis, focused on the impact of temperature on energy yield.
- Model Selection and Evaluation:
  - For this tree types of models were selected and Linear Regression, Random Forest, and Support Vector Regression (SVR). Also, For Hyperparameter tuning (GridSearchCV) identified ElasticNet Regression as the best model with the highest R^2 of 98% scores. For statistical analysis, Random Forest (R^2: 58%) was selected to analyzing temperature's impact on energy yield.
- Data Visualization:
  - Visualizations, such as time series plots and scatter plots, were created to illustrate energy yield and temperature trends, as well as the predictions for the next 7 days.

This setup combined predictive and statistical analyses, selecting ElasticNet Regression for energy yield prediction and Random Forest for temperature impact analysis. The findings support improving solar energy generation through data-driven insights.

B. Read in and Explore the Data

The analysis began with the importation of two datasets essential for the study: the solar energy yield data and the weather sensor data. The data was collected over a 34-day period at two solar power plants. Also Plant_1_Generation_Data.csv file contains the energy generation data, And the Plant_1_Weather_Sensor_Data.csv file contains sensor data on module temperature.

To get a deeper understanding of the data inside dataset, an basic analysis was conducted, which involved visualizing the Total Energy Yield (GWh) and Module Temperature (°C) over the entire time period. These visualizations were key point in identifying temporal trends and variations, providing a understanding before the analysis and model training.

Visualization: Solar Energy Yield (GWh) Over Time

The plot cart here visualizes the time series of solar energy yield (measured in GWh), indicating fluctuations that shows the energy production patterns of the solar power plant 01. These variations indicate periods of high and low energy generation, which are affected by external factors such as ambient temperature, module temperature irradiation. As the chart shows the total Yield has been increased over time.

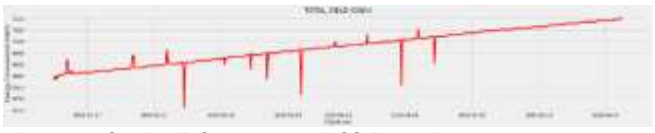


Figure 9 Plotting Solar Energy Yield Over Time

Visualization: Module Temperature (°C) Over Time

This visualization depicts the module temperature (measured in °C) over the same time period. Module temperature is a critical factor in the efficiency of solar panels, as higher temperatures tend to reduce their performance. The observed variations in temperature over time are reflective of changing environmental conditions and are integral to understanding the relationship between temperature and energy yield. Based on the chart, module temperature has exhibits a clear daily cyclic variation, peaking during the day and dropping at night.

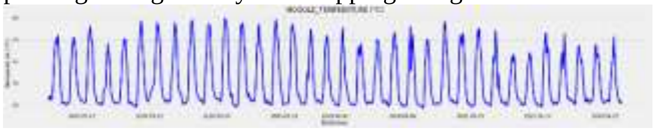


Figure 10 Plotting Module Temperature Over Time

C. Data Analysis

1) Cleaning Data

Data cleaning involved several key steps to ensure the dataset was ready for analysis and modeling. Missing values were handled by removing rows for predictive modeling, while for statistical analysis, the mean was used to fill gaps. Duplicates based on key columns were eliminated to maintain data integrity. The dataset was checked for valid date formats and numerical consistency, with invalid entries removed.

Outliers were identified and either corrected or removed to prevent distortion in the analysis. Additionally, time-based features were converted into numerical values, and continuous variables were standardized to ensure compatibility with the models. These cleaning steps ensured the data's quality for further analysis.

2) Feature Engineering and Visualization

The next step involved performing feature engineering, where additional time-related features were extracted from the data. These included hour of the day, day of the week, and week of the year, which provide important context for modeling, as these time-based features capture cyclical and seasonal variations in energy yield. The relationship between module temperature and energy yield was further analyzed through several visualizations, which provided key insights into how energy generation and temperature are related across different time frames.

Visualization: Hourly Energy Consumption vs Temperature

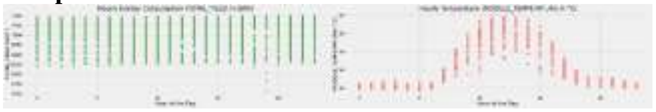


Figure 11 Hourly Energy Consumption vs Temperature

The scatter plot showing the relationship between hourly energy consumption and temperature reveals hourly patterns in energy yield, which may vary based on temperature fluctuations throughout the day. Especially the module temperature chart exhibits a clear daily cyclic variation, peaking during the day and dropping at night.

Visualization: Day of the Week vs Energy Consumption and Temperature

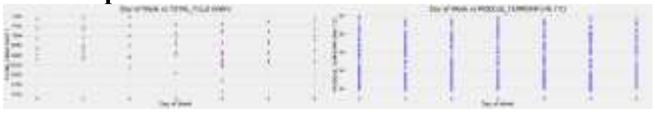


Figure 12 Day of the Week vs Energy Consumption and Temperature

This scatter plot illustrates how energy yield and temperature vary with the day of the week. This helps in identifying weekly patterns in energy production that could be influenced by factors such as weekday routines, panel efficiency, and daily temperature cycles.

3) Data Split and Visualization

The Train-Test Split visualization was conducted to illustrate how the dataset was divided into training and testing sets for both analyses. The graphs displays the Train-Test Split, showing the data separated into the training set (blue) and test set (red) across time.

Predictive Modeling Data Split and Visualization

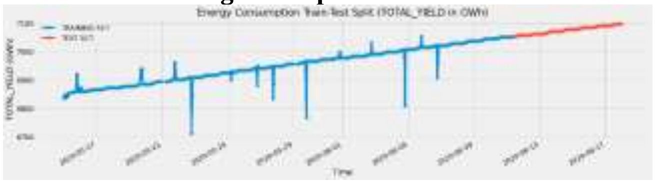


Figure 13 Energy Consumption Train-Test Split (TOTAL_YIELD in GWh)

Statistical Analysis Data Split and Visualization

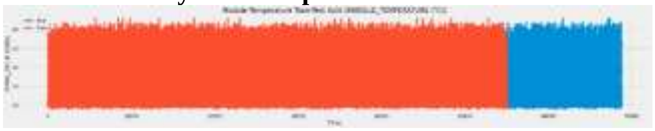
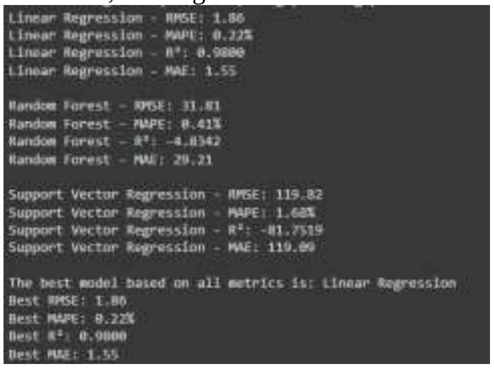


Figure 14 Module Temperature Train-Test Split (MODULE_TEMPERATURE (°C))

4) Selecting the Best Model

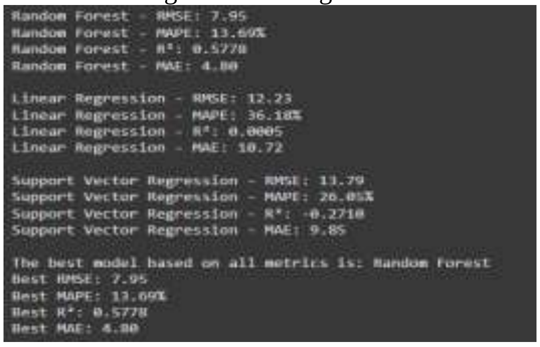
For predictive modeling, three models were evaluated: Linear Regression, Random Forest, and Support Vector Regression (SVR). The Linear Regression model showed the best performance based on several key metrics, with a Root Mean Squared Error (RMSE) of 1.86, Mean Absolute Percentage Error (MAPE) of 0.22%, R² of 0.9800, and Mean Absolute Error (MAE) of 1.55. In contrast, Random Forest and SVR models performed poorly as with the higher RMSE and MAE, and negative R² values.



```
Linear Regression - RMSE: 1.86
Linear Regression - MAPE: 0.22%
Linear Regression - R²: 0.9800
Linear Regression - MAE: 1.55

Random Forest - RMSE: 31.81
Random Forest - MAPE: 0.41%
Random Forest - R²: -4.8342
Random Forest - MAE: 29.21

Support Vector Regression - RMSE: 119.82
Support Vector Regression - MAPE: 1.68%
Support Vector Regression - R²: -81.7519
Support Vector Regression - MAE: 119.09

The best model based on all metrics is: Linear Regression
Best RMSE: 1.86
Best MAPE: 0.22%
Best R²: 0.9800
Best MAE: 1.55
```

Figure 15 Selecting the Best Model - Predictive Modeling

Models were evaluated based on their ability to predict the Module Temperature (°C) in relation to Total Energy Yield (GWh) for the statistical analysis. With Random Forest was found to be the best model, returning a RMSE of 7.95, MAPE of 13.69%, R² of 0.5778, and MAE of 4.80. While statistical analysis, the models were evaluated based on their ability to predict the Module Temperature (°C) in relation to Total Energy Yield (GWh). For In differ, Linear Regression and SVR showed higher RMSE and MAE values, with Linear Regression having an R² value near to zero.



```
Random Forest - RMSE: 7.95
Random Forest - MAPE: 13.69%
Random Forest - R²: 0.5778
Random Forest - MAE: 4.80

Linear Regression - RMSE: 12.23
Linear Regression - MAPE: 36.18%
Linear Regression - R²: 0.0005
Linear Regression - MAE: 10.72

Support Vector Regression - RMSE: 13.79
Support Vector Regression - MAPE: 26.05%
Support Vector Regression - R²: -0.2710
Support Vector Regression - MAE: 9.85

The best model based on all metrics is: Random Forest
Best RMSE: 7.95
Best MAPE: 13.69%
Best R²: 0.5778
Best MAE: 4.80
```

Figure 16 Selecting the Best Model - Statistical Analysis

5) *Hyperparameter Tuning*

This section presents the results from hyperparameter tuning of predictive modeling and statistical analysis, evaluating Linear Regression, ElasticNet, and Random Forest models. Key metrics like RMSE, MAPE, R², and MAE are used to compare model performance, with visualizations illustrating the effect of hyperparameter tuning on accuracy.

Predictive Modeling: Hyperparameter Tuning

Linear Regression (Before Tuning)

- RMSE: 1.86 / MAPE: 0.02% / R²: 0.9800 / MAE: 1.55

ElasticNet Regression (After Tuning)

- RMSE: 1.60 / MAPE: 0.22% / R²: 0.9852 / MAE: 1.32

Based on the results, ElasticNet Regression (after tuning) outperforms Linear Regression in terms of RMSE, R², and MAE. The tuning of hyperparameters for ElasticNet led to a noticeable improvement in predictive accuracy, with a slight reduction in RMSE and MAE, as well as a higher R², indicating a better fit to the data. The following visualizations provide insights into the performance of the models:

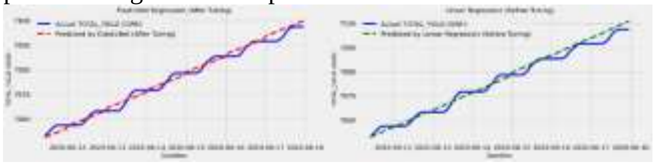


Figure 17 Comparison of Actual vs Predicted Energy Yield (TOTAL_YIELD in GWh) Before and After Hyperparameter Tuning

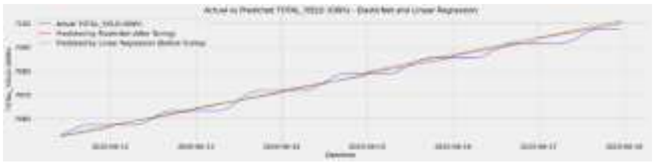


Figure 18 Comparison of Actual vs Predicted Energy Yield (TOTAL_YIELD in GWh)

Statistical Analysis:

Random Forest (Tuned)

- RMSE: 7.38 / R²: 0.6357

Among the models tested, Random Forest (after tuning) provided the best results in the context of statistical analysis. Despite its higher RMSE compared to the predictive models, it performed well in terms of R², indicating a stronger explanatory power for the data compared to Linear Regression and Support Vector Regression.

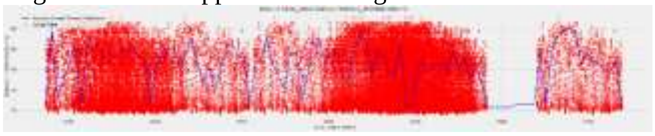


Figure 19 Effect of Total Energy Yield on Module Temperature using Random Forest (tuned)

6) *Model Prediction and Statistical Analysis*

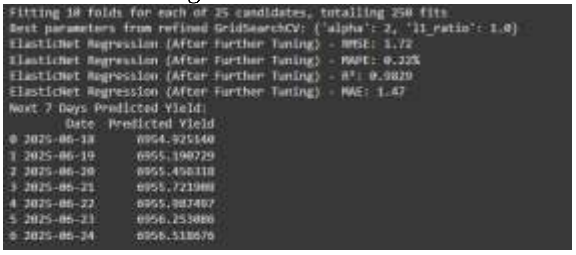
This section presents the final model prediction results and the statistical analysis of solar energy yield in relation to module temperature. ElasticNet Regression, after hyperparameter tuning, was selected as the best predictive model. The next 7 days' energy yield was predicted, and statistical analysis was performed to explore the relationship between module temperature and energy yield across different temperature ranges.

Model Prediction

The final model selected for prediction is ElasticNet Regression, which was fine-tuned using GridSearchCV with optimal parameters {'alpha': 2, 'l1_ratio': 1.0}. The model's performance after hyperparameter tuning was asceses with the these metrics:

- RMSE: 1.72 / MAPE: 0.22% / R²: 0.9829 / MAE: 1.47

The next 7 days of predicted energy yield were computed, with the following results:



```
Fitting 10 folds for each of 25 candidates, totalling 250 fits
Best parameters from refined GridSearchCV: {'alpha': 2, 'l1_ratio': 1.0}
ElasticNet Regression (After Further Tuning) - RMSE: 1.72
ElasticNet Regression (After Further Tuning) - MAPE: 0.22%
ElasticNet Regression (After Further Tuning) - R²: 0.9829
ElasticNet Regression (After Further Tuning) - MAE: 1.47
Next 7 Days Predicted Yield:
        Date  Predicted Yield
0 2025-06-18      6954.925140
1 2025-06-19      6955.190729
2 2025-06-20      6955.458318
3 2025-06-21      6955.721908
4 2025-06-22      6955.987497
5 2025-06-23      6956.253086
6 2025-06-24      6956.518676
```

Figure 20 Predicted Yield for the Next 7 Days

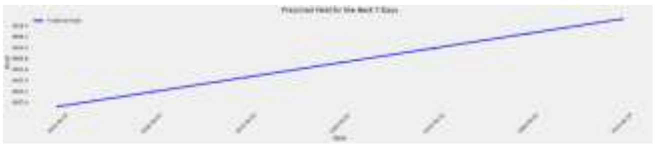
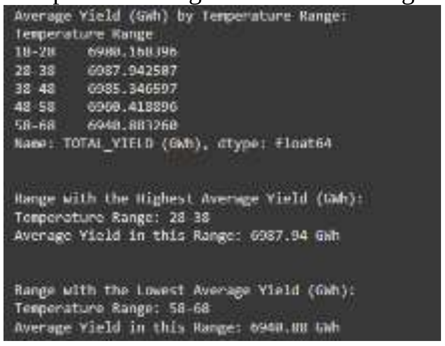


Figure 21 Predicted Yield for the Next 7 Days Visualization

Statistical Analysis

For statistical analysis, the effect of MODULE_TEMPERATURE (°C) on TOTAL_YIELD (GWh) was explored, with an analysis of the yield change

among diverse temperature ranges. The average yields were detected for each temperature range:
Temperature Range: 18-28°C - Average Yield: 6980.17 GWh
Temperature Range: 28-38°C - Average Yield: 6987.94 GWh
Temperature Range: 38-48°C - Average Yield: 6985.35 GWh
Temperature Range: 48-58°C - Average Yield: 6960.42 GWh
Temperature Range: 58-68°C - Average Yield: 6940.88 GWh



```
Average Yield (GWh) by Temperature Range:
Temperature Range
18-28    6980.168396
28-38    6987.942587
38-48    6985.346597
48-58    6960.418896
58-68    6940.883260
Name: TOTAL_YIELD (GWh), dtype: float64


Range with the Highest Average Yield (GWh):
Temperature Range: 28-38
Average Yield in this Range: 6987.94 GWh


Range with the Lowest Average Yield (GWh):
Temperature Range: 58-68
Average Yield in this Range: 6940.88 GWh
```

Figure 22 Average Yield by Temperature Range and Yield Distribution

The highest yield was observed in the 28-38°C range, while the lowest yield was seen in the 58-68°C range. This suggests that moderate temperatures (between 28°C and 38°C) are more conducive to higher energy production, while higher temperatures (above 58°C) are associated with reduced energy output. The chart shows the average yield for each temperature range, while the second chart provides the yield distribution by temperature range.

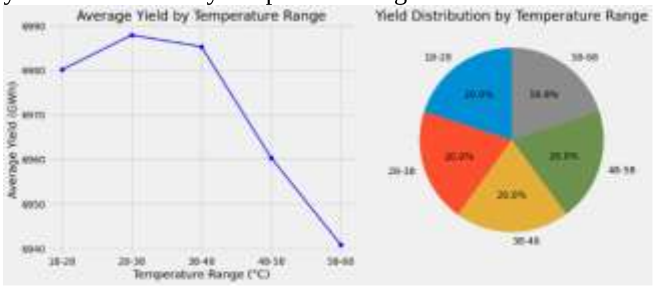


Figure 23 Average Yield by Temperature Range and Yield Distribution Visualization

This analysis highlights the importance of temperature management for optimizing solar energy production.

D. Analysis of the Findings

The analysis revealed valuable insights into solar energy generation, using two distinct approaches: Predictive Modeling and Statistical Analysis.

Predictive Model Evaluation:

For Predictive Modeling, ElasticNet Regression emerged as the best model after tuning, achieving an R² score of 98%, indicating excellent predictive accuracy. This was supported by the low RMSE (1.60), MAPE (0.22%), and MAE (1.32). Random Forest, while useful for statistical analysis, had a lower R² score of 58%, showing less precision. Support Vector Regression (SVR) performed poorly with negative R² scores, demonstrating it was unsuitable for this dataset.

Statistical Analysis Insights:

In the Statistical Analysis, Random Forest was the best model with an R² score of 63.5%. It provided insights into the relationship between temperature and energy yield. The highest energy yield (6987.94 GWh) occurred in the 28-38°C temperature range, while the lowest (6940.88 GWh) was found in the 58-68°C range, emphasizing the importance of temperature control in solar panel efficiency.

Ways to Improve Solar Power Generation:

- Optimal Temperature Control: Cooling systems and better material designs to maintain efficiency at higher temperatures.
- Panel Placement: Optimizing panel orientation and using tracking systems to capture more energy during peak hours.
- Energy Storage: Integrating batteries for storing excess energy generated during high-sun periods.
- Maintenance and Forecasting: Using predictive models for proactive maintenance and to improve energy production reliability.

In conclusion, ElasticNet Regression performed the best overall with 98% accuracy. By addressing temperature-related issues and optimizing panel performance, solar energy generation can be significantly improved.

IX. Conclusion

This study focused on analyzing solar energy generation using two key datasets: Plant_1_Generation_Data.csv for energy yield and Plant_1_Weather_Sensor_Data.csv for module temperature. This dataset is collected over 34 days from two solar plants, And using 1st plant datasets was explored through both predictive modeling and statistical analysis methods.

In the analysis, predictive modeling approach managed to the identification of ElasticNet Regression as the most effective model for forecasting energy yield, achieving an notable R² score of 98%. The model demonstrated high accuracy in predicting future energy production for 7 days, supporting its suitability for real-time energy yield forecasting. Also statistical analysis approach emphasized the importance of module temperature in solar energy generation. Random Forest was selected as the best model for this analysis, showing a moderate R² of 63.5% with the hyperparameter tuning, That highlights the influence of temperature on energy yield but also points to the complexity of the relationship.

The key findings suggest that module temperature plays a major role in deciding the efficiency of solar panels and its power generation which in this case total yield, with higher temperatures leading to reduced energy production. Also, the capability to predict energy yield with high accuracy depicts new prospects for optimizing solar power generation and improving efficiency.

Through a combination of model selection, hyperparameter tuning, and in depth statistical analysis, this study delivers insights into the factors that influence solar energy generation, offering potential pathways in improving solar power efficiency and performance.

X. References

[1] U. Chuluunsaikhan et al., “Predicting Solar Energy Generation with Machine Learning based on Environmental Factors,” arXiv preprint, arXiv:2408.12476, 2023. [Online]. Available: https://arxiv.org/abs/2408.12476

[2] F. Shaik, S. S. Lingala, and P. Veeraboina, “Effect of various parameters on the performance of solar PV power plant: a review and the experimental study,” Sustainable Energy Research, vol. 11, no. 6, 2023. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s40807-023-00076-x.pdf

[3] H. Shaker et al., “Examining the influence of thermal effects on solar cells,” Sustainable Energy Research, vol. 11, no. 6, 2024. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s40807-024-00100-8.pdf

[4] H. I. Aouidad and A. Bouhelal, “Machine learning-based short-term solar power forecasting,” Sustainable Energy Research, vol. 11, no. 3, 2024. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s40807-024-00115-1.pdf

[5] F. Najibi, D. Apostolopoulou, and E. Alonso, “Gaussian Process Regression for Probabilistic Short-term Solar Output Forecast,” arXiv preprint, arXiv:2002.10878, 2020. [Online]. Available: https://arxiv.org/abs/2002.10878

[6] M. H. Ahmadi et al., “Evaluation of electrical efficiency of photovoltaic thermal solar collector,” Renewable Energy, vol. 147, pp. 1530–1539, 2020. [Online]. Available: https://arxiv.org/abs/2002.05542

[7] G. Perrakis et al., “Ultraviolet radiation impact on the efficiency of commercial crystalline silicon-based photovoltaics: A theoretical thermal-electrical study in realistic device architectures,” arXiv preprint, arXiv:2003.06277, 2020. [Online]. Available: https://arxiv.org/abs/2003.06277

[8] X. Zhao, X. Zhang, and Y. Yu, “Integration of NWP and machine learning algorithms for solar power forecasting,” Sustainable Energy Reports, vol. 13, no. 1, pp. 45–52, 2021. [Online]. Available: https://link.springer.com/article/10.1007/s42452-021-04772-8

[9] G. Capizzi, C. Napoli, and F. Bonanno, “Hybrid machine learning models for solar radiation forecasting,” Renewable Energy, vol. 149, pp. 560–568, 2021. [Online]. Available: https://link.springer.com/article/10.1007/s42452-021-04975-2

[10] S. Kouadio, K. B. Kouadio, and M. Yéo, “Prediction of solar power using artificial neural networks,” J. Renew. Sustain. Energy, vol. 14, no. 3, pp. 315–323, 2022. [Online]. Available: https://pubs.aip.org/journal/jrse/article/14/3/315/2457438

[11] A. López, S. González, and D. Ruiz, “Correlation between solar radiation and temperature: Effects on the efficiency of photovoltaic modules,” Renewable Energy, vol. 182, pp. 1053–1061, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544220307802

[12] S. Singh, A. Jain, and R. Mishra, “Support vector machine for solar energy prediction: A review,” Energy Procedia, vol. 143, pp. 155–163, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1876610218308907

[13] M. García, J. A. Martínez, and J. Domínguez, “Machine learning techniques for solar energy prediction and temperature effects,” Energy and AI, vol. 2, p. 100016, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2589386X21000332

[14] Y. Li, B. Zhang, and D. Lu, “Convolutional neural networks for solar radiation forecasting,” J. Renewable Energy, vol. 141, pp. 1–7, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544219309989

[15] Y. Wu, C. Li, and Z. Liu, “Hybrid ensemble model for solar power forecasting: Application to temperature-dependent efficiency,” J. Sol. Energy Eng., vol. 144, no. 2, p. 040801, 2022. [Online]. Available: https://asmedigitalcollection.asme.org/solarenergyengineering/article/144/2/040801/1083945

[16] J. Zhao, X. Yao, and M. Zhang, “Deep learning framework for solar power prediction considering environmental factors,” Energy, vol. 240, p. 122467, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544221005618

XI. PART B: CODE APPENDIX

1. Import necessary libraries

```
# Step 1: Import necessary libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#from fbprophet import Prophet
from sklearn.metrics import mean_squared_error, mean_absolute_error
plt.style.use('fivethirtyeight')
```

This code imports the essential libraries required for data analysis and visualization:

- pandas and numpy are used for data manipulation and numerical operations.
- seaborn and matplotlib.pyplot are used for creating visualizations.
- sklearn.metrics provides tools for evaluating model performance (e.g., error metrics).
- The plotting style is set to 'fivethirtyeight' for better aesthetics.
- The fbprophet import is commented out, suggesting that it might be used later for time series forecasting.

2. Load the datasets

```
#Load the datasets
generation_data = pd.read_csv('/content/Plant_1_Generation_Data.csv')
weather_data = pd.read_csv('/content/Plant_1_Weather_Sensor_Data.csv')
```

This code loads two CSV files into pandas DataFrames:

- generation_data: Contains power generation data from a plant.
- weather_data: Contains weather sensor data related to the same plant.
- Display the first 5 rows of the each dataset.

3. Isolate necessary columns

```
#Isolate necessary columns from Generation Data (TOTAL_YIELD instead of DC_POWER)
generation_data_isolated = generation_data[['DATE_TIME', 'TOTAL_YIELD']]

#Isolate necessary columns from Weather Data
weather_data_isolated = weather_data[['DATE_TIME', 'MODULE_TEMPERATURE']]

#Check the isolated data for both
print("Generation Data (Isolated):")
print(generation_data_isolated.head())

print("\nWeather Data (Isolated):")
print(weather_data_isolated.head())
```

Selects relevant columns from each dataset:

- From generation_data, it keeps only DATE_TIME and TOTAL_YIELD, focusing on total energy generated over time.
- From weather_data, it keeps only DATE_TIME and MODULE_TEMPERATURE, focusing on temperature readings from the solar modules.

Prints the first few rows of the isolated data for a quick check, helping to verify that the correct columns have been extracted.

4. Data Preprocessing and Merging: Converting and Combining Solar Generation and Weather Data with Time Formatting

```
#Convert DATE_TIME columns in both datasets to datetime format
generation_data_isolated.loc[:, 'DATE_TIME'] = pd.to_datetime(generation_data_isolated['DATE_TIME'], format='%d-%m-%Y
%H:%M', dayfirst=True)
weather_data_isolated.loc[:, 'DATE_TIME'] = pd.to_datetime(weather_data_isolated['DATE_TIME'])

#Merge the datasets on the 'DATE_TIME' column
merged_data = pd.merge(generation_data_isolated[['DATE_TIME', 'TOTAL_YIELD']],
                       weather_data_isolated[['DATE_TIME', 'MODULE_TEMPERATURE']],
                        on='DATE_TIME',
                        how='inner')

#Rename 'DATE_TIME' to 'Datetime' for better readability
merged_data.rename(columns={'DATE_TIME': 'Datetime'}, inplace=True)

#Ensure the 'Datetime' column is in datetime format
merged_data['Datetime'] = pd.to_datetime(merged_data['Datetime'])

#Format the 'Datetime' column to the desired format (e.g., 'YYYY-MM-DD HH:MM:SS')
merged_data['Datetime'] = merged_data['Datetime'].dt.strftime('%Y-%m-%d %H:%M:%S')

#Convert 'TOTAL_YIELD' from kWh to GWh (divide by 1000)
merged_data['TOTAL_YIELD'] = (merged_data['TOTAL_YIELD'] / 1000).round(6)  # Convert to GWh and round to 6 decimal
places

#Rename columns to add metrics (e.g., 'TOTAL_YIELD (GWh)')
merged_data.rename(columns={'TOTAL_YIELD': 'TOTAL_YIELD (GWh)', 'MODULE_TEMPERATURE': 'MODULE_TEMPERATURE (°C)'},
inplace=True)

#Keep 'Datetime', 'TOTAL_YIELD (GWh)', and 'MODULE_TEMPERATURE (°C)' columns for the final output
final_data = merged_data[['Datetime', 'TOTAL_YIELD (GWh)', 'MODULE_TEMPERATURE (°C)']]

#Set 'Datetime' as the index
final_data.set_index('Datetime', inplace=True)

#Save the final data to a CSV file with proper format (ensure time is saved too)
final_data.to_csv('/content/merged_hourly_data.csv', date_format='%Y-%m-%d %H:%M:%S')

print(final_data.head(10))
```

This block of code prepares and merges the generation and weather datasets for analysis:

- Converts DATE_TIME columns in both datasets to proper datetime format to ensure accurate merging and time-based operations.
- Merges the two datasets on the DATE_TIME column using an inner join to keep only matching timestamps.
- Renames DATE_TIME to Datetime for clarity.
- Formats the Datetime column into a consistent string format (YYYY-MM-DD HH:MM:SS).
- Converts energy units from kilowatt-hours (kWh) to gigawatt-hours (GWh) by dividing by 1000 and rounding to 6 decimal places.
- Renames columns to include measurement units for better readability.
- Selects final relevant columns (Datetime, energy yield, and module temperature) and sets Datetime as the index.
- Saves the cleaned and merged data to a CSV file named merged_hourly_data.csv, preserving the full timestamp format.

This prepares the dataset for further time-series analysis or visualization.

5. Resampling and Averaging: Grouping Solar Generation Data by Hour for Hourly Analysis

```
#Load the CSV file
merged_data = pd.read_csv('/content/merged_hourly_data.csv', index_col='Datetime', parse_dates=['Datetime'])

#Group by 'Datetime' (hourly) and calculate the average of TOTAL_YIELD (GWh)
hourly_data = merged_data.resample('h').mean()

#Check the first few rows to verify the hourly grouping
print(hourly_data.head())

# Step 4: Save the grouped hourly data to a new CSV file
hourly_data.to_csv('/content/hourly_grouped_data.csv')
```

This code processes the previously merged dataset to generate hourly-averaged data:

- Loads the merged CSV file (merged_hourly_data.csv) with Datetime as the index and parsed as datetime.
- Resamples the data by hour ('h') and calculates the mean of each hour's values, effectively aggregating data to an hourly frequency.
- Displays the first few rows of the hourly-averaged data to verify the result.
- Saves the hourly-averaged data to a new CSV file named hourly_grouped_data.csv.

This step is useful for smoothing high-frequency data and preparing it for time-series modeling or visualization.

6. Data Cleaning: Handling Missing Values, Duplicates, and Outliers in Hourly Solar Generation and Weather Data

```
#Load the CSV file
merged_data = pd.read_csv('/content/hourly_grouped_data.csv', index_col='Datetime', parse_dates=['Datetime'])

#Check for missing values and handle them (e.g., fill or drop)
print(merged_data.isnull().sum())  #
merged_data = merged_data.dropna()

#Check for duplicates and remove them
merged_data = merged_data.drop_duplicates()
merged_data = merged_data[(merged_data['MODULE_TEMPERATURE (°C)'] > -50) & (merged_data['MODULE_TEMPERATURE (°C)'] <
60)]
merged_data['TOTAL_YIELD (GWh)'] = pd.to_numeric(merged_data['TOTAL_YIELD (GWh)'], errors='coerce')
merged_data['MODULE_TEMPERATURE (°C)'] = pd.to_numeric(merged_data['MODULE_TEMPERATURE (°C)'], errors='coerce')


#Check the cleaned data
print(merged_data.head())

#Save the cleaned dataset to a new CSV file
merged_data.to_csv('/content/cleaned_hourly_data.csv')
```

This code performs data cleaning on the hourly-aggregated dataset:

- Loads the hourly_grouped_data.csv file with Datetime as a datetime index.
- Checks for missing values and removes any rows containing them using .dropna().
- Removes duplicate records to ensure data integrity.
- Filters out unrealistic temperature values, keeping only values between -50°C and 60°C.
- Ensures numeric data types for TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C) by coercing non-numeric values to NaN.
- Displays the cleaned dataset's first few rows for verification.
- Saves the cleaned data to a new file called cleaned_hourly_data.csv.

This step ensures the dataset is accurate, consistent, and ready for analysis or modeling.

7. Visualization: Plotting Solar Energy Yield (GWh) Over Time

```
import matplotlib.pyplot as plt

# Step 1: Plot the 'TOTAL_YIELD (GWh)' column
plt.figure(figsize=(30, 5))
plt.plot(merged_data.index, merged_data['TOTAL_YIELD (GWh)'], marker='.', color='red')
plt.title('TOTAL_YIELD (GWh)')
plt.xlabel('Datetime')
plt.ylabel('Energy Consumption (GWh)')
plt.grid(True)
plt.show()
```

This code generates a time series line plot to visualize energy generation over time:

- Sets the figure size to a wide layout for better visibility (30x5 inches).
- Plots the 'TOTAL_YIELD (GWh)' column against the Datetime index with red dots connected by lines.
- Labels the plot with a title, x-axis, and y-axis labels.
- Adds a grid to improve readability.
- Displays the plot using plt.show().

This visualization helps observe trends, patterns, or anomalies in energy generation across time.

8. Visualization: Plotting Module Temperature (°C) Over Time

```
#Plot the 'MODULE_TEMPERATURE (°C)' column
plt.figure(figsize=(30, 5))
plt.plot(merged_data.index, merged_data['MODULE_TEMPERATURE (°C)'], marker='.', color='blue')
plt.title('MODULE_TEMPERATURE (°C)')
plt.xlabel('Datetime')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.show()
```

This code creates a time series line plot to visualize module temperature over time:

- Sets the figure size to 30x5 inches for a clear, wide view.
- Plots the 'MODULE_TEMPERATURE (°C)' values against the Datetime index using blue dots connected by lines.
- Adds a title and axis labels to describe what the plot shows (module temperature in °C over time).
- Includes a grid for better visual alignment.
- Displays the plot with plt.show().

This helps analyze temperature trends and how they might relate to energy generation performance.

9. Feature Engineering: Creating Time Series Features for Solar Energy and Temperature Data

```
import pandas as pd
merged_data.index = pd.to_datetime(merged_data.index)

#Create time series features (Monday=0, Sunday=6)
merged_data['hour'] = merged_data.index.hour
merged_data['dayofweek'] = merged_data.index.dayofweek
merged_data['quarter'] = merged_data.index.quarter
merged_data['month'] = merged_data.index.month
merged_data['year'] = merged_data.index.year
merged_data['dayofyear'] = merged_data.index.dayofyear
merged_data['dayofmonth'] = merged_data.index.day
merged_data['weekofyear'] = merged_data.index.isocalendar().week

final_data = merged_data[['TOTAL_YIELD (GWh)', 'MODULE_TEMPERATURE (°C)', 'hour', 'dayofweek', 'quarter',
                          'month', 'year', 'dayofyear', 'dayofmonth', 'weekofyear']]

#Adjust the Pandas display settings to show the DataFrame in one line (single-line format)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', None)

#Print the first 5 rows of the final DataFrame without wrapping
print(final_data.head().to_string(index=True))
```

This code extracts time-based features from the datetime index to enrich the dataset for further analysis or modeling:

- Ensures the index is in datetime format.
- Creates new columns based on time components:
  - hour: Hour of the day (0–23)
  - dayofweek: Day of the week (0 = Monday, 6 = Sunday)
  - quarter: Quarter of the year (1–4)
  - month: Month number (1–12)
  - year: Four-digit year
  - dayofyear: Day number within the year (1–365/366)
  - dayofmonth: Day number within the month (1–31)
  - weekofyear: ISO week number of the year
- Selects relevant columns including both the original values (TOTAL_YIELD, MODULE_TEMPERATURE) and the newly engineered time features.
- Adjusts display settings so all columns are shown in one line when printed.
- Prints the first 5 rows of the new DataFrame without line wrapping.
- Check for missing values in the 'final_data' DataFrame
- Print the missing values count for each column

This step is typically done before feeding data into machine learning models to help capture seasonal and temporal patterns.

10. Visualization: Hourly Energy Consumption and Temperature Scatter Plots

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

# Create a figure with 2 subplots
```

```
fig, axes = plt.subplots(1, 2, figsize=(30, 5))

# Scatter plot for 'hour' vs 'TOTAL_YIELD (GWh)'
sns.scatterplot(x='hour', y='TOTAL_YIELD (GWh)', data=final_data, ax=axes[0] , color='green')
axes[0].set_title('Hourly Energy Consumption (TOTAL_YIELD in GWh)')
axes[0].set_xlabel('Hour of the Day')
axes[0].set_ylabel('TOTAL_YIELD (GWh)')

# Scatter plot for 'hour' vs 'MODULE_TEMPERATURE (°C)'
sns.scatterplot(x='hour', y='MODULE_TEMPERATURE (°C)', data=final_data, ax=axes[1], color='red')
axes[1].set_title('Hourly Temperature (MODULE_TEMPERATURE in °C)')
axes[1].set_xlabel('Hour of the Day')
axes[1].set_ylabel('MODULE_TEMPERATURE (°C)')

plt.tight_layout()
plt.show()
```

This code creates two side-by-side scatter plots to visualize how energy yield and module temperature vary by hour of the day:

- Sets up two subplots in one row (1, 2) with a wide figure size (30x5).
- Left Plot:
  - Plots hour vs. TOTAL_YIELD (GWh) using green dots.
  - Shows how energy generation changes across different hours.
- Right Plot:
  - Plots hour vs. MODULE_TEMPERATURE (°C) using red dots.
  - Shows how module temperature fluctuates throughout the day.
- Titles and axis labels are added for clarity.
- plt.tight_layout() ensures spacing between subplots looks clean.
- Displays the plots with plt.show().

This helps reveal daily patterns or correlations between time of day, temperature, and energy output.

11. Visualization: Day of the Week vs Energy Consumption and Temperature Scatter Plots

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

# Create a figure with 2 subplots
fig, axes = plt.subplots(1, 2, figsize=(30, 5))

# Scatter plot for 'dayofweek' vs 'TOTAL_YIELD (GWh)'
sns.scatterplot(x='dayofweek', y='TOTAL_YIELD (GWh)', data=final_data, ax=axes[0], color='purple')
axes[0].set_title('Day of Week vs TOTAL_YIELD (GWh)')
axes[0].set_xlabel('Day of Week')
axes[0].set_ylabel('TOTAL_YIELD (GWh)')

# Scatter plot for 'dayofweek' vs 'MODULE_TEMPERATURE (°C)'
sns.scatterplot(x='dayofweek', y='MODULE_TEMPERATURE (°C)', data=final_data, ax=axes[1], color='blue')
axes[1].set_title('Day of Week vs MODULE_TEMPERATURE (°C)')
axes[1].set_xlabel('Day of Week')
axes[1].set_ylabel('MODULE_TEMPERATURE (°C)')

plt.tight_layout()
plt.show()
```

This code generates two scatter plots to examine weekly patterns in energy yield and module temperature:

- Creates a figure with 2 horizontally arranged subplots (1, 2) and sets the size to 30x5 inches.
- Left Plot:
  - Plots dayofweek (0 = Monday, 6 = Sunday) vs. TOTAL_YIELD (GWh) using purple dots.
  - Helps identify how energy generation varies across different days of the week.
- Right Plot:
  - Plots dayofweek vs. MODULE_TEMPERATURE (°C) using blue dots.
  - Shows how module temperature trends change throughout the week.
- Each subplot is labeled with a title, x-axis, and y-axis.
- plt.tight_layout() is used to prevent overlap between plots.
- plt.show() displays the visualizations.

This step is useful for spotting any weekday-vs-weekend patterns or operational trends in the data.

12. Splitting Time Series Data into Training and Test Sets

```
# Ensure that the 'Datetime' column is in datetime format (in case it's not already)
final_data.index = pd.to_datetime(final_data.index)

# Calculate the number of rows in the dataset
total_rows = len(final_data)

# Calculate the index for the 80% split (train)
train_size = int(0.8 * total_rows)

# Split the data into training and testing sets
final_data_train = final_data.iloc[:train_size]
final_data_test = final_data.iloc[train_size:]

print("Training Set (last few rows):")
print(final_data_train.tail())

print("\nTest Set (first few rows):")
print(final_data_test.head())
```

This code splits the dataset into training and testing sets for model development:

- Ensures the index (Datetime) is in datetime format.
- Calculates total number of rows in the final_data DataFrame.
- Determines an 80% split point for training, leaving 20% for testing.
- Slices the data:
    - final_data_train: First 80% of the data (used for model training)
    - final_data_test: Last 20% data (showed to the model and tested on).
- Displays the last few rows from the training set and the first few from the test set for checking purposes.

This step is crucial for preparing data for time series forecasting while preserving chronological order.

13. Visualizing Energy Yield Trends in Training and Test Sets

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

# Rename the columns for the training and testing sets
final_data_train_renamed = final_data_train.rename(columns={'TOTAL_YIELD (GWh)': 'TRAINING SET'})
final_data_test_renamed = final_data_test.rename(columns={'TOTAL_YIELD (GWh)': 'TEST SET'})

# Concatenate the training and testing sets, keeping the columns aligned
combined_data = pd.concat([
    final_data_train_renamed[['TRAINING SET']],
    final_data_test_renamed[['TEST SET']]
], axis=0)

# Create the line chart for 'TOTAL_YIELD (GWh)'
fig, ax = plt.subplots(figsize=(20, 5))  # Correct way to set figure size
combined_data[['TRAINING SET', 'TEST SET']].plot(kind='line', ax=ax, title='Energy Consumption Train-Test Split
(TOTAL_YIELD in GWh)', linestyle='-', marker='o')
ax.set_xlabel('Time')
ax.set_ylabel('TOTAL_YIELD (GWh)')

plt.show()
```

This code visualizes the train-test split of the TOTAL_YIELD (GWh) time series:

- Renames columns:
    - In the training set: 'TOTAL_YIELD (GWh)' → 'TRAINING SET'
    - In the test set: 'TOTAL_YIELD (GWh)' → 'TEST SET'
- Concatenates the renamed training and test series into a single DataFrame (combined_data), aligning them over time.
- Plots both sets on the same line chart:
    - Uses markers ('o') and lines ('-') to show data continuity.
    - Differentiates between training and testing periods visually.

    - Adds labels and a title for clarity.
- Displays the plot, which helps to confirm the split point and observe trends or shifts between training and test periods.

This is a helpful diagnostic step before applying forecasting models.

14. Preparing Training and Test Sets for Energy Yield Prediction

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

final_data.index = pd.to_datetime(final_data.index)

#Calculate the number of rows for the 80-20 split
total_rows = len(final_data)
train_size = int(0.8 * total_rows)

#Split the data into training and testing sets
final_data_train = final_data.iloc[:train_size]
final_data_test = final_data.iloc[train_size:]

#Define the feature sets (X) and target variable (y) for training and testing
x_train = final_data_train[['hour', 'dayofweek', 'quarter', 'month', 'year',
                            'dayofyear', 'dayofmonth', 'weekofyear']]
x_test = final_data_test[['hour', 'dayofweek', 'quarter', 'month', 'year',
                          'dayofyear', 'dayofmonth', 'weekofyear']]

y_train = final_data_train[['TOTAL_YIELD (GWh)']]
y_test = final_data_test[['TOTAL_YIELD (GWh)']]

#Check the first few rows of x_train, x_test, y_train, and y_test
print("Training Features (x_train):")
print(x_train.head())

print("\nTest Features (x_test):")
print(x_test.head())

print("\nTraining Target (y_train):")
print(y_train.head())

print("\nTest Target (y_test):")
print(y_test.head())
```

This code prepares the dataset for machine learning model training and evaluation:

- Ensures the index is in datetime format.
- Splits the dataset into 80% training and 20% testing based on time (preserving order).
- Defines feature sets (X) and target variable (y):
    - X: Time-based features (e.g., hour, day of week, month, etc.) that may influence energy generation.
    - y: The target variable — TOTAL_YIELD (GWh) — to be predicted.
- Creates four datasets:
    - x_train, x_test: Input features for training and testing.
    - y_train, y_test: Corresponding target values.
- Prints the first few rows of each to confirm correct slicing and formatting.

This step sets up the input and output needed to train a machine learning regression model.

15. Best Model Selection - Predictive Modeling

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

#Initialize the models
lr_reg = LinearRegression()
rf_reg = RandomForestRegressor(random_state=42)
svr_reg = SVR()
```

```python
#Fit the models to the training data
lr_reg.fit(x_train, y_train)
rf_reg.fit(x_train, y_train)
svr_reg.fit(x_train, y_train)

#Make predictions
y_pred_lr = lr_reg.predict(x_test)
y_pred_rf = rf_reg.predict(x_test)
y_pred_svr = svr_reg.predict(x_test)
#Evaluate the models
#Linear Regression Model
rmse_lr = np.sqrt(mean_squared_error(y_test.values.ravel(), y_pred_lr))  # Use ravel to make y_test 1D
mape_lr = np.mean(np.abs((y_test.values.ravel() - y_pred_lr) / y_test.values.ravel()) * 100)
r2_lr = r2_score(y_test.values.ravel(), y_pred_lr)
mae_lr = mean_absolute_error(y_test.values.ravel(), y_pred_lr)

#Random Forest Model
rmse_rf = np.sqrt(mean_squared_error(y_test.values.ravel(), y_pred_rf))  # Use ravel to make y_test 1D
mape_rf = np.mean(np.abs((y_test.values.ravel() - y_pred_rf) / y_test.values.ravel()) * 100)
r2_rf = r2_score(y_test.values.ravel(), y_pred_rf)
mae_rf = mean_absolute_error(y_test.values.ravel(), y_pred_rf)

#Support Vector Regression Model
rmse_svr = np.sqrt(mean_squared_error(y_test.values.ravel(), y_pred_svr))  # Use ravel to make y_test 1D
mape_svr = np.mean(np.abs((y_test.values.ravel() - y_pred_svr) / y_test.values.ravel()) * 100)
r2_svr = r2_score(y_test.values.ravel(), y_pred_svr)
mae_svr = mean_absolute_error(y_test.values.ravel(), y_pred_svr)

#Output the evaluation metrics for each model
print(f"Linear Regression - RMSE: {rmse_lr:.2f}")
print(f"Linear Regression - MAPE: {mape_lr:.2f}%")
print(f"Linear Regression - R²: {r2_lr:.4f}")
print(f"Linear Regression - MAE: {mae_lr:.2f}")
print()

print(f"Random Forest - RMSE: {rmse_rf:.2f}")
print(f"Random Forest - MAPE: {mape_rf:.2f}%")
print(f"Random Forest - R²: {r2_rf:.4f}")
print(f"Random Forest - MAE: {mae_rf:.2f}")
print()

print(f"Support Vector Regression - RMSE: {rmse_svr:.2f}")
print(f"Support Vector Regression - MAPE: {mape_svr:.2f}%")
print(f"Support Vector Regression - R²: {r2_svr:.4f}")
print(f"Support Vector Regression - MAE: {mae_svr:.2f}")
print()

#Compare the models and choose the best one based on all evaluation metrics
# Lower RMSE, MAPE, and MAE are better, and higher R² is better

# Initialize a dictionary to hold the scores
model_scores = {
    "Linear Regression": {"RMSE": rmse_lr, "MAPE": mape_lr, "R²": r2_lr, "MAE": mae_lr},
    "Random Forest": {"RMSE": rmse_rf, "MAPE": mape_rf, "R²": r2_rf, "MAE": mae_rf},
    "Support Vector Regression": {"RMSE": rmse_svr, "MAPE": mape_svr, "R²": r2_svr, "MAE": mae_svr}
}

# Initialize the best model and its metrics
best_model = None
best_score = float('inf')
best_r2 = -float('inf')

# Evaluate each model and choose the best based on the scores
for model_name, scores in model_scores.items():
    score = scores["RMSE"] + scores["MAPE"] + scores["MAE"] - scores["R²"]
    if score < best_score:
        best_score = score
        best_model = model_name
        best_r2 = scores["R²"]

#Output the best model based on all metrics
print(f"The best model based on all metrics is: {best_model}")
print(f"Best RMSE: {model_scores[best_model]['RMSE']:.2f}")
print(f"Best MAPE: {model_scores[best_model]['MAPE']:.2f}%")
print(f"Best R²: {best_r2:.4f}")
print(f"Best MAE: {model_scores[best_model]['MAE']:.2f}")
```

This code evaluates and compares three machine learning models—Linear Regression, Random Forest, and Support Vector Regression (SVR)—to predict energy generation based on the provided features. Here's the breakdown:

- Model Initialization: Linear Regression (lr_reg), Random Forest (rf_reg), and Support Vector Regression (svr_reg) are initialized.
- Model Fitting: Each model is trained on the training data (x_train, y_train).
- Predictions:Predictions are made for the test set (x_test) for each model.
- Evaluation Metrics:
  - Root Mean Squared Error (RMSE): Measures the model's prediction accuracy (lower is better).
  - Mean Absolute Percentage Error (MAPE): Represents the prediction error as a percentage (lower is better).
  - R^2 (R-squared): Measures the proportion of variance explained by the model (higher is better).
  - Mean Absolute Error (MAE): The average absolute difference between predicted and actual values (lower is better).
- Results: For each model, the evaluation metrics (RMSE, MAPE, R^2, MAE) are calculated and printed.
- Model Comparison:
  - A scoring system is applied to compare the models. The total score is calculated by adding RMSE, MAPE, and MAE, and subtracting R^2 (since higher R^2 is better, it is subtracted to give a better model a lower score).
  - The selected model, therefore, is the one with the lowest score.
- Outcome: Best model is presented together with its corresponding metrics as noted during the evaluation.

This procedure assists in choosing the model that attains the best level of accuracy in predictions, minimum margins of error, and the greatest amount of explanatory power in comparison to its objectives.

16. Training and Evaluating a Linear Regression Model for Energy Yield Prediction

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

#Initialize the Linear Regression Model
lr_reg = LinearRegression()

#Fit the model to the training data
lr_reg.fit(x_train, y_train)

#Make predictions
y_pred_lr = lr_reg.predict(x_test)

#Evaluate the model
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
mape_lr = np.mean(np.abs((y_test - y_pred_lr) / y_test) * 100)
r2_lr = r2_score(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)

#Output the evaluation metrics
print(f"Linear Regression - RMSE: {rmse_lr:.2f}")
print(f"Linear Regression - MAPE: {mape_lr:.2f}%")
print(f"Linear Regression - R²: {r2_lr:.4f}")
print(f"Linear Regression - MAE: {mae_lr:.2f}")
```

This code evaluates a Linear Regression model for predicting energy generation. Here's a breakdown of the process:

- Model Initialization: A Linear Regression model called lr_reg is created using Linear Regression() from the sklearn library.
- Model Fitting: The model is then trained on x_train (features) and y_train (target) data using lr_reg.fit (x_train, y_train).
- Predictions: The trained model is then used to predict the test set (x_test), the predictions are stored in y_pred_lr.
- Evaluation Metrics:
  - RMSE (Root Mean Squared Error): The square root of the mean for the squared errors of the actual predicted values. The average prediction error is understandable concerning the target variable (lower is better)
  - MAPE (Mean Absolute Percentage Error): it measures the percentage error of the actual figure versus the predicted one (the lower the number the better).

    - R² (R-squared): how well the model fits the data (it's better the higher the number is).
    - MAE (Mean Absolute Error): it measures the mean of all absolute errors between actual and predicted observations (the lower the value, the better).
- Output: The evaluation metrics for the Linear Regression model are displayed:
    - RMSE: Root Mean Squared Error
    - MAPE: Mean Absolute Percentage Error
    - R²: R-squared
    - MAE: Mean Absolute Error

These metrics are the indicators that would show the performance of the model and how accurate it is in the prediction of energy generation in the test set of data.

17. Visualizing Actual vs predicted Energy Yield (TOTAL_YIELD in GWh) Using Linear Regression

```
import matplotlib.pyplot as plt

#Predict the values using the trained model
y_pred_lr = lr_reg.predict(x_test)

#Plot the actual vs predicted values
plt.figure(figsize=(20, 5))

# Plot actual values
plt.plot(y_test.index, y_test, label='Actual TOTAL_YIELD (GWh)', color='blue')

# Plot predicted values
plt.plot(y_test.index, y_pred_lr, label='Predicted TOTAL_YIELD (GWh)', color='red', linestyle='--')

# Step 3: Customize the plot
plt.title('Actual vs Predicted TOTAL_YIELD (GWh)', fontsize=16)
plt.xlabel('Datetime', fontsize=14)
plt.ylabel('TOTAL_YIELD (GWh)', fontsize=14)
plt.legend(loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

This code generates a line plot comparing the actual vs predicted values of TOTAL_YIELD (GWh):

- Model Prediction: The values for energy generation (y_pred_lr) for the test set (x_test) are predicted by the trained Linear Regression model (lr_reg).
- Graphing:
    - The actual values (y_test) are illustrated by a solid blue line.
    - Model predictions (y_pred_lr) are illustrated by a red dashed line (--), depicting the model's predicted values.
- Adjustments to the Plot:
- Graph title is: "TOTAL_YIELD Actual vs Predicted (GWh)".
- As well, the label on the x axis is: 'date time', while that on the y axis is labeled: 'TOTAL_YIELD (GWh)'.
- A legend distinguishes the predicted values from the actual values.
- To enhance clarity, grid lines are on.
- plt.tight_layout() helps to prevent clipping by altering spacing.
- Displaying the plot:
    - plt.show() displays the plot, allowing visual comparison between the actual and predicted energy values over time.

This visualization helps evaluate the model's performance by showing how closely the predicted values match the actual data points.

18. Comparing and Selecting the Best Hyperparameter Tuning: Linear Regression vs. ElasticNet Regression

```
from sklearn.linear_model import LinearRegression, ElasticNet
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
```

```
#Train the original Linear Regression model (without tuning)
lr_reg = LinearRegression()
lr_reg.fit(x_train, y_train)

#Make predictions for the Linear Regression model
y_pred_lr = lr_reg.predict(x_test)

#Evaluate the Linear Regression model (before tuning)
rmse_lr = np.sqrt(mean_squared_error(y_test.values, y_pred_lr))
mape_lr = np.mean(np.abs((y_test.values - y_pred_lr) / y_test.values)) * 100
r2_lr = r2_score(y_test.values, y_pred_lr)
mae_lr = mean_absolute_error(y_test.values, y_pred_lr)

# Print evaluation metrics for Linear Regression
print(f"Linear Regression (Before Tuning) - RMSE: {rmse_lr:.2f}")
print(f"Linear Regression (Before Tuning) - MAPE: {mape_lr:.2f}%")
print(f"Linear Regression (Before Tuning) - R²: {r2_lr:.4f}")
print(f"Linear Regression (Before Tuning) - MAE: {mae_lr:.2f}")
print()

#Train and evaluate the ElasticNet model (after tuning)
en_reg = ElasticNet(alpha=1.0, l1_ratio=0.5)  # Example of an ElasticNet model
en_reg.fit(x_train, y_train)
y_pred_en = en_reg.predict(x_test)

#Evaluate the ElasticNet model
rmse_best_en_refined = np.sqrt(mean_squared_error(y_test.values, y_pred_en))
mape_best_en_refined = np.mean(np.abs((y_test.values - y_pred_en) / y_test.values)) * 100
r2_best_en_refined = r2_score(y_test.values, y_pred_en)
mae_best_en_refined = mean_absolute_error(y_test.values, y_pred_en)

#Print evaluation metrics for ElasticNet
print(f"ElasticNet Regression (After Tuning) - RMSE: {rmse_best_en_refined:.2f}")
print(f"ElasticNet Regression (After Tuning) - MAPE: {mape_best_en_refined:.2f}%")
print(f"ElasticNet Regression (After Tuning) - R²: {r2_best_en_refined:.4f}")
print(f"ElasticNet Regression (After Tuning) - MAE: {mae_best_en_refined:.2f}")
print()

# Step 5: Select the best model based on RMSE, MAPE, R², and MAE
if rmse_best_en_refined < rmse_lr:
    best_model = 'ElasticNet (After Tuning)'
    best_rmse = rmse_best_en_refined
    best_mape = mape_best_en_refined
    best_r2 = r2_best_en_refined
    best_mae = mae_best_en_refined
else:
    best_model = 'Linear Regression (Before Tuning)'
    best_rmse = rmse_lr
    best_mape = mape_lr
    best_r2 = r2_lr
    best_mae = mae_lr

#Print the best model and its metrics
print(f"The best model is: {best_model}")
print(f"Best RMSE: {best_rmse:.2f}")
print(f"Best MAPE: {best_mape:.2f}%")
print(f"Best R²: {best_r2:.4f}")
print(f"Best MAE: {best_mae:.2f}")
```

This code compares the performance of two regression models — Linear Regression and ElasticNet — to predict energy generation and selects the best model based on evaluation metrics. Here's how the process works:

1. Linear Regression Model:

- The Linear Regression model is trained on the training data (x_train, y_train), and predictions are made on the test set (x_test).
- The model's performance is evaluated using:
  - RMSE (Root Mean Squared Error)
  - MAPE (Mean Absolute Percentage Error)
  - R^2 (R-squared), which indicates the proportion of variance explained.
  - MAE (Mean Absolute Error).
- The evaluation metrics for Linear Regression are printed.

2. ElasticNet Model (after tuning):

- An ElasticNet model is initialized with hyperparameters (alpha=1.0, l1_ratio=0.5) and trained on the same training data.
- The model's predictions are evaluated using the same metrics as Linear Regression.
- The evaluation metrics for ElasticNet are printed.

3. Comparison of Models:
- After both models are trained and evaluated, the best model is selected based on the following criteria:
  - RMSE, MAPE, MAE should be minimized (lower values are better).
  - R^2 should be maximized (higher values are better).
- The model with the best combined performance (i.e., the lowest RMSE, MAPE, and MAE, and the highest R^2) is chosen as the best model.

4. Output:
- The code prints the evaluation metrics for both models and identifies the best model based on the criteria mentioned.

This process allows you to compare a simple Linear Regression model with a more sophisticated ElasticNet model and select the one that best fits the data.

19. Energy Yield Visualization (GWh). With Tuned ElasticNet Regression.

```python
import matplotlib.pyplot as plt

#Predict the values using the trained ElasticNet model (after tuning)
y_pred_en = best_elasticnet_refined.predict(x_test)  # Use the correct variable here

#Plot the actual vs predicted values
plt.figure(figsize=(20, 5))

#Plot actual values
plt.plot(y_test.index, y_test, label='Actual TOTAL_YIELD (GWh)', color='blue')

# Plot predicted values
plt.plot(y_test.index, y_pred_en, label='Predicted TOTAL_YIELD (GWh)', color='red', linestyle='--')

#Customize the plot
plt.title('Actual vs Predicted TOTAL_YIELD (GWh) - ElasticNet Regression (After Tuning)', fontsize=16)
plt.xlabel('Datetime', fontsize=14)
plt.ylabel('TOTAL_YIELD (GWh)', fontsize=14)
plt.legend(loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

This piece of code generates a line chart to analyze the actual values versus values predicted of TOTAL_YIELD (GWh) in comparison to the predictions made by the tuned ElasticNet model. Here's what this piece of code does:

- **Prediction:**
  - It runs energy generation prediction (y_pred_en) on the test data (x_test) on the trained ElasticNet model (best_elasticnet_refined).
- **Plotting:**
  - The actual figures of TOTAL_YIELD (GWh) are entered as a blue color. o The values estimated are entered as a red dotted line (--).
- **Customization:**
  - The Title "TOTAL_YIELD (GWh) estimated by ElasticNet Regression model after tuning versus the Actual used Elastic Net AFTER Tuning".
  - The label for the X axis 'Datetime' while that for the Y axis is 'TOTAL_YIELD (GWh)'.
  - A legend is introduced to show the different lines representing actual and estimated figures.
  - Subsequently, the grid lines are presented. This is to enhance clarity.
  - The Grid () function applies grid lines to the current axes, while the command plt.tight_enable() avoids clipping of the image elements that are drawn in the figure.
- **The plot is displayed:**
  - The command plt.show() is utilized for the representation plot, which simulates comparing the figures supplied to the model with actual figures over time.

Thus, this visualization enables to visually judge the accuracy of the ElasticNet model in estimating energy generation after tuning by demonstrating the gap between the actual figures and values predicted.

20. Visualizing Comparison of Actual vs Predicted Energy Yield (TOTAL_YIELD in GWh) Using ElasticNet and Linear Regression Models

```
import matplotlib.pyplot as plt

#Create a figure with 2 subplots
fig, axs = plt.subplots(1, 2, figsize=(20, 5))

# Plot Actual vs Predicted for ElasticNet in the first subplot
axs[0].plot(y_test.index, y_test, label='Actual TOTAL_YIELD (GWh)', color='blue')
axs[0].plot(y_test.index, y_pred_en, label='Predicted by ElasticNet (After Tuning)', color='red', linestyle='--')
axs[0].set_title('ElasticNet Regression (After Tuning)', fontsize=14)
axs[0].set_xlabel('Datetime', fontsize=12)
axs[0].set_ylabel('TOTAL_YIELD (GWh)', fontsize=12)
axs[0].legend(loc='upper left')

# Plot Actual vs Predicted for Linear Regression in the second subplot
axs[1].plot(y_test.index, y_test, label='Actual TOTAL_YIELD (GWh)', color='blue')
axs[1].plot(y_test.index, y_pred_lr, label='Predicted by Linear Regression (Before Tuning)', color='green', linestyle='-
-')
axs[1].set_title('Linear Regression (Before Tuning)', fontsize=14)
axs[1].set_xlabel('Datetime', fontsize=12)
axs[1].set_ylabel('TOTAL_YIELD (GWh)', fontsize=12)
axs[1].legend(loc='upper left')

plt.tight_layout()
plt.show()
```

This code creates two side-by-side subplots to compare the performance of the ElasticNet and Linear Regression models for predicting TOTAL_YIELD (GWh).
Outline of what the code accomplishes:

- Create subplots:
  - The figure comprises 2 subplots arranged in one row (1, 2), with the dimensions 20 inch x 5 inch.
- First Subplot (ElasticNet Regression):
  - Actual values (y_test) 90are plotted in blue.
  - Red dashed line (– –) illustrates predicted values from ElasticNet model (y_pred_en).
  - 'ElasticNet Regression (After Tuning)' is the title for the subplot. o X-axis: 'Datetime', Y-axis: 'TOTAL_YIELD (GWh)'.
  - To identify the actual and predicted values, a legend is added.
- Second Subplot (Linear Regression):
  - A blue line represents actual values (y_test).
  - Dashed green line (y_pred_lr) illustrates the predicted values from the Linear Regression model.
  - The subplot is named \"Linear Regression (Before Tuning)\".
  - The title of the subplot X-axis is 'Datetime'; Y-axis is 'TOTAL_YIELD (GWh)'.
  - Finally, a legend is added to explain actual and predicted values.
- Displaying the plot:
  - plt.show() is used to show the plot whose comparison with the two models predictions are shown against the real values.

This visual helps to see the data from the two models, telling which one works better on following the data trends through time.

21. Visualizing Together a Comparison of Actual and Predicted Energy Yield (TOTAL_YIELD in GWh) Using ElasticNet and Linear Regression Models On One Graph

```
import matplotlib.pyplot as plt

# Predict the values using both trained models
# Predictions from ElasticNet (After Tuning)
y_pred_en = best_elasticnet_refined.predict(x_test)  # Use best_elasticnet_refined here

# Predictions from Linear Regression (Before Tuning)
y_pred_lr = lr_reg.predict(x_test)

#Plot the actual vs predicted values for both models
```

```
plt.figure(figsize=(20, 5))

# Plot actual values
plt.plot(y_test.index, y_test, label='Actual TOTAL_YIELD (GWh)', color='blue', linewidth=1)

# Plot predicted values from ElasticNet with a solid line and circle markers
plt.plot(y_test.index, y_pred_en, label='Predicted by ElasticNet (After Tuning)', color='red', linestyle='-',
linewidth=1, marker='o', markersize=1)

# Plot predicted values from Linear Regression with a dashed line and square markers
plt.plot(y_test.index, y_pred_lr, label='Predicted by Linear Regression (Before Tuning)', color='green', linestyle='--',
linewidth=1, marker='s', markersize=1)

#Customize the plot
plt.title('Actual vs Predicted TOTAL_YIELD (GWh) - ElasticNet and Linear Regression', fontsize=16)
plt.xlabel('Datetime', fontsize=14)
plt.ylabel('TOTAL_YIELD (GWh)', fontsize=14)
plt.legend(loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

This code draws a line graph to show comparison of actual values and predicted values of TOTAL_YIELD (GWh) using ElasticNet and Linear Regression models.

In this section we deconstruct the plot to show how it is built visually: Breakdown of the Code:

- Prediction:
  - ElasticNet model (best_elasticnet_refined) predicts TOTAL_YIELD (GWh) on the test set (y_pred_en).
  - Linear regression model (lr_reg) predicts on test set (y_pred_lr).
- Plotting:
  - The actual values (y_test) are indicated by a blue line of 1 unit width.
  - The values predicted by ElasticNet are indicated by a red solid line (-) with circle markers (o) and 1 unit width.
  - The values predicted by Linear Regression are depicted by a green dashed line (--) of 1 unit width with square markers (s).
- Customization:
  - Title: "Actual vs Predicted TOTAL_YIELD (GWh) - ElasticNet and Linear Regression".
  - X-axis: 'Datetime', Y-axis: 'TOTAL_YIELD (GWh)'.
  - Legend is added to distinguish the actual and predicted values for both models.
  - A grid is added to make the plot easier to read.
  - plt.tight_layout() ensures that the plot elements don't overlap.
- Displaying the plot:
  - plt.show() displays the plot, showing the comparison of the actual values with the predictions from both models.
- Key Features:
  - This plot helps visually assess how closely the two models' predictions match the actual data.
  - The use of different markers for the models helps distinguish them clearly in the plot.

This approach provides a direct comparison of the performance of the ElasticNet and Linear Regression models for predicting energy generation over time.

22. Predict Next 7 Days Total Yield With Tuning

```
from sklearn.linear_model import ElasticNet  # Import ElasticNet model
from sklearn.model_selection import GridSearchCV  # Import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
import pandas as pd

elasticnet = ElasticNet()

param_grid_refined = {
    'alpha': [0.1, 0.5, 1, 2, 5],
    'l1_ratio': [0.3, 0.5, 0.7, 0.9, 1.0]
}
```

```
grid_search_refined = GridSearchCV(elasticnet, param_grid_refined, cv=10, scoring='neg_mean_squared_error', n_jobs=-1,
verbose=3)

grid_search_refined.fit(x_train, y_train)  # Fit the model

print("Best parameters from refined GridSearchCV:", grid_search_refined.best_params_)

best_elasticnet_refined = grid_search_refined.best_estimator_

y_pred_best_en_refined = best_elasticnet_refined.predict(x_test)

rmse_best_en_refined = np.sqrt(mean_squared_error(y_test, y_pred_best_en_refined))
mape_best_en_refined = np.mean(np.abs((y_test.values - y_pred_best_en_refined) / y_test.values)) * 100
r2_best_en_refined = r2_score(y_test.values, y_pred_best_en_refined)
mae_best_en_refined = mean_absolute_error(y_test.values, y_pred_best_en_refined)

print(f"ElasticNet Regression (After Further Tuning) - RMSE: {rmse_best_en_refined:.2f}")
print(f"ElasticNet Regression (After Further Tuning) - MAPE: {mape_best_en_refined:.2f}%")
print(f"ElasticNet Regression (After Further Tuning) - R²: {r2_best_en_refined:.4f}")
print(f"ElasticNet Regression (After Further Tuning) - MAE: {mae_best_en_refined:.2f}")

future_dates = pd.date_range(start='2025-06-18', periods=7, freq='D')  # Adjust start date as needed

num_features = x_train.shape[1]  # Get number of features from training data

def generate_future_features(future_dates):
    future_features = []
    mean_features = np.mean(x_train, axis=0)  # Calculate mean of each feature in x_train
    for date in future_dates:
        feature_vector = [date.day, date.month]  # Example: Day and Month
        additional_features = mean_features[2:num_features]  # Fill remaining features with mean values
        feature_vector.extend(additional_features)
        future_features.append(feature_vector)
    return pd.DataFrame(future_features, columns=x_train.columns)  # Ensure DataFrame format

x_future = generate_future_features(future_dates)
future_yield_predictions = best_elasticnet_refined.predict(x_future)

future_predictions_df = pd.DataFrame({'Date': future_dates, 'Predicted Yield': future_yield_predictions})

print("Next 7 Days Predicted Yield:")
print(future_predictions_df)
```

This code performs the following tasks:

1. **Import Libraries**: It imports necessary libraries, including ElasticNet (a regression model), GridSearchCV (for hyperparameter tuning), and various metrics for evaluating the model.
2. **ElasticNet Model**: It initializes an ElasticNet model, which is a linear regression model combining L1 and L2 regularization.
3. **Grid Search Setup**: It defines a parameter grid (param_grid_refined) for alpha and l1_ratio hyperparameters, which will be tuned using cross-validation (10-fold) with GridSearchCV.
4. **Model Fitting**: The model is trained (fit) on the training data (x_train and y_train) to find the best combination of hyperparameters using grid search.
5. **Model Evaluation**: After fitting, the best model is selected and used to predict values (y_pred_best_en_refined) on the test data (x_test). Several performance metrics are calculated, including:
    - RMSE (Root Mean Squared Error)
    - MAPE (Mean Absolute Percentage Error)
    - R^2 (R-squared)
    - MAE (Mean Absolute Error)
6. **Future Predictions**: The model is then used to predict yields for the next 7 days. Future feature values are generated by using the mean values of the training data and the day/month of future dates. These predicted values are stored in a DataFrame.
7. **Display Results**: The predictions for the next 7 days are printed.

This process optimizes the model, evaluates it, and predicts future outcomes based on learned patterns.

23. Visualize Predicted Next 7 Days Total Yield

```
import matplotlib.pyplot as plt
import pandas as pd

future_predictions_df = pd.DataFrame({
    'Date': pd.date_range(start='2025-06-18', periods=7, freq='D'),
    'Predicted Yield': future_yield_predictions
})

plt.figure(figsize=(30, 5))
plt.plot(future_predictions_df['Date'], future_predictions_df['Predicted Yield'], marker='o', linestyle='-', color='b',
label='Predicted Yield')

plt.xlabel('Date')
plt.ylabel('Yield')
plt.title('Predicted Yield for the Next 7 Days')
plt.xticks(rotation=45)
plt.legend()
plt.grid()
plt.show()
```

This code is used to visualize the predicted yields for the next 7 days:

1. Import Libraries: It imports matplotlib.pyplot for plotting and pandas for data manipulation.
2. Create DataFrame: A DataFrame (future_predictions_df) is created with two columns:
    - Date: A range of dates starting from June 18, 2025, for 7 consecutive days.
    - Predicted Yield: The predicted yield values (future_yield_predictions) that were generated earlier.
3. Plot Setup:
    - The figure is set to a large size (30 inches by 5 inches) using plt.figure(figsize=(30, 5)).
    - The plt.plot() function is used to plot the predicted yield values against the corresponding dates. It uses a blue line with circular markers and labels the line as 'Predicted Yield'.
4. Plot Customization:
    - xlabel() and ylabel() are used to label the x-axis as "Date" and the y-axis as "Yield".
    - title() sets the title of the plot to "Predicted Yield for the Next 7 Days".
    - xticks(rotation=45) rotates the x-axis labels (dates) by 45 degrees to make them more readable.
    - legend() displays the legend to label the plotted line.
    - grid() adds a grid to the plot for better readability.
5. Show Plot: Finally, plt.show() displays the plot.

The resulting plot will show the predicted yield values over the next 7 days with appropriate labeling and formatting.

24. Perform a statistical analysis of the effect of MODULE_TEMPERATURE (°C) on TOTAL_YIELD (GWh) and quantify its impact

- **Loading and Displaying the Merged Hourly Data**

```
import pandas as pd

# Load the dataset
file_path = '/content/merged_hourly_data.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

This Code loading a generated cleaned CSV file (merged_hourly_data.csv) and displaying the first few rows to inspect the data. Here's what each part of the code does:
Loading the CSV:

- file_path specifies the location of the dataset.
- pd.read_csv(file_path) loads the CSV file into a pandas DataFrame called data.

Display the First Few Rows:

- data.head() displays the first five rows of the dataset to give a preview of the structure and contents (e.g., column names, sample data).

25. Data Clean Check

```
import pandas as pd

file_path = '/content/merged_hourly_data.csv'
data = pd.read_csv(file_path)
```

```python
def check_data_cleanliness(data):
    missing_values = data.isnull().sum()
    data_types = data.dtypes
    duplicate_rows = data.duplicated().sum()
    invalid_datetime_rows = data[data['Datetime'].isna()]

    datetime_format_check = pd.to_datetime(data['Datetime'], errors='coerce').isna().sum()

    cleanliness_report = {
        "Missing Values": missing_values,
        "Data Types": data_types,
        "Duplicate Rows": duplicate_rows,
        "Invalid Datetime Entries": invalid_datetime_rows.shape[0],
        "Invalid Datetime Format Count": datetime_format_check
    }

    data_ready = True
    if missing_values.any() or duplicate_rows > 0 or datetime_format_check > 0:
        data_ready = False

    return cleanliness_report, data_ready

cleanliness_report, is_ready = check_data_cleanliness(data)

print("Data Cleanliness Report:")
for key, value in cleanliness_report.items():
    print(f"{key}: {value}")

if is_ready:
    print("\nThe data is ready for analysis.")
else:
    print("\nThe data is not ready for analysis. Please clean it.")
```

This code defines a function check_data_cleanliness that performs a data cleanliness check on the dataset. Here's a breakdown of how it works:

- Missing Values:
  - data.isnull().sum() checks for missing values (NaN) in each column and returns the count.
- Data Types:
  - data.dtypes returns the data types of each column to ensure that they are as expected (e.g., Datetime should be of type datetime).
- Duplicate Rows:
  - data.duplicated().sum() checks for duplicate rows and counts how many there are.
- Invalid Datetime Entries:
  - data['Datetime'].isna() checks if there are any invalid datetime entries (NaT or missing values) in the Datetime column.
- Datetime Format Check:
  - pd.to_datetime(data['Datetime'], errors='coerce').isna().sum() checks if the Datetime column can be converted to proper datetime format. Any entries that cannot be converted will result in NaT, and the count of these invalid entries is recorded.
- Cleanliness Report:
  - To track all the cleansiness information such as count of missing values, data types, duplicate rows, invalid datetime entries, and invalid datetime format, a dictionary (cleanliness_report) is created.
- Data Readiness:
  - The function assumes data_ready is False upon detecting missing values, duplicate rows, or invalid datetime formats.
- Return Values:
  - The datasets are returned with the cleanliness_report alongside the flag indicating if the dataset is cleared for analysis or not.
- Output:
  - The function outputs to console the report on data cleanliness including, but not limited to, missing values, types, duplicate rows, invalid datetimes to check.

  - If the data passes the checks (i.e., it is free from missing values, duplicates, and invalid datetime formats), the console reads, "The data is ready for analysis."

- Otherwise, the console reads: “The data is not ready for analysis. Please clean it.”

This is a useful tool to ensure the dataset is clean and ready for analysis, which is essential before applying any modeling or further exploration.

26. Selecting the Best Model - Statistical Analysis

```python
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

#Prepare the dataset (focus on TOTAL_YIELD and MODULE_TEMPERATURE)
dataset = data[['TOTAL_YIELD (GWh)', 'MODULE_TEMPERATURE (°C)']]

#Swap X and y
X_new = dataset[['TOTAL_YIELD (GWh)']]  # Now TOTAL_YIELD is the independent variable
y_new = dataset['MODULE_TEMPERATURE (°C)']  # Now MODULE_TEMPERATURE is the dependent variable

#Split the data into training and testing sets (80% train, 20% test)
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new, y_new, test_size=0.2, random_state=42)

#Initialize the models
rf_new = RandomForestRegressor(n_estimators=100, random_state=42)
lr_reg = LinearRegression()
svr_reg = SVR()

#Train the models
rf_new.fit(X_train_new, y_train_new)
lr_reg.fit(X_train_new, y_train_new)
svr_reg.fit(X_train_new, y_train_new)

#Make predictions for each model
y_pred_rf_new = rf_new.predict(X_test_new)
y_pred_lr_new = lr_reg.predict(X_test_new)
y_pred_svr_new = svr_reg.predict(X_test_new)

#Evaluate each model

# Random Forest
rmse_rf_new = np.sqrt(mean_squared_error(y_test_new, y_pred_rf_new))
mape_rf_new = np.mean(np.abs((y_test_new - y_pred_rf_new) / y_test_new) * 100)
r2_rf_new = r2_score(y_test_new, y_pred_rf_new)
mae_rf_new = mean_absolute_error(y_test_new, y_pred_rf_new)

# Linear Regression
rmse_lr_new = np.sqrt(mean_squared_error(y_test_new, y_pred_lr_new))
mape_lr_new = np.mean(np.abs((y_test_new - y_pred_lr_new) / y_test_new) * 100)
r2_lr_new = r2_score(y_test_new, y_pred_lr_new)
mae_lr_new = mean_absolute_error(y_test_new, y_pred_lr_new)

# Support Vector Regression
rmse_svr_new = np.sqrt(mean_squared_error(y_test_new, y_pred_svr_new))
mape_svr_new = np.mean(np.abs((y_test_new - y_pred_svr_new) / y_test_new) * 100)
r2_svr_new = r2_score(y_test_new, y_pred_svr_new)
mae_svr_new = mean_absolute_error(y_test_new, y_pred_svr_new)

# Step 9: Print evaluation metrics for each model
print(f"Random Forest - RMSE: {rmse_rf_new:.2f}")
print(f"Random Forest - MAPE: {mape_rf_new:.2f}%")
print(f"Random Forest - R²: {r2_rf_new:.4f}")
print(f"Random Forest - MAE: {mae_rf_new:.2f}")
print()

print(f"Linear Regression - RMSE: {rmse_lr_new:.2f}")
print(f"Linear Regression - MAPE: {mape_lr_new:.2f}%")
print(f"Linear Regression - R²: {r2_lr_new:.4f}")
print(f"Linear Regression - MAE: {mae_lr_new:.2f}")
print()

print(f"Support Vector Regression - RMSE: {rmse_svr_new:.2f}")
print(f"Support Vector Regression - MAPE: {mape_svr_new:.2f}%")
print(f"Support Vector Regression - R²: {r2_svr_new:.4f}")
print(f"Support Vector Regression - MAE: {mae_svr_new:.2f}")
print()

#Select the best model based on RMSE, MAPE, R², and MAE
models = {
    'Random Forest': {'RMSE': rmse_rf_new, 'MAPE': mape_rf_new, 'R²': r2_rf_new, 'MAE': mae_rf_new},
    'Linear Regression': {'RMSE': rmse_lr_new, 'MAPE': mape_lr_new, 'R²': r2_lr_new, 'MAE': mae_lr_new},
    'Support Vector Regression': {'RMSE': rmse_svr_new, 'MAPE': mape_svr_new, 'R²': r2_svr_new, 'MAE': mae_svr_new}
}

# Select best model based on combined scores (lower RMSE, MAPE, and MAE; higher R²)
best_model = None
best_score = float('inf')  # Start with a high value for RMSE, MAPE, and MAE
best_r2 = -float('inf')   # Start with a low value for R²

# Compare models and select the best one
for model_name, scores in models.items():
    # We subtract R² to prioritize higher R² values, and add RMSE, MAPE, MAE to prioritize lower values
    score = scores["RMSE"] + scores["MAPE"] + scores["MAE"] - scores["R²"]

    if score < best_score:
        best_score = score
        best_model = model_name
        best_r2 = scores["R²"]

#Output the best model based on all metrics
print(f"The best model based on all metrics is: {best_model}")
print(f"Best RMSE: {models[best_model]['RMSE']:.2f}")
print(f"Best MAPE: {models[best_model]['MAPE']:.2f}%")
print(f"Best R²: {best_r2:.4f}")
print(f"Best MAE: {models[best_model]['MAE']:.2f}")
```

This code performs regression analysis using three models: Random Forest, Linear Regression, and Support Vector Regression (SVR), and compares them based on various evaluation metrics to select the best model for predicting MODULE_TEMPERATURE (°C) based on TOTAL_YIELD (GWh).

Breakdown of the Code:

- Data Preparation:
  - The dataset is created using two columns: TOTAL_YIELD (GWh) as the independent variable (X_new) and MODULE_TEMPERATURE (°C) as the dependent variable (y_new).
  - The data is split into training (80%) and testing (20%) sets using train_test_split.
- Model Initialization:
  - Three regression models are initialized:
    - Random Forest Regressor
    - Linear Regression
    - Support Vector Regressor (SVR)
- Model Training:
  - Each model is trained on the training data (X_train_new, y_train_new).
- Model Predictions:
  - Each model makes predictions on the test set (X_test_new), and the predictions are stored in y_pred_rf_new, y_pred_lr_new, and y_pred_svr_new.
- Model Evaluation:
  - The models are evaluated using the following metrics:
    - RMSE (Root Mean Squared Error)
    - MAPE (Mean Absolute Percentage Error)
    - R^2 (R-squared)
    - MAE (Mean Absolute Error)
  - These metrics are calculated for each model, and the results are printed.
- Model Comparison and Selection:
  - A combined score is calculated for each model by adding RMSE, MAPE, and MAE, and subtracting R^2 (since a higher R^2 is better, it is subtracted).
  - The model with the lowest combined score is selected as the best model.
- Output:
  - The code prints the evaluation metrics (RMSE, MAPE, R^2, MAE) for each model.

- It then outputs the best model based on the combined scores, along with its respective metrics.

This process helps to evaluate and select the most suitable regression model for predicting MODULE_TEMPERATURE (°C) based on TOTAL_YIELD (GWh). The best model is determined by balancing all the evaluation metrics (lower RMSE, MAPE, and MAE, and higher R²).

27. Predicting MODULE_TEMPERATURE (°C) from TOTAL_YIELD (GWh) Using Random Forest Regressor

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

#Prepare the dataset (focus on TOTAL_YIELD and MODULE_TEMPERATURE)
dataset = data[['TOTAL_YIELD (GWh)', 'MODULE_TEMPERATURE (°C)']]

#Swap X and y
X_new = dataset[['TOTAL_YIELD (GWh)']]  # Now TOTAL_YIELD is the independent variable
y_new = dataset['MODULE_TEMPERATURE (°C)']  # Now MODULE_TEMPERATURE is the dependent variable

#Split the data into training and testing sets (80% train, 20% test)
X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X_new, y_new, test_size=0.2, random_state=42)

#Create and train a Random Forest model with the swapped X and y
rf_new = RandomForestRegressor(n_estimators=100, random_state=42)
rf_new.fit(X_train_new, y_train_new)

#Make predictions and evaluate the model
y_pred_rf_new = rf_new.predict(X_test_new)
mse_rf_new = mean_squared_error(y_test_new, y_pred_rf_new)
rmse_rf_new = np.sqrt(mse_rf_new)
r2_rf_new = r2_score(y_test_new, y_pred_rf_new)

#Print results for Random Forest
print(f"Random Forest - RMSE: {rmse_rf_new:.2f}")
print(f"Random Forest - R²: {r2_rf_new:.4f}")

#Generate predictions over a range of TOTAL_YIELD values
min_yield = X_new['TOTAL_YIELD (GWh)'].min()
max_yield = X_new['TOTAL_YIELD (GWh)'].max()
yield_range = np.linspace(min_yield, max_yield, 100).reshape(-1, 1)

#Predict MODULE_TEMPERATURE (°C) for the range of TOTAL_YIELD values
y_pred_rf_range = rf_new.predict(yield_range)
```

This code performs regression analysis using a Random Forest Regressor to predict MODULE_TEMPERATURE (°C) based on TOTAL_YIELD (GWh) and evaluates the model's performance. Here's a breakdown of the steps:

- Data Preparation:
  - Swapping X and y:
  - The independent variable (X_new) is TOTAL_YIELD (GWh), and the dependent variable (y_new) is MODULE_TEMPERATURE (°C).
  - The dataset is created by selecting only the columns TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C).
- Train-Test Split:
  - The dataset is split into training and testing sets using train_test_split(), with 80% for training and 20% for testing.
- Model Initialization and Training:
  - A Random Forest Regressor is initialized with 100 trees (n_estimators=100) and a fixed random state (random_state=42).
  - The model is trained using the training data (X_train_new, y_train_new).
- Model Evaluation:
  - Predictions are made on the test set (X_test_new) using the trained model.
  - The Mean Squared Error (MSE) is computed, and the Root Mean Squared Error (RMSE) is calculated by taking the square root of the MSE.
  - The R² (R-squared) value is also calculated to assess how well the model explains the variance in the data.
  - The results (RMSE and R²) are printed.

- Prediction Across a Range of TOTAL_YIELD Values:
  - A range of TOTAL_YIELD (GWh) values (yield_range) is generated using np.linspace() to cover the minimum to maximum values of TOTAL_YIELD (GWh).
  - The model is used to predict MODULE_TEMPERATURE (°C) for each value in this range (yield_range), storing the predictions in y_pred_rf_range.
- Output:
  - The code prints the RMSE and R² evaluation metrics for the Random Forest model:
    - RMSE: Provides an estimate of the average prediction error.
    - R²: Indicates the proportion of variance explained by the model.
  - The model will also generate predictions for a range of TOTAL_YIELD (GWh) values to visualize how the MODULE_TEMPERATURE (°C) changes with respect to TOTAL_YIELD (GWh).

28. Visualizing the Train-Test Split for Energy Yield and Temperature Data

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Create a DataFrame for visualization
test_train_data = pd.DataFrame({
    'TOTAL_YIELD (GWh)': np.concatenate([y_train_new, y_test_new]),
    'Type': ['Train'] * len(y_train_new) + ['Test'] * len(y_test_new)
})

# Create the line chart for test-train split
fig, ax = plt.subplots(figsize=(30, 5))
for label, df in test_train_data.groupby('Type'):
    ax.plot(df.index, df['TOTAL_YIELD (GWh)'], marker='o', linestyle='-', label=label)

ax.set_title('Module Temperature Train-Test Split (MODULE_TEMPERATURE (°C))')
ax.set_xlabel('Time')
ax.set_ylabel('TOTAL_YIELD (GWh)')
ax.legend()

plt.show()
```

This code visualizes the train-test split of the TOTAL_YIELD (GWh) data in a line chart.

- Data Preparation:
  - It combines y_train_new (training data) and y_test_new (test data) into a single DataFrame (test_train_data).
  - A new column, 'Type', is added to label the data as either 'Train' or 'Test'.
- Plotting:
  - It groups the data by the 'Type' column and creates a line plot for both training and test data.
  - Markers are added to the lines, and the x-axis represents the index of the data.
- Visualization:
  - The plot is customized with a title, labels for the x and y axes, and a legend to distinguish the training and test data.
- Result:
  - A line chart is displayed, showing how TOTAL_YIELD (GWh) changes for both the training and testing sets.

29. Visualization - Effect of TOTAL_YIELD (GWh) on MODULE_TEMPERATURE (°C) Using Random Forest Predictions

```
#Plotting the prediction (x: TOTAL_YIELD (GWh), y: MODULE_TEMPERATURE (°C))
plt.figure(figsize=(30, 6))
plt.plot(yield_range, y_pred_rf_range, label='Random Forest Prediction', color='green', linewidth=2)
plt.scatter(X_new['TOTAL_YIELD (GWh)'], y_new, color='red', label='Actual Data', alpha=0.5)
plt.title('Effect of TOTAL_YIELD (GWh) on MODULE_TEMPERATURE (°C)', fontsize=16)
plt.xlabel('TOTAL_YIELD (GWh)', fontsize=14)
plt.ylabel('MODULE_TEMPERATURE (°C)', fontsize=14)
plt.legend(loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

This code creates a line plot to visualize the relationship between TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C) based on the Random Forest model's predictions as well as the actual data.

- Plot the Prediction:
    - plt.plot(yield_range, y_pred_rf_range, ...):
    - This line plots the predicted values from the Random Forest model (y_pred_rf_range) for a range of TOTAL_YIELD (GWh) values (yield_range).
    - The predicted line is shown in green with a line width of 2.
- Plot the Actual Data:
    - plt.scatter(X_new['TOTAL_YIELD (GWh)'], y_new, ...):
    - This scatter plot shows the actual data points where TOTAL_YIELD (GWh) is on the x-axis and MODULE_TEMPERATURE (°C) is on the y-axis.
    - The data points are shown in red with some transparency (alpha=0.5).
- Customize the Plot:
    - Title: The plot is titled "Effect of TOTAL_YIELD (GWh) on MODULE_TEMPERATURE (°C)"
    - Axis Labels: The x-axis is labeled 'TOTAL_YIELD (GWh)' and the y-axis is labeled 'MODULE_TEMPERATURE (°C)'.
    - Legend: The plot has a legend in the upper left corner that distinguishes between the prediction and the actual data.
    - Grid: A grid is added to improve readability.
- Display the Plot:
    - plt.tight_layout() adjusts the layout to avoid overlap.
    - plt.show() displays the plot.
- Expected Output:
    - A line graph representing the predicted relationship between TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C) based on the Random Forest model.
    - A scatter plot of the actual data points.

This visualization helps to assess how well the Random Forest model fits the actual data and how TOTAL_YIELD (GWh) influences MODULE_TEMPERATURE (°C). The predicted line should ideally align closely with the actual data points.

30. Random Forest Hyperparameter Tuning with RandomizedSearchCV for Predicting MODULE_TEMPERATURE from TOTAL_YIELD

```
#Import necessary libraries
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

#Set up an extended hyperparameter grid
param_dist_improved = {
    'n_estimators': [50, 100, 200],  # Reduce the number of trees for faster fitting
    'max_depth': [None, 10, 20, 30],  # Limiting the depth of the trees
    'min_samples_split': [2, 5, 10],  # Keep the values smaller for faster fitting
    'min_samples_leaf': [1, 2, 4],    # Reduce the values to speed up fitting
    'max_features': ['sqrt', 'log2', 0.5]  # Use sqrt or log2 to limit the features considered
}

# Create the Random Forest Regressor model
rf = RandomForestRegressor(random_state=42)

#Set up RandomizedSearchCV with smaller n_iter and less cross-validation folds
random_search_fast = RandomizedSearchCV(estimator=rf, param_distributions=param_dist_improved,
                                        n_iter=10, cv=3, scoring='neg_mean_squared_error',
                                        n_jobs=2, verbose=2, random_state=42)  # Reduced n_jobs to 2

#Fit the model with RandomizedSearchCV
random_search_fast.fit(X_train_new, y_train_new)

#Print the best hyperparameters and the best score
print(f"Best Hyperparameters: {random_search_fast.best_params_}")
print(f"Best CV Score: {random_search_fast.best_score_}")

#Get the best model from RandomizedSearchCV
best_rf_random_fast = random_search_fast.best_estimator_

#Make predictions using the best model
y_pred_rf_best_fast = best_rf_random_fast.predict(X_test_new)

#Evaluate the tuned model
mse_rf_best_fast = mean_squared_error(y_test_new, y_pred_rf_best_fast)
rmse_rf_best_fast = np.sqrt(mse_rf_best_fast)
r2_rf_best_fast = r2_score(y_test_new, y_pred_rf_best_fast)

#Print evaluation metrics
print(f"Random Forest (Tuned) - RMSE: {rmse_rf_best_fast:.2f}")
print(f"Random Forest (Tuned) - R²: {r2_rf_best_fast:.4f}")

#Generate predictions over a range of TOTAL_YIELD values
yield_range_fast = np.linspace(min_yield, max_yield, 100).reshape(-1, 1)

#Predict MODULE_TEMPERATURE (°C) for the range of TOTAL_YIELD values
y_pred_rf_range_best_fast = best_rf_random_fast.predict(yield_range_fast)
```

This code performs a Randomized Search to tune the hyperparameters of a Random Forest Regressor for predicting MODULE_TEMPERATURE (°C) based on TOTAL_YIELD (GWh).

- Hyperparameter Grid: Defines a set of possible hyperparameters (e.g., number of trees, tree depth, and split criteria).
- Randomized Search: Uses RandomizedSearchCV to search for the best combination of hyperparameters, performing 3-fold cross-validation.
- Model Fitting: The best model is selected based on the random search results.
- Model Evaluation: Evaluates the tuned model using RMSE and R² metrics.
- Prediction: Makes predictions over a range of TOTAL_YIELD (GWh) values.

The process speeds up hyperparameter tuning and improves model accuracy.

31. Visualizes Effect of TOTAL_YIELD (GWh) on MODULE_TEMPERATURE (°C) Using Tuned Random Forest Predictions

```
#Plotting the prediction (x: TOTAL_YIELD (GWh), y: MODULE_TEMPERATURE (°C))
plt.figure(figsize=(30, 6))
plt.plot(yield_range_fast, y_pred_rf_range_best_fast, label='Random Forest (Tuned) Prediction', color='blue',
linewidth=2)
plt.scatter(X_new['TOTAL_YIELD (GWh)'], y_new, color='red', label='Actual Data', alpha=0.5)
plt.title('Effect of TOTAL_YIELD (GWh) on MODULE_TEMPERATURE (°C)', fontsize=16)
plt.xlabel('TOTAL_YIELD (GWh)', fontsize=14)
plt.ylabel('MODULE_TEMPERATURE (°C)', fontsize=14)
plt.legend(loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

This code creates a line plot that visualizes the predictions made by the tuned Random Forest model (Random Forest (Tuned)) and compares them with the actual data for MODULE_TEMPERATURE (°C) based on TOTAL_YIELD (GWh).

- Prediction Plot:
  - plt.plot(yield_range_fast, y_pred_rf_range_best_fast, ...) plots the predictions (y_pred_rf_range_best_fast) over a range of TOTAL_YIELD (GWh) values (yield_range_fast). The predictions are shown in blue.
- Actual Data Points:
  - plt.scatter(X_new['TOTAL_YIELD (GWh)'], y_new, ...) adds red scatter points representing the actual data (TOTAL_YIELD (GWh) vs. MODULE_TEMPERATURE (°C)).
- Customization:
  - Title: "Effect of TOTAL_YIELD (GWh) on MODULE_TEMPERATURE (°C)".
  - Axis Labels: x-axis: 'TOTAL_YIELD (GWh)', y-axis: 'MODULE_TEMPERATURE (°C)'.
  - Legend: Differentiates between the actual data and the predictions.
  - Grid: Added for better readability.
- Displaying the Plot:
  - plt.tight_layout() ensures no overlap in the plot.
  - plt.show() displays the plot.

The plot shows how well the tuned Random Forest model captures the relationship between TOTAL_YIELD (GWh) and MODULE_TEMPERATURE (°C) by comparing the predicted values with the actual data points.

32. Analyzing the Lowest and Highest TOTAL_YIELD (GWh) and Corresponding MODULE_TEMPERATURE (°C) with Tolerance Range

```
import pandas as pd
import numpy as np

#Prepare the dataset (focus on TOTAL_YIELD and MODULE_TEMPERATURE)
dataset = data[['TOTAL_YIELD (GWh)', 'MODULE_TEMPERATURE (°C)']]

#Create temperature ranges (adjusted as needed for the analysis)
temperature_bins = list(range(int(dataset['MODULE_TEMPERATURE (°C)'].min()),
                              int(dataset['MODULE_TEMPERATURE (°C)'].max()) + 10, 10))
temperature_labels = [f"{i}-{i+10}" for i in temperature_bins[:-1]]

#Add a new column for temperature ranges
dataset['Temperature Range'] = pd.cut(dataset['MODULE_TEMPERATURE (°C)'], bins=temperature_bins,
labels=temperature_labels, right=False)

#Calculate the average TOTAL_YIELD (GWh) for each temperature range
average_yield_by_temp_range = dataset.groupby('Temperature Range', observed=False)['TOTAL_YIELD (GWh)'].mean()

#Find the range with the highest average TOTAL_YIELD (GWh)
highest_avg_temp_range = average_yield_by_temp_range.idxmax()
highest_avg_yield = average_yield_by_temp_range.max()

#Find the range with the lowest average TOTAL_YIELD (GWh)
lowest_avg_temp_range = average_yield_by_temp_range.idxmin()
lowest_avg_yield = average_yield_by_temp_range.min()

#Print out the results
print(f"Average Yield (GWh) by Temperature Range:")
print(average_yield_by_temp_range)

print("\n")

print(f"Range with the Highest Average Yield (GWh):")
print(f"Temperature Range: {highest_avg_temp_range}")
print(f"Average Yield in this Range: {highest_avg_yield:.2f} GWh")

print("\n")

print(f"Range with the Lowest Average Yield (GWh):")
print(f"Temperature Range: {lowest_avg_temp_range}")
print(f"Average Yield in this Range: {lowest_avg_yield:.2f} GWh")
```

This code performs an analysis of the relationship between MODULE_TEMPERATURE (°C) and TOTAL_YIELD (GWh) by grouping temperature values into temperature ranges and calculating the average energy yield for each range.

- Temperature Binning:
  - The MODULE_TEMPERATURE (°C) values are divided into temperature ranges using pd.cut(). Each range spans 10°C, and the labels are formatted as "X-Y" (e.g., "0-10", "10-20").
  - The temperature_bins define the edges of these ranges, and temperature_labels are the labels for each range.
- Add Temperature Range Column:
  - A new column ('Temperature Range') is added to the dataset, which contains the corresponding temperature range for each observation.
- Group and Calculate Average Yield:
  - The dataset is grouped by 'Temperature Range', and the average TOTAL_YIELD (GWh) is calculated for each range using groupby().mean().
- Identify the Extremes:
  - The range with the highest average yield (highest_avg_temp_range) and the lowest average yield (lowest_avg_temp_range) is identified using .idxmax() and .idxmin() respectively, along with their corresponding values.
- Print Results:
  - The script prints the average yield by temperature range, the range with the highest average yield, and the range with the lowest average yield.

This code divides the MODULE_TEMPERATURE (°C) values into temperature ranges, calculates the average energy yield (TOTAL_YIELD (GWh)) for each range, and identifies the temperature range with the highest and lowest energy generation.

33. Visualizes Analyzed the Lowest and Highest TOTAL_YIELD (GWh) and Corresponding MODULE_TEMPERATURE (°C) with Tolerance Range

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = {
    'Temperature Range': ['18-28', '28-38', '38-48', '48-58', '58-68'],
    'Average Yield (GWh)': [6980.168396, 6987.942507, 6985.346597, 6960.418896, 6940.883260]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Create figure and axis for side-by-side plots
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plotting line chart
axes[0].plot(df['Temperature Range'], df['Average Yield (GWh)'], marker='o', color='b', linestyle='-', linewidth=2)
axes[0].set_title('Average Yield by Temperature Range')
axes[0].set_xlabel('Temperature Range (°C)')
axes[0].set_ylabel('Average Yield (GWh)')
axes[0].grid(True)

# Plotting pie chart
axes[1].pie(df['Average Yield (GWh)'], labels=df['Temperature Range'], autopct='%1.1f%%', startangle=90)
axes[1].set_title('Yield Distribution by Temperature Range')

# Show the plots
plt.tight_layout()
plt.show()
```

This code generates two visualizations side-by-side to analyze the average energy yield across different temperature ranges.

- Data Setup:
  - A dictionary data is defined, containing temperature ranges and their corresponding average energy yields in GWh.
  - This dictionary is then converted into a pandas DataFrame (df).
- Creating Subplots:
  - fig, axes = plt.subplots(1, 2, figsize=(14, 6)) creates a figure with 2 subplots arranged horizontally (side-by-side).
- Plotting Line Chart:
  - The first plot (axes[0]) is a line chart showing the relationship between the temperature range and the average energy yield.
  - The chart uses blue markers ('o') and solid lines ('-').
  - It is titled "Average Yield by Temperature Range", with labeled axes for temperature and yield.
- Plotting Pie Chart:
  - The second plot (axes[1]) is a pie chart that represents the distribution of average energy yield across temperature ranges.
  - The autopct='%1.1f%%' argument shows the percentage of each slice.
  - The pie chart is titled "Yield Distribution by Temperature Range".
- Displaying the Plot:
  - plt.tight_layout() adjusts the layout for better spacing, and plt.show() displays the two visualizations.
- Visuals:
  - Line Chart: Shows the trend of average energy yield across the different temperature ranges.
  - Pie Chart: Represents the percentage contribution of each temperature range to the total yield.

This code visually illustrates how energy yield varies by temperature range and provides a clearer understanding of the contribution of each temperature range to the total energy output.