



# Contents

<b>1. Abstract .....</b>	<b>4</b>
<b>2. Introduction .....</b>	<b>5</b>
<b>2.1 Background study .....</b>	<b>5</b>
<b>2.2 Proposed Model and Key Findings .....</b>	<b>6</b>
<b>2.3 Objective .....</b>	<b>6</b>
<b>3. Dataset Description .....</b>	<b>7</b>
<b>3.1 Data Source .....</b>	<b>7</b>
<b>3.2 Column Description .....</b>	<b>8</b>
<b>4. Methodology.....</b>	<b>11</b>
<b>4.1 The Approach and Justification .....</b>	<b>11</b>
<b>4.2 The Model .....</b>	<b>12</b>
<b>4.2.1 Importing Tools and Libraries .....</b>	<b>13</b>
<b>4.2.2 Loading and Merging the Dataset.....</b>	<b>14</b>
<b>4.2.3 Exploratory Data Analysis (EDA) .....</b>	<b>16</b>
<b>4.2.3.1 Basics and Statistical Summary .....</b>	<b>17</b>
<b>4.2.3.2 Exploring and handling Missing Values .....</b>	<b>18</b>
<b>4.2.3.3 Distribution of Data .....</b>	<b>20</b>
<b>4.2.3.4 Relationships between Features .....</b>	<b>21</b>
<b>4.2.4 Feature Engineering.....</b>	<b>23</b>
<b>4.2.4.1 Applying Feature Engineering.....</b>	<b>23</b>
<b>4.2.4.2 Defining the Features and Target Variable .....</b>	<b>25</b>
<b>4.2.5 Data Preprocessing.....</b>	<b>26</b>
<b>4.2.5.1 Handling Missing Values by dropping columns.....</b>	<b>26</b>
<b>4.2.5.2 Splitting the Dataset.....</b>	<b>26</b>
<b>4.2.5.3 Handling Missing Values by Imputation.....</b>	<b>28</b>
<b>4.2.5.4 Scaling the Dataset.....</b>	<b>28</b>
<b>4.2.6 The Model Training .....</b>	<b>29</b>
<b>4.2.6.1 Defining the models .....</b>	<b>29</b>
<b>4.2.6.1.1 Linear Regression .....</b>	<b>29</b>
<b>4.2.6.1.2 Decision Tree .....</b>	<b>30</b>

4.2.6.1.3	Random Forest .....	30
4.2.6.1.4	Gradient Boosting .....	31
4.2.6.1.5	KNN.....	31
4.2.6.1.6	Neural Network .....	32
4.2.6.2	Training and Evaluation of the models.....	33
4.2.6.3	Visualizing the model evaluation.....	35
4.2.7	Saving Artifacts and the Model .....	36
4.2.8	Testing the Model.....	37
4.2.9	Creating a file to Save History.....	39
4.2.10	Functions Required for Using the Model.....	40
4.2.10.1	Applying Feature Engineering for Incoming Data .....	40
4.2.10.2	Real-time AC Power Prediction Function.....	41
4.2.10.3	Function for Forecasting AC Power for Few Hours Ahead.....	42
4.2.11	Application of the Model .....	43
4.2.11.1	Real-time AC Power Prediction.....	43
4.2.11.2	Forecast AC Power for Few Hours Ahead.....	44
5.	Discussion .....	45
5.1	Usage of the Model.....	45
5.2	Risks and Potential Fixes .....	45
5.3	Future Enhancements .....	46
6.	Conclusion .....	47
7.	References.....	48

## 1. Abstract

The project is based on a machine learning inspired **AC power predicting model** where it helps to **predict the Realtime AC power generation output and forecasted AC power prediction for few hours ahead**. The inputs taken by the user are feature engineered to get all the needed features for predicting a better accurate answer. Since an accurate answer is required, we have also trained many models at once and got a summary of **RMSE** and  **$R^2$**  values in order to decide which is the best model. Then the best model is saved separately to use when using with Realtime data. Even this model is developed for academic purposes, it also can be enhanced and used in real world scenarios to increase revenue generated and reduce the power wastage. The enhanced model of this sample project will be an significant turning point for all the stakeholders related to Solar power plants such as Engineers, users, owners etc... In conclusion the model covers the Solar Prediction process smoothly for better outcomes.

## **2. Introduction**

### **2.1 Background study**

With the significant increase in population of all around the world, the renewable energy crisis also has been arisen. The global demand for the renewable energy is also estimated to be increased by 25% at the end of the year 2040. In terms of the renewable energy natural gas, fossil fuels, nuclear which were also affecting the threats of nuclear hazards, health problems and environmental solution. By the time world moved into the renewable energy sources such as Solar power, wind energy, biomass, geothermal energy and hydropower to increase the sustainability and eco friendliness. Among these, solar energy plays a major role since it makes a big impact on global renewable energy requirement. It has following significant benefits in compared to other sources.

- The availability of the energy source is ensured since the earth receives a considerable amount of solar energy per day consistently. It has estimated that the amount of solar energy received by each is nearly 10,000 times of the global energy requirement.
- Since solar energy does not emit direct carbon to the environment it is eco-friendly and sustainable.

As a result, initiative towards the solar power plants has been clearly identified. Even though, the solar energy output cannot be stated accurately since it depends on various factors such as climatic changes, hardware equipment used, structure of the solar panels and light intensity received. Therefore, it is a major requirement for having a rough idea on the amount of power output in-order-to increase the efficiency of solar power plants. It will give a clear overview of how the usage verses the power output maps, which is required to manage the power output for reducing the wastage and increasing the usability. Other than that, it also helps for calculating and increasing the revenue generated by these solar power plants for the stakeholders.

## **2.2 Proposed Model and Key Findings**

As a solution we trained a model, which does following predictions based on required inputs.

- Real time AC power prediction
- Forecasting AC power for few hours ahead

Real time power prediction helps to capture the live updates and performance of the solar power plant, and it will be useful for mitigating the unexpected sudden drops. The power forecast for few days ahead is a feature which helps crucially in terms of the revenue management, because it helps to proceed with the short-term plans of power generation and maintenance. It will also help in balancing the grids based on the upcoming power generation for reducing the wastage of energy.

## **2.3 Objective**

Objective of this project is to train a model which outputs actionable insights based on the data collected through the solar power plants. Primary expected gains of this project are,

- Identify the fluctuations in power production and energy usage to map the gap in between them.
- Detection of breakdowns and shortage of energy production.
- Expand the insights gained to make decisions of backups, storage and maintenance.
- Find-out hidden patterns in power generation over the time, climatic changes and other factors for better efficiency.

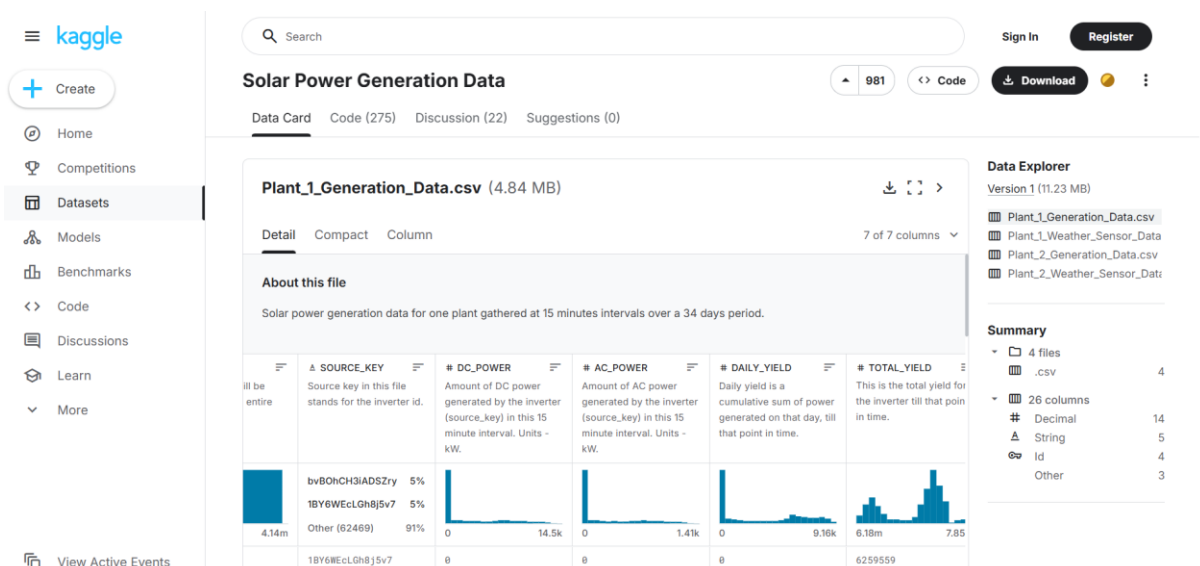
### 3. Dataset Description

#### 3.1 Data Source

The data source for the model training is selected through the Kaggle platform. The dataset can be found through the link - <https://www.kaggle.com/datasets/anikannal/solar-power-generation-data>. The data source is generated through collecting weather sensor data and power generation data from two solar power stations located in India considering 34 days continuously. The datasets accuracy is considerably high since the data were collected considering each inverter level by level. The dataset includes 04 files namely,

- Plant\_1\_Genarataion\_Data.csv
- Plant\_1\_Weather\_Sensor\_Data.csv
- Plant\_2\_Genarataion\_Data.csv
- Plant\_2\_Weather\_Sensor\_Data.csv

According to the above datasets it is clear that the dataset includes data points separately based on the power plant and the type of data gathered. Each column in the dataset will be discussed in following chapters.



## 3.2 Column Description

“Plant\_1\_Genarataion\_Data.csv” and “Plant\_2\_Genarataion\_Data.csv” includes the following data columns.

- **DATE\_TIME** – The observed time and date is mentioned. As the description mentions these data are observed for each 15 minutes.
- **PLANT\_ID** – The unique identifier for the power plant is notated. This attribute will be common per each file.
- **SOURCE\_KEY** – The identifier used to uniquely identify each inverter.
- **DC\_POWER** – DC power output by the selected 15-minute interval in kilowatts (kW) by the respective inverter.
- **AC\_POWER** - AC power output by the selected 15-minute interval in kilowatts (kW) by the respective inverter.
- **DAILY\_YIELD** – Cumulative sum of power output of the relevant day up to given time by the power plant.
- **TOTAL\_YIELD** – Energy output of the relevant day up to given time by the selected inverter.

DATE_TIME	PLANT_ID	SOURCE_KEY	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD
15-05-2020 00:00	4130001	1819WELC0h9b7	0	0	0	6298059
15-05-2020 00:00	4130001	1P33a7x0U06Y	0	0	0	6183645
15-05-2020 00:00	4130001	3P2a0A020Wc2hD	0	0	0	6087750
15-05-2020 00:00	4130001	719VvLSPdwe4	0	0	0	7020960
15-05-2020 00:00	4130001	Mu0l0eGh9W7Ca	0	0	0	7158964
15-05-2020 00:00	4130001	1P4H0K0h9W7Ca	0	0	0	7206408
15-05-2020 00:00	4130001	W0ng0K7a0P0W0b	0	0	0	7038673
15-05-2020 00:00	4130001	Zu0D0P0U1G0g6	0	0	0	6522172
15-05-2020 00:00	4130001	Zu0D0P0U1G0g6	0	0	0	7086599
15-05-2020 00:00	4130001	a0LQ0D70eN858	0	0	0	6271355
15-05-2020 00:00	4130001	h0N0CH0AD0Dy	0	0	0	6338803
15-05-2020 00:00	4130001	0R0M0R0W0c3	0	0	0	7177882
15-05-2020 00:00	4130001	h0u0K40G0u02	0	0	0	6185184
15-05-2020 00:00	4130001	p0c03g0h0g0b	0	0	0	7189102
15-05-2020 00:00	4130001	0u0L0g0h0P0u0L0V	0	0	0	7111453
15-05-2020 00:00	4130001	0u0L0g0h0P0u0L0V	0	0	0	7016832
15-05-2020 00:00	4130001	u0u0u0Q00W70u0c	0	0	0	7038681
15-05-2020 00:00	4130001	v0J0E0E0E0P0u0c0	0	0	0	6782598
15-05-2020 00:00	4130001	0P0H0T0P0W0u0G	0	0	0	7007966
15-05-2020 00:00	4130001	0B0e0d0r0W0D0Y	0	0	0	6338380
15-05-2020 00:00	4130001	v0P0d0L0P75b0e0	0	0	0	7118151
15-05-2020 00:15	4130001	1819WELC0h9b7	0	0	0	6298059
15-05-2020 00:15	4130001	1P33a7x0U06Y	0	0	0	6183645
15-05-2020 00:15	4130001	3P2a0A020Wc2hD	0	0	0	6087750
15-05-2020 00:15	4130001	719VvLSPdwe4	0	0	0	7020960
15-05-2020 00:15	4130001	Mu0l0eGh9W7Ca	0	0	0	7158964



DATE_TIME	PLANT_ID	SOURCE_KEY	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD
5/15/2020 0:00	4136001	4UPUqMR7TRMqmi	0	0	9425	2429011
5/15/2020 0:00	4136001	81aH1q11NBPmL	0	0	0	1215278796
5/15/2020 0:00	4136001	9kRkVW6d5G4CqJR	0	0	3675.333333	2247718577
5/15/2020 0:00	4136001	EdBgqMD729KT4	0	0	269.9333333	1704250
5/15/2020 0:00	4136001	Q2u7wF4YD6u11Q	0	0	3177	199411526
5/15/2020 0:00	4136001	LYwCqW7kuaH5C2s	0	0	1872.5	1764869634
5/15/2020 0:00	4136001	L172YUhhqhg5Sw	0	0	1094.357143	282508010
5/15/2020 0:00	4136001	M2u2CDyR6DPv	0	0	5682.2	2433648
5/15/2020 0:00	4136001	NgU113aMa2j17u	0	0	1866.2	111512591
5/15/2020 0:00	4136001	Pe6FFyGkUqRNM	0	0	651.2	1348350801
5/15/2020 0:00	4136001	QH4G5u1pJa5T6c6	0	0	0	838421377
5/15/2020 0:00	4136001	Quc111FVW2pV4Wk	0	0	5485	329509985
5/15/2020 0:00	4136001	V54E3Bv11ThEDV	0	0	0	1412083119
5/15/2020 0:00	4136001	WcnaY2VAPhApt	0	0	0	181695261
5/15/2020 0:00	4136001	mqpcP2170TTP	0	0	1238.533333	58350025
5/15/2020 0:00	4136001	v2DkuaaG2Qv	0	0	1281.466667	165965051
5/15/2020 0:00	4136001	v2DkuaaG2Qv	0	0	0	1708083348
5/15/2020 0:00	4136001	q513kA8wDQre	0	0	4315	339923
5/15/2020 0:00	4136001	vuaBwF1gR1yRv	0	0	280.2132857	120964108
5/15/2020 0:00	4136001	v0uHqM2uqLmb	0	0	0	2211962
5/15/2020 0:00	4136001	aPhagap2P78B	0	0	9186	10656621
5/15/2020 0:00	4136001	vauBwF1gR1yRv	0	0	0	299143983
5/15/2020 0:15	4136001	4UPUqMR7TRMqmi	0	0	9425	2429011
5/15/2020 0:15	4136001	81aH1q11NBPmL	0	0	0	1215278796
5/15/2020 0:15	4136001	9kRkVW6d5G4CqJR	0	0	0	2247718577
5/15/2020 0:15	4136001	EdBgqMD729KT4	0	0	0	1704250

“Plant\_1\_Weather\_Sensor\_Data.csv” and “Plant\_2\_Weather\_Sensor\_Data.csv” includes the following data columns.

- **DATE\_TIME** – The observed time and date is mentioned. As the description mentions these data are observed for each 15 minutes.
- **PLANT\_ID** – The unique identifier for the power plant is notated. This attribute will be common per each file.
- **SOURCE\_KEY** – The identifier used to uniquely identify each inverter.
- **AMBIENT\_TEMPERATURE** – The temperature around the power plant at the given time
- **MODULE\_TEMPERATURE** – The temperature which is read by the sensor module of the panel.
- **IRRADIATION** – Amount of solar power received on the surface of the module for the given 15 minutes.

Plant\_1\_Weather\_Sensor\_Data.xlsx

DATE TIME	PLANT_ID	SOURCE_KEY	AMBIENT TEMPE	MODULE TEMPER	IRRADIATION
5/15/2020 0:00	4135001	Hmy02TTLFHqNe	25.18431613	22.8575074	0
5/15/2020 0:15	4135001	Hmy02TTLFHqNe	25.68458667	22.76186787	0
5/15/2020 0:30	4135001	Hmy02TTLFHqNe	24.8357526	22.58320633	0
5/15/2020 0:45	4135001	Hmy02TTLFHqNe	24.8461304	22.36085213	0
5/15/2020 1:00	4135001	Hmy02TTLFHqNe	24.62152536	22.16542264	0
5/15/2020 1:15	4135001	Hmy02TTLFHqNe	24.6386922	21.96637087	0
5/15/2020 1:30	4135001	Hmy02TTLFHqNe	24.63867387	22.35292567	0
5/15/2020 1:45	4135001	Hmy02TTLFHqNe	24.87302233	23.1699102	0
5/15/2020 2:00	4135001	Hmy02TTLFHqNe	24.93859547	23.1026113	0
5/15/2020 2:15	4135001	Hmy02TTLFHqNe	25.012476	23.34229207	0
5/15/2020 2:30	4135001	Hmy02TTLFHqNe	25.00514933	23.62945807	0
5/15/2020 2:45	4135001	Hmy02TTLFHqNe	24.96011953	24.2094658	0
5/15/2020 3:00	4135001	Hmy02TTLFHqNe	25.01630943	24.38413557	0
5/15/2020 3:15	4135001	Hmy02TTLFHqNe	24.98521527	24.35150773	0
5/15/2020 3:30	4135001	Hmy02TTLFHqNe	24.93775193	24.86029653	0
5/15/2020 3:45	4135001	Hmy02TTLFHqNe	24.87909953	23.70979413	0
5/15/2020 4:00	4135001	Hmy02TTLFHqNe	24.6788022	22.58994153	0
5/15/2020 4:15	4135001	Hmy02TTLFHqNe	24.2033598	21.76384253	0
5/15/2020 4:30	4135001	Hmy02TTLFHqNe	24.0626222	21.85253493	0
5/15/2020 4:45	4135001	Hmy02TTLFHqNe	24.032242	22.306315	0
5/15/2020 5:00	4135001	Hmy02TTLFHqNe	24.1771058	22.55190647	0
5/15/2020 5:15	4135001	Hmy02TTLFHqNe	24.364688	22.97948007	0
5/15/2020 5:30	4135001	Hmy02TTLFHqNe	24.32872727	23.45238047	0
5/15/2020 5:45	4135001	Hmy02TTLFHqNe	24.28921113	23.09669193	0.000692721
5/15/2020 6:00	4135001	Hmy02TTLFHqNe	24.68646007	22.2067766	0.004869857
5/15/2020 6:15	4135001	Hmy02TTLFHqNe	24.01163527	22.35345867	0.02291607

Plant\_2\_Weather\_Sensor\_Data.xlsx

DATE TIME	PLANT_ID	SOURCE_KEY	AMBIENT TEMPERATURE	MODULE TEMPERATURE	IRRADIATION
5/15/2020 0:00	4136001	q8h72N4Mmm3u0	27.00476337	25.0607889	0
5/15/2020 0:15	4136001	q8h72N4Mmm3u0	26.88001143	24.42136603	0
5/15/2020 0:30	4136001	q8h72N4Mmm3u0	26.68205534	24.42729031	0
5/15/2020 0:45	4136001	q8h72N4Mmm3u0	26.5005889	24.4206776	0
5/15/2020 1:00	4136001	q8h72N4Mmm3u0	26.596148	25.08021041	0
5/15/2020 1:15	4136001	q8h72N4Mmm3u0	26.51274003	25.31796967	0
5/15/2020 1:30	4136001	q8h72N4Mmm3u0	26.49433897	25.21719253	0
5/15/2020 1:45	4136001	q8h72N4Mmm3u0	26.42641021	25.85058231	0
5/15/2020 2:00	4136001	q8h72N4Mmm3u0	26.40116613	24.69146937	0
5/15/2020 2:15	4136001	q8h72N4Mmm3u0	26.22607821	24.55948079	0
5/15/2020 2:30	4136001	q8h72N4Mmm3u0	26.2603992	24.48240603	0
5/15/2020 2:45	4136001	q8h72N4Mmm3u0	26.2979261	24.4869862	0
5/15/2020 3:00	4136001	q8h72N4Mmm3u0	26.3282491	24.50628962	0
5/15/2020 3:15	4136001	q8h72N4Mmm3u0	26.11670973	24.59519743	0
5/15/2020 3:30	4136001	q8h72N4Mmm3u0	26.07849933	24.80131437	0
5/15/2020 3:45	4136001	q8h72N4Mmm3u0	26.00130397	24.60781745	0
5/15/2020 4:00	4136001	q8h72N4Mmm3u0	25.6972106	24.626179	0
5/15/2020 4:15	4136001	q8h72N4Mmm3u0	25.62770241	24.77214507	0
5/15/2020 4:30	4136001	q8h72N4Mmm3u0	25.48204579	24.79689075	0
5/15/2020 4:45	4136001	q8h72N4Mmm3u0	25.23093169	24.43960426	0
5/15/2020 5:00	4136001	q8h72N4Mmm3u0	25.1175996	24.10437963	0
5/15/2020 5:15	4136001	q8h72N4Mmm3u0	25.0918621	23.897232	0
5/15/2020 5:30	4136001	q8h72N4Mmm3u0	24.91696783	23.83947131	0
5/15/2020 5:45	4136001	q8h72N4Mmm3u0	24.7412736	23.786618	0.00236015
5/15/2020 6:00	4136001	q8h72N4Mmm3u0	24.74276637	24.67722653	0.012961976
5/15/2020 6:15	4136001	q8h72N4Mmm3u0	24.7829109	24.35618545	0.021537669

## 4. Methodology

### 4.1 The Approach and Justification

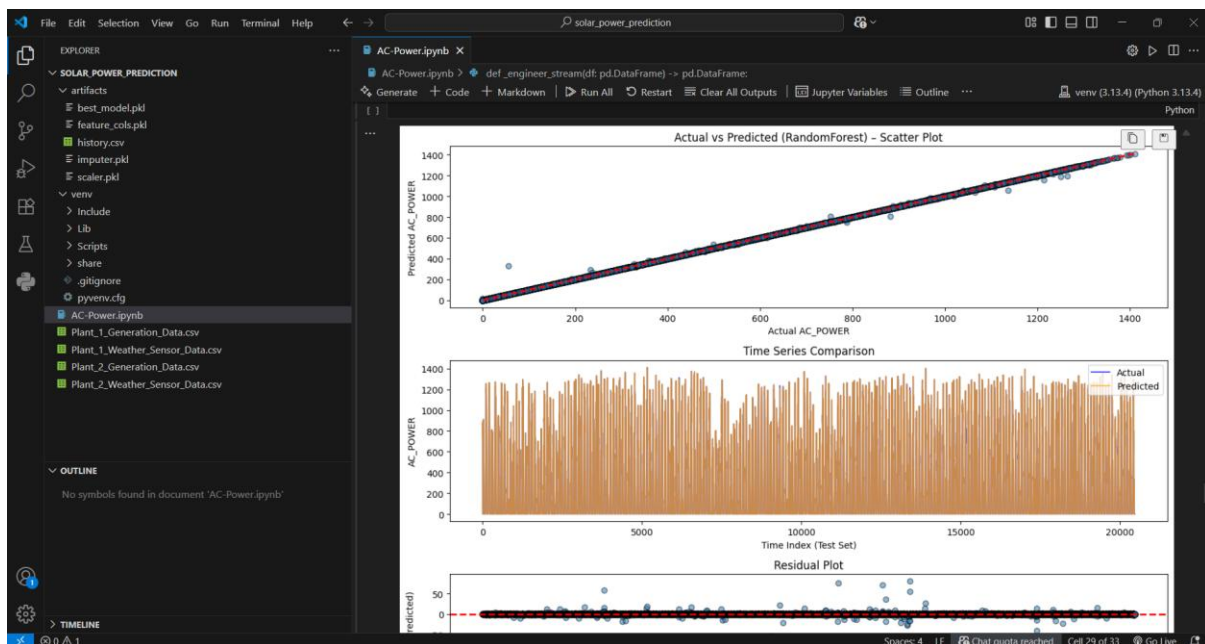
There are three machine language methodologies used namely,

- Supervised learning – where the model trained from a dataset with both input features and corresponding outputs. It is also outputs continues real world numbers as predictions.
- Unsupervised learning – the mechanism works on datasets where no labels are presented. This leads to learn hidden patterns and relationships in the data points for exploring key insights.
- Reinforcement learning - an agent is used for learning the model and make insights with the contribution of a surrounding with the aim of achieving the goal.

Among these mechanisms, our model uses **Supervised learning**. The reason for using Supervised learning is that the data source itself mentions the AC power as a field. Since our model is trained to predict the AC power real-time and many days ahead, supervised works as the best approach to train the model.

## 4.2 The Model

The trained model can be explained by dividing it into following sections. Those are provided with a clear understanding of methodologies and concepts with the reasonings, justification and theoretical explanations. Since the model has been trained in **VS code – Jupiter notebook**, each cell will be explained in following chapters.



The GitHub Repository :

[https://github.com/Ilmaa2003/Solar\\_Power\\_Prediction\\_using\\_Machine\\_Language](https://github.com/Ilmaa2003/Solar_Power_Prediction_using_Machine_Language)

### 4.2.1 Importing Tools and Libraries

```
> ~
import os
import warnings
warnings.filterwarnings("ignore")

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score, mean_squared_error

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor

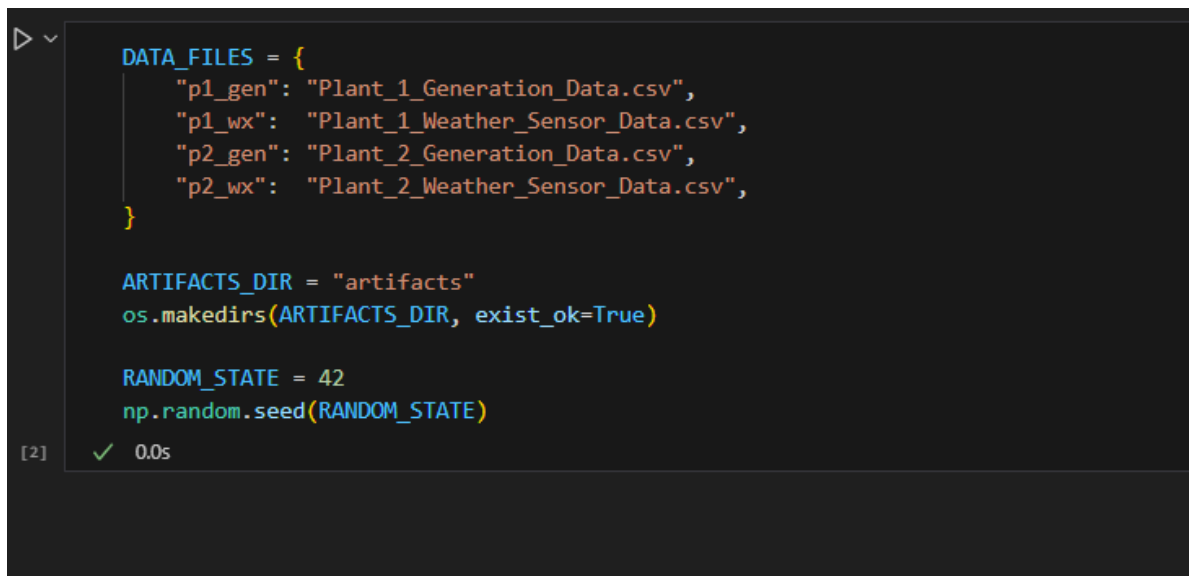
[1] ✓ 8.2s
```

As the initial step, the required libraries for training the model have been imported. The required libraries and relevant usages are as follows.

- **os** - gains the access of commands by the Operating system of the computer.
- **warnings** – since python shows some harmless warnings, this is used to import them and hide in order to make the output look cleaner.
- **numpy** – used when python performs mathematical operations and array related operations. In our case it is used when the model requires to perform calculations, array related operations and handling statistical values such as standard deviations, means and Nans.
- **pandas** – used when dealing with tabular formatted dataset in python. In our model, when the model reads data from CSV format files and merging them together to create a large dataset, this library is used. This also required when performing feature engineering as well.
- **joblib** – required to deal with python objects such as scalers, imputers and saving the .pkl files.

- **matplotlib** – this is a library required when visualizing the data. Some of the supported data visualizations are bar charts, scatter plots which are used to represent the relationships and patterns over factors.
- **sklearn (scikit-learn)** – the library is responsible for providing required tools needed in model training , evaluating , selecting .... throughout the process. Some of those tools are ,
  - ✓ For model selection for training and testing – **train\_test\_split**
  - ✓ For scaling the model when preprocessing - **StandardScaler**
  - ✓ For missing value handling when preprocessing - **SimpleImputer**
  - ✓ For evaluating the model to define metrics such as – **r2\_score**, **mean\_squared\_error**
  - ✓ For working with algorithms – **LinearRegression**, **DecisionTreeRegressor**, **RandomForestRegressor**, **GradientBoostingRegressor**, **MLPRegressor**, **KNeighborsClassifier**

#### 4.2.2 Loading and Merging the Dataset



```
DATA_FILES = {
    "p1_gen": "Plant_1_Generation_Data.csv",
    "p1_wx": "Plant_1_Weather_Sensor_Data.csv",
    "p2_gen": "Plant_2_Generation_Data.csv",
    "p2_wx": "Plant_2_Weather_Sensor_Data.csv",
}

ARTIFACTS_DIR = "artifacts"
os.makedirs(ARTIFACTS_DIR, exist_ok=True)

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
```

[2] ✓ 0.0s

**DATA\_FILES** section stores all the data files in the same place for easy accessibility . Next it creates a directory named “**artifacts**” in-order-to save all the outputs. The outputs stand for the visualized plots, models, objects etc... **RANDOM\_STATE** is the variable stored to

ensure the accuracy of model when randomly chosen data are utilized. It ensures the reproducibility of the model.

```
def load_and_merge(gen_path: str, wx_path: str) -> pd.DataFrame:

    gen = pd.read_csv(gen_path)
    wx = pd.read_csv(wx_path)

    gen["DATE_TIME"] = pd.to_datetime(gen["DATE_TIME"], errors="coerce")
    wx["DATE_TIME"] = pd.to_datetime(wx["DATE_TIME"], errors="coerce")

    wx_keep = ["PLANT_ID", "DATE_TIME", "AMBIENT_TEMPERATURE", "MODULE_TEMPERATURE", "IRRADIATION"]
    wx = wx[wx_keep]

    df = pd.merge(gen, wx, on=["PLANT_ID", "DATE_TIME"], how="left")

    return df
```

[3] ✓ 0.0s

This function is used to loading and merging the dataset together to create a dataset where it has both weather – sensor data and power generation data based on the date , time and unique plant ID. The methodology stands with following steps.

- Loading the data points from datasets.
- Converting the **DATE\_TIME** into datetime type.
- For invalid dates, passes a value as **NaT** which stands for “Not a Time”
- Filter only required columns from weather related datasets. In our case we select only,
  - ✓ DATE\_TIME
  - ✓ PLANT\_ID
  - ✓ AMBIENT\_TEMPERATURE
  - ✓ MODULE\_TEMPERATURE
  - ✓ IRRADIATION
- Then the data files are merged into one using **PLANT\_ID** and **DATE\_TIME** columns.

```
print("Loading & merging...")
plant1 = load_and_merge(DATA_FILES["p1_gen"], DATA_FILES["p1_wx"])
plant2 = load_and_merge(DATA_FILES["p2_gen"], DATA_FILES["p2_wx"])

[4] ✓ 0.2s
... Loading & merging...
```

Next the above function is applied to the two plants separately. As the results it creates two datasets for two power plants with the combination of power generation and weather data.

```
raw = pd.concat([plant1, plant2], ignore_index=True)
raw = raw.sort_values(["SOURCE_KEY", "DATE_TIME"]).reset_index(drop=True)
print(f"Combined shape: {raw.shape}")

[5] ✓ 0.0s
... Combined shape: (136476, 10)
```

In-order-to convert the merged two datasets into a single dataset, the new two datasets were combined using the **SOURCE\_KEY** and **DATE\_TIME** columns. Shape of the dataset stands for the rows by column value. It has output as the final merged dataset includes 136476 rows and 10 columns.

### 4.2.3 Exploratory Data Analysis (EDA)

This is a mechanism used in machine learning for getting an idea of the dataset before we are preprocessing the data. This mechanism helps for

- Getting a clear idea of datasets such as columns, rows and counts of each
- Gaining a statistical summary of the dataset
- Checking for missing values



- Visualizations to get an idea of how certain variables are distributed, relationships in between them and trends across the time.

#### 4.2.3.1 Basics and Statistical Summary

```
print("First 5 rows:")
display(raw.head())

print("Last 5 rows:")
display(raw.tail())

print(f"Dataset shape: {raw.shape}")

print("\nColumn names:")
print(list(raw.columns))

print("\nData info:")
raw.info()

print("\nStatistical summary:")
display(raw.describe())
```

```
... First 5 rows:
...
  DATE_TIME  PLANT_ID  SOURCE_KEY  DC_POWER  AC_POWER  DAILY_YIELD  TOTAL_YIELD  AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  IRRADIATION
0  2020-05-15 00:00:00  4135001  18Y6WecLGH8j5v7  0.0  0.0  0.0  6259559.0  25.184316  22.857507  0.0
1  2020-05-15 00:15:00  4135001  18Y6WecLGH8j5v7  0.0  0.0  0.0  6259559.0  25.084589  22.781668  0.0
2  2020-05-15 00:30:00  4135001  18Y6WecLGH8j5v7  0.0  0.0  0.0  6259559.0  24.935753  22.592306  0.0
3  2020-05-15 00:45:00  4135001  18Y6WecLGH8j5v7  0.0  0.0  0.0  6259559.0  24.846130  22.360852  0.0
4  2020-05-15 01:00:00  4135001  18Y6WecLGH8j5v7  0.0  0.0  0.0  6259559.0  24.621525  22.165423  0.0
...
... Last 5 rows:
...
  DATE_TIME  PLANT_ID  SOURCE_KEY  DC_POWER  AC_POWER  DAILY_YIELD  TOTAL_YIELD  AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  IRRADIATION
136471  2020-06-17 22:45:00  4135001  zVJPv84UY57bAof  0.0  0.0  5910.0  7363272.0  22.150570  21.480377  0.0
136472  2020-06-17 23:00:00  4135001  zVJPv84UY57bAof  0.0  0.0  5910.0  7363272.0  22.129816  21.389024  0.0
136473  2020-06-17 23:15:00  4135001  zVJPv84UY57bAof  0.0  0.0  5910.0  7363272.0  22.008275  20.709211  0.0
136474  2020-06-17 23:30:00  4135001  zVJPv84UY57bAof  0.0  0.0  5910.0  7363272.0  21.969495  20.734963  0.0
136475  2020-06-17 23:45:00  4135001  zVJPv84UY57bAof  0.0  0.0  5910.0  7363272.0  21.909288  20.427972  0.0
...
Dataset shape: (136476, 10)

Column names:
['DATE_TIME', 'PLANT_ID', 'SOURCE_KEY', 'DC_POWER', 'AC_POWER', 'DAILY_YIELD', 'TOTAL_YIELD', 'AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE', 'IRRADIATION']

Data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 136476 entries, 0 to 136475
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype
---  ---
0  DATE_TIME             136476 non-null  datetime64[ns]
1  PLANT_ID              136476 non-null  int64
2  SOURCE_KEY            136476 non-null  object
3  DC_POWER              136476 non-null  float64
4  AC_POWER              136476 non-null  float64
5  DAILY_YIELD           136476 non-null  float64
6  TOTAL_YIELD           136476 non-null  float64
7  AMBIENT_TEMPERATURE    136472 non-null  float64
8  MODULE_TEMPERATURE    136472 non-null  float64
9  IRRADIATION            136472 non-null  float64
dtypes: datetime64[ns](1), float64(7), int64(1), object(1)
memory usage: 10.4+ MB

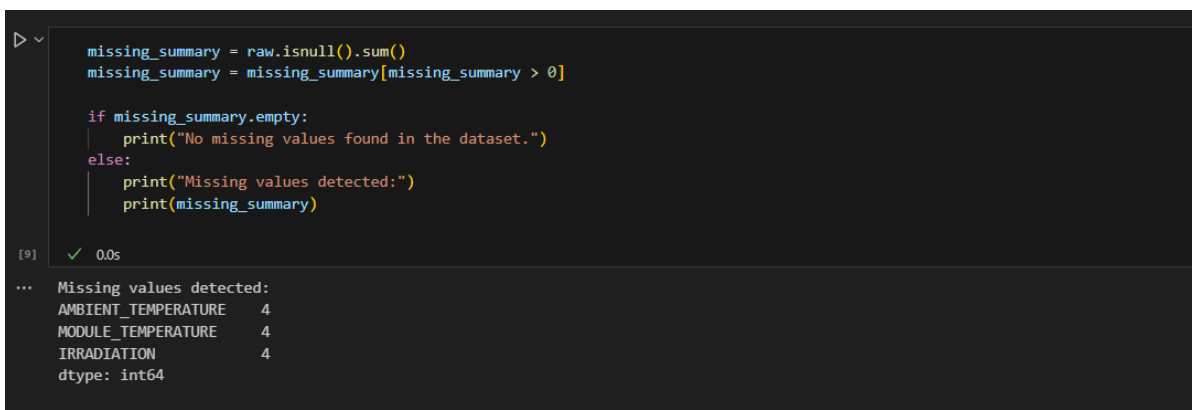
Statistical summary:
...
  count  DATE_TIME  PLANT_ID  DC_POWER  AC_POWER  DAILY_YIELD  TOTAL_YIELD  AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  IRRADIATION
mean  2020-06-01 09:23:03.157477888  4.135497e+06  1708.541497  274.803511  3295.433783  3.303821e+08  26.763086  31.920744  0.230767
min    2020-05-15 00:00:00  4.135001e+06  0.000000  0.000000  0.000000  0.000000e+00  20.398505  18.140415  0.000000
25%    2020-05-23 23:00:00  4.135001e+06  0.000000  0.000000  28.321429  6.520020e+06  23.637804  22.411098  0.000000
50%    2020-06-01 18:45:00  4.135001e+06  6.050000  3.506905  2834.803572  7.269333e+06  25.908122  26.413755  0.026213
75%    2020-06-09 21:45:00  4.136001e+06  1155.788333  532.673333  5992.000000  2.826096e+08  29.266583  40.778583  0.442961
max    2020-06-17 23:45:00  4.136001e+06  14471.125000  1410.950000  9873.000000  2.247916e+09  39.181638  66.635953  1.221652
std      NaN  4.999862e+02  3222.181273  380.182569  3035.294425  6.085705e+08  3.897340  3.897340  0.305652
```

In order to get a basic understanding of the dataset,

- `head()` – used to output the first five rows
- `tail()` – used to output the last five rows
- `shape` – for output the row and column count
- `columns` – for output the columns in the dataset
- `info()` – this function outputs the information related to each column such as data type, non-Null count
- `describe()` – for statistical analysis of the dataset such as,
  - ✓ `min` – minimum value of each column
  - ✓ `max` - maximum value of each column
  - ✓ `mean` – average value of each column in dataset
  - ✓ `count` – No of rows in each column
  - ✓ `standard deviation` – Standard deviation value of each column
  - ✓ `percentiles` – it stands for 25%, 75%, 50% value in each column

These values provide a basic summary of the dataset. It works as the foundation for data preprocessing, evaluation and training of the dataset.

#### 4.2.3.2 Exploring and handling Missing Values



```
missing_summary = raw.isnull().sum()
missing_summary = missing_summary[missing_summary > 0]

if missing_summary.empty:
    print("No missing values found in the dataset.")
else:
    print("Missing values detected:")
    print(missing_summary)
```

[9] ✓ 0.0s

```
... Missing values detected:
AMBIENT_TEMPERATURE    4
MODULE_TEMPERATURE     4
IRRADIATION            4
dtype: int64
```

The code block works for finding missing values in each column of the dataset. It outputs the columns with missing values with the count. According to the output, our model had missing values in 03 columns where the datatype is **int64**.

```
▶ # Forward-fill missing values
raw = raw.fillna(method="ffill")

# Check any missing values
print("Missing values after forward fill:")
print(raw.isnull().sum())

[10] ✓ 0.0s

... Missing values after forward fill:
DATE_TIME      0
PLANT_ID        0
SOURCE_KEY      0
DC_POWER        0
AC_POWER        0
DAILY_YIELD     0
TOTAL_YIELD     0
AMBIENT_TEMPERATURE  0
MODULE_TEMPERATURE  0
IRRADIATION     0
dtype: int64
```

Once the missing values are explored, it should be handled in-order-to make the dataset accurately for prediction of solar power with minimal errors. There are many approaches where the missing values can be handled. Some of them are,

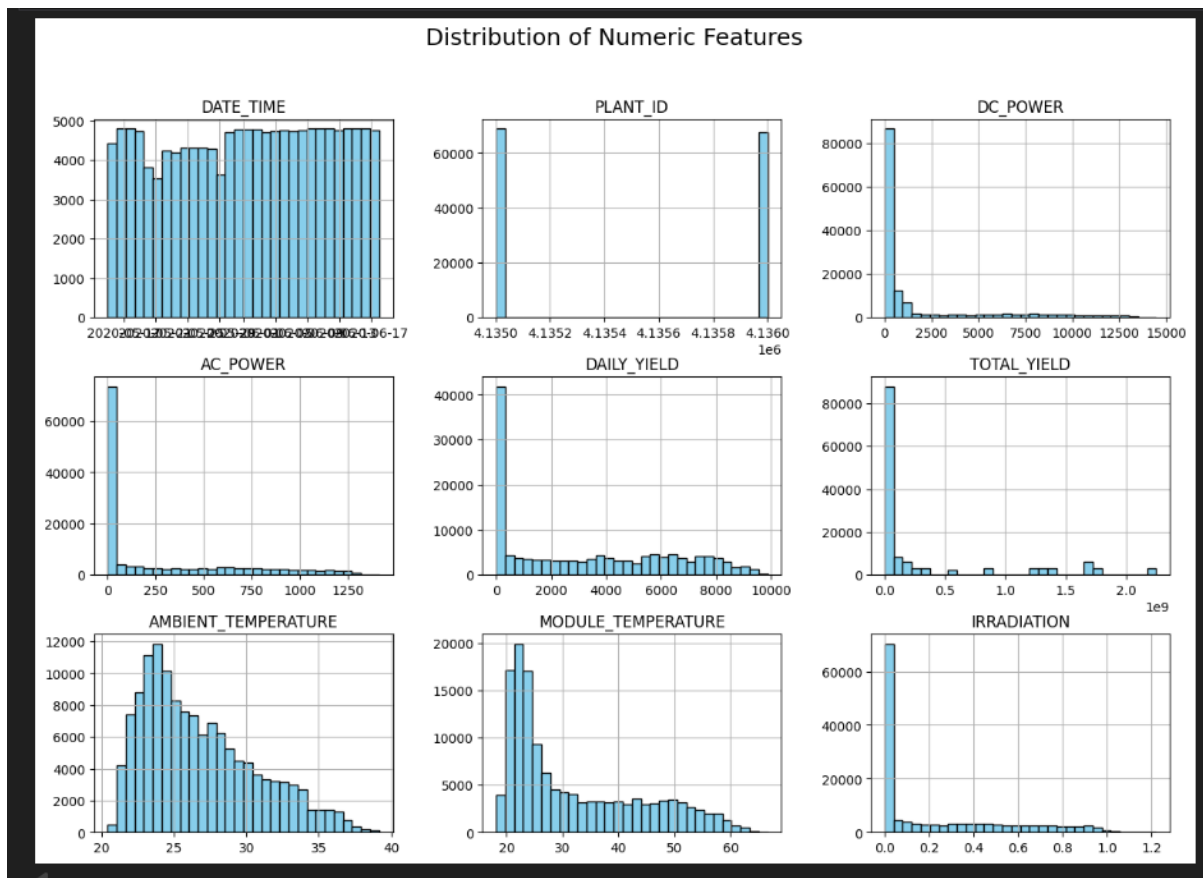
- Forward Fill – missing values are gained from last filled values
- Backword Fill - missing values are gained from next filled values
- Dropping – dropping the rows where the missing values are included
- Imputation - the missing values represented based on the mean, median or mode of the relevant column

Among the above, technique we have used is **Forward Filling (ffill)** since the dataset is a 15-minute time-based and it can satisfy with previous readings. Those values can be considered reasonable for the upcoming values.

After handling the missing values, again checked the columns in-order-to ensure that there is no nulls.

### 4.2.3.3 Distribution of Data

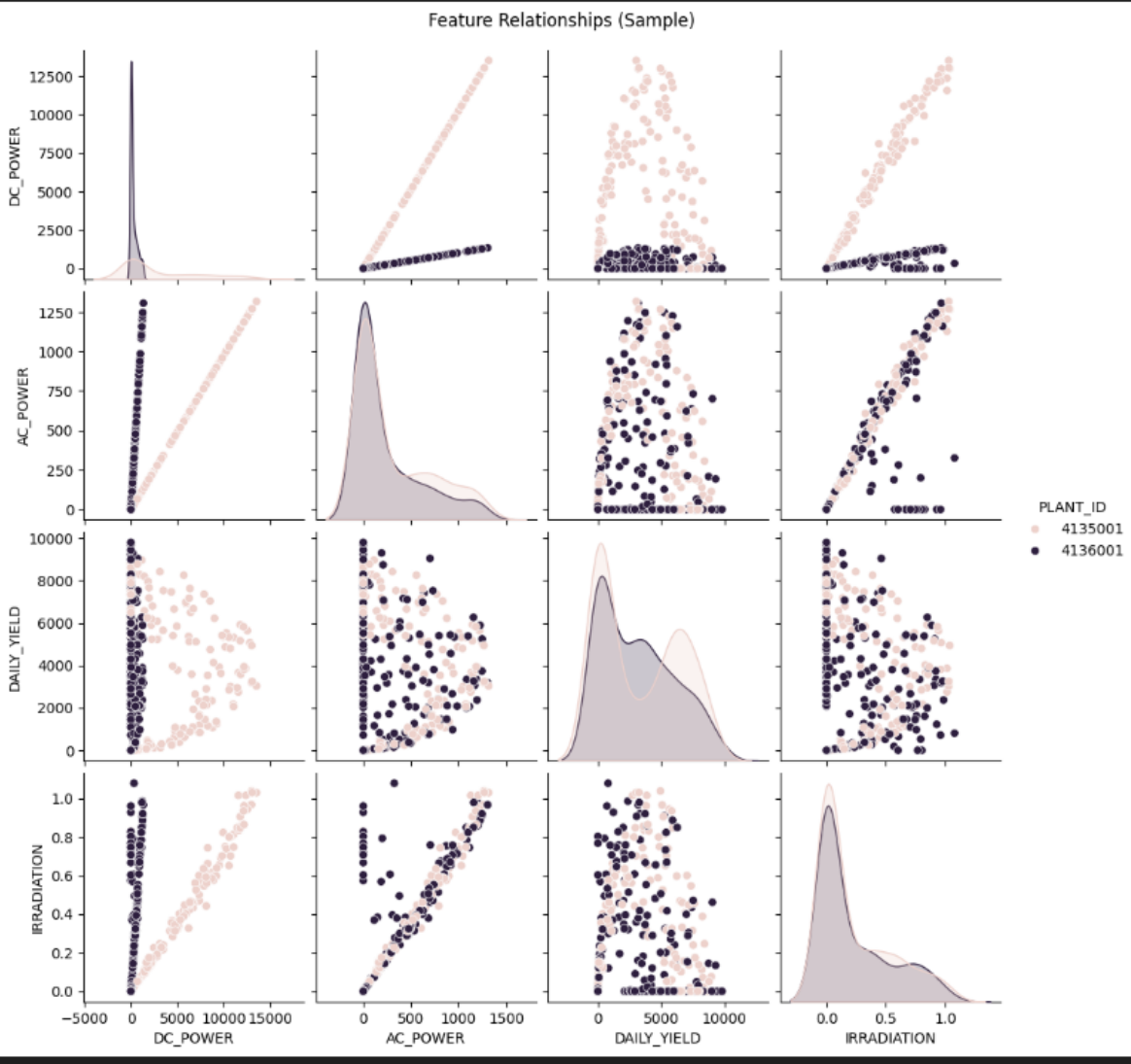
```
raw.hist(figsize=(15,10), bins=30, color="skyblue", edgecolor="black")  
plt.suptitle("Distribution of Numeric Features", fontsize=18)  
plt.show()  
[11] ✓ 0.7s
```



Then the insights and data were mapped into a histogram, where it represents the distribution of data. It visually shows how the data are distributed in overall and clearly shows the outliers, skewedness and variability of the data columns in the dataset. The distribution works for all the numerical values in the dataset to have an overall idea.

#### 4.2.3.4 Relationships between Features

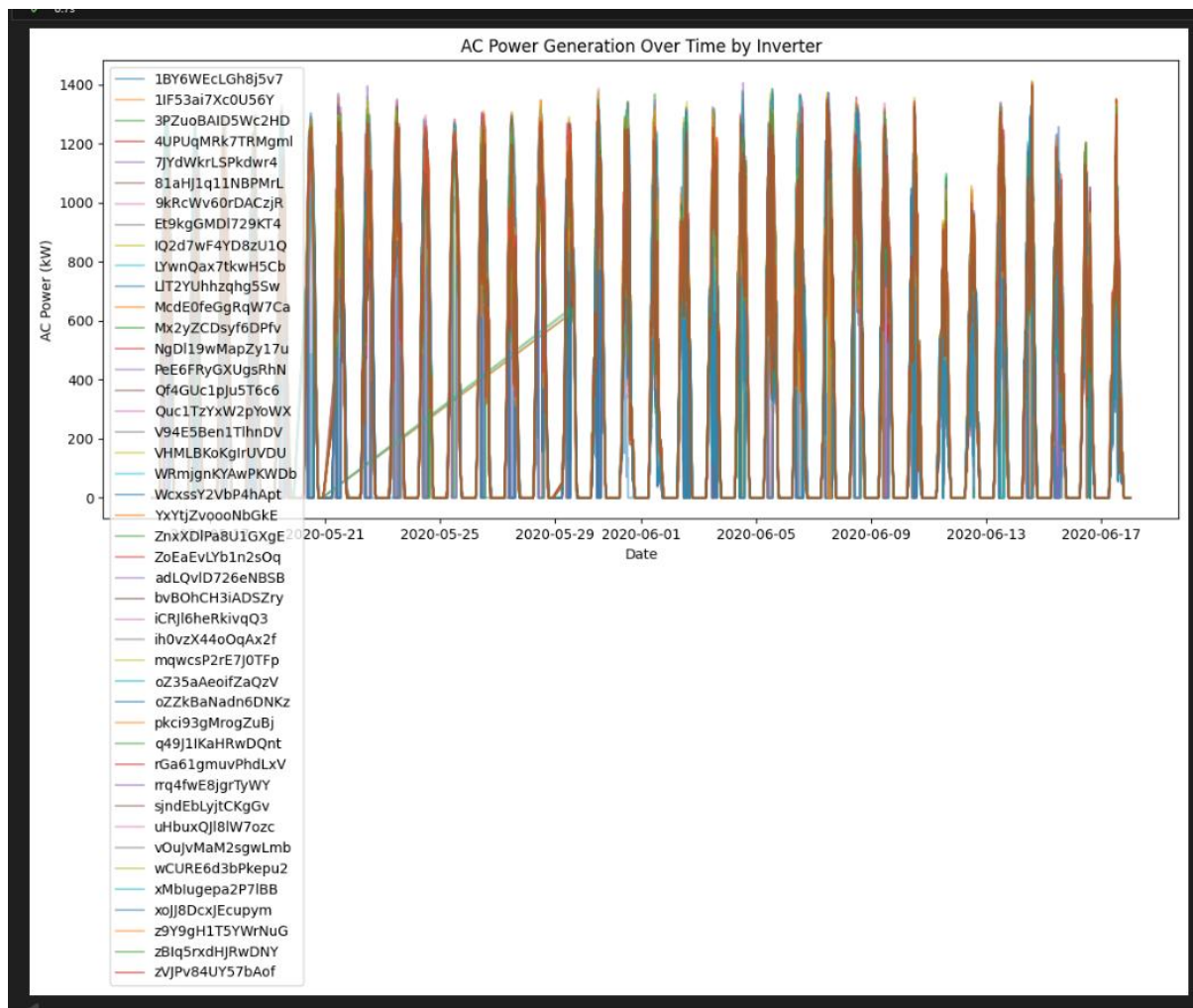
```
sns.pairplot(raw.sample(500), vars=["DC_POWER", "AC_POWER", "DAILY_YIELD", "IRRADIATION"], hue="PLANT_ID")  
plt.suptitle("Feature Relationships (Sample)", y=1.02)  
plt.show()
```



The above plots show how the correlation between following features look when mapped across each other and based on the selected 02 power plants with different colours. It depicts the clear difference of relationship between the given features of both power plants. Those features are DC\_POWER, AC\_POWER, DAILY\_YIELD and IRRADIATION.

```
plt.figure(figsize=(14,6))
for key, group in raw.groupby("SOURCE_KEY"):
    plt.plot(group["DATE_TIME"], group["AC_POWER"], label=key, alpha=0.5)
plt.xlabel("Date")
plt.ylabel("AC Power (kW)")
plt.title("AC Power Generation Over Time by Inverter")
plt.legend()
plt.show()
```

✓ 0.7s



The given plot also a significant visualization because it shows the AC power output of each inverter over the date and time of the day. It helps to identify any anomalies and performance issues related to the inverters successfully. Also, it provides a power generation summary as well.

## 4.2.4 Feature Engineering

“Feature Engineering” stands for converting raw data columns in a dataset to meaningful features which can be used by a machine learning model to understand and learn through the model accurately and efficiently by recognizing the hidden patterns.

### 4.2.4.1 Applying Feature Engineering

```
def make_features(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()

    # Extract hour of day & date from timestamp
    df["HOUR"] = df["DATE_TIME"].dt.hour
    df["DATE"] = df["DATE_TIME"].dt.date

    # Rolling statistics per inverter (SOURCE_KEY). Assuming 15-min sampling rate.
    df["DC_POWER_ROLLING_MEAN_1H"] = (
        df.groupby("SOURCE_KEY")["DC_POWER"].transform(lambda x: x.rolling(4, min_periods=1).mean())
    )
    df["DC_POWER_ROLLING_STD_1H"] = (
        df.groupby("SOURCE_KEY")["DC_POWER"].transform(lambda x: x.rolling(4, min_periods=1).std())
    )
    df["IRRADIATION_ROLLING_MAX_3H"] = (
        df.groupby("SOURCE_KEY")["IRRADIATION"].transform(lambda x: x.rolling(12, min_periods=1).max())
    )

    # Lag features (previous interval values)
    df["DC_POWER_LAG_1"] = df.groupby("SOURCE_KEY")["DC_POWER"].shift(1)
    df["IRRADIATION_LAG_1"] = df.groupby("SOURCE_KEY")["IRRADIATION"].shift(1)

    # Daily cumulative yield (resets per inverter each day)
    df["DAILY_YIELD_CUMSUM"] = df.groupby(["SOURCE_KEY", "DATE"])["DAILY_YIELD"].cumsum()

    # Interaction term + efficiency metric
    df["IRRADIATION_x_AMBIENT_TEMP"] = df["IRRADIATION"] * df["AMBIENT_TEMPERATURE"]
    eps = 1e-6
    df["EFFICIENCY"] = df["DC_POWER"] / (df["IRRADIATION"] + eps)

    return df
```

17] ✓ 0.0s

```
# Apply the feature creation to cleaned dataset
feat_df = make_features(raw)
```

3] ✓ 0.2s

In terms of feature engineering, our model performs the above function. The following features are created through feature engineering using the already available data.

1. HOUR

- By extracting the hour value from the DATE\_TIME column in dataset

2. DATE

- By extracting the date value from the DATE\_TIME column in dataset

3. DC\_POWER\_ROLLING\_MEAN\_1H

- Calculated by rolling the mean of DC power of four rows together since one row shows 15 minutes. It helps to enhance the smoothness of the data flow since it hides the random up and downs.

4. DC\_POWER\_ROLLING\_STD\_1H

- Calculated by rolling the mean of DC power of four rows together since one row shows 15 minutes. Instability of data can be identified by high rolling standard deviation values.

5. IRRADIATION\_ROLLING\_MAX\_3H

- Calculated by 04 rows in IRRADIATION column since one row calculates the 15 minutes. It allows to explore which hour's sunlight detection to the panel is high.

6. DC\_POWER\_LAG\_1

- Uses the previous timing value to get to know about the dependency.

7. IRRADIATION\_LAG\_1

- Uses the previous timing value to get to know about the dependency.

8. DAILY\_YIELD\_CUMSUM

- Calculated cumulative sum for each day per the inverter.

9. IRRADIATION\_x\_AMBIENT\_TEMP



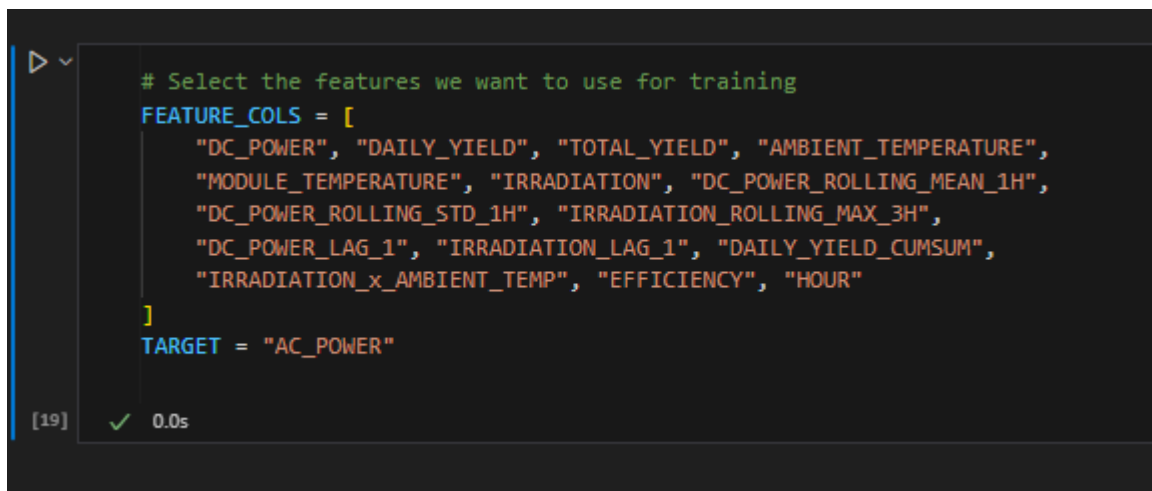
- This is the multiplied value of sunlight detected by panel and surrounding temperature. Multiplied answer of these values provide an idea of how the overall temperature affects the power production based on low and high temperatures in the environment.

#### 10. EFFICIENCY

- Efficiency of the power output is measured by dividing the DC power output and Irradiation. It detects how much of sunlight is converted to power.

For performing the feature engineering, we have also created a copy of the dataset to avoid any damage to original dataset.

#### 4.2.4.2 Defining the Features and Target Variable



```
# Select the features we want to use for training
FEATURE_COLS = [
    "DC_POWER", "DAILY_YIELD", "TOTAL_YIELD", "AMBIENT_TEMPERATURE",
    "MODULE_TEMPERATURE", "IRRADIATION", "DC_POWER_ROLLING_MEAN_1H",
    "DC_POWER_ROLLING_STD_1H", "IRRADIATION_ROLLING_MAX_3H",
    "DC_POWER_LAG_1", "IRRADIATION_LAG_1", "DAILY_YIELD_CUMSUM",
    "IRRADIATION_x_AMBIENT_TEMP", "EFFICIENCY", "HOUR"
]
TARGET = "AC_POWER"
```

[19] ✓ 0.0s

After feature engineering performed, the finalized features (inputs) which are planned to be used in final model training are captured in to **FEATURE\_COLS**. It includes all the features which were selected and newly featured variables together.

The **TARGET** represents, the value what is planned to be predicted using the model.

### 4.2.5 Data Preprocessing

“Data Preprocessing” is an important mechanism followed in machine learning where it prepares the data before the model is trained. It performs functions such as,

- Missing value handling
- Scaling the features
- Splitting the dataset to train, test and validate
- Outlier detection

This creates a clean feature set and data where the modelling process is smooth and accurate.

#### 4.2.5.1 Handling Missing Values by dropping columns

```
model_df = feat_df.dropna(subset=[TARGET, "DC_POWER_LAG_1", "IRRADIATION_LAG_1"]).reset_index(drop=True)
```

[21] ✓ 0.0s

Since the DC\_POWER\_LAG\_1 and IRRADIATION\_LAG\_1 are feature engineered columns using combination of four consecutive data points, there can be still missing values obviously. This code is responsible for dropping the rows where these values are missing (NaNs).

#### 4.2.5.2 Splitting the Dataset

```
X = model_df[FEATURE_COLS]  
Y = model_df[TARGET]
```

[22] ✓ 0.0s

Here the FEATURE\_COLS and TARGET variables are assigned to X and Y variables for making the model ready for training. X stands for inputs while Y stands for target output.

```
> ✓  
N = len(model_df)  
tr_end = int(N * 0.70)  
va_end = int(N * 0.85)  
[23] ✓ 0.0s
```

```
▶ ✓  
X_train, y_train = X.iloc[:tr_end], Y.iloc[:tr_end]  
X_valid, y_valid = X.iloc[tr_end:va_end], Y.iloc[tr_end:va_end]  
X_test, y_test = X.iloc[va_end:], Y.iloc[va_end:]  
[28] ✓ 0.0s
```

Then the dataset is split to three parts where it includes training, testing and validating the model. Those splits stand for following functionalities.

- Training data – used to train the model. It is 70% of the dataset.
- Validating data – for finetuning the hyper parameters of the model. It is 15% of the dataset.
- Testing data - for evaluating the dataset. It included 15% of the dataset.

Splitting the dataset is done with no biasness. Since the dataset is based on the time, it is clear that the earlier datapoints are involved in training while recent data points involved to validation and testing.

#### 4.2.5.3 Handling Missing Values by Imputation

```
imputer = SimpleImputer(strategy="mean")
X_train_imp = imputer.fit_transform(X_train)
X_valid_imp = imputer.transform(X_valid)
X_test_imp = imputer.transform(X_test)
```

[29] ✓ 0.0s

This is another technique where the missing values can be replaced by statistical summaries such as mean, median, mode etc... Since the dataset is based on a solar power plant some data can be recorded with missing values where the missing of values may make inaccuracy. In such cases, applying imputations make the dataset more efficient.

#### 4.2.5.4 Scaling the Dataset

```
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train_imp)
X_valid_sc = scaler.transform(X_valid_imp)
X_test_sc = scaler.transform(X_test_imp)
```

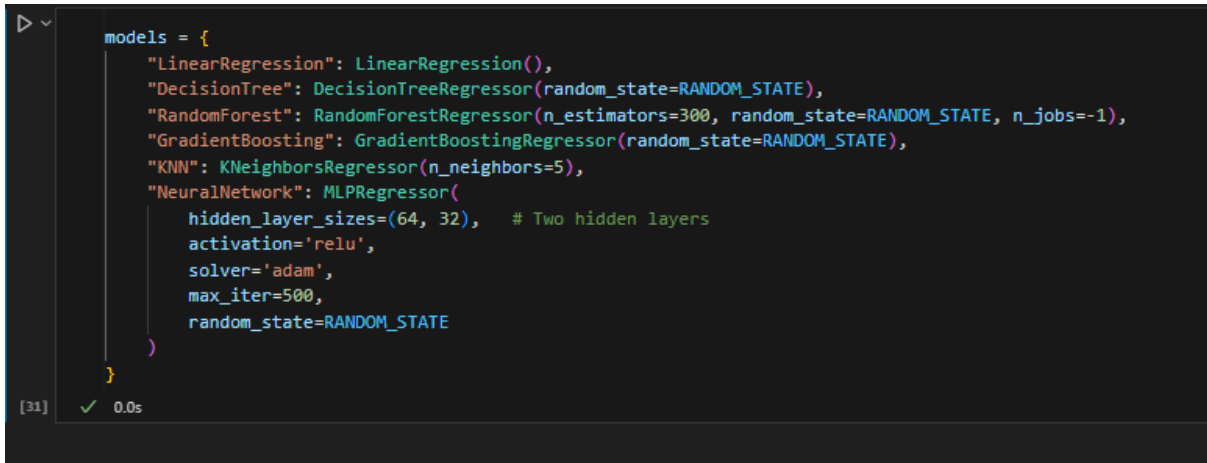
[30] ✓ 0.0s

Scaling the dataset is also a significant process since the values are centralized to a similar range. It makes the models to perform the training by enhancing the accuracy, speed and efficiency.

## 4.2.6 The Model Training

### 4.2.6.1 Defining the models

After Preprocessing the data which were planned to train, in-order-to find the best model among them many algorithms were trained and saved in a python dictionary.



```
models = {  
    "LinearRegression": LinearRegression(),  
    "DecisionTree": DecisionTreeRegressor(random_state=RANDOM_STATE),  
    "RandomForest": RandomForestRegressor(n_estimators=300, random_state=RANDOM_STATE, n_jobs=-1),  
    "GradientBoosting": GradientBoostingRegressor(random_state=RANDOM_STATE),  
    "KNN": KNeighborsRegressor(n_neighbors=5),  
    "NeuralNetwork": MLPRegressor(  
        hidden_layer_sizes=(64, 32), # Two hidden layers  
        activation='relu',  
        solver='adam',  
        max_iter=500,  
        random_state=RANDOM_STATE  
    )  
}
```

[31] ✓ 0.0s

According to above code block it is planned to train following models.

- Linear Regression
- Decision Tree
- Random Forest
- Gradient Boosting
- KNN
- Neural Network

The way how the above models work will be discussed below.

#### 4.2.6.1.1 Linear Regression

The commonly used regression algorithm used to develop machine language models. It outputs real world numbers as labels (predicted values). It maps a relationship between the inputs and the label using a formula to calculate and visualize the accuracy. There are two types of regression algorithms namely Multi linear regression and single linear regression

where single linear regression allows to map one input to an output while the multi linear regression allows to map multiple inputs with the output.

In our model we have trained a multi linear regression model. The reasons why a multi linear regression model suits are,

- Model is simple to understand by anyone
- It does not consume much time for training the model
- This gives a quick overview of the relationship between variables can be taken

#### **4.2.6.1.2 Decision Tree**

This is a classification-based model where the inputs are categorised based on the conditions to derive at a final answer. Each node in the decision tree is responsible for a condition and in that manner the final prediction is concluded.

As one of the models we have also trained a decision tree. The reasons for choosing such a model are given below.

- It helps to capture the non-linear relationship between features
- Detects the thresholds of data well to understand outliers well
- Easy understandability

#### **4.2.6.1.3 Random Forest**

This is a machine learning algorithm where multiple decision trees are interconnected, and more accurate decisions are made. The model works as bellows.

- If the predicting output is a label (classification), majority predicted label is taken.
- If the predicting output is a numerical value (regression), it gets the average value of all final predicted values from trees.

In our solar power predicting model we used this algorithm because,

- The outputs are highly accurate since each decision tree captures each aspect of the model.
- It handles the major problems such as overfitting and underfitting of the model
- Captures non-linear relationships among inputs
- Allows to create a hierarchy based on importance of each input (feature)

#### **4.2.6.1.4 Gradient Boosting**

This machine learning algorithm is based on a sequential decision tree where multiple decision trees are interconnected, without training them independently. The model works in following steps.

- First a decision tree model is trained simply.
- Detects the errors by comparing the predicted value and actual labels.
- Then a new tree is modelled by fixing that residual error.
- The above three steps are repeated until the expected requirement is met.

This fits our solar power predicting model since it has following significant benefits.

- Detects the importance of inputs over each other
- A successful accurate type of model
- suitable for any kind of dataset whether is it larger or smaller

Even it is beneficial in above ways, it also takes time for training because it is too much sensitive when it comes to the process of tuning.

#### **4.2.6.1.5 KNN**

KNN stands for “K Nearest Neighbours” where the predictive value is taken without a model but only considering the nearest K number of data points from the training dataset. Usually, the average of previous labels is used for the prediction purpose.

The model has below characteristics and benefits.

- Easy implementation and understandability
- No complex algorithms to practice

It has some drawbacks also. They are,

- Takes too much time to train
- Irrelevancy in some cases
- K number of datapoints may contain results where it goes under outliers where it causes inaccurate results.

#### **4.2.6.1.6 Neural Network**

One of the most powerful modelling algorithms used in machine learning where it is inspired by biological neuron structure. The neural network structure includes three main layers called,

1. Input layers
2. Hidden layers
3. Output layers

The input layer is responsible for receiving the input features required to make predictions. The hidden layers are responsible of processing using the summation and activation functions. There can be one or more hidden layers based on the requirement and complexity of the model. The output layer is responsible for predicting label value.

It has the following benefits other than usual models.

- It also detects linear and non-linear relationships between variables
- A flexible, accurate and efficient model
- Best for larger datasets

Even though, the following challenges may also arise

- Need or large number of data points
- Takes much time to train



#### 4.2.6.2 Training and Evaluation of the models

```
results = {}
trained = {}

# Train and evaluate
for name, mdl in models.items():
    mdl.fit(X_train_sc, y_train) # Train on scaled training data
    pred_val = mdl.predict(X_valid_sc) # Predict on validation set

    # Performance metrics
    r2 = r2_score(y_valid, pred_val)
    rmse = float(np.sqrt(mean_squared_error(y_valid, pred_val)))

    # Store results
    results[name] = {"R2_val": float(r2), "RMSE_val": rmse}
    trained[name] = mdl

    print(f"{name:16s} | Val R²: {r2: .4f} | Val RMSE: {rmse: .3f}")
```

[32] ✓ 1m 41.4s

LinearRegression	Val R²: 0.8501	Val RMSE: 146.159
DecisionTree	Val R²: 0.9995	Val RMSE: 8.363
RandomForest	Val R²: 0.9998	Val RMSE: 4.655
GradientBoosting	Val R²: 0.9987	Val RMSE: 13.450
KNN	Val R²: 0.9781	Val RMSE: 55.896
NeuralNetwork	Val R²: 0.9994	Val RMSE: 9.397

In this function the models are trained using the training dataset and evaluated using the validating dataset. It calculates the **R<sup>2</sup>** value and **RMSE** value related to each model. In supervised learning these two values are used to understand how well the model performs.

- **R<sup>2</sup> (Coefficient of Determination)**

This value distributes in between 0 and 1. The measure explains whether up to which extend does the model explains the variability of the dataset. When the value is getting near to 1 means the model performs well.

- **RMSE (Root Mean Squared Error)**

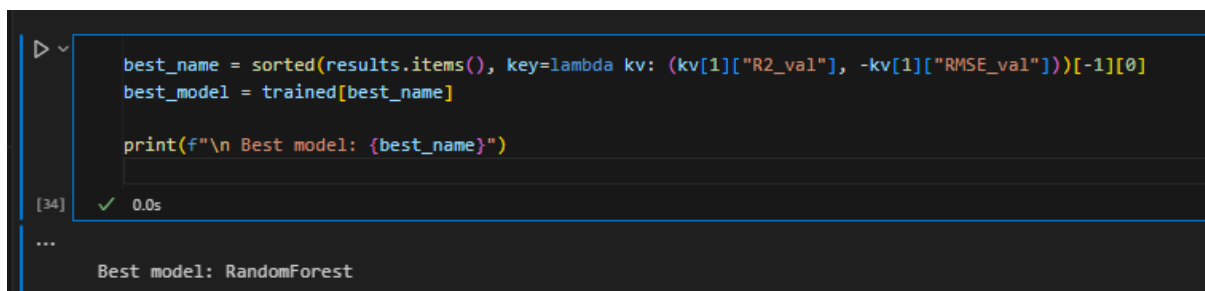
These values represent the error on the prediction data. When the RMSE value increases, accuracy decreases while when the RMSE value decreases, the accuracy increases.

Therefore, in overall the when the RMSE value is low and  $R^2$  value increases, the better model can be found.

The summary output table can be shown as follows.

Model Name	$R^2$ value	RMSE value
Linear Regression	0.8501	146.159
Decision Tree	0.9995	8.363
Random Forest	0.9998	4.655
Gradient Boosting	0.9997	13.450
KNN	0.9781	55.896
Neural Network	0.9994	9.397

According to the above table, **Random Forest** can be shown as the best model for further predictions.



```
best_name = sorted(results.items(), key=lambda kv: (kv[1]["R2_val"], -kv[1]["RMSE_val"]))[-1][0]
best_model = trained[best_name]

print(f"\n Best model: {best_name}")
```

[34] ✓ 0.0s

...

Best model: RandomForest

The above code block outputs the best model using the same manual way we used to decide.

### 4.2.6.3 Visualizing the model evaluation

```
res_df = pd.DataFrame(results).T.reset_index().rename(columns={'index': 'Model'})

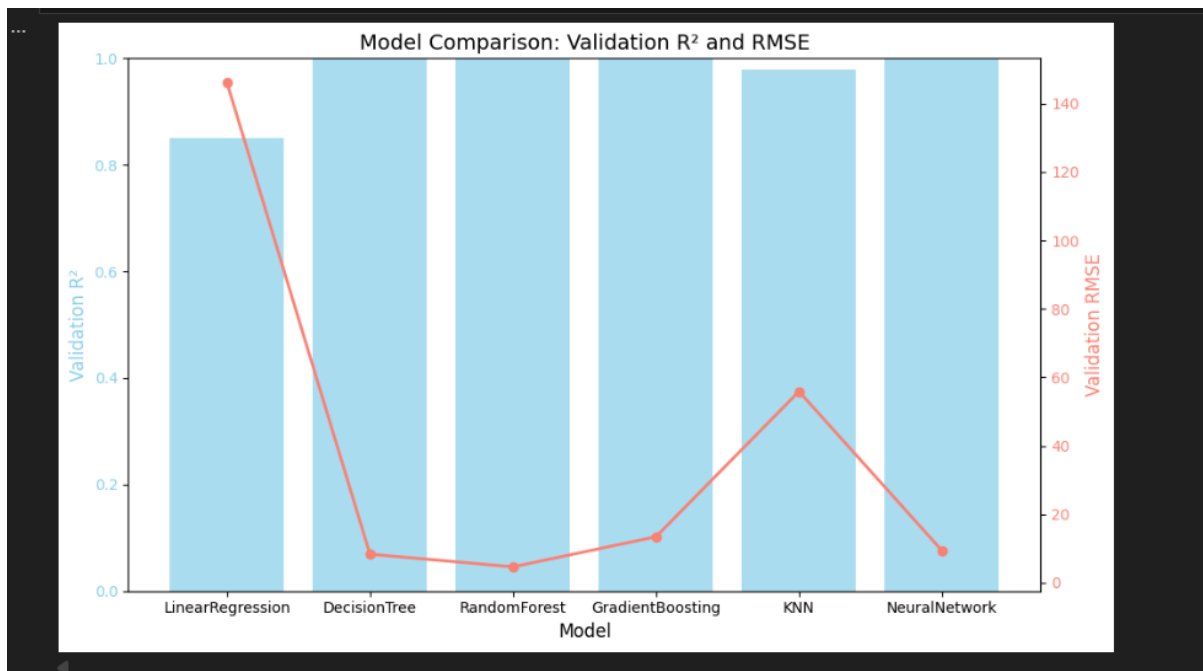
fig, ax1 = plt.subplots(figsize=(10, 6))

color1 = 'skyblue'
ax1.set_xlabel('Model', fontsize=12)
ax1.set_ylabel('Validation R2', color=color1, fontsize=12)
ax1.bar(res_df['Model'], res_df['R2_val'], color=color1, alpha=0.7, label='Validation R2')
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 1)

ax2 = ax1.twinx()
color2 = 'salmon'
ax2.set_ylabel('Validation RMSE', color=color2, fontsize=12)
ax2.plot(res_df['Model'], res_df['RMSE_val'], color=color2, marker='o', linewidth=2, label='Validation RMSE')
ax2.tick_params(axis='y', labelcolor=color2)

plt.title("Model Comparison: Validation R2 and RMSE", fontsize=14)
fig.tight_layout()
plt.show()
```

[36] ✓ 0.2s



RMSE value of each model is represented by the red line while  $R^2$  value is represented by the blue bars. This graphical visualization provides a quick overview of the best model at a glance. `twinx()` is the function used to map these two values with the model in the same diagram and sharing the same bar.

### 4.2.7 Saving Artifacts and the Model

```
joblib.dump(best_model, os.path.join(ARTIFACTS_DIR, "best_model.pkl"))
joblib.dump(imputer, os.path.join(ARTIFACTS_DIR, "imputer.pkl"))
joblib.dump(scaler, os.path.join(ARTIFACTS_DIR, "scaler.pkl"))
joblib.dump(FEATURE_COLS, os.path.join(ARTIFACTS_DIR, "feature_cols.pkl"))
print(f"Saved artifacts to: {ARTIFACTS_DIR}/")
```

[37] ✓ 1.5s

... Saved artifacts to: artifacts/

A new directory named artifacts is created to save the below resources.

- **best\_model.pkl**  
In terms of best model, “Random Forest” is saved here to utilize it once required.
- **imputer.pkl**  
To ensure that the same imputing strategies will be used in future as well as same as the training data.
- **scaler.pkl**  
to ensure the consistency of the data before predictions arrived.
- **feature\_cols.pkl**  
Saves all the features which were used to train the model for a later requirement and to ensure that they are orderly arranged.

### 4.2.8 Testing the Model

In terms of testing the model, the data which were split for testing is used. In this scenario the model tests the ability of the model to generalize when unseen data is given. It compares the outputs which were already given verses the model's predicted corresponding value in order to track the outliers, noises and accuracy.

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

pred_test = best_model.predict(X_test_sc)

# Calculate residuals
residuals = y_test.values - pred_test

plt.figure(figsize=(12, 10))

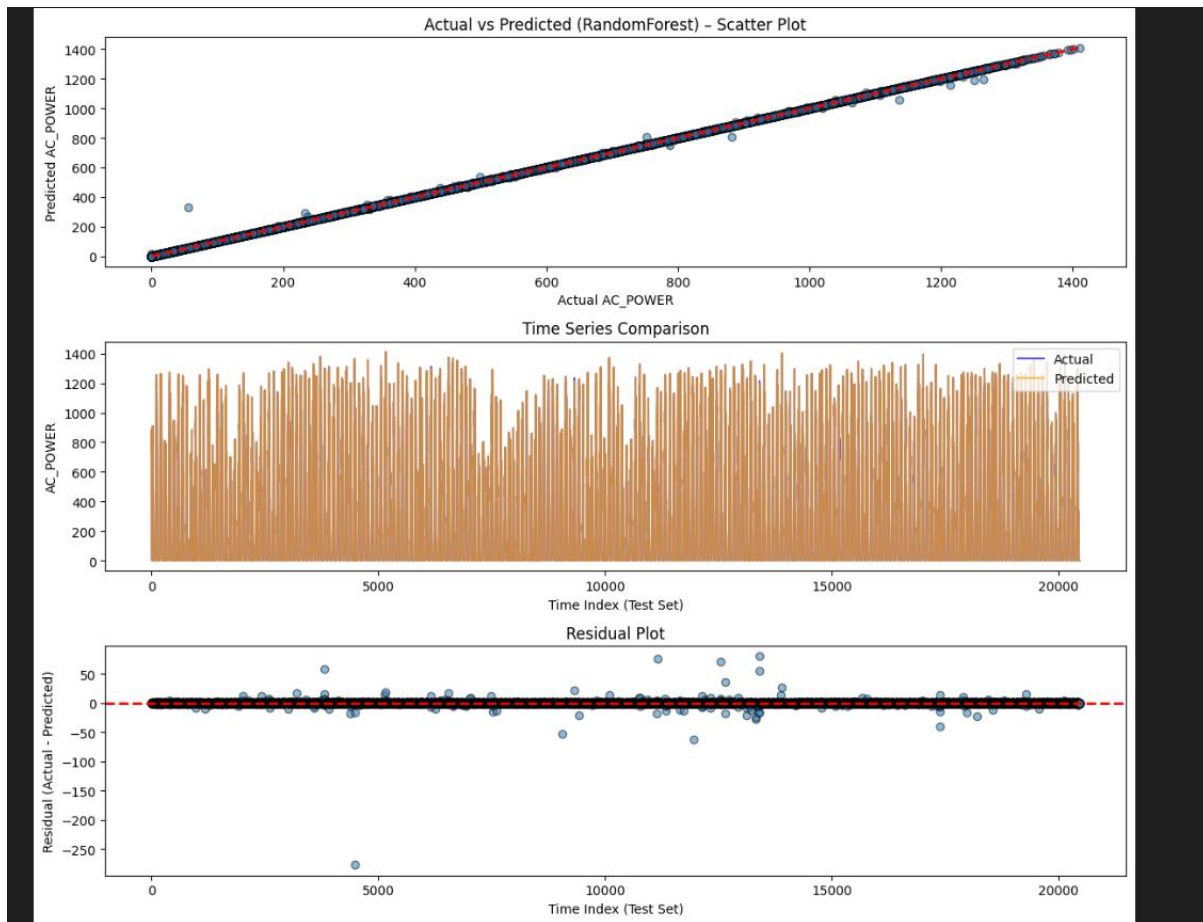
# Scatter plot (Actual vs Predicted)
plt.subplot(3, 1, 1)
plt.scatter(y_test, pred_test, alpha=0.5, edgecolor='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Actual AC_POWER')
plt.ylabel('Predicted AC_POWER')
plt.title(f'Actual vs Predicted ({best_name}) Scatter Plot')

# Time series comparison
plt.subplot(3, 1, 2)
plt.plot(y_test.values, label='Actual', color='blue', alpha=0.7)
plt.plot(pred_test, label='Predicted', color='orange', alpha=0.7)
plt.xlabel('Time Index (Test Set)')
plt.ylabel('AC_POWER')
plt.title('Time Series Comparison')
plt.legend()

# Residual plot
plt.subplot(3, 1, 3)
plt.scatter(range(len(residuals)), residuals, alpha=0.5, edgecolor='k')
plt.axhline(y=0, color='red', linestyle='--', linewidth=2)
plt.xlabel('Time Index (Test Set)')
plt.ylabel('Residual (Actual - Predicted)')
plt.title('Residual Plot')

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

[37] ✓ 1.3s



In our model three visualizations were created in order to get an idea of how predicted variable and actual variables relies on each other. The diagrams used are,

### 1. Scatter Plot

- This is a visualization mapped to get the relationship between the two variables, Actual and the predicted value.
- Blue dots are predicted values while the black dots denote the actual values.
- When both predicted values and actual values lies on the given red dashed line, we can say that the predicted values are exactly the same as the actual values.
- When the blue dots go above the dashed line means, the model has overfitted in that case while if the blue dots are below the red dashed line the model has been underfitted in the that case.

### 2. Time Series Plot

- This visualization plots the two variables along with the time.

- The blue series shows the actual values while the orange series shows the predicted values.
- Apart from the comparison it also shows the days and what times of the day does the power prediction is high.

### 3. Residual Plot

- This visualization plots the two variable residuals along with the time index which is plotted around zero.
- If the residual is plotted as positive, it is underfitted while if the residual is predicted as negative, the model is overfitted.

#### 4.2.9 Creating a file to Save History

```
HISTORY_PATH = os.path.join(ARTIFACTS_DIR, 'history.csv')

if not os.path.exists(HISTORY_PATH):
    pd.DataFrame(columns=[
        'DATE_TIME', 'PLANT_ID', 'SOURCE_KEY', 'DC_POWER', 'DAILY_YIELD', 'TOTAL_YIELD',
        'AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE', 'IRRADIATION', 'AC_POWER'
    ]).to_csv(HISTORY_PATH, index=False)
```

38] ✓ 0.0s

The code block creates a history file to store the data which were input by the user when using the model in real world. It will help to increase the accuracy of the machine language model. This saves all the features and the predicted output in a newly created file in artifacts directory called “history.csv”.

## 4.2.10 Functions Required for Using the Model

### 4.2.10.1 Applying Feature Engineering for Incoming Data

```
def _engineer_stream(df: pd.DataFrame) -> pd.DataFrame:

    # Sort by time to maintain sequence
    df = df.sort_values(['SOURCE_KEY', 'DATE_TIME']).copy()

    # Ensure numeric columns are actually numeric
    numeric_cols = [
        'DC_POWER', 'DAILY_YIELD', 'TOTAL_YIELD',
        'AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE',
        'IRRADIATION', 'AC_POWER'
    ]
    for col in numeric_cols:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')

    # Create basic time features
    df['HOUR'] = df['DATE_TIME'].dt.hour
    df['DATE'] = df['DATE_TIME'].dt.date

    # Rolling features
    df['DC_POWER_ROLLING_MEAN_1H'] = df.groupby('SOURCE_KEY')['DC_POWER'].transform(lambda x: x.rolling(4, min_periods=1).mean())
    df['DC_POWER_ROLLING_STD_1H'] = df.groupby('SOURCE_KEY')['DC_POWER'].transform(lambda x: x.rolling(4, min_periods=1).std())
    df['IRRADIATION_ROLLING_MAX_3H'] = df.groupby('SOURCE_KEY')['IRRADIATION'].transform(lambda x: x.rolling(12, min_periods=1).max())

    # Lag features
    df['DC_POWER_LAG_1'] = df.groupby('SOURCE_KEY')['DC_POWER'].shift(1)
    df['IRRADIATION_LAG_1'] = df.groupby('SOURCE_KEY')['IRRADIATION'].shift(1)
    if 'USE_AC_LAG' in globals() and USE_AC_LAG:
        df['AC_POWER_LAG_1'] = df.groupby('SOURCE_KEY')['AC_POWER'].shift(1)

    # Cumulative daily yield
    df['DAILY_YIELD_CUMSUM'] = df.groupby(['SOURCE_KEY', 'DATE'])['DAILY_YIELD'].cumsum()

    # Interaction feature
    df['IRRADIATION_x_AMBIENT_TEMP'] = df['IRRADIATION'] * df['AMBIENT_TEMPERATURE']

    # Efficiency
    eps = 1e-6
    df['EFFICIENCY'] = df['DC_POWER'] / (df['IRRADIATION'] + eps)

    return df
```

The function is responsible for engineering the below columns based on the given input data.(discussed in a previous chapter)

- HOUR
- DATE
- DC\_POWER\_ROLLING\_MEAN\_1H
- DC\_POWER\_ROLLING\_STD\_1H
- IRRADIATION\_ROLLING\_MAX\_3H
- DC\_POWER\_LAG\_1
- IRRADIATION\_LAG\_1
- DAILY\_YIELD\_CUMSUM



- IRRADIATION\_x\_AMBIENT\_TEMP
- EFFICIENCY

#### 4.2.10.2 Real-time AC Power Prediction Function

```
def forecast_future(future_df: pd.DataFrame, use_neural: bool = False) -> pd.DataFrame:
    """future_df must contain: DATE_TIME, PLANT_ID, SOURCE_KEY, DC_POWER, DAILY_YIELD, TOTAL_YIELD, AMBIENT_TEMPERATURE, MODULE_TEMPERATURE, IRRADIATION"""
    model_path = os.path.join(ARTIFACTS_DIR, 'neural_model.pkl' if use_neural else 'best_model.pkl')
    model = joblib.load(model_path)
    imp = joblib.load(os.path.join(ARTIFACTS_DIR, 'imputer.pkl'))
    scl = joblib.load(os.path.join(ARTIFACTS_DIR, 'scaler.pkl'))
    feats = joblib.load(os.path.join(ARTIFACTS_DIR, 'feature_cols.pkl'))

    hist = pd.read_csv(HISTORY_PATH)
    hist['DATE_TIME'] = pd.to_datetime(hist['DATE_TIME'], errors='coerce')

    fut = future_df.copy()
    fut['DATE_TIME'] = pd.to_datetime(fut['DATE_TIME'], errors='coerce')

    combo = pd.concat([hist, fut], ignore_index=True)
    combo = _engineer_stream(combo)

    ask = combo[combo['DATE_TIME'].isin(fut['DATE_TIME'])][feats]
    ask_imp = imp.transform(ask)
    ask_sc = scl.transform(ask_imp)
    preds = model.predict(ask_sc)

    out = fut[['DATE_TIME']].copy()
    out['PRED_AC_POWER'] = preds
    return out
```

The models first target is to find the real-time AC power output. The above function applies the inputs given by user as parameters to this function. The steps of the function for predicting the real-time AC power are given below.

1. Loading the following from artifacts directory
  - Model to predict the value
  - Imputer to preprocessing
  - Scaler to scaling the dataset
  - Featured columns to feature engineering
2. Loading the “history.csv” file to enter the new values using the user inputs
3. Defining the finalized feature set by combining the inputs and featured columns
4. Perform data processing by,
  - Handling missing values using forward filling and imputation

- Scaling the features ...
5. Saving the “history.csv” by only the user inputs, excluding the featured columns
  6. Outputs the predicted value

#### 4.2.10.3 Function for Forecasting AC Power for Few Hours Ahead

```
def forecast_future(future_df: pd.DataFrame, use_neural: bool = False) -> pd.DataFrame:
    """future_df must contain: DATE_TIME, PLANT_ID, SOURCE_KEY, DC_POWER, DAILY_YIELD, TOTAL_YIELD, AMBIENT_TEMPERATURE, MODULE_TEMPERATURE, IRRADIATION"""
    model_path = os.path.join(ARTIFACTS_DIR, 'neural_model.pkl' if use_neural else 'best_model.pkl')
    model = joblib.load(model_path)
    imp = joblib.load(os.path.join(ARTIFACTS_DIR, 'imputer.pkl'))
    scl = joblib.load(os.path.join(ARTIFACTS_DIR, 'scaler.pkl'))
    feats = joblib.load(os.path.join(ARTIFACTS_DIR, 'feature_cols.pkl'))

    hist = pd.read_csv(HISTORY_PATH)
    hist['DATE_TIME'] = pd.to_datetime(hist['DATE_TIME'], errors='coerce')

    fut = future_df.copy()
    fut['DATE_TIME'] = pd.to_datetime(fut['DATE_TIME'], errors='coerce')

    combo = pd.concat([hist, fut], ignore_index=True)
    combo = _engineer_stream(combo)

    ask = combo[combo['DATE_TIME'].isin(fut['DATE_TIME'])][feats]
    ask_imp = imp.transform(ask)
    ask_sc = scl.transform(ask_imp)
    preds = model.predict(ask_sc)

    out = fut[['DATE_TIME']].copy()
    out['PRED_AC_POWER'] = preds
    return out
```

[43] ✓ 0.0s

The second target of the model is to output the AC power for few hours ahead. The above function does that according to below steps. (same as above)

1. Loading the following from artifacts directory
  - Model to predict the value
  - Imputer to preprocessing
  - Scaler to scaling the dataset
  - Featured columns to feature engineering
2. Loading the “history.csv” file to enter the new values using the user inputs
3. Defining the finalized feature set by combining the inputs and featured columns

4. Perform data processing by,
  - Handling missing values using forward filling and imputation
  - Scaling the features ...
5. Saving the “history.csv” by only the user inputs, excluding the featured columns
6. Outputs the predicted value along with the time, since the hour is required to predict when it is few hours ahead.

#### 4.2.11 Application of the Model

##### 4.2.11.1 Real-time AC Power Prediction

```
pred_now = predict_ac_power_raw(  
    plant_id=1,  
    source_key='INV_01',  
    timestamp='2025-08-14 14:00:00',  
    dc_power=520,  
    daily_yield=45,  
    total_yield=2100,  
    ambient_temp=26,  
    module_temp=32,  
    irradiation=780  
)  
  
print('Realtime prediction:', pred_now)
```

[48] ✓ 2.9s

... Realtime prediction: 509.3710158730158

When the above data are given to the relevant function it successfully predicts the Realtime AC power generation in watts.

#### 4.2.11.2 Forecast AC Power for Few Hours Ahead

```
fut = pd.DataFrame({
    'DATE_TIME': pd.date_range('2025-08-15 06:00', periods=12, freq='H'),
    'PLANT_ID': 1,
    'SOURCE_KEY': 'INV_01',
    'DC_POWER': np.linspace(100, 900, 12),
    'DAILY_YIELD': np.linspace(0, 120, 12),
    'TOTAL_YIELD': 2500 + np.linspace(0, 120, 12),
    'AMBIENT_TEMPERATURE': np.linspace(22, 32, 12),
    'MODULE_TEMPERATURE': np.linspace(25, 40, 12),
    'IRRADIATION': np.linspace(50, 900, 12)
})

fut_pred = forecast_future(fut)
print(fut_pred.head())
```

[49] ✓ 2.6s

...	DATE_TIME	PRED_AC_POWER
0	2025-08-15 06:00:00	97.281329
1	2025-08-15 07:00:00	169.745752
2	2025-08-15 08:00:00	240.676973
3	2025-08-15 09:00:00	311.620168
4	2025-08-15 10:00:00	383.000805

As same as the above function when the input features with values are fed as shown above it outputs the predicted AC power for few hours ahead with relevant times.

## **5. Discussion**

### **5.1 Usage of the Model**

The trained model is used to predict the followings.

- The Realtime power prediction
- Few hours ahead power forecasting

The model helps in following factors when dealing with power optimization in solar power plants and related stakeholders.

- Helps to manage grid optimization and maintenance of power plants based on power generation summaries
- To analyse how weather, time of the day and other relevant factors affect the power generation
- To manage energy prediction to reduce power wastage

### **5.2 Risks and Potential Fixes**

The following problems may arise when and after the model is trained.

- The risk related to dataset such as high number of outliers; dataset is not matching the timeline may affect improper outcomes. Therefore, it is important to select a proper time aligning dataset for training the model.
- High probability of underfitting and overfitting due to too much and purely training on the dataset. As a prefix it is required to choose a considerably fair data modelling algorithm.
- Model may fail when it comes to deployment stage, the format of input may affect due to input data types and issues in preprocessing. Therefore, handling scalers and imputers are important.
- With the time the model trained dataset will not match the time aligned patterns. Therefore, it is important to maintain the model along with the time.
- Also to ensure that the software breakdowns does not affect the performance of the model, it is recommended to maintain the tech stack continuously.

### **5.3 Future Enhancements**

To increase the efficiency of the model and predicted value, the following enhancements can be included in future.

- Increase the accuracy of data (input features), IoT based sensors and satellite integrated data can be added.
- Can train the model with deep learning models
- A software can be implemented to connect the stakeholders with a user-friendly interface with a real-time dashboard.
- To ensure the scalability, a cloud based storage to save the artifacts can be implemented.
- The insights gained can be turned into innovative business decisions which generates revenue income.

## 6. Conclusion

In conclusion, it is clear that the proposed solar power predicting model helps to generate the predicted output using weather, sensor and power generation data successfully. Since the model is trained using multiple modelling algorithms, a clear comparison also identified. The project study highlights the following key points.

- Accurate predicted models lead better predictions
- Performance of the model is based on the quality of dataset algorithm and best model selection
- Continuous maintenance leads to a long-term successful model

Therefore, a well-trained model helps to get proper business insights to increase the revenue. And also, the model can be enhanced using proper mitigation techniques and future enhancements.

## 7. References

- <https://www.kaggle.com/datasets/anikannal/solar-power-generation-data>
- [https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning)
- <https://www.ibm.com/think/topics/neural-networks>
- <https://www.geeksforgeeks.org/machine-learning/machine-learning-algorithms/>
- <https://www.solarpowereurope.org/>
- <https://www.edfenergy.com/energywise/renewable-energy-sources>
- <https://medium.com/@aaabulkhair/so-which-ml-algorithm-to-use-d2484239f448>