**Name:** Ananya Godse **SAP ID:** 60009220161 **Batch:** D1 – 2

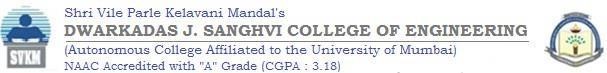
# 

# **Title:** Solar Power Generation Forecasting

## Aim: To Predict the Daily Yield of Solar Power of Solar Power Plants using Weather Sensor Data

## Justification:

1. Describe your problem in detail and discuss why it is a data science problem. Our world is on the brink of a climate crisis, driven primarily by the accumulation of greenhouse gases in the Earth's atmosphere. These greenhouse gases are released when fossil fuels are burned. According to the Government of India’s NITI Aayog website, in 2022, 58.63% of our energy supply came from the burning of coal and 29.32% from oil. That means that close to 88% of our energy supply comes from non-renewable, climate change causing sources. Clearly, renewable sources of energy are the need of the hour.

Fortunately, India is making strides in this area. One such source of renewable energy is solar energy. Solar Power grids are being laid down every day, increasing our power generation capacity. But there is an inherent variability to the production of solar energy. Its dependent on weather conditions, time of the day, seasonal changes, and geographic factors.

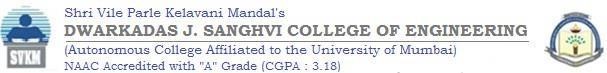
If we are going to rely on solar energy to fulfil a larger slice of energy consumption, we need to ensure that it will be enough. Solar Power Generation Forecasting is thus necessary to manage the logistics of electricity supply and optimize grid management.

The problem here is to predict how much power a solar power plant will generate on any given day based on the weather.

This is a data science problem because it involves analysing large volumes of data from various sources (weather forecasts, historical energy production data, geographical information, etc.) to build accurate predictive models that can anticipate fluctuations in solar energy production. These models are crucial for optimizing the efficiency and reliability of solar energy systems and integrating them effectively into the broader energy infrastructure.

2. Justify that the data chosen is appropriate to build a model to solve the problem.

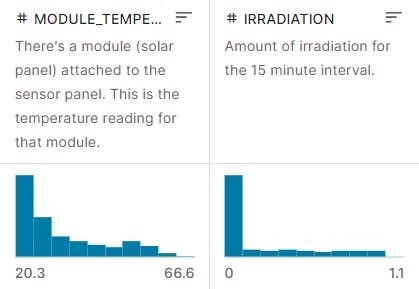
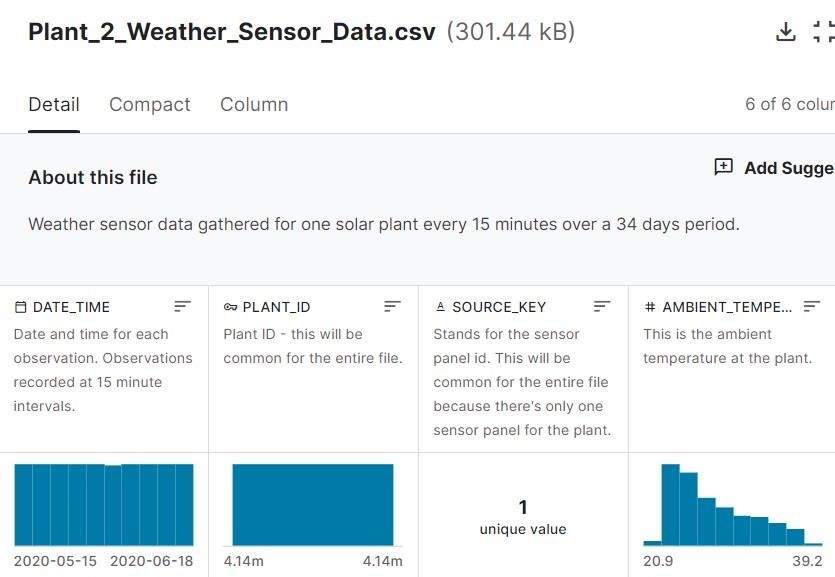
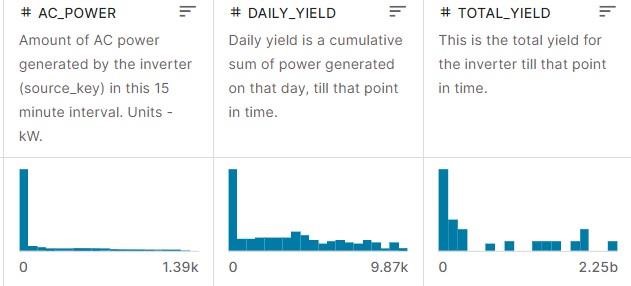
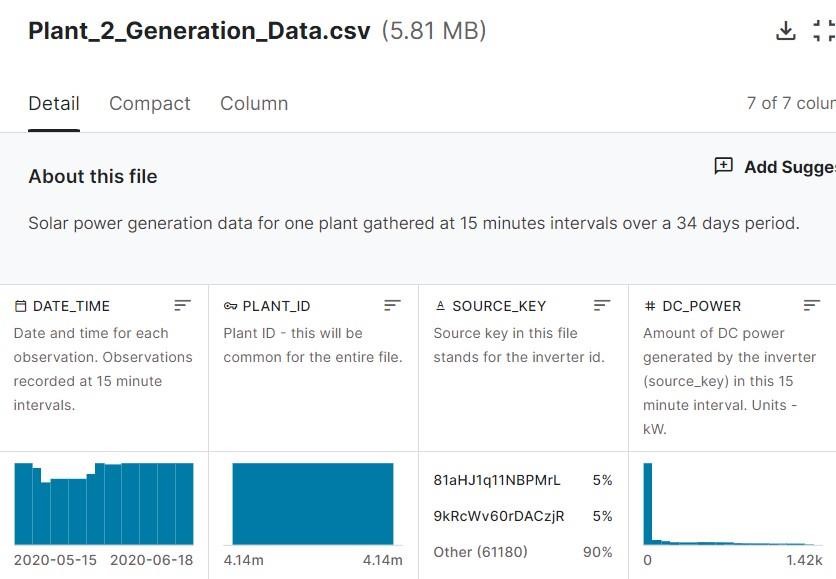
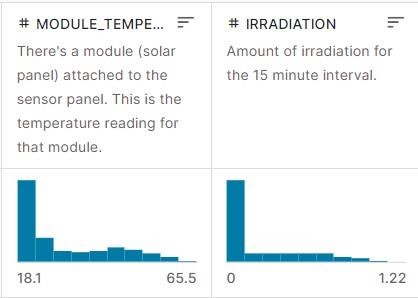
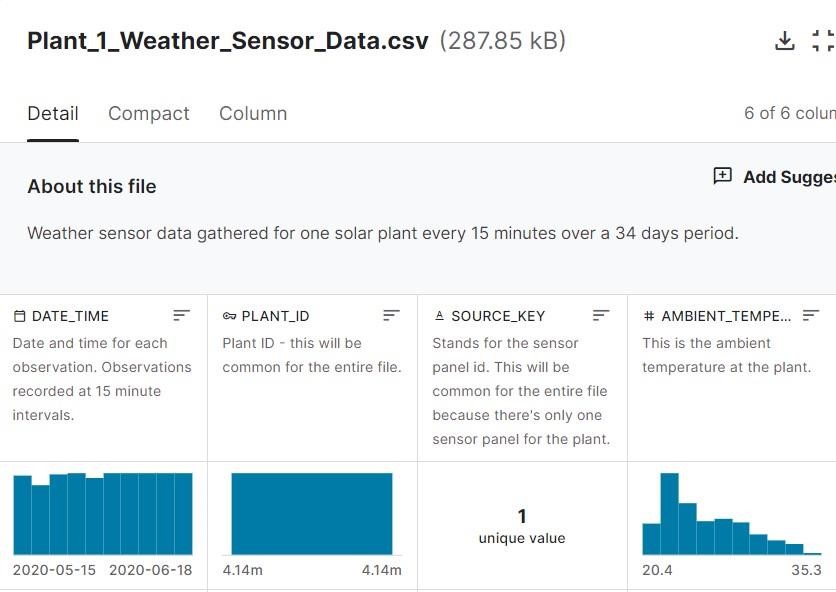
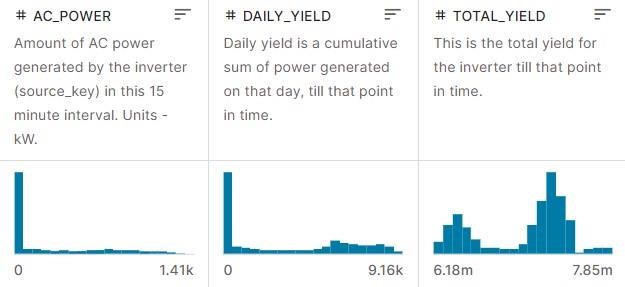
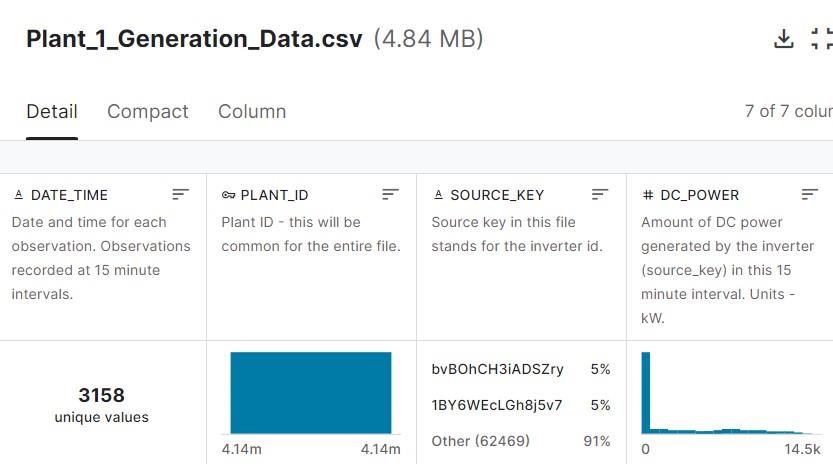
**Dataset Link:** [https://www.kaggle.com/datasets/anikannal/solar-power-generationdata?resource=download&select=Plant\_1\_Generation\_Data.csv](https://www.kaggle.com/datasets/anikannal/solar-power-generation-data?resource=download&select=Plant_1_Generation_Data.csv)

This data has been gathered at two solar power plants in India over a 34-day period. It has two pairs of files - each pair has one power generation dataset and one sensor readings dataset. The power generation datasets are gathered at the inverter level - each inverter has multiple lines of solar panels attached to it. The sensor data is gathered at a plant level - single array of sensors optimally placed at the plant.

Since this is data is collected from a solar power plant in India and there is data about the solar energy yield and data from weather sensors, this dataset is perfect for figuring out how much solar energy will be produced on any given day based on weather factors.

## Data Description:

This data has been gathered at two solar power plants in India over a 34-day period. It has two pairs of files - each pair has one power generation dataset and one sensor readings dataset. The power generation datasets are gathered at the inverter level - each inverter has multiple lines of solar panels attached to it. The sensor data is gathered at a plant level - single array of sensors optimally placed at the plant.



## Exploratory Data Analysis & Pre-Processing:

**Name: Ananya Godse SAP ID: 60009220161**

### Importing the necessary libraries

|  |
| --- |
| **import** pandas **as** pd  **import** matplotlib.pyplot **as** plt **%matplotlib** inline **import** seaborn **as** sns **from** datetime **import** datetime |

In [1]:

### Importing the power generation data and weather sensor data for both plants

|  |
| --- |
| plant1\_generation **=** pd**.**read\_csv(r"Solar Power Generation  Data\Plant\_1\_Generation\_Data.csv") print("PLANT 1 GENERATION DATA") display(plant1\_generation)  plant1\_sensor **=** pd**.**read\_csv(r"Solar Power Generation  Data\Plant\_1\_Weather\_Sensor\_Data.csv") print("PLANT 1 WEATHER SENSOR DATA") display(plant1\_sensor)  plant2\_generation **=** pd**.**read\_csv(r"Solar Power Generation  Data\Plant\_2\_Generation\_Data.csv") print("PLANT 2 GENERATION DATA") display(plant2\_generation)  plant2\_sensor **=** pd**.**read\_csv(r"Solar Power Generation  Data\Plant\_2\_Weather\_Sensor\_Data.csv") print("PLANT 2 WEATHER SENSOR DATA") display(plant1\_sensor) |

In [2]:

PLANT 1 GENERATION DATA

#### 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 0.000 6259559.0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 15-05-2020 00:00 | 4135001 | 1IF53ai7Xc0U56Y | 0.0 | 0.0 | 0.000 | 6183645.0 |
| **2** | 15-05-2020 00:00 | 4135001 | 3PZuoBAID5Wc2HD | 0.0 | 0.0 | 0.000 | 6987759.0 |
| **3** | 15-05-2020 00:00 | 4135001 | 7JYdWkrLSPkdwr4 | 0.0 | 0.0 | 0.000 | 7602960.0 |
| **4** | 15-05-2020 00:00 | 4135001 | McdE0feGgRqW7Ca | 0.0 | 0.0 | 0.000 | 7158964.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **68773** | 17-06-2020 23:45 | 4135001 | uHbuxQJl8lW7ozc | 0.0 | 0.0 | 5967.000 | 7287002.0 |
| **68774** | 17-06-2020 23:45 | 4135001 | wCURE6d3bPkepu2 | 0.0 | 0.0 | 5147.625 | 7028601.0 |
| **68775** | 17-06-2020 23:45 | 4135001 | z9Y9gH1T5YWrNuG | 0.0 | 0.0 | 5819.000 | 7251204.0 |
| **68776** | 17-06-2020 23:45 | 4135001 | zBIq5rxdHJRwDNY | 0.0 | 0.0 | 5817.000 | 6583369.0 |

1. 17-06-2020 23:45 4135001 zVJPv84UY57bAof 0.0 0.0 5910.000 7363272.0
2. rows × 7 columns PLANT 1 WEATHER SENSOR DATA

#### DATE\_TIME PLANT\_ID SOURCE\_KEY AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2020-05-15  **0**  00:00:00 | 4135001 | HmiyD2TTLFNqkNe | 25.184316 | 22.857507 | 0.0 |
| 2020-05-15  **1** 00:15:00 | 4135001 | HmiyD2TTLFNqkNe | 25.084589 | 22.761668 | 0.0 |

2020-05-15

**2** 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 |
| **...** | ... | ... | ... | ... | ... | ... |
| **3177** | 2020-06-17 22:45:00 | 4135001 | HmiyD2TTLFNqkNe | 22.150570 | 21.480377 | 0.0 |
| **3178** | 2020-06-17 23:00:00 | 4135001 | HmiyD2TTLFNqkNe | 22.129816 | 21.389024 | 0.0 |

2020-06-17

**3179** 23:15:00 4135001 HmiyD2TTLFNqkNe 22.008275 20.709211 0.0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **3180** | 2020-06-17 23:30:00 | 4135001 | HmiyD2TTLFNqkNe |  | 21.969495 |  | 20.734963 | 0.0 |
| **3181** | 2020-06-17 23:45:00 | 4135001 | HmiyD2TTLFNqkNe |  | 21.909288 |  | 20.427972 | 0.0 |

3182 rows × 6 columns

PLANT 2 GENERATION DATA

#### DATE\_TIME PLANT\_ID SOURCE\_KEY DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

**0** 2020-05-15 00:00:00 4136001 4UPUqMRk7TRMgml 0.0 0.0 9425.000000 2.429011e+06

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 2020-05-15 00:00:00 | 4136001 | 81aHJ1q11NBPMrL | 0.0 |  | 0.0 | 0.000000 | 1.215279e+09 |
| **2** | 2020-05-15 00:00:00 | 4136001 | 9kRcWv60rDACzjR | 0.0 |  | 0.0 | 3075.333333 | 2.247720e+09 |
| **3** | 2020-05-15 00:00:00 | 4136001 | Et9kgGMDl729KT4 | 0.0 |  | 0.0 | 269.933333 | 1.704250e+06 |
| **4** | 2020-05-15 00:00:00 | 4136001 | IQ2d7wF4YD8zU1Q | 0.0 |  | 0.0 | 3177.000000 | 1.994153e+07 |
| **...** | ... | ... | ... | ... |  | ... | ... | ... |
| **67693** | 2020-06-17 23:45:00 | 4136001 | q49J1IKaHRwDQnt | 0.0 |  | 0.0 | 4157.000000 | 5.207580e+05 |
| **67694** | 2020-06-17 23:45:00 | 4136001 | rrq4fwE8jgrTyWY | 0.0 |  | 0.0 | 3931.000000 | 1.211314e+08 |

**67695** 2020-06-17 23:45:00 4136001 vOuJvMaM2sgwLmb 0.0 0.0 4322.000000 2.427691e+06

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **67696** 2020-06-17 23:45:00 | 4136001 | xMbIugepa2P7lBB | 0.0 |  | 0.0 | 4218.000000 | 1.068964e+08 |
| **67697** 2020-06-17 23:45:00 | 4136001 | xoJJ8DcxJEcupym | 0.0 |  | 0.0 | 4316.000000 | 2.093357e+08 |

67698 rows × 7 columns PLANT 2 WEATHER SENSOR DATA

#### DATE\_TIME PLANT\_ID SOURCE\_KEY AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

2020-05-15

**0** 4135001 HmiyD2TTLFNqkNe 25.184316 22.857507 0.0

00:00:00

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |
| **...** | ... | ... | ... |  | ... |  | ... | ... |
| **3177** | 2020-06-17 22:45:00 | 4135001 | HmiyD2TTLFNqkNe |  | 22.150570 |  | 21.480377 | 0.0 |
| **3178** | 2020-06-17 23:00:00 | 4135001 | HmiyD2TTLFNqkNe |  | 22.129816 |  | 21.389024 | 0.0 |
| **3179** | 2020-06-17 23:15:00 | 4135001 | HmiyD2TTLFNqkNe |  | 22.008275 |  | 20.709211 | 0.0 |
| **3180** | 2020-06-17 23:30:00 | 4135001 | HmiyD2TTLFNqkNe |  | 21.969495 |  | 20.734963 | 0.0 |
| **3181** | 2020-06-17 23:45:00 | 4135001 | HmiyD2TTLFNqkNe |  | 21.909288 |  | 20.427972 | 0.0 |

3182 rows × 6 columns

In [3]: plant1\_generation**.**head()

#### Out[3]: 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 |

In [4]: plant1\_sensor**.**head()

#### Out[4]: DATE\_TIME PLANT\_ID SOURCE\_KEY AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

2020-05-15

**0** 4135001 HmiyD2TTLFNqkNe 25.184316 22.857507 0.0

00:00:00

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

2020-05-15

**4** 01:00:00 4135001 HmiyD2TTLFNqkNe 24.621525 22.165423 0.0

In [5]: plant2\_generation**.**head()

#### Out[5]: DATE\_TIME PLANT\_ID SOURCE\_KEY DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

**0** 2020-05-15 00:00:00 4136001 4UPUqMRk7TRMgml 0.0 0.0 9425.000000 2.429011e+06

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1** 2020-05-15 00:00:00 | 4136001 | 81aHJ1q11NBPMrL | 0.0 | 0.0 | 0.000000 1.215279e+09 |
| **2** 2020-05-15 00:00:00 | 4136001 | 9kRcWv60rDACzjR | 0.0 | 0.0 | 3075.333333 2.247720e+09 |
| **3** 2020-05-15 00:00:00 | 4136001 | Et9kgGMDl729KT4 | 0.0 | 0.0 | 269.933333 1.704250e+06 |
| **4** 2020-05-15 00:00:00 | 4136001 | IQ2d7wF4YD8zU1Q | 0.0 | 0.0 | 3177.000000 1.994153e+07 |

In [6]: plant2\_sensor**.**head()

#### Out[6]: DATE\_TIME PLANT\_ID SOURCE\_KEY AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

2020-05-15

**0** 4136001 iq8k7ZNt4Mwm3w0 27.004764 25.060789 0.0

00:00:00

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 2020-05-15 00:15:00 | 4136001 iq8k7ZNt4Mwm3w0 |  | 26.880811 |  | 24.421869 | 0.0 |
| **2** | 2020-05-15 00:30:00 | 4136001 iq8k7ZNt4Mwm3w0 |  | 26.682055 |  | 24.427290 | 0.0 |
| **3** | 2020-05-15 00:45:00 | 4136001 iq8k7ZNt4Mwm3w0 |  | 26.500589 |  | 24.420678 | 0.0 |
| **4** | 2020-05-15 01:00:00 | 4136001 iq8k7ZNt4Mwm3w0 |  | 26.596148 |  | 25.088210 | 0.0 |

In [7]: plant1\_generation**.**info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 68778 entries, 0 to 68777 Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. DATE\_TIME 68778 non-null object
2. PLANT\_ID 68778 non-null int64
3. SOURCE\_KEY 68778 non-null object
4. DC\_POWER 68778 non-null float64
5. AC\_POWER 68778 non-null float64 5 DAILY\_YIELD 68778 non-null float64 6 TOTAL\_YIELD 68778 non-null float64 dtypes: float64(4), int64(1), object(2) memory usage:

3.7+ MB

In [8]: plant1\_sensor**.**info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 3182 entries, 0 to 3181 Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. DATE\_TIME 3182 non-null object
2. PLANT\_ID 3182 non-null int64
3. SOURCE\_KEY 3182 non-null object 3 AMBIENT\_TEMPERATURE 3182 non-null float64

4 MODULE\_TEMPERATURE 3182 non-null float64 5 IRRADIATION 3182 non-null float64 dtypes: float64(3), int64(1), object(2) memory usage: 149.3+ KB

In [9]: plant2\_generation**.**info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 67698 entries, 0 to 67697 Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. DATE\_TIME 67698 non-null object
2. PLANT\_ID 67698 non-null int64
3. SOURCE\_KEY 67698 non-null object
4. DC\_POWER 67698 non-null float64
5. AC\_POWER 67698 non-null float64 5 DAILY\_YIELD 67698 non-null float64 6 TOTAL\_YIELD 67698 non-null float64 dtypes: float64(4), int64(1), object(2) memory usage:

3.6+ MB

In [10]: plant2\_sensor**.**info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 3259 entries, 0 to 3258 Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. DATE\_TIME 3259 non-null object
2. PLANT\_ID 3259 non-null int64
3. SOURCE\_KEY 3259 non-null object 3 AMBIENT\_TEMPERATURE 3259 non-null float64

4 MODULE\_TEMPERATURE 3259 non-null float64 5 IRRADIATION 3259 non-null float64 dtypes: float64(3), int64(1), object(2) memory usage: 152.9+ KB

**Observations:**

1. DATE\_TIME column data type needs to converted to Date time for all the datasets.
2. We know from the data description that the SOURCE\_KEY column in the generation datasets is the Inverter ID and the Sensor Panel ID in the Weather Sensor Datasets. We'll rename the columns.

In [11]: plant1\_generation["PLANT\_ID"]**.**value\_counts()

PLANT\_ID Out[11]:

4135001 68778 Name: count, dtype: int64

**Observations:**

As we know from data description and as proven above, all the records from the PLANT 1 GENERATION DATA belong to Plant 1. Since this doesn't provide any actionable insight, we'll drop the column.

In [12]: plant1\_generation["SOURCE\_KEY"]**.**value\_counts()

SOURCE\_KEY

Out[12]: bvBOhCH3iADSZry

3155 1BY6WEcLGh8j5v7

3154

7JYdWkrLSPkdwr4 3133

VHMLBKoKgIrUVDU 3133 ZnxXDlPa8U1GXgE 3130 ih0vzX44oOqAx2f 3130 z9Y9gH1T5YWrNuG 3126 wCURE6d3bPkepu2 3126 uHbuxQJl8lW7ozc 3125 pkci93gMrogZuBj 3125 iCRJl6heRkivqQ3 3125 rGa61gmuvPhdLxV 3124 sjndEbLyjtCKgGv 3124 McdE0feGgRqW7Ca 3124 zVJPv84UY57bAof 3124 ZoEaEvLYb1n2sOq 3123 1IF53ai7Xc0U56Y 3119 adLQvlD726eNBSB 3119 zBIq5rxdHJRwDNY 3119 WRmjgnKYAwPKWDb 3118

3PZuoBAID5Wc2HD 3118

YxYtjZvoooNbGkE 3104 Name: count, dtype: int64

As we know from the data description, the SOURCE\_KEY column in the PLANT 1 GENERATION DATA SET has the INVERTER ID

In [13]: print(f"No. of Inverters in Plant 1: {len(plant1\_generation['SOURCE\_KEY']**.**value\_counts())}") No. of Inverters in Plant 1: 22

In [14]: plant1\_sensor["PLANT\_ID"]**.**value\_counts()

PLANT\_ID Out[14]:

4135001 3182

Name: count, dtype: int64

All records in PLANT 1 WEATHER SENSOR DATA belong to Plant 1. Since this doesn't provide any actionable insight, we'll be dropping this column.

In [15]: plant1\_sensor["SOURCE\_KEY"]**.**value\_counts()

SOURCE\_KEY Out[15]:

HmiyD2TTLFNqkNe 3182

Name: count, dtype: int64

As we know from the data description, the SOURCE\_KEY column in the PLANT 1 WEATHER SENSOR DATA SET has the SENSOR PANEL ID and there is only one Sensor Panel in Plant 1. So since it doesn't provide any insight we can drop the column.

In [16]: plant2\_generation["PLANT\_ID"]**.**value\_counts()

PLANT\_ID Out[16]:

4136001 67698

Name: count, dtype: int64

**Observations:**

As we know from data description and as proven above, all the records from the PLANT 2 GENERATION DATA belong to Plant 2. Since this doesn't provide any actionable insight, we'll be dropping the column.

In [17]: plant2\_generation["SOURCE\_KEY"]**.**value\_counts()

SOURCE\_KEY

Out[17]: xoJJ8DcxJEcupym

3259 WcxssY2VbP4hApt

3259 9kRcWv60rDACzjR

3259 vOuJvMaM2sgwLmb

3259 rrq4fwE8jgrTyWY

3259 LYwnQax7tkwH5Cb

3259 LlT2YUhhzqhg5Sw

3259 q49J1IKaHRwDQnt

3259 oZZkBaNadn6DNKz

3259 PeE6FRyGXUgsRhN

3259

81aHJ1q11NBPMrL 3259 V94E5Ben1TlhnDV 3259 oZ35aAeoifZaQzV 3195

4UPUqMRk7TRMgml 3195 Qf4GUc1pJu5T6c6 3195

Mx2yZCDsyf6DPfv 3195

Et9kgGMDl729KT4 3195 Quc1TzYxW2pYoWX 3195 mqwcsP2rE7J0TFp 2355 NgDl19wMapZy17u 2355

IQ2d7wF4YD8zU1Q 2355

xMbIugepa2P7lBB 2355 Name: count, dtype: int64

As we know from the data description, the SOURCE\_KEY column in the PLANT 2 GENERATION DATA SET has the INVERTER ID

In [18]: print(f"No. of Inverters in Plant 2: {len(plant2\_generation['SOURCE\_KEY']**.**value\_counts())}") No. of Inverters in Plant 2: 22

In [19]: plant2\_sensor["PLANT\_ID"]**.**value\_counts()

PLANT\_ID Out[19]:

4136001 3259

Name: count, dtype: int64

All records in PLANT 2 WEATHER SENSOR DATA belong to Plant 2. Since this doesn't provide any actionable insight, we'll be dropping this column.

In [20]: plant2\_sensor["SOURCE\_KEY"]**.**value\_counts()

SOURCE\_KEY

Out[20]: iq8k7ZNt4Mwm3w0 3259 Name: count, dtype:

int64

As we know from the data description, the SOURCE\_KEY column in the PLANT 2 WEATHER SENSOR DATA SET has the SENSOR PANEL ID and there is only one Sensor Panel in Plant 2. Since it doesn't provide any insight we can drop the column.

**Renaming & Dropping Columns:**

In [21]: plant1\_generation**.**rename(columns**=**{"SOURCE\_KEY":"INVERTER\_ID"}, inplace**=True**) plant1\_generation

#### Out[21]: DATE\_TIME PLANT\_ID INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** 15-05-2020 00:00 |  | 4135001 1IF53ai7Xc0U56Y | 0.0 | 0.0 | 0.000 | 6183645.0 |
| **2** 15-05-2020 00:00 |  | 4135001 3PZuoBAID5Wc2HD | 0.0 | 0.0 | 0.000 | 6987759.0 |
| **3** 15-05-2020 00:00 |  | 4135001 7JYdWkrLSPkdwr4 | 0.0 | 0.0 | 0.000 | 7602960.0 |

**0**  1BY6WEcLGh8j5v7 0.0 0.0 0.000 6259559.0

**4** 15-05-2020 4135001 McdE0feGgRqW7Ca 0.0 0.0 0.000 7158964.0

00:00

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **68773** | 17-06-2020  23:45 | 4135001 | uHbuxQJl8lW7ozc | 0.0 | 0.0 | 5967.000 | 7287002.0 |
| **68774** | 17-06-2020  23:45 | 4135001 | wCURE6d3bPkepu2 | 0.0 | 0.0 | 5147.625 | 7028601.0 |
| **68775** | 17-06-2020  23:45 | 4135001 | z9Y9gH1T5YWrNuG | 0.0 | 0.0 | 5819.000 | 7251204.0 |
| **68776** | 17-06-2020  23:45 | 4135001 | zBIq5rxdHJRwDNY | 0.0 | 0.0 | 5817.000 | 6583369.0 |
| **68777** | 17-06-2020  23:45 | 4135001 | zVJPv84UY57bAof | 0.0 | 0.0 | 5910.000 | 7363272.0 |

68778 rows × 7 columns

|  |
| --- |
| plant1\_generation**.**drop("PLANT\_ID", axis**=**1, inplace**=True**) plant1\_generation |

In [22]:

#### Out[22]: DATE\_TIME INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

**0** 15-05-2020 00:00 1BY6WEcLGh8j5v7 0.0 0.0 0.000 6259559.0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** 15-05-2020 00:00 | 1IF53ai7Xc0U56Y | 0.0 |  | 0.0 | 0.000 | 6183645.0 |

**2** 15-05-2020 00:00 3PZuoBAID5Wc2HD 0.0 0.0 0.000 6987759.0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **3** | 15-05-2020 00:00 | 7JYdWkrLSPkdwr4 | 0.0 |  | 0.0 | 0.000 | 7602960.0 |
| **4** | 15-05-2020 00:00 | McdE0feGgRqW7Ca | 0.0 |  | 0.0 | 0.000 | 7158964.0 |
| **...** | ... | ... | ... |  | ... | ... | ... |
| **68773** | 17-06-2020 23:45 | uHbuxQJl8lW7ozc | 0.0 |  | 0.0 | 5967.000 | 7287002.0 |
| **68774** | 17-06-2020 23:45 | wCURE6d3bPkepu2 | 0.0 |  | 0.0 | 5147.625 | 7028601.0 |

**68775** 17-06-2020 23:45 z9Y9gH1T5YWrNuG 0.0 0.0 5819.000 7251204.0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **68776** 17-06-2020 23:45 | zBIq5rxdHJRwDNY | 0.0 |  | 0.0 | 5817.000 | 6583369.0 |
| **68777** 17-06-2020 23:45 | zVJPv84UY57bAof | 0.0 |  | 0.0 | 5910.000 | 7363272.0 |

68778 rows × 6 columns

In [23]: plant1\_sensor**.**drop(["SOURCE\_KEY", "PLANT\_ID"], axis**=**1, inplace**=True**) plant1\_sensor

#### Out[23]: DATE\_TIME AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

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

**0** 2020-05-15 00:00:00  22.857507 0.0

3182 rows × 4 columns

In [24]: plant2\_generation**.**rename(columns**=**{"SOURCE\_KEY":"INVERTER\_ID"}, inplace**=True**) plant2\_generation

#### Out[24]: DATE\_TIME PLANT\_ID INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

**0** 2020-05-15 00:00:00 4136001 4UPUqMRk7TRMgml 0.0 0.0 9425.000000 2.429011e+06

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 2020-05-15 00:00:00 | 4136001 | 81aHJ1q11NBPMrL | 0.0 | 0.0 | 0.000000 | 1.215279e+09 |
| **2** | 2020-05-15 00:00:00 | 4136001 | 9kRcWv60rDACzjR | 0.0 | 0.0 | 3075.333333 | 2.247720e+09 |
| **3** | 2020-05-15 00:00:00 | 4136001 | Et9kgGMDl729KT4 | 0.0 | 0.0 | 269.933333 | 1.704250e+06 |
| **4** | 2020-05-15 00:00:00 | 4136001 | IQ2d7wF4YD8zU1Q | 0.0 | 0.0 | 3177.000000 | 1.994153e+07 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **67693** | 2020-06-17 23:45:00 | 4136001 | q49J1IKaHRwDQnt | 0.0 | 0.0 | 4157.000000 | 5.207580e+05 |
| **67694** | 2020-06-17 23:45:00 | 4136001 | rrq4fwE8jgrTyWY | 0.0 | 0.0 | 3931.000000 | 1.211314e+08 |
| **67695** | 2020-06-17 23:45:00 | 4136001 | vOuJvMaM2sgwLmb | 0.0 | 0.0 | 4322.000000 | 2.427691e+06 |
| **67696** | 2020-06-17 23:45:00 | 4136001 | xMbIugepa2P7lBB | 0.0 | 0.0 | 4218.000000 | 1.068964e+08 |
| **67697** | 2020-06-17 23:45:00 | 4136001 | xoJJ8DcxJEcupym | 0.0 | 0.0 | 4316.000000 | 2.093357e+08 |

67698 rows × 7 columns

In [25]: plant2\_generation**.**drop("PLANT\_ID", axis**=**1, inplace**=True**) plant2\_generation

#### Out[25]: DATE\_TIME INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

**0** 2020-05-15 00:00:00 4UPUqMRk7TRMgml 0.0 0.0 9425.000000 2.429011e+06

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** 2020-05-15 00:00:00 | 81aHJ1q11NBPMrL |  | 0.0 | 0.0 | 0.000000 | 1.215279e+09 |
| **2** 2020-05-15 00:00:00 | 9kRcWv60rDACzjR |  | 0.0 | 0.0 | 3075.333333 | 2.247720e+09 |
| **3** 2020-05-15 00:00:00 | Et9kgGMDl729KT4 |  | 0.0 | 0.0 | 269.933333 | 1.704250e+06 |

**4** 2020-05-15 00:00:00 IQ2d7wF4YD8zU1Q 0.0 0.0 3177.000000 1.994153e+07

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **...** | ... | ... | ... | ... | ... | ... |
| **67693** | 2020-06-17 23:45:00 | q49J1IKaHRwDQnt | 0.0 | 0.0 | 4157.000000 | 5.207580e+05 |
| **67694** | 2020-06-17 23:45:00 | rrq4fwE8jgrTyWY | 0.0 | 0.0 | 3931.000000 | 1.211314e+08 |
| **67695** | 2020-06-17 23:45:00 | vOuJvMaM2sgwLmb | 0.0 | 0.0 | 4322.000000 | 2.427691e+06 |
| **67696** | 2020-06-17 23:45:00 | xMbIugepa2P7lBB | 0.0 | 0.0 | 4218.000000 | 1.068964e+08 |

1. 2020-06-17 23:45:00 xoJJ8DcxJEcupym 0.0 0.0 4316.000000 2.093357e+08
2. rows × 6 columns

In [26]: plant2\_sensor**.**drop(["SOURCE\_KEY", "PLANT\_ID"], axis**=**1, inplace**=True**) plant2\_sensor

#### Out[26]: DATE\_TIME AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

**0** 2020-05-15 00:00:00 27.004764 25.060789 0.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | 2020-05-15 00:15:00 | 26.880811 | 24.421869 | 0.0 |
| **2** | 2020-05-15 00:30:00 | 26.682055 | 24.427290 | 0.0 |
| **3** | 2020-05-15 00:45:00 | 26.500589 | 24.420678 | 0.0 |
| **4** | 2020-05-15 01:00:00 | 26.596148 | 25.088210 | 0.0 |
| **...** | ... | ... | ... | ... |
| **3254** | 2020-06-17 22:45:00 | 23.511703 | 22.856201 | 0.0 |
| **3255** | 2020-06-17 23:00:00 | 23.482282 | 22.744190 | 0.0 |
| **3256** | 2020-06-17 23:15:00 | 23.354743 | 22.492245 | 0.0 |
| **3257** | 2020-06-17 23:30:00 | 23.291048 | 22.373909 | 0.0 |
| **3258** | 2020-06-17 23:45:00 | 23.202871 | 22.535908 | 0.0 |

3259 rows × 4 columns

### Handling Missing & Duplicate Values

In [27]: plant1\_generation**.**isnull()**.**sum()

Out[27]: DATE\_TIME 0

INVERTER\_ID 0

DC\_POWER 0

AC\_POWER 0

DAILY\_YIELD 0

TOTAL\_YIELD 0 dtype: int64

In [28]: plant1\_sensor**.**isnull()**.**sum()

Out[28]: DATE\_TIME 0 AMBIENT\_TEMPERATURE 0 MODULE\_TEMPERATURE 0 IRRADIATION 0 dtype: int64

In [29]: plant2\_generation**.**isnull()**.**sum()

DATE\_TIME 0

Out[29]: INVERTER\_ID 0

DC\_POWER 0

AC\_POWER 0

DAILY\_YIELD 0

TOTAL\_YIELD 0 dtype: int64

In [30]: plant2\_sensor**.**isnull()**.**sum()

Out[30]: DATE\_TIME 0 AMBIENT\_TEMPERATURE 0 MODULE\_TEMPERATURE 0 IRRADIATION 0 dtype: int64

In [31]: plant1\_generation**.**duplicated()**.**sum() Out[31]: 0

In [32]: plant1\_sensor**.**duplicated()**.**sum() Out[32]: 0

In [33]: plant2\_generation**.**duplicated()**.**sum() Out[33]: 0

In [34]: plant2\_sensor**.**duplicated()**.**sum()

0 Out[34]:

There are no missing values or duplicated values in any of the datasets.

### Changing the data type of DATE\_TIME to datetime

In [35]: plant1\_generation["DATE\_TIME"] **=** pd**.**to\_datetime(plant1\_generation["DATE\_TIME"], format**=**'%d-%m-%Y

%H:% In [36]: plant1\_generation**.**dtypes Out[36]: DATE\_TIME datetime64[ns] INVERTER\_ID object

DC\_POWER float64

AC\_POWER float64 DAILY\_YIELD float64 TOTAL\_YIELD float64 dtype: object

In [37]: plant1\_sensor["DATE\_TIME"] **=** pd**.**to\_datetime(plant1\_generation["DATE\_TIME"], format**=**"%Y-%m-%d

%H:%M:%S

In [38]: plant1\_sensor**.**dtypes

DATE\_TIME datetime64[ns] Out[38]:

AMBIENT\_TEMPERATURE float64

MODULE\_TEMPERATURE float64

IRRADIATION float64 dtype: object

In [39]: plant2\_generation["DATE\_TIME"] **=** pd**.**to\_datetime(plant1\_generation["DATE\_TIME"], format**=**"%Y-%m-%d

%H:% In [40]: plant2\_generation**.**dtypes Out[40]: DATE\_TIME datetime64[ns] INVERTER\_ID object

DC\_POWER float64

AC\_POWER float64 DAILY\_YIELD float64 TOTAL\_YIELD float64 dtype: object

In [41]: plant2\_sensor["DATE\_TIME"] **=** pd**.**to\_datetime(plant1\_generation["DATE\_TIME"], format**=**"%Y-%m-%d

%H:%M:%S In [42]: plant2\_sensor**.**dtypes

DATE\_TIME datetime64[ns] Out[42]:

AMBIENT\_TEMPERATURE float64 MODULE\_TEMPERATURE float64

IRRADIATION float64 dtype: object

### Summary Statistics

PLANT 1

In [43]: plant1\_generation**.**describe()

#### Out[43]: DATE\_TIME DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

**count** 68778 68778.000000 68778.000000 68778.000000 6.877800e+04

|  |  |  |
| --- | --- | --- |
| **mean** 2020-06-01 08:02:49.458256896 3147.426211 | 307.802752 | 3295.968737 6.978712e+06 |

**min** 2020-05-15 00:00:00 0.000000 0.000000 0.000000 6.183645e+06

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **25%** |  | 2020-05-24 00:45:00 | 0.000000 | 0.000000 | 0.000000 | 6.512003e+06 |
| **50%** |  | 2020-06-01 14:30:00 | 429.000000 | 41.493750 | 2658.714286 | 7.146685e+06 |
| **75%** |  | 2020-06-09 20:00:00 | 6366.964286 | 623.618750 | 6274.000000 | 7.268706e+06 |
| **max** |  | 2020-06-17 23:45:00 | 14471.125000 | 1410.950000 | 9163.000000 | 7.846821e+06 |
| **std** |  | NaN | 4036.457169 | 394.396439 | 3145.178309 | 4.162720e+05 |

**Observations:**

1. The data was collected from 15 May 2020 to 17 June 2020. According to the India Meteorological

Department, monsoon covered the whole country by 26 June 2020 and hit Kerala on June 1. So if the plants are in south-west India then the values from 1st June onwards may be affected by rain.

1. The difference between the avg. DC power and the avg. AC power is a lot. Something seems wrong because only around 10% of the DC power is being converted into AC.
2. There's a pretty big jump in the Q2 to Q3 and from Q3 to Q4 values in DC\_POWER & AC\_POWER.

In [44]: plant1\_sensor**.**describe()

#### Out[44]: DATE\_TIME AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

**count** 3182 3182.000000 3182.000000 3182.000000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **mean** | 2020-05-15 19:36:36.543054592 | 25.531606 | 31.091015 | 0.228313 |
| **min** | 2020-05-15 00:00:00 | 20.398505 | 18.140415 | 0.000000 |
| **25%** | 2020-05-15 09:15:00 | 22.705182 | 21.090553 | 0.000000 |
| **50%** | 2020-05-15 18:15:00 | 24.613814 | 24.618060 | 0.024653 |
| **75%** | 2020-05-16 06:45:00 | 27.920532 | 41.307840 | 0.449588 |

**max** 2020-05-16 15:45:00 35.252486 65.545714 1.221652

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **std** |  | NaN | 3.354856 | 12.261222 | 0.300836 |

**Observations:**

There is a pretty big difference between the AMBIENT\_TEMPERATURE & MODULE\_TEMPERATURE values at Q3 & Q4.

PLANT 2

In [45]: plant2\_generation**.**describe()

#### Out[45]: DATE\_TIME DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD

**count** 67698 67698.000000 67698.000000 67698.000000 6.769800e+04

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **mean** 2020-06-01 01:45:59.159798016 | | | | 246.701961 | | 241.277825 | 3294.890295 6.589448e+08 |
| **min** 2020-05-15 00:00:00 | | | | 0.000000 | | 0.000000 | 0.000000 0.000000e+00 |
| **25%** 2020-05-23 21:15:00 | | | | 0.000000 | | 0.000000 | 272.750000 1.996494e+07 |
| **50%** 2020-06-01 08:15:00 | | | | 0.000000 | | 0.000000 | 2911.000000 2.826276e+08 |
| **75%** |  | 2020-06-09 10:45:00 | 446.591667 | | 438.215000 | | 5534.000000 1.348495e+09 |
| **max** |  | 2020-06-17 11:30:00 | 1420.933333 | | 1385.420000 | | 9873.000000 2.247916e+09 |
| **std** |  | NaN | 370.569597 | | 362.112118 | | 2919.448386 7.296678e+08 |

**Observations:**

1. The data collection dates of both plants are the same.
2. Unlike Plant 1, the DC\_POWER & AC\_POWER of Plant 2 is in line.
3. Consequently, there isn't much difference in the Q3 & Q4 values of DC\_POWER & AC\_POWER.

In [46]: plant2\_sensor**.**describe()

#### Out[46]: DATE\_TIME AMBIENT\_TEMPERATURE MODULE\_TEMPERATURE IRRADIATION

**count** 3259 3259.000000 3259.000000 3259.000000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **mean** |  | 28.069400 | 32.772408 | 0.232737 |
| **min** | 2020-05-15 00:00:00 | 20.942385 | 20.265123 | 0.000000 |
| **25%** | 2020-05-15 09:15:00 | 24.602135 | 23.716881 | 0.000000 |
| **50%** | 2020-05-15 18:30:00 | 26.981263 | 27.534606 | 0.019040 |
| **75%** | 2020-05-16 07:30:00 | 31.056757 | 40.480653 | 0.438717 |
| **max** | 2020-05-16 16:45:00 | 39.181638 | 66.635953 | 1.098766 |
| **std** | NaN | 4.061556 | 11.344034 | 0.312693 |

**Observation:**

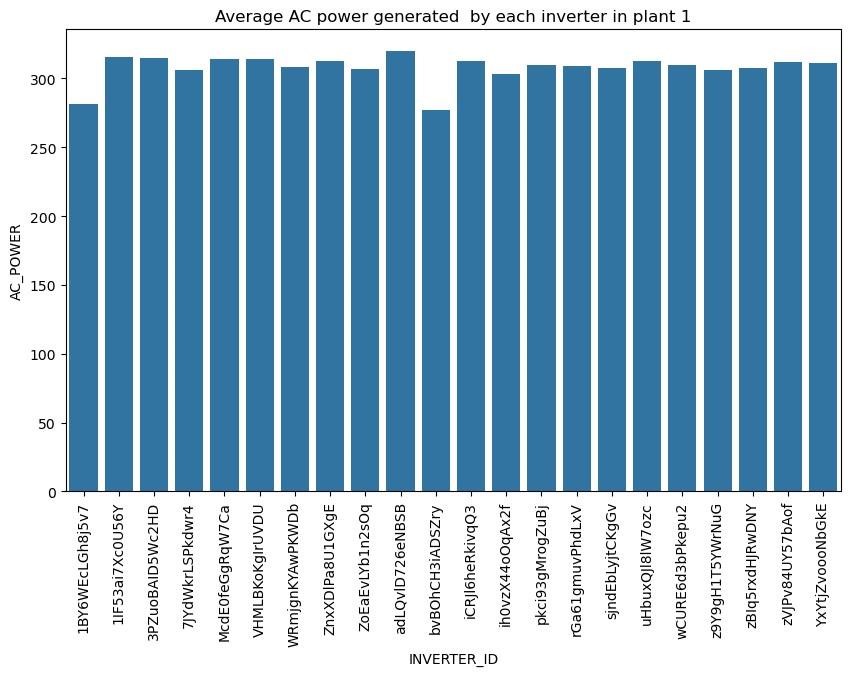
1. There isn't much difference in the avg, Q1, Q2 & Q3 values of AMBIENT\_TEMPERATURE & MODULE\_TEMPERATURE.
2. The max value of MODULE\_TEMPERATURE is much higher than the max value of AMBIENT\_TEMPERATURE.

**Comparison between Plant 1 & Plant 2:**

1. The average DC Power produced by Plant 1 is 13x the average DC power produced by Plant 2.
2. But the average AC Power produced by both is almost the same. There is something definitely wrong with Plant 1's DC Power data.
3. The daily yield of both the plants is similar.
4. But the average TOTAL\_YIELD OF Plant 2 is 7x of Plant 1.
5. Plant 1 & Plant 2 get the same amount of irradiation.
6. The average module & ambient temperatures of both plants is also similar.

### Analyzing the Inverters in both plants

In [47]: plt**.**figure(figsize**=**(10, 6)) sns**.**barplot(data**=**plant1\_generation, x**=**"INVERTER\_ID", y**=**"AC\_POWER", errorbar**=None**)**.**set(title**=**"Average plt**.**xticks(rotation**=**90) plt**.**show()

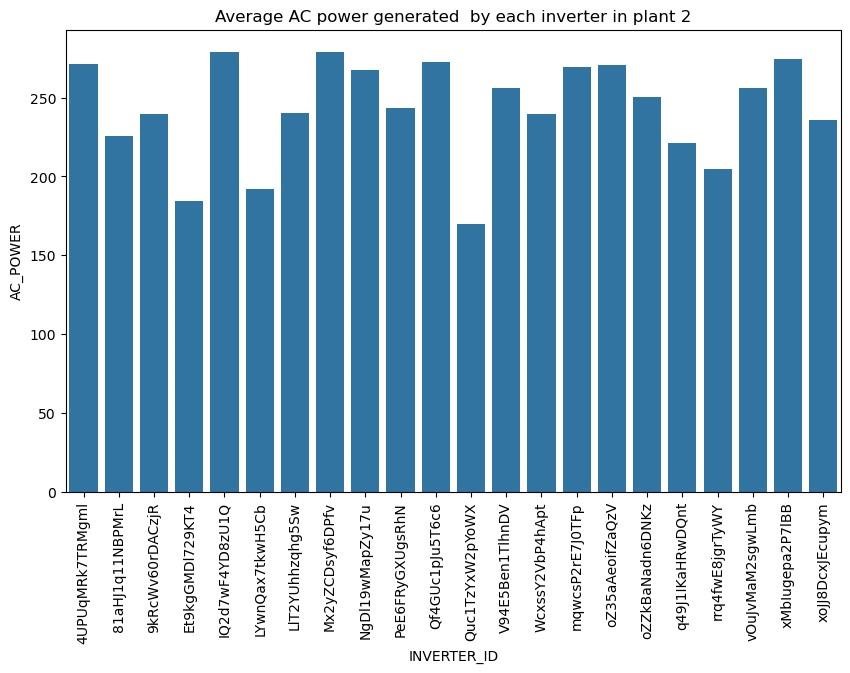


**Observation:**

All inverters in plant 1 produce the same amount of AC POWER except for two that produce less.

|  |
| --- |
| plt**.**figure(figsize**=**(10,6))  sns**.**barplot(data**=**plant2\_generation, x**=**"INVERTER\_ID", y**=**"AC\_POWER", errorbar**=None**)**.**set(title**=**"Average plt**.**xticks(rotation**=**90) plt**.**show() |

In [48]:



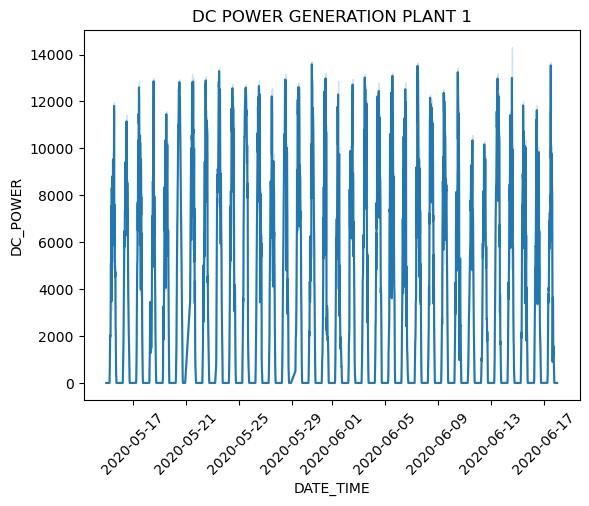
**Observation:**

The AC POWER production of inverters in plant 2 is all over the place, with 4 inverters performing very poorly.

### DC POWER Generation in the plants

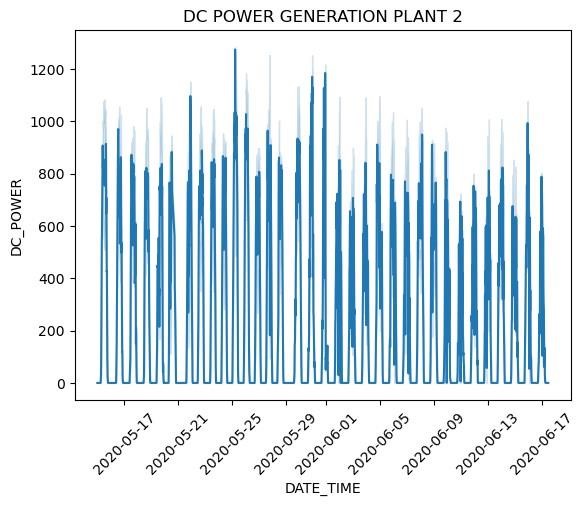
|  |
| --- |
| sns**.**lineplot(data**=**plant1\_generation, x**=**'DATE\_TIME', y**=**'DC\_POWER')**.**set(title**=**"DC POWER GENERATION PLAN plt**.**xticks(rotation**=**45) plt**.**show() |

In [49]:



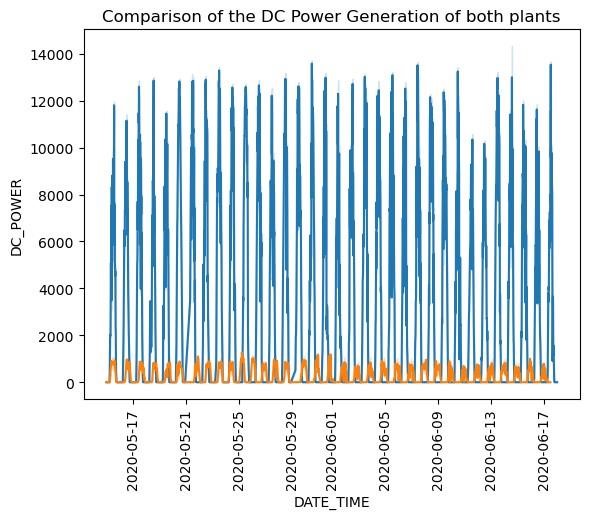
|  |
| --- |
| sns**.**lineplot(data**=**plant2\_generation, x**=**'DATE\_TIME', y**=**'DC\_POWER')**.**set(title**=**"DC POWER GENERATION PLAN plt**.**xticks(rotation**=**45) plt**.**show() |

In [50]:



|  |
| --- |
| sns**.**lineplot(data**=**plant1\_generation, x**=**'DATE\_TIME', y**=**'DC\_POWER') sns**.**lineplot(data**=**plant2\_generation,  x**=**'DATE\_TIME', y**=**'DC\_POWER') plt**.**title("Comparison of the DC Power Generation of both plants") plt**.**xticks(rotation**=**90) plt**.**show() |

In [51]:



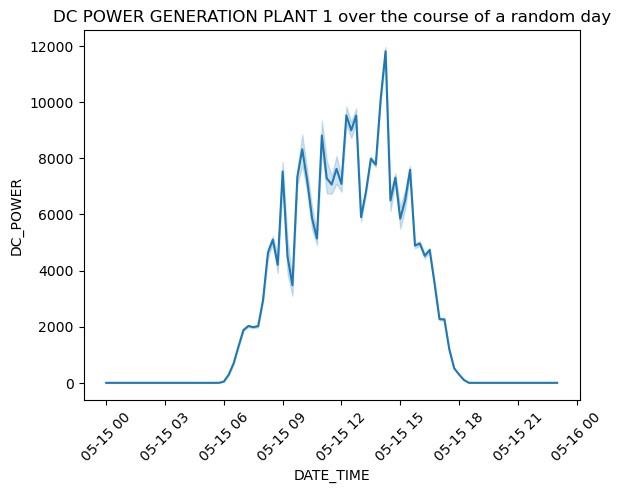
**Observations:**

The DC power produced by Plant 1 (blue) is way higher than plant 2 (orange) In [52]:

|  |  |
| --- | --- |
| selected\_day **=** '2020-05-15'  df\_single\_day **=** plant1\_generation[plant1\_generation['DATE\_TIME']**.**dt**.**date pd**.**to\_datetime(selected\_d |  |
| sns**.**lineplot(data**=**df\_single\_day, x**=**'DATE\_TIME', y**=**'DC\_POWER')**.**set(title**=**"DC POWER GENERATION PLANT 1 plt**.**xticks(rotation**=**45) plt**.**show() |  |

**==**

In [53]:



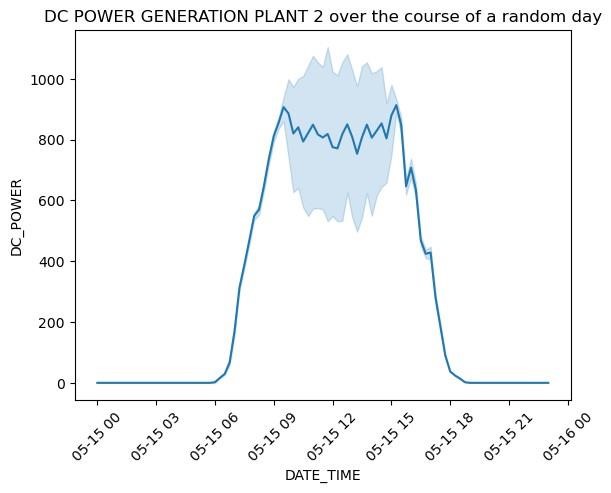
**Observation:**

DC power is generated only between 6 am to 6 pm which makes sense since those are the daylight hours. Most power was produced between the hours of 10 am to 3 pm.

|  |  |  |
| --- | --- | --- |
| df\_single\_day2 **=** pd**.**to\_datetime(selected\_ | plant2\_generation[plant2\_generation['DATE\_TIME']**.**dt**.**date |  |
| sns**.**lineplot(data**=**df\_single\_day2, x**=**'DATE\_TIME', y**=**'DC\_POWER')**.**set(title**=**"DC POWER GENERATION PLANT plt**.**xticks(rotation**=**45) plt**.**show() | |  |

In [54]: **==**

In [55]:

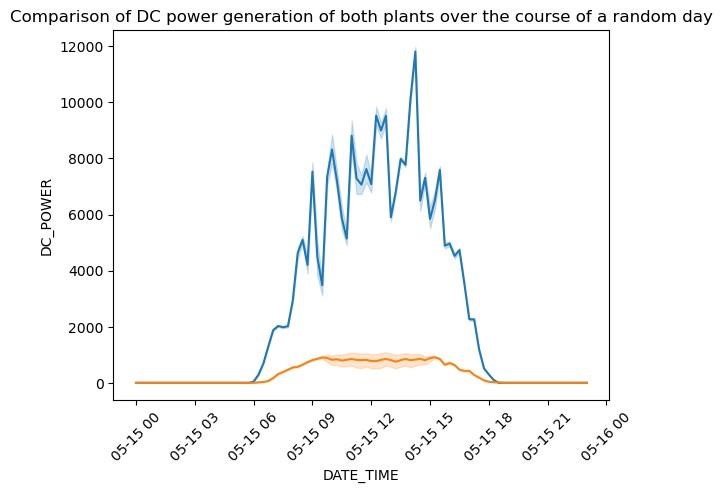


**Observations:**

Power was only produced during 6 am to 6 pm which makes sense since those are the daylight hours. Power production remained mostly constant throughout the day.

|  |
| --- |
| sns**.**lineplot(data**=**df\_single\_day, x**=**'DATE\_TIME', y**=**'DC\_POWER') sns**.**lineplot(data**=**df\_single\_day2, x**=**'DATE\_TIME', y**=**'DC\_POWER') plt**.**title("Comparison of DC power generation of both plants over the course of a random day") plt**.**xticks(rotation**=**45) plt**.**show() |

In [56]:



**Observations:**

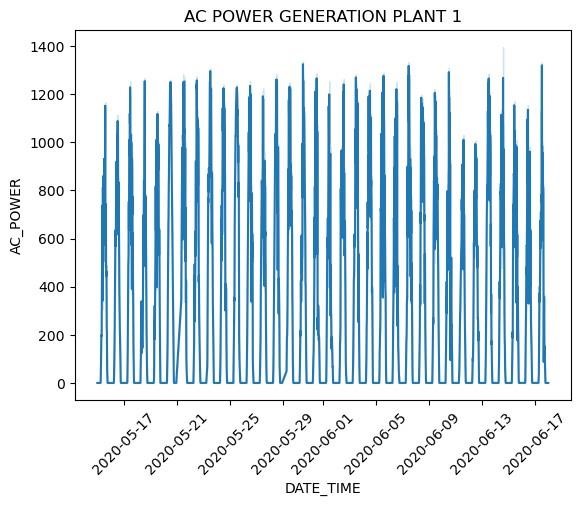
Plant 1 produces more DC power than plant 2

### AC Power generation in the plants

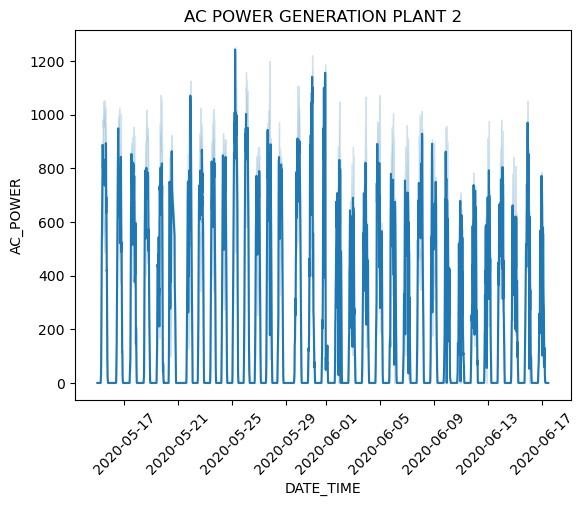
|  |
| --- |
| sns**.**lineplot(data**=**plant1\_generation, x**=**'DATE\_TIME', y**=**'AC\_POWER')**.**set(title**=**"AC POWER GENERATION PLAN plt**.**xticks(rotation**=**45) plt**.**show() |

In [57]:

sns**.**lineplot(data**=**plant2\_generation, x**=**'DATE\_TIME', y**=**'AC\_POWER')**.**set(title**=**"AC POWER GENERATION PLAN plt**.**xticks(rotation**=**45) plt**.**show()

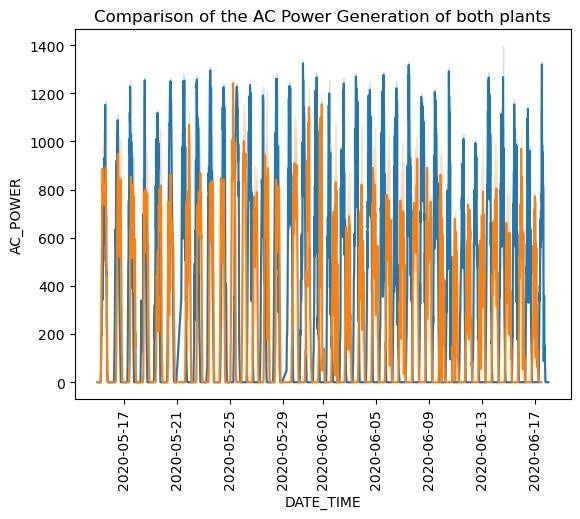


In [58]:



|  |
| --- |
| sns**.**lineplot(data**=**plant1\_generation, x**=**'DATE\_TIME', y**=**'AC\_POWER') sns**.**lineplot(data**=**plant2\_generation,  x**=**'DATE\_TIME', y**=**'AC\_POWER') plt**.**title("Comparison of the AC Power Generation of both plants") plt**.**xticks(rotation**=**90) plt**.**show() |

In [59]:



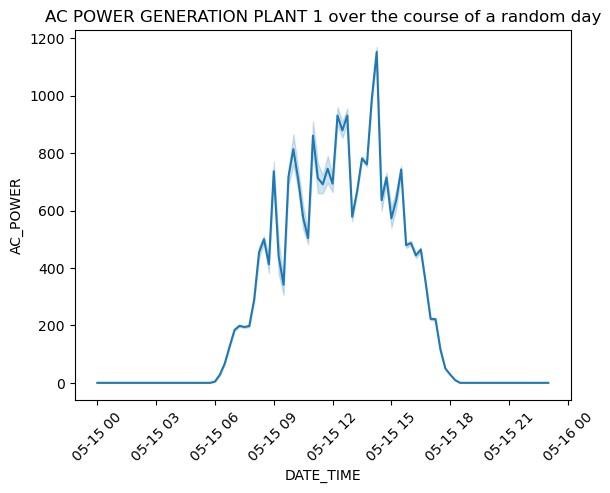
**Observations:**

The AC power produced by Plant 1 (blue) is way higher than plant 2 (orange)

|  |  |  |
| --- | --- | --- |
| df\_single\_day3 **=** pd**.**to\_datetime(selected\_ | plant1\_generation[plant1\_generation['DATE\_TIME']**.**dt**.**date |  |
| sns**.**lineplot(data**=**df\_single\_day3, x**=**'DATE\_TIME', y**=**'AC\_POWER')**.**set(title**=**"AC POWER GENERATION PLANT 1 plt**.**xticks(rotation**=**45) plt**.**show() | |  |

In [60]: **==**

In [61]:



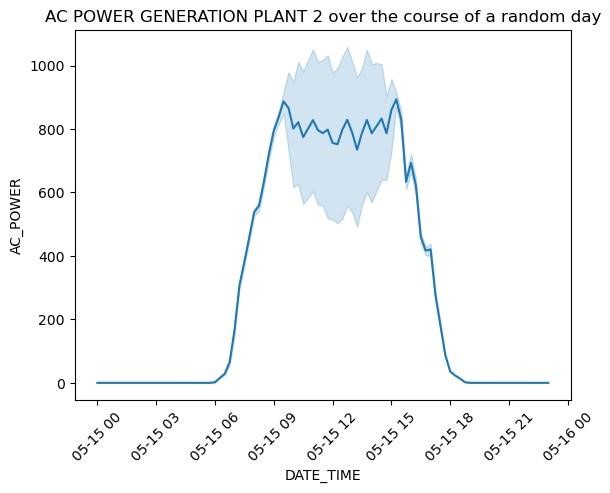
**Observation:**

DC power is generated only between 6 am to 6 pm which makes sense since those are the daylight hours. Most power was produced between the hours of 10 am to 3 pm.

|  |  |  |
| --- | --- | --- |
| df\_single\_day4 **=** pd**.**to\_datetime(selected\_ | plant2\_generation[plant2\_generation['DATE\_TIME']**.**dt**.**date |  |
| sns**.**lineplot(data**=**df\_single\_day4, x**=**'DATE\_TIME', y**=**'AC\_POWER')**.**set(title**=**"AC POWER GENERATION PLANT plt**.**xticks(rotation**=**45) plt**.**show() | |  |

In [62]: **==**

In [63]:

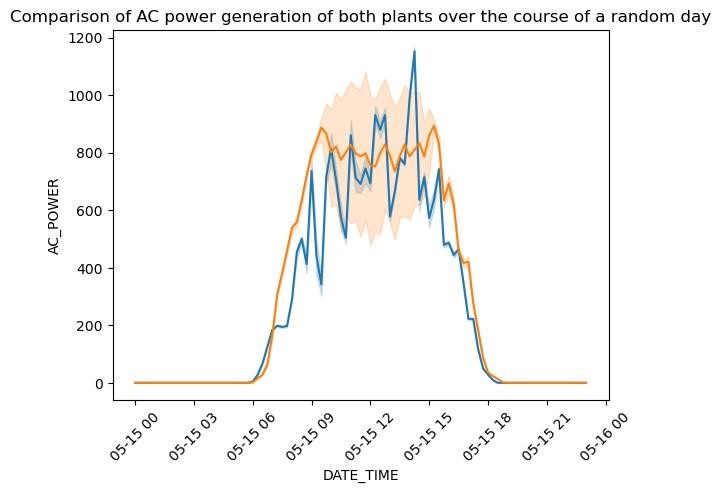


**Observations:**

Power was only produced during 6 am to 6 pm which makes sense since those are the daylight hours. Power production remained mostly constant throughout the day.

|  |
| --- |
| sns**.**lineplot(data**=**df\_single\_day3, x**=**'DATE\_TIME', y**=**'AC\_POWER') sns**.**lineplot(data**=**df\_single\_day4, x**=**'DATE\_TIME', y**=**'AC\_POWER')  plt**.**title("Comparison of AC power generation of both plants over the course of a random day") plt**.**xticks(rotation**=**45) plt**.**show() |

In [64]:

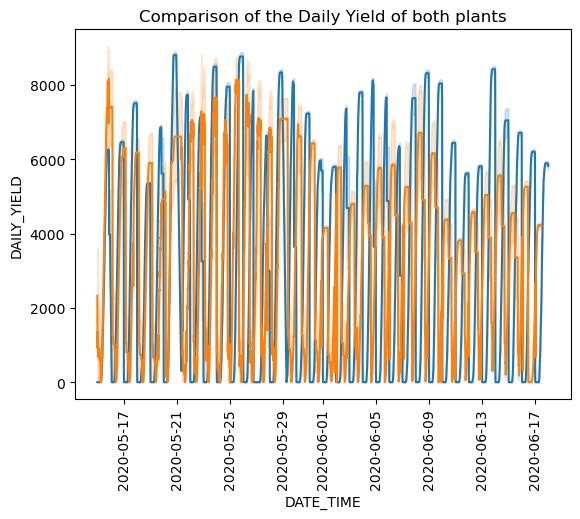


**Observations:**

AC Power produced by plant 1 throughout a day fluctatuates a lot but plant 2 remains fairly constant. But overall, there isn't a massive difference in scale the way there is with the DC power.

**Comparison of Daily Yield of Both Plants:**

|  |
| --- |
| In [65]: sns**.**lineplot(data**=**plant1\_generation, x**=**'DATE\_TIME', y**=**'DAILY\_YIELD') sns**.**lineplot(data**=**plant2\_generation,  x**=**'DATE\_TIME', y**=**'DAILY\_YIELD') plt**.**title("Comparison of the Daily Yield of both plants") plt**.**xticks(rotation**=**90) plt**.**show() |



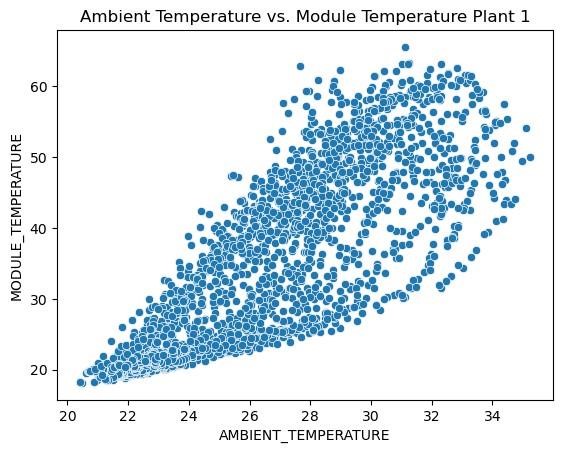
**Observations:**

1. On average, the daily yield of plant 1 (blue) is much greater than plant 2 (orange).
2. The daily yield of plant 2 dropped after June 1. We can only assume because of monsoon.

**Relationship between Ambient Temperature & Module Temperature:**

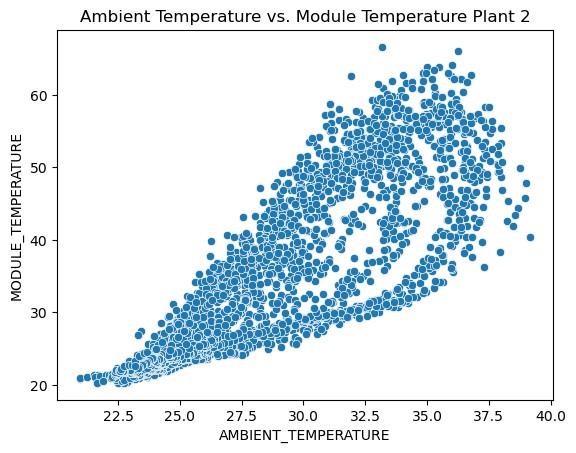
In [66]: sns**.**scatterplot(data**=**plant1\_sensor, x**=**"AMBIENT\_TEMPERATURE",

y**=**"MODULE\_TEMPERATURE")**.**set(title**=**"Ambie plt**.**show()



In [67]: sns**.**scatterplot(data**=**plant2\_sensor, x**=**"AMBIENT\_TEMPERATURE",

y**=**"MODULE\_TEMPERATURE")**.**set(title**=**"Ambie plt**.**show()

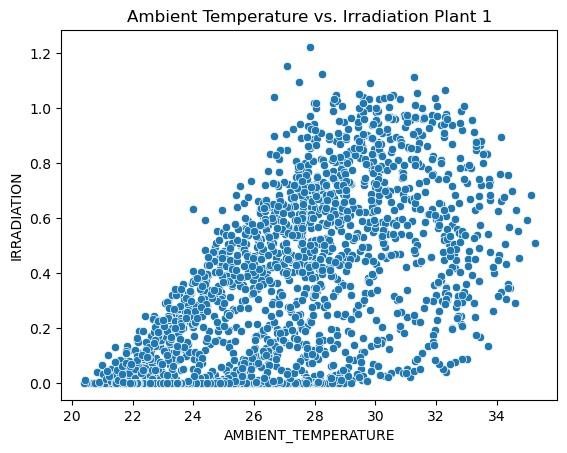


**Observation:**

There is a Positive Correlation between Module Temperature and Ambient Temperature. The Module Temperature increases as the Ambient Temperature increases.

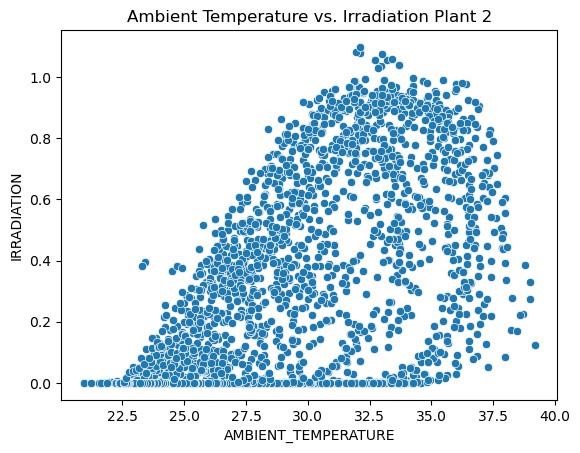
**Relationship between Ambient Temperature & Irradiation:**

|  |
| --- |
| In [68]: sns**.**scatterplot(data**=**plant1\_sensor, x**=**"AMBIENT\_TEMPERATURE", y**=**"IRRADIATION")**.**set(title**=**"Ambient  Tem plt**.**show() |



|  |
| --- |
| sns**.**scatterplot(data**=**plant2\_sensor, x**=**"AMBIENT\_TEMPERATURE", y**=**"IRRADIATION")**.**set(title**=**"Ambient  Tem plt**.**show() |

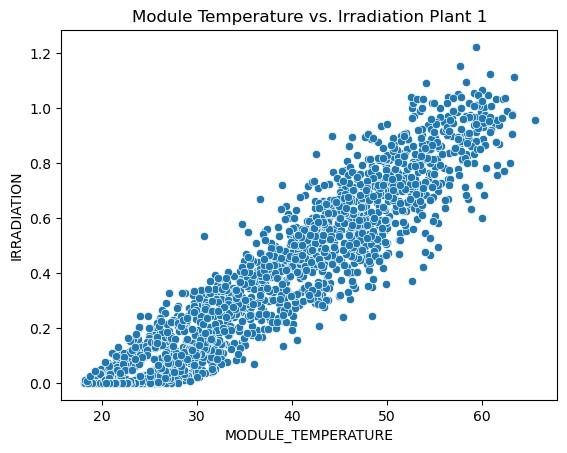
In [69]:



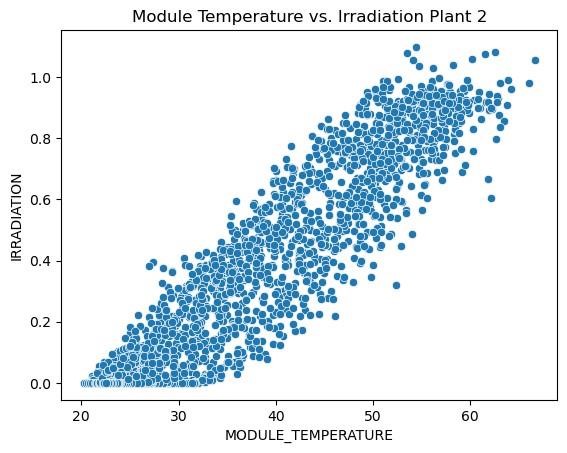
**Relationship between Module Temperature & Irradiation:**

|  |
| --- |
| sns**.**scatterplot(data**=**plant1\_sensor, x**=**"MODULE\_TEMPERATURE", y**=**"IRRADIATION")**.**set(title**=**"Module  Tempe plt**.**show() |

In [70]:



|  |
| --- |
| In [71]: sns**.**scatterplot(data**=**plant2\_sensor, x**=**"MODULE\_TEMPERATURE", y**=**"IRRADIATION")**.**set(title**=**"Module  Tempe plt**.**show() |



**Observations:**

There is a Positive Correlation between Irradiation and Module Temperature. Irradiation increases as Module Temperature increases.

### Merging the Power Generation + Weather Sensor Data

In [81]: plant1 **=** pd**.**merge(plant1\_generation, plant1\_sensor, on**=**"DATE\_TIME", how**=**"left") plant1**.**reset\_index(drop**=True**)

#### Out[81]: DATE\_TIME INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD AMBIENT\_TEMPERATURE

2020-05-15

**0** 1BY6WEcLGh8j5v7 0.0 0.0 0.000 6259559.0 25.184316

00:00:00

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 2020-05-15 00:00:00 | | 1BY6WEcLGh8j5v7 | | | 0.0 | 0.0 | 0.000 | | 6259559.0 | 25.084589 | |
| **2** | 2020-05-15 00:00:00 | | 1BY6WEcLGh8j5v7 | | | 0.0 | 0.0 | 0.000 | | 6259559.0 | 24.935753 | |
| **3** | 2020-05-15 00:00:00 | | 1BY6WEcLGh8j5v7 | | | 0.0 | 0.0 | 0.000 | | 6259559.0 | 24.846130 | |
| **4** | 2020-05-15 00:00:00 | | 1BY6WEcLGh8j5v7 | | | 0.0 | 0.0 | 0.000 | | 6259559.0 | 24.621525 | |
| **...** | ... | | ... | | | ... | ... | ... | | ... | ... | |
| **134125** | 2020-06-17 23:45:00 | | uHbuxQJl8lW7ozc | | | 0.0 | 0.0 | 5967.000 | | 7287002.0 | NaN | |
| **134126** | 2020-06-17 23:45:00 | | wCURE6d3bPkepu2 | | | 0.0 | 0.0 | 5147.625 | | 7028601.0 | NaN | |
| **134127** | 2020-06-17  23:45:00 z9Y9gH1T5YWrNuG | | | | | 0.0 | 0.0 | 5819.000 | | 7251204.0 | NaN | |
| **134128** | | 2020-06-17 23:45:00 | | zBIq5rxdHJRwDNY |  | 0.0 | 0.0 | | 5817.000 | 6583369.0 | | NaN |
| **134129** | | 2020-06-17 23:45:00 | | zVJPv84UY57bAof |  | 0.0 | 0.0 | | 5910.000 | 7363272.0 | | NaN |

134130 rows × 9 columns

|  |
| --- |
| plant2 **=** pd**.**merge(plant2\_generation, plant2\_sensor, on**=**"DATE\_TIME", how**=**"left") plant2**.**reset\_index(drop**=True**) |

In [82]:

#### Out[82]: DATE\_TIME INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD AMBIENT\_TEMPERATURE

2020-05-15

**0** 4UPUqMRk7TRMgml 0.0 0.0 9425.0 2429011.0 27.004764

00:00:00

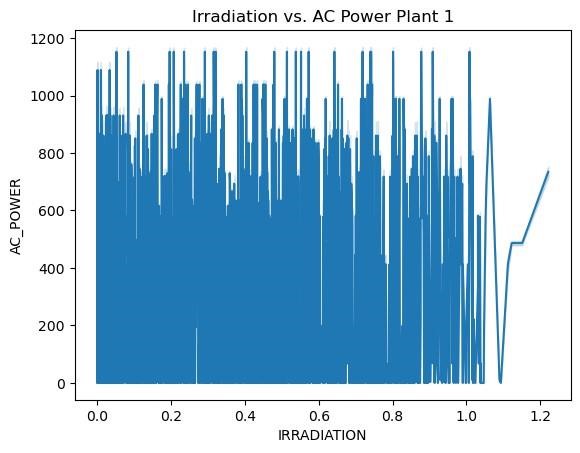
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 2020-05-15  00:00:00 4UPUqMRk7TRMgml | | 0.0 | 0.0 | 9425.0 | 2429011.0 | 26.880811 |
| **2** | 2020-05-15  00:00:00 4UPUqMRk7TRMgml | | 0.0 | 0.0 | 9425.0 | 2429011.0 | 26.682055 |
| **3** | 2020-05-15  00:00:00 4UPUqMRk7TRMgml | | 0.0 | 0.0 | 9425.0 | 2429011.0 | 26.500589 |
| **4** | 2020-05-15  00:00:00 4UPUqMRk7TRMgml | | 0.0 | 0.0 | 9425.0 | 2429011.0 | 26.596148 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **134651** | 2020-06-17 11:30:00 | q49J1IKaHRwDQnt | 0.0 | 0.0 | 4157.0 | 520758.0 | NaN |
| **134652** | 2020-06-17 11:30:00 | rrq4fwE8jgrTyWY | 0.0 | 0.0 | 3931.0 | 121131356.0 | NaN |
| **134653** | 2020-06-17 11:30:00 | vOuJvMaM2sgwLmb | 0.0 | 0.0 | 4322.0 | 2427691.0 | NaN |
| **134654** | 2020-06-17 11:30:00 | xMbIugepa2P7lBB | 0.0 | 0.0 | 4218.0 | 106896394.0 | NaN |
| **134655** | 2020-06-17 11:30:00 | xoJJ8DcxJEcupym | 0.0 | 0.0 | 4316.0 | 209335741.0 | NaN |

134656 rows × 9 columns

### Exploring the Relationship between AC Power and Irradiation

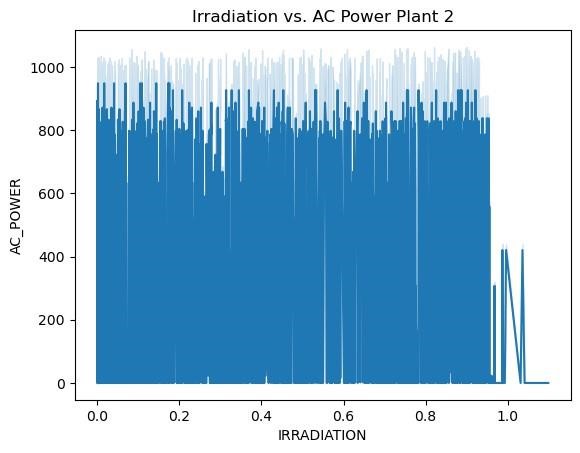
|  |
| --- |
| sns**.**lineplot(data**=**plant1, x**=**"IRRADIATION", y**=**"AC\_POWER")**.**set(title**=**"Irradiation vs. AC Power  Plant 1 plt**.**show() |

In [83]:



|  |
| --- |
| sns**.**lineplot(data**=**plant2, x**=**"IRRADIATION", y**=**"AC\_POWER")**.**set(title**=**"Irradiation vs. AC Power  Plant 2 plt**.**show() |

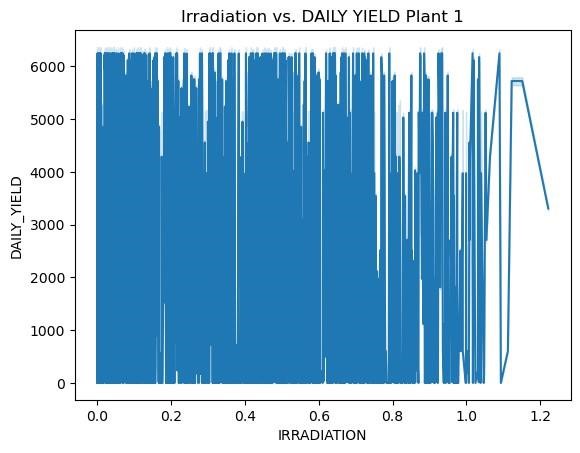
In [84]:



### Exploring the Relationship between Daily Yield and Irradiation

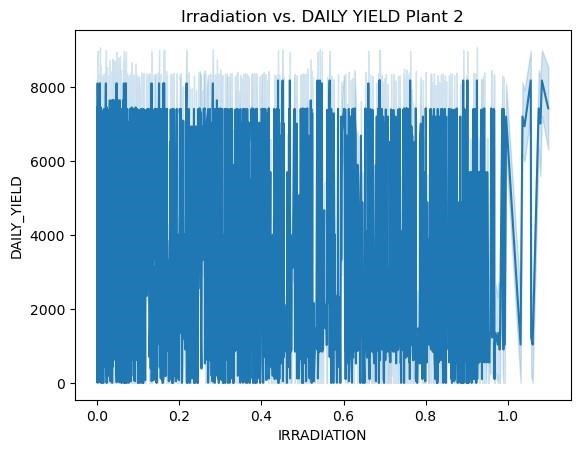
|  |
| --- |
| sns**.**lineplot(data**=**plant1, x**=**"IRRADIATION", y**=**"DAILY\_YIELD")**.**set(title**=**"Irradiation vs. DAILY  YIELD P plt**.**show() |

In [85]:



|  |
| --- |
| sns**.**lineplot(data**=**plant2, x**=**"IRRADIATION", y**=**"DAILY\_YIELD")**.**set(title**=**"Irradiation vs. DAILY  YIELD P plt**.**show() |

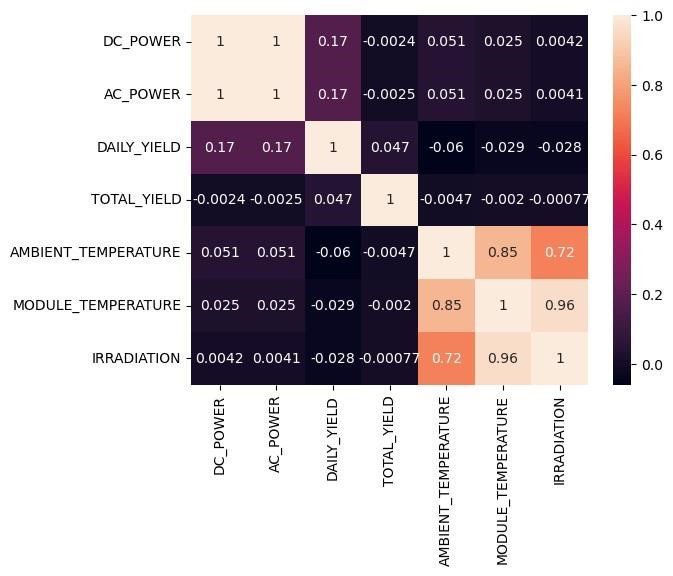
In [86]:



**Exploring Correlation between features:**

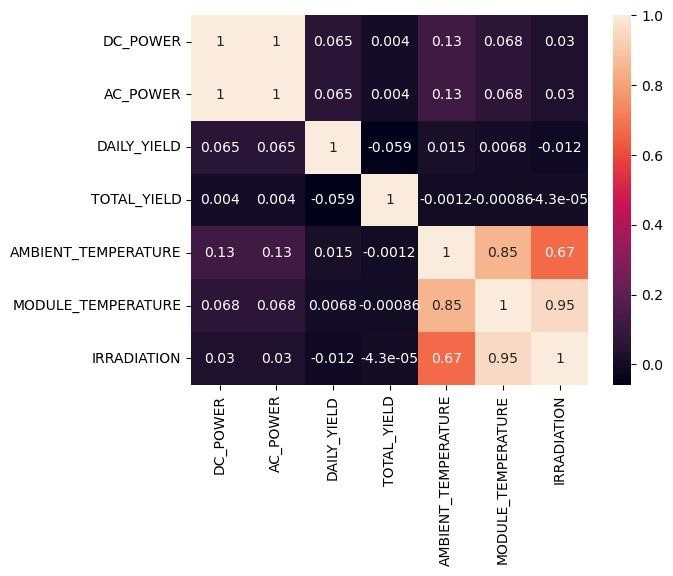
|  |
| --- |
| sns**.**heatmap(plant1**.**corr(numeric\_only**=True**), annot**=True**, fmt**=**'.2g') plt**.**show() |

In [87]:



|  |
| --- |
| sns**.**heatmap(plant2**.**corr(numeric\_only**=True**), annot**=True**, fmt**=**'.2g') plt**.**show() |

In [88]:



**Observations:**

1. Highest Positive Correlation is between Irradiation & Module Temperature.
2. A very high positive correlation between Module Temperature & Ambient Temperature
3. A high positive correlation between Irradiation & Ambient Temperature
4. Positive Correlations between all features except the ones that involve Total Yield.

### Exporting Merged Dataframes as csv

|  |
| --- |
| plant1**.**to\_csv('plant1\_merged.csv') plant2**.**to\_csv('plant2\_merged.csv') |

In [89]:

## Model Building:

**Name: Ananya Godse SAP ID: 60009220161**

### Importing the basic libraries

|  |
| --- |
| **import** pandas **as** pd  **import** matplotlib.pyplot **as** plt  **%matplotlib** inline **import** numpy **as** np **import** seaborn **as** sns  **from** sklearn.model\_selection **import** train\_test\_split, GridSearchCV |

In [1]:

### Importing the datasets

|  |
| --- |
| plant1 **=** pd**.**read\_csv("plant1\_merged.csv") plant1 |

In [2]:

Out[2]: **Unnamed:**

**DATE\_TIME INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD AMBIENT\_TEM**

**0**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 2020-05-15 00:00:00 | 1BY6WEcLGh8j5v7 | 0.0 | 0.0 | 0.000 | 6259559.0 |
| **1** | 1 | 2020-05-15 00:00:00 | 1BY6WEcLGh8j5v7 | 0.0 | 0.0 | 0.000 | 6259559.0 |
| **2** | 2 | 2020-05-15 00:00:00 | 1BY6WEcLGh8j5v7 | 0.0 | 0.0 | 0.000 | 6259559.0 |
| **3** | 3 | 2020-05-15 00:00:00 | 1BY6WEcLGh8j5v7 | 0.0 | 0.0 | 0.000 | 6259559.0 |
| **4** | 4 | 2020-05-15 00:00:00 | 1BY6WEcLGh8j5v7 | 0.0 | 0.0 | 0.000 | 6259559.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **134125** | 134125 | 2020-06-17 23:45:00 | uHbuxQJl8lW7ozc | 0.0 | 0.0 | 5967.000 | 7287002.0 |
| **134126** | 134126 | 2020-06-17 23:45:00 | wCURE6d3bPkepu2 | 0.0 | 0.0 | 5147.625 | 7028601.0 |
| **134127** | 134127 | 2020-06-17 23:45:00 | z9Y9gH1T5YWrNuG | 0.0 | 0.0 | 5819.000 | 7251204.0 |
| **134128** | 134128 | 2020-06-17 23:45:00 | zBIq5rxdHJRwDNY | 0.0 | 0.0 | 5817.000 | 6583369.0 |
| **134129** | 134129 | 2020-06-17 23:45:00 | zVJPv84UY57bAof | 0.0 | 0.0 | 5910.000 | 7363272.0 |

134130 rows × 10 columns

|  |
| --- |
| plant2 **=** pd**.**read\_csv("plant2\_merged.csv") plant2 |

In [3]:

Out[3]: **Unnamed:**

**DATE\_TIME INVERTER\_ID DC\_POWER AC\_POWER DAILY\_YIELD TOTAL\_YIELD AMBIENT\_TEM**

**0**

2020-05-15

**0** 0 4UPUqMRk7TRMgml 0.0 0.0 9425.0 2429011.0

00:00:00

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 1 | 2020-05-15 00:00:00 | 4UPUqMRk7TRMgml |  | 0.0 |  | 0.0 | 9425.0 | 2429011.0 |
| **2** | 2 | 2020-05-15 00:00:00 | 4UPUqMRk7TRMgml |  | 0.0 |  | 0.0 | 9425.0 | 2429011.0 |
| **3** | 3 | 2020-05-15 00:00:00 | 4UPUqMRk7TRMgml |  | 0.0 |  | 0.0 | 9425.0 | 2429011.0 |
| **4** | 4 | 2020-05-15 00:00:00 | 4UPUqMRk7TRMgml |  | 0.0 |  | 0.0 | 9425.0 | 2429011.0 |
| **...** | ... | ... | ... |  | ... |  | ... | ... | ... |
| **134651** | 134651 | 2020-06-17 11:30:00 | q49J1IKaHRwDQnt |  | 0.0 |  | 0.0 | 4157.0 | 520758.0 |
| **134652** | 134652 | 2020-06-17 11:30:00 | rrq4fwE8jgrTyWY |  | 0.0 |  | 0.0 | 3931.0 | 121131356.0 |
| **134653** | 134653 | 2020-06-17 11:30:00 | vOuJvMaM2sgwLmb |  | 0.0 |  | 0.0 | 4322.0 | 2427691.0 |
| **134654** | 134654 | 2020-06-17 11:30:00 | xMbIugepa2P7lBB |  | 0.0 |  | 0.0 | 4218.0 | 106896394.0 |
| **134655** | 134655 | 2020-06-17 11:30:00 | xoJJ8DcxJEcupym |  | 0.0 |  | 0.0 | 4316.0 | 209335741.0 |

134656 rows × 10 columns

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| plant1**.**drop(["Unnamed: 0", "INVERTER\_ID"], plant2**.**drop(["Unnamed: 0", "INVERTER\_ID"], | | | axis**=**1, axis**=**1, | inplace**=True**) inplace**=True**) |
|  | | |  |  |
| plant1**.**dtypes | | |  |  |
| DATE\_TIME object DC\_POWER float64  AC\_POWER float64  DAILY\_YIELD float64  TOTAL\_YIELD float64 AMBIENT\_TEMPERATURE float64  MODULE\_TEMPERATURE float64 IRRADIATION float64 dtype: object | | |  |  |
| plant1["DATE\_TIME"] | **=** | pd**.**to\_datetime(plant1["DATE\_TIME"], format**=**"%Y-%m-%d %H:%M:%S") | | |
| plant2["DATE\_TIME"] | **=** | pd**.**to\_datetime(plant2["DATE\_TIME"], format**=**"%Y-%m-%d %H:%M:%S") | | |
|  |  |  | | |
| plant1**.**dtypes |  |  | | |

In [4]:

In [5]:

Out[5]:

In [6]:

In [7]:

DATE\_TIME datetime64[ns] Out[7]:

DC\_POWER float64

AC\_POWER float64

DAILY\_YIELD float64

TOTAL\_YIELD float64 AMBIENT\_TEMPERATURE float64

MODULE\_TEMPERATURE float64 IRRADIATION float64 dtype: object

In [8]: plant2**.**dtypes

DATE\_TIME datetime64[ns] Out[8]:

DC\_POWER float64 AC\_POWER float64

DAILY\_YIELD float64

TOTAL\_YIELD float64

AMBIENT\_TEMPERATURE float64 MODULE\_TEMPERATURE float64 IRRADIATION float64 dtype: object

**Some Assumptions:**

We're going to keep DAILY\_YIELD as our target variable. So in a real life scenario, we'd just have the weather sensor data to predict the daily yield of a power plant and so we'll move forward with just that.

### Train Test Split

In [9]: reduced\_plant1 **=** plant1[["AMBIENT\_TEMPERATURE", "MODULE\_TEMPERATURE", "IRRADIATION", "DAILY\_YIELD"]]

In [10]: reduced\_plant2 **=** plant2[["AMBIENT\_TEMPERATURE", "MODULE\_TEMPERATURE", "IRRADIATION", "DAILY\_YIELD"]] In [11]: reduced\_plant1**.**isnull()**.**sum()

|  |  |
| --- | --- |
| Out[11]: | AMBIENT\_TEMPERATURE 65594  MODULE\_TEMPERATURE 65594  IRRADIATION 65594 DAILY\_YIELD 0 dtype: int64 |

In [12]: reduced\_plant1**.**fillna(value**=**0, inplace**=True**) reduced\_plant2**.**fillna(value**=**0, inplace**=True**)

C:\Users\Ananya\AppData\Local\Temp\ipykernel\_18472\3200288954.py:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexi ng.html#returning-a-view-versus-a-copy reduced\_plant1.fillna(value=0, inplace=True)

C:\Users\Ananya\AppData\Local\Temp\ipykernel\_18472\3200288954.py:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexi ng.html#returning-a-view-versus-a-copy reduced\_plant2.fillna(value=0, inplace=True)

|  |  |  |
| --- | --- | --- |
| train\_features **=** ["AMBIENT\_TEMPERATURE", "MODULE\_TEMPERATURE", "IRRADIATION"] X\_p1 **=** reduced\_plant1[train\_features] | | |
| y\_p1 **=** | reduced\_plant1["DAILY\_YIELD"] |  |
|  |  |  |
| X\_p2 **=** | reduced\_plant2[train\_features] |  |
| y\_p2 **=** | reduced\_plant2["DAILY\_YIELD"] |  |
|  |  |  |
| X\_p1\_train, X\_p1\_test, y\_p1\_train, y\_p1\_test **=** train\_test\_split(X\_p1, | | y\_p1, test\_size**=**0.2, random\_sta |
|  | |  |
| X\_p2\_train, X\_p2\_test, y\_p2\_train, y\_p2\_test **=** train\_test\_split(X\_p2, | | y\_p2, test\_size**=**0.2, random\_sta |

In [13]:

In [14]:

In [15]:

In [16]:

### LINEAR REGRESSION

In [17]: **from** sklearn.linear\_model **import** LinearRegression **from** sklearn.metrics **import** mean\_squared\_error

In [18]: linreg **=** LinearRegression() linreg**.**fit(X\_p1\_train, y\_p1\_train) pred **=** linreg**.**predict(X\_p1\_test)

In [19]: mean\_squared\_error(y\_p1\_test, pred)

7887607.970225091 Out[19]:

In [20]: linreg2 **=** LinearRegression() *# PLANT 2* linreg2**.**fit(X\_p2\_train, y\_p2\_train) pred2 **=** linreg2**.**predict(X\_p2\_test)

In [21]: mean\_squared\_error(y\_p2\_test, pred2)

9666845.977933416 Out[21]:

Default Linear Regression gives a pretty high MSE value which is not good. Now, we know from EDA that Module

Temperature, Ambient Temperature and Irradiation are all very highly correlated. So let's try and do Linear Regression with only Irradiation as the other two are a consequence of Irradiation.

In [22]: X\_p1\_rev **=** reduced\_plant1["IRRADIATION"]

In [23]: X\_p1\_train\_rev, X\_p1\_test\_rev, y\_p1\_train, y\_p1\_test **=** train\_test\_split(X\_p1\_rev, y\_p1, test\_size**=**0.

|  |
| --- |
| X\_p1\_train\_rev **=** X\_p1\_train\_rev**.**to\_numpy()  X\_p1\_test\_rev **=** X\_p1\_test\_rev**.**to\_numpy()  X\_p1\_train\_rev **=** X\_p1\_train\_rev**.**reshape(**-**1, 1) X\_p1\_test\_rev **=** X\_p1\_test\_rev**.**reshape(**-**1, 1)  linreg **=** LinearRegression()  linreg**.**fit(X\_p1\_train\_rev, y\_p1\_train) pred **=** linreg**.**predict(X\_p1\_test\_rev) |

In [24]:

In [25]: mean\_squared\_error(y\_p1\_test, pred)

8109974.187599193 Out[25]:

Okay the mse is still too high.

**using GridSearchCV to find the best parameters**

|  |
| --- |
| model **=** LinearRegression()  param\_grid **=** {  'fit\_intercept': [**True**, **False**],  'copy\_X': [**True**, **False**]  } grid\_search **=** GridSearchCV(estimator**=**model, param\_grid**=**param\_grid, cv**=**5, scoring**=**'neg\_mean\_squared\_e grid\_search**.**fit(X\_p1\_train\_rev, y\_p1\_train) print("Best hyperparameters:", grid\_search**.**best\_params\_) best\_model **=** grid\_search**.**best\_estimator\_  y\_pred **=** best\_model**.**predict(X\_p1\_test\_rev) mse **=** mean\_squared\_error(y\_p1\_test, y\_pred) print("Mean Squared Error:", mse) |

In [26]:

Best hyperparameters: {'copy\_X': True, 'fit\_intercept': True}

Mean Squared Error: 8109974.187599193

Even using GridSearchCV the mse is way too big.

### DECISION TREE REGRESSOR

|  |
| --- |
| **from** sklearn.tree **import** DecisionTreeRegressor decreg **=** DecisionTreeRegressor(random\_state **=** 42)  decreg**.**fit(X\_p1\_train, y\_p1\_train) *# using all weather sensor features* pred **=** decreg**.**predict(X\_p1\_test) |

In [27]:

In [28]: mean\_squared\_error(y\_p1\_test, pred)

5189993.872284841 Out[28]:

In [29]: decreg **=** DecisionTreeRegressor(random\_state **=** 42) decreg**.**fit(X\_p1\_train\_rev, y\_p1\_train) *# using only irradiation* pred **=** decreg**.**predict(X\_p1\_test\_rev)

In [30]: mean\_squared\_error(y\_p1\_test, pred)

|  |  |
| --- | --- |
| Out[30]: | 6589861.20479132 |

|  |
| --- |
| tree\_reg **=** DecisionTreeRegressor()  param\_grid **=** {  'max\_depth': [**None**, 10, 20, 30],  'min\_samples\_split': [2, 5, 10],  'min\_samples\_leaf': [1, 2, 4]  } grid\_search **=** GridSearchCV(tree\_reg, param\_grid, cv**=**5, scoring**=**'neg\_mean\_squared\_error', verbose**=**1) grid\_search**.**fit(X\_p1\_train, y\_p1\_train) print("Best parameters:", grid\_search**.**best\_params\_) best\_tree\_reg **=** grid\_search**.**best\_estimator\_  y\_pred **=** best\_tree\_reg**.**predict(X\_p1\_test) mse **=** mean\_squared\_error(y\_p1\_test, pred) print("Mean squared error on test set:", mse) |

In [31]:

Fitting 5 folds for each of 36 candidates, totalling 180 fits

Best parameters: {'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2}

Mean squared error on test set: 6589861.20479132 Decision Tree Regressor is a horrible model for this.

### LASSO REGRESSION

In [32]: **from** sklearn.linear\_model **import** Lasso lasso **=** Lasso(alpha**=**0.1) lasso**.**fit(X\_p1\_train, y\_p1\_train) pred **=** lasso**.**predict(X\_p1\_test)

C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.416e+10, tolerance:

8.788e+07 model = cd\_fast.enet\_coordinate\_descent(

In [33]: mean\_squared\_error(y\_p1\_test, pred)

7887634.380124176 Out[33]:

|  |
| --- |
| lasso\_reg **=** Lasso()  param\_grid **=** {  'alpha': [0.001, 0.01, 0.1, 1, 10]  } grid\_search **=** GridSearchCV(lasso\_reg, param\_grid, cv**=**5, scoring**=**'neg\_mean\_squared\_error', verbose**=**1) grid\_search**.**fit(X\_p1\_train, y\_p1\_train) print("Best parameters:", grid\_search**.**best\_params\_) best\_lasso\_reg **=** grid\_search**.**best\_estimator\_  y\_pred **=** best\_lasso\_reg**.**predict(X\_p1\_test) mse **=** mean\_squared\_error(y\_p1\_test, y\_pred) print("Mean squared error on test set:", mse) |

In [34]:

C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.257e+11, tolerance:

7.028e+07

model = cd\_fast.enet\_coordinate\_descent(

Fitting 5 folds for each of 5 candidates, totalling 25 fits

|  |
| --- |
| C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.248e+11, tolerance:  7.006e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.258e+11, tolerance:  7.035e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.253e+11, tolerance:  7.035e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.256e+11, tolerance:  7.050e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.450e+11, tolerance:  7.028e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.466e+11, tolerance:  7.006e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.469e+11, tolerance:  7.035e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.394e+11, tolerance:  7.035e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.416e+11, tolerance:  7.050e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.413e+10, tolerance:  7.028e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.601e+10, tolerance:  7.006e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.571e+10, tolerance:  7.035e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.975e+10, tolerance:  7.035e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.124e+10, tolerance:  7.050e+07  model = cd\_fast.enet\_coordinate\_descent(  C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.342e+08, tolerance:  7.028e+07  model = cd\_fast.enet\_coordinate\_descent( |

C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.841e+08, tolerance:

7.006e+07

model = cd\_fast.enet\_coordinate\_descent(

C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.712e+08, tolerance:

7.035e+07

model = cd\_fast.enet\_coordinate\_descent(

C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.372e+08, tolerance:

7.035e+07

model = cd\_fast.enet\_coordinate\_descent(

C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.586e+08, tolerance:

7.050e+07

model = cd\_fast.enet\_coordinate\_descent(

Best parameters: {'alpha': 0.001}

Mean squared error on test set: 7887607.837941845

C:\Users\Ananya\anaconda3\Lib\site-packages\sklearn\linear\_model\\_coordinate\_descent.py:628: Converg enceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.068e+11, tolerance:

8.788e+07 model = cd\_fast.enet\_coordinate\_descent(

### RIDGE REGRESSION

In [35]: **from** sklearn.linear\_model **import** Ridge ridge **=** Ridge(alpha**=**1.0) ridge**.**fit(X\_p1\_train, y\_p1\_train) pred **=** ridge**.**predict(X\_p1\_test)

In [36]: mean\_squared\_error(y\_p1\_test, pred)

7887612.985658665 Out[36]:

Both Lasso & Ridge Regression don't work out.

### New Training & Testing data (with DATE\_TIME)

In [37]: t\_reduced\_plant1 **=** plant1[["DATE\_TIME","DAILY\_YIELD"]] t\_reduced\_plant2 **=** plant2[["DATE\_TIME","DAILY\_YIELD"]]

In [38]: t\_reduced\_plant1**.**set\_index("DATE\_TIME", inplace**=True**) t\_reduced\_plant1

Out[38]: **DAILY\_YIELD**

**DATE\_TIME**

**2020-05-15 00:00:00** 0.000

|  |  |
| --- | --- |
| **2020-05-15 00:00:00** | 0.000 |
| **2020-05-15 00:00:00** | 0.000 |
| **2020-05-15 00:00:00** | 0.000 |
| **2020-05-15 00:00:00** | 0.000 |
| **...** | ... |
| **2020-06-17 23:45:00** | 5967.000 |
| **2020-06-17 23:45:00** | 5147.625 |
| **2020-06-17 23:45:00** | 5819.000 |
| **2020-06-17 23:45:00** | 5817.000 |
| **2020-06-17 23:45:00** | 5910.000 |

134130 rows × 1 columns

In [39]: t\_reduced\_plant2**.**set\_index("DATE\_TIME", inplace**=True**) t\_reduced\_plant2

Out[39]: **DAILY\_YIELD**

**DATE\_TIME**

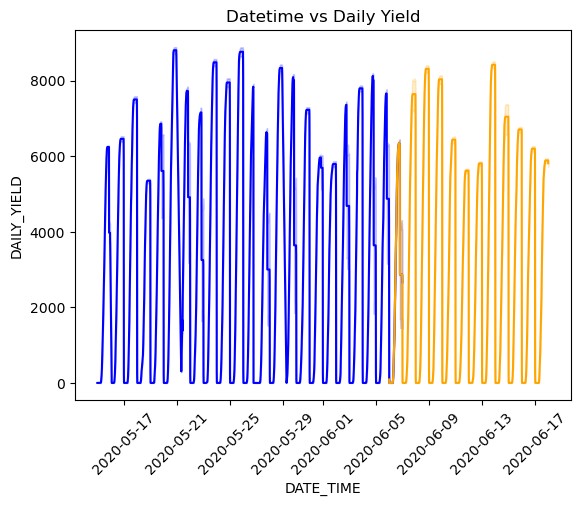
**2020-05-15 00:00:00** 9425.0

|  |  |  |
| --- | --- | --- |
| **2020-05-15 00:00:00** |  | 9425.0 |
| **2020-05-15 00:00:00** |  | 9425.0 |
| **2020-05-15 00:00:00** |  | 9425.0 |
| **2020-05-15 00:00:00** |  | 9425.0 |
| **...** |  | ... |
| **2020-06-17 11:30:00** |  | 4157.0 |
| **2020-06-17 11:30:00** |  | 3931.0 |
| **2020-06-17 11:30:00** |  | 4322.0 |
| **2020-06-17 11:30:00** |  | 4218.0 |
| **2020-06-17 11:30:00** |  | 4316.0 |

134656 rows × 1 columns

In [40]: split\_date **=** '2020-06-06' plant1\_train **=** t\_reduced\_plant1**.**loc[:split\_date] plant1\_test **=** t\_reduced\_plant1**.**loc[split\_date:]

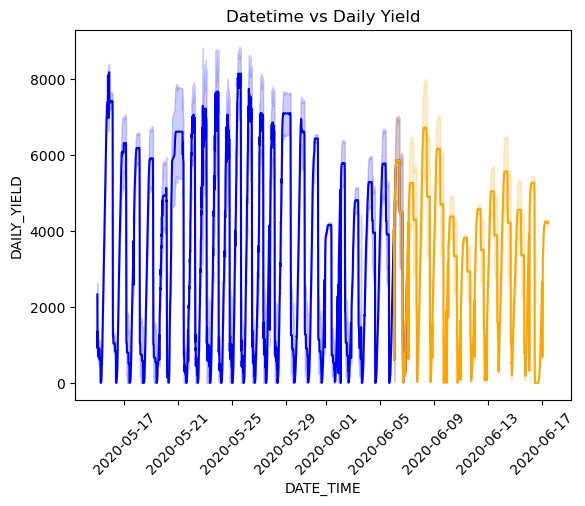
In [41]: sns**.**lineplot(data**=**plant1\_train, x**=**plant1\_train**.**index, y**=**plant1\_train["DAILY\_YIELD"], c**=**"blue") sns**.**lineplot(data**=**plant1\_test, x**=**plant1\_test**.**index, y**=**plant1\_test["DAILY\_YIELD"], c**=**"orange") plt**.**title("Datetime vs Daily Yield") plt**.**xticks(rotation**=**45) plt**.**show()



|  |
| --- |
| plant2\_train **=** t\_reduced\_plant2**.**loc[:split\_date] plant2\_test **=** t\_reduced\_plant2**.**loc[split\_date:] |

|  |
| --- |
| sns**.**lineplot(data**=**plant2\_train, x**=**plant2\_train**.**index, y**=**plant2\_train["DAILY\_YIELD"], c**=**"blue") sns**.**lineplot(data**=**plant2\_test, x**=**plant2\_test**.**index, y**=**plant2\_test["DAILY\_YIELD"], c**=**"orange") plt**.**title("Datetime vs Daily Yield") plt**.**xticks(rotation**=**45) plt**.**show() |

In [42]: In [43]:



|  |
| --- |
| **def** create\_features(df):  df **=** df**.**copy() df['hour'] **=** df**.**index**.**hour  df['day\_of\_week'] **=** df**.**index**.**dayofweek df['month'] **=** df**.**index**.**month df['year'] **=** df**.**index**.**year df['week\_of\_year'] **=** df**.**index**.**isocalendar()**.**week **return** df  t\_reduced\_plant1 **=** create\_features(t\_reduced\_plant1) plant1\_train **=** create\_features(plant1\_train) plant1\_test **=** create\_features(plant1\_test)  t\_reduced\_plant2 **=** create\_features(t\_reduced\_plant2) plant2\_train **=** create\_features(plant2\_train) plant2\_test **=** create\_features(plant2\_test) |

In [44]:

In [45]: X\_p1\_train **=** plant1\_train**.**iloc[:, 1:]

y\_p1\_train **=** plant1\_train**.**iloc[:, 0]

X\_p1\_test **=** plant1\_test**.**iloc[:, 1:] y\_p1\_test **=** plant1\_test**.**iloc[:, 0]

**XG BOOST REGRESSOR:**

In [46]: **import** xgboost **as** xgb

**PLANT 1:**

In [47]: reg **=** xgb**.**XGBRegressor(n\_estimators**=**1000, learning\_rate**=**0.01) reg**.**fit(X\_p1\_train, y\_p1\_train)

Out[47]: ▾ XGBRegressor

XGBRegressor(base\_score=None, booster=None, callbacks=None, colsample\_bylevel=None, colsample\_bynode=None, colsample\_bytree=None, device=None, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=None, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, multi\_strategy=None, n\_estimators=1000, n\_jobs=None, num parallel tree=None, random state=None, ...)

In [48]: predictions **=** reg**.**predict(X\_p1\_test) predictions

array([ 47.04413, 47.04413, 47.04413, ..., 7617.5845 , 7617.5845 ,

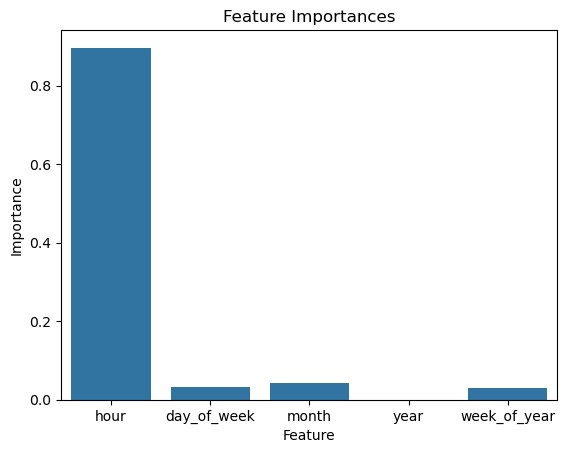
Out[48]:

7617.5845 ], dtype=float32)

In [49]: mse **=** mean\_squared\_error(y\_p1\_test, predictions) print(f"MSE = {mse:.2f}")

MSE = 2536624.12

In [50]: sns**.**barplot(x**=**reg**.**feature\_names\_in\_, y**=**reg**.**feature\_importances\_)**.**set(xlabel**=**'Feature', ylabel**=**'Import plt**.**show()

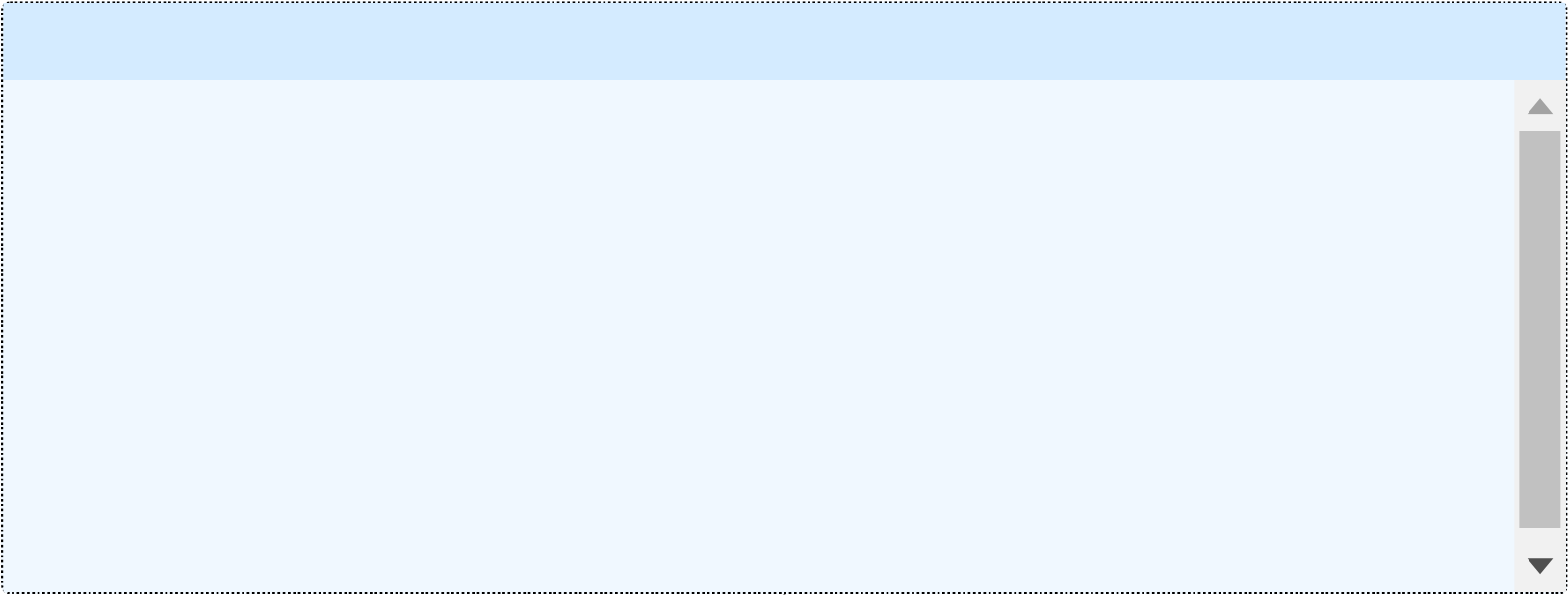


In [51]: X\_p1\_train **=** plant1\_train**.**iloc[:, 1] y\_p1\_train **=** plant1\_train**.**iloc[:, 0]

X\_p1\_test **=** plant1\_test**.**iloc[:, 1] y\_p1\_test **=** plant1\_test**.**iloc[:, 0]

In [52]: reg **=** xgb**.**XGBRegressor(n\_estimators**=**1000, learning\_rate**=**0.01) reg**.**fit(X\_p1\_train, y\_p1\_train)

Out[52]: ▾ XGBRegressor

XGBRegressor(base\_score=None, booster=None, callbacks=None, colsample\_bylevel=None, colsample\_bynode=None, colsample\_bytree=None, device=None, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=None, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, multi\_strategy=None, n\_estimators=1000, n\_jobs=None, num parallel tree=None, random state=None, ...)

In [53]: predictions **=** reg**.**predict(X\_p1\_test) predictions

array([ 25.496717, 25.496717, 25.496717, ..., 4983.6963 ,

Out[53]:

4983.6963 , 4983.6963 ], dtype=float32)

In [56]: rmse **=** np**.**sqrt(mean\_squared\_error(y\_p1\_test, predictions)) print(f"RMSE = {rmse:.2f}")

|  |
| --- |
| model **=** xgb**.**XGBRegressor() param\_grid **=** {  'n\_estimators': [100, 500, 1000],  'learning\_rate': [0.01, 0.05, 0.1],  }  grid\_search **=** GridSearchCV(estimator**=**model, param\_grid**=**param\_grid, cv**=**5, scoring**=**'neg\_mean\_squared\_e grid\_search**.**fit(X\_p1\_train, y\_p1\_train) print("Best parameters:", grid\_search**.**best\_params\_) |

RMSE = 1128.25 In [57]:

best\_model **=** grid\_search**.**best\_estimator\_

test\_predictions **=** best\_model**.**predict(X\_p1\_test) mse **=** np**.**mean((test\_predictions **-** y\_p1\_test) **\*\*** 2) print("Mean Squared Error on test set:", mse)

Fitting 5 folds for each of 9 candidates, totalling 45 fits

[CV] END ...............learning\_rate=0.01, n\_estimators=100; total time= 0.0s

[CV] END ...............learning\_rate=0.01, n\_estimators=100; total time= 0.0s

[CV] END ...............learning\_rate=0.01, n\_estimators=100; total time= 0.0s

[CV] END ...............learning\_rate=0.01, n\_estimators=100; total time= 0.0s [CV] END ...............learning\_rate=0.01, n\_estimators=100; total time= 0.0s

[CV] END ...............learning\_rate=0.01, n\_estimators=500; total time= 0.6s

[CV] END ...............learning\_rate=0.01, n\_estimators=500; total time= 0.6s

[CV] END ...............learning\_rate=0.01, n\_estimators=500; total time= 0.5s [CV] END ...............learning\_rate=0.01, n\_estimators=500; total time= 0.6s

[CV] END ...............learning\_rate=0.01, n\_estimators=500; total time= 0.5s

[CV] END ..............learning\_rate=0.01, n\_estimators=1000; total time= 1.0s

[CV] END ..............learning\_rate=0.01, n\_estimators=1000; total time= 1.1s [CV] END ..............learning\_rate=0.01, n\_estimators=1000; total time= 1.5s

[CV] END ..............learning\_rate=0.01, n\_estimators=1000; total time= 1.7s

[CV] END ..............learning\_rate=0.01, n\_estimators=1000; total time= 1.5s

[CV] END ...............learning\_rate=0.05, n\_estimators=100; total time= 0.2s [CV] END ...............learning\_rate=0.05, n\_estimators=100; total time= 0.1s

[CV] END ...............learning\_rate=0.05, n\_estimators=100; total time= 0.1s

[CV] END ...............learning\_rate=0.05, n\_estimators=100; total time= 0.1s

[CV] END ...............learning\_rate=0.05, n\_estimators=100; total time= 0.1s [CV] END ...............learning\_rate=0.05, n\_estimators=500; total time= 0.7s

[CV] END ...............learning\_rate=0.05, n\_estimators=500; total time= 0.7s

[CV] END ...............learning\_rate=0.05, n\_estimators=500; total time= 0.7s

[CV] END ...............learning\_rate=0.05, n\_estimators=500; total time= 0.7s [CV] END ...............learning\_rate=0.05, n\_estimators=500; total time= 0.7s

[CV] END ..............learning\_rate=0.05, n\_estimators=1000; total time= 1.4s

[CV] END ..............learning\_rate=0.05, n\_estimators=1000; total time= 1.4s

[CV] END ..............learning\_rate=0.05, n\_estimators=1000; total time= 1.4s [CV] END ..............learning\_rate=0.05, n\_estimators=1000; total time= 1.5s

[CV] END ..............learning\_rate=0.05, n\_estimators=1000; total time= 1.4s

[CV] END ................learning\_rate=0.1, n\_estimators=100; total time= 0.1s

[CV] END ................learning\_rate=0.1, n\_estimators=100; total time= 0.1s [CV] END ................learning\_rate=0.1, n\_estimators=100; total time= 0.1s

[CV] END ................learning\_rate=0.1, n\_estimators=100; total time= 0.2s

[CV] END ................learning\_rate=0.1, n\_estimators=100; total time= 0.1s

[CV] END ................learning\_rate=0.1, n\_estimators=500; total time= 0.7s [CV] END ................learning\_rate=0.1, n\_estimators=500; total time= 0.9s

[CV] END ................learning\_rate=0.1, n\_estimators=500; total time= 0.8s

[CV] END ................learning\_rate=0.1, n\_estimators=500; total time= 0.8s

[CV] END ................learning\_rate=0.1, n\_estimators=500; total time= 0.7s [CV] END ...............learning\_rate=0.1, n\_estimators=1000; total time= 1.6s

[CV] END ...............learning\_rate=0.1, n\_estimators=1000; total time= 1.5s

[CV] END ...............learning\_rate=0.1, n\_estimators=1000; total time= 1.6s

[CV] END ...............learning\_rate=0.1, n\_estimators=1000; total time= 1.4s

[CV] END ...............learning\_rate=0.1, n\_estimators=1000; total time= 1.6s

Best parameters: {'learning\_rate': 0.01, 'n\_estimators': 500}

Mean Squared Error on test set: 1282434.1400564422

The best model for plant 1 is XGBoostRegressor with only the feature hour, n\_estimators=1000 and learning rate=0.01

**PLANT 2:**

|  |
| --- |
| plant2\_train |

In [58]:

Out[58]: **DAILY\_YIELD hour day\_of\_week month year week\_of\_year**

**DATE\_TIME**

**2020-05-15 00:00:00** 9425.000000 0 4 5 2020 20

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2020-05-15 00:00:00** | 9425.000000 | 0 |  | 4 | 5 | 2020 |  | 20 |
| **2020-05-15 00:00:00** | 9425.000000 | 0 |  | 4 | 5 | 2020 |  | 20 |
| **2020-05-15 00:00:00** | 9425.000000 | 0 |  | 4 | 5 | 2020 |  | 20 |
| **2020-05-15 00:00:00** | 9425.000000 | 0 |  | 4 | 5 | 2020 |  | 20 |
| **...** | ... | ... |  | ... | ... | ... |  | ... |
| **2020-06-06 23:45:00** | 1078.000000 | 23 |  | 5 | 6 | 2020 |  | 23 |
| **2020-06-06 23:45:00** | 4292.428571 | 23 |  | 5 | 6 | 2020 |  | 23 |
| **2020-06-06 23:45:00** | 4162.533333 | 23 |  | 5 | 6 | 2020 |  | 23 |
| **2020-06-06 23:45:00** | 4616.133333 | 23 |  | 5 | 6 | 2020 |  | 23 |
| **2020-06-06 23:45:00** | 1079.000000 | 23 |  | 5 | 6 | 2020 |  | 23 |

112548 rows × 6 columns

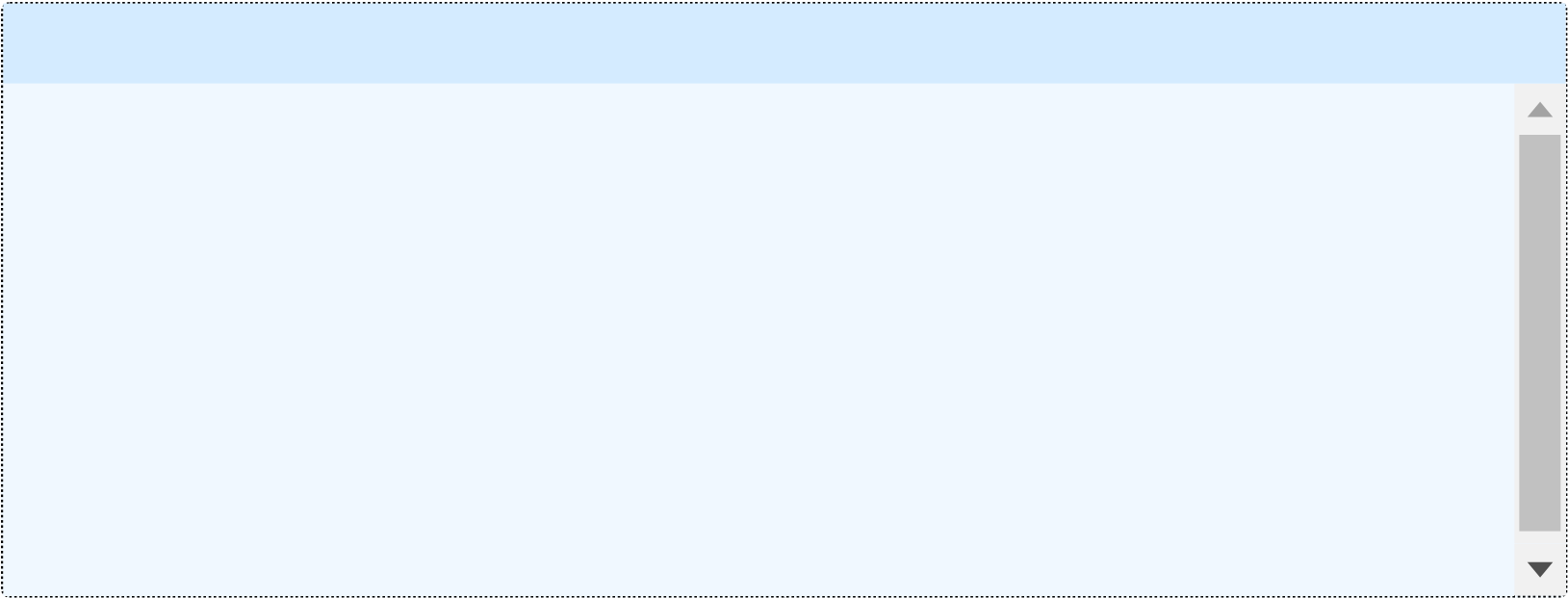
In [59]: X\_p2\_train **=** plant2\_train**.**iloc[:, 1:]

y\_p2\_train **=** plant2\_train**.**iloc[:, 0]

X\_p2\_test **=** plant2\_test**.**iloc[:, 1:] y\_p2\_test **=** plant2\_test**.**iloc[:, 0]

In [60]: reg **=** xgb**.**XGBRegressor(n\_estimators**=**1000, learning\_rate**=**0.01) reg**.**fit(X\_p2\_train, y\_p2\_train)

Out[60]: ▾ XGBRegressor

XGBRegressor(base\_score=None, booster=None, callbacks=None, colsample\_bylevel=None, colsample\_bynode=None, colsample\_bytree=None, device=None, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=None, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, multi\_strategy=None, n\_estimators=1000, n\_jobs=None, num\_parallel\_tree=None, random\_state=None, ...)

In [61]: predictions **=** reg**.**predict(X\_p2\_test) predictions

array([3644.4348, 3644.4348, 3644.4348, ..., 975.2371, 975.2371,

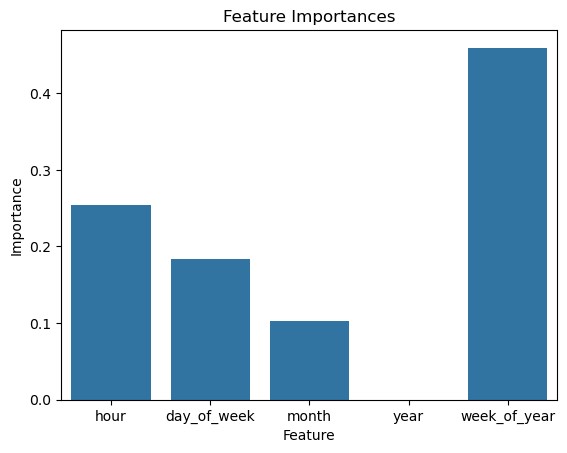
Out[61]:

975.2371], dtype=float32)

In [62]: rmse **=** np**.**sqrt(mean\_squared\_error(y\_p2\_test, predictions)) print(f"RMSE = {rmse:.2f}")

RMSE = 1282434.14

In [63]: sns**.**barplot(x**=**reg**.**feature\_names\_in\_, y**=**reg**.**feature\_importances\_)**.**set(xlabel**=**'Feature', ylabel**=**'Import plt**.**show()

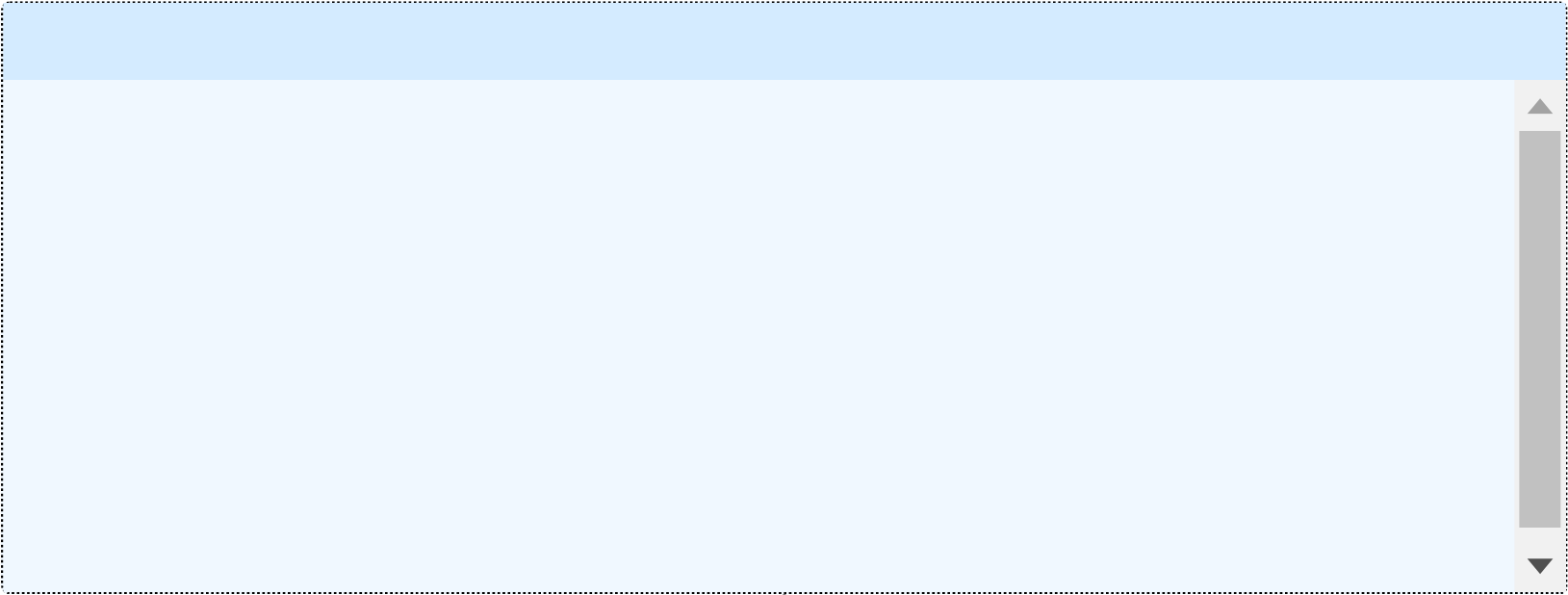


In [64]: X\_p2\_train **=** plant2\_train**.**drop(['year', 'month'], axis**=**1) y\_p2\_train **=** plant2\_train**.**iloc[:, 0]

X\_p2\_test **=** plant2\_test**.**drop(['year', 'month'], axis**=**1) y\_p2\_test **=** plant2\_test**.**iloc[:, 0]

In [65]: reg **=** xgb**.**XGBRegressor(n\_estimators**=**1000, learning\_rate**=**0.01) reg**.**fit(X\_p2\_train, y\_p2\_train)

Out[65]: ▾ XGBRegressor

XGBRegressor(base\_score=None, booster=None, callbacks=None, colsample\_bylevel=None, colsample\_bynode=None, colsample\_bytree=None, device=None, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=None, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, multi\_strategy=None, n\_estimators=1000, n\_jobs=None, num parallel tree=None, random state=None, ...)

In [66]: predictions **=** reg**.**predict(X\_p2\_test) predictions

array([1067.047 , 5523.4956, 5374.6084, ..., 4315.8247, 4209.1377,

Out[66]:

4315.8247], dtype=float32)

In [76]: rmse **=** np**.**sqrt(mean\_squared\_error(y\_p2\_test, predictions)) print(f"RMSE = {rmse:.2f}")

RMSE = 15.60

The best model for Plant 2 is XGBoostRegressor with the features: hour, day\_of\_week, week\_of\_year, n\_Estimators=1000, learning\_rate=0.01

**FINAL MODEL:**

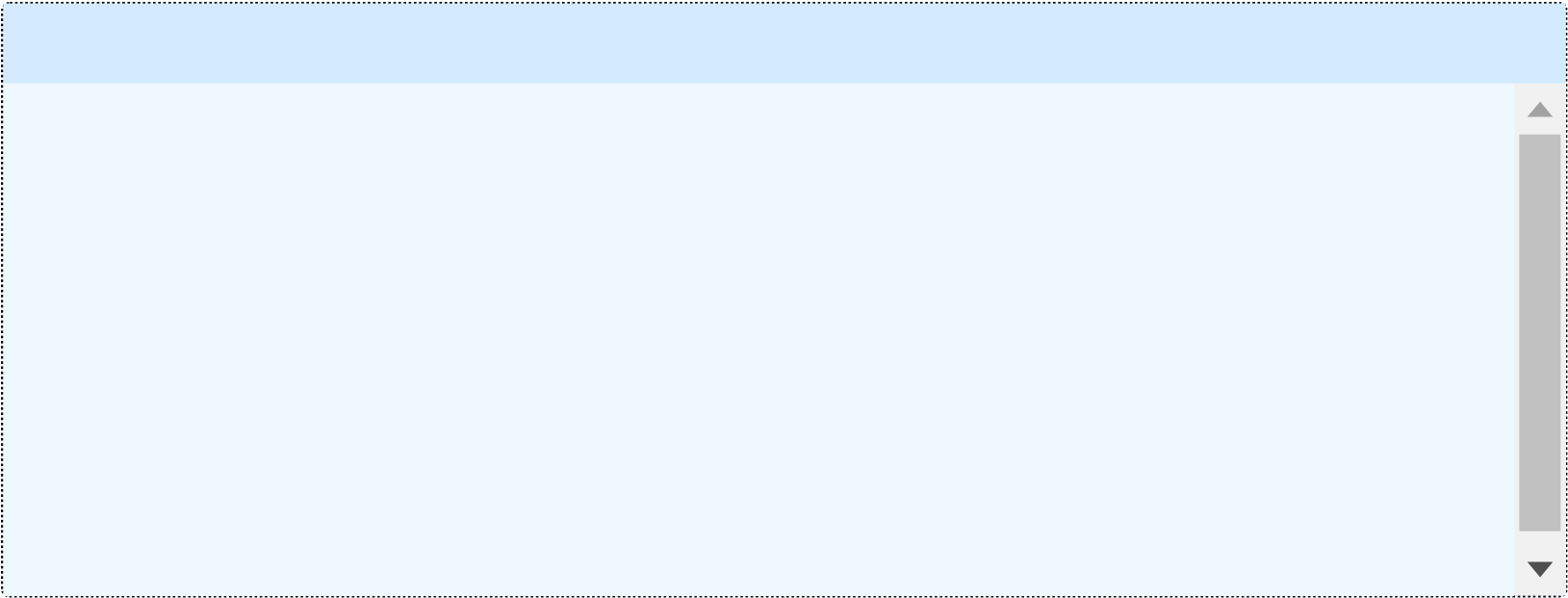
**PLANT 1:**

In [68]: X\_p1\_train\_final **=** plant1\_train**.**iloc[:, 1] *# only hour data* y\_p1\_train\_final **=** plant1\_train**.**iloc[:, 0]

|  |  |
| --- | --- |
| X\_p1\_test\_final **=** plant1\_test**.**iloc[:, y\_p1\_test\_final **=** plant1\_test**.**iloc[:, | 1]  0] |
|  |  |
| reg\_final\_p1 **=** xgb**.**XGBRegressor(n\_estimators**=**1000, learning\_rate**=**0.01) reg\_final\_p1**.**fit(X\_p1\_train\_final, y\_p1\_train\_final) | |

In [69]:

Out[69]: ▾ XGBRegressor

XGBRegressor(base\_score=None, booster=None, callbacks=None, colsample\_bylevel=None, colsample\_bynode=None, colsample\_bytree=None, device=None, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=None, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, multi\_strategy=None, n\_estimators=1000, n\_jobs=None, num\_parallel\_tree=None, random\_state=None, ...)

In [70]: redictions\_final\_p1 **=** reg\_final\_p1**.**predict(X\_p1\_test\_final)p predictions\_final\_p1

array([ 25.496717, 25.496717, 25.496717, ..., 4983.6963 ,

Out[70]:

4983.6963 , 4983.6963 ], dtype=float32)

In [71]: rmse\_final\_p1 **=** np**.**sqrt(mean\_squared\_error(y\_p1\_test\_final, predictions\_final\_p1)) print(f"RMSE = {rmse\_final\_p1:.2f}")

RMSE = 1128.25 **PLANT 2:**

|  |
| --- |
| X\_p2\_train\_final **=** plant2\_train**.**drop(['year', 'month'], axis**=**1) *# using week\_of\_year, day\_of\_week, ho* y\_p2\_train\_final **=** plant2\_train**.**iloc[:, 0]  X\_p2\_test\_final **=** plant2\_test**.**drop(['year', 'month'], axis**=**1) y\_p2\_test\_final **=** plant2\_test**.**iloc[:, 0] |

In [72]:

In [73]: reg\_final\_p2 **=** xgb**.**XGBRegressor(n\_estimators**=**1000, learning\_rate**=**0.01) reg\_final\_p2**.**fit(X\_p2\_train\_final, y\_p2\_train\_final)

Out[73]: ▾ XGBRegressor

XGBRegressor(base\_score=None, booster=None, callbacks=None, colsample\_bylevel=None, colsample\_bynode=None, colsample\_bytree=None, device=None, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=None, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, multi\_strategy=None, n\_estimators=1000, n\_jobs=None, num parallel tree=None, random state=None, ...)

In [74]: predictions\_final\_p2 **=** reg**.**predict(X\_p2\_test\_final) predictions\_final\_p2

array([1067.047 , 5523.4956, 5374.6084, ..., 4315.8247, 4209.1377,

Out[74]:

4315.8247], dtype=float32)

In [75]: rmse\_final\_p2 **=** np**.**sqrt(mean\_squared\_error(y\_p2\_test\_final, predictions\_final\_p2)) print(f"RMSE = {rmse\_final\_p2:.2f}")

RMSE = 15.60

**Exporting the final train, test sets:**

|  |
| --- |
| X\_p1\_train\_final**.**to\_csv('X\_plant1\_train.csv') X\_p1\_test\_final**.**to\_csv('X\_plant1\_test.csv')  y\_p1\_train\_final**.**to\_csv('y\_plant1\_train.csv') y\_p1\_test\_final**.**to\_csv('y\_plant1\_test.csv')  X\_p2\_train\_final**.**to\_csv('X\_plant2\_train.csv') X\_p2\_test\_final**.**to\_csv('X\_plant2\_test.csv')  y\_p2\_train\_final**.**to\_csv('y\_plant2\_train.csv') y\_p2\_test\_final**.**to\_csv('y\_plant2\_test.csv') |

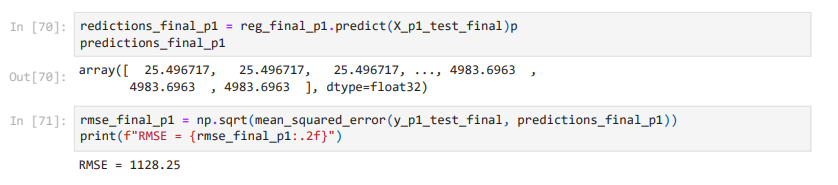
|  |
| --- |
|  |

In [77]: In [ ]:

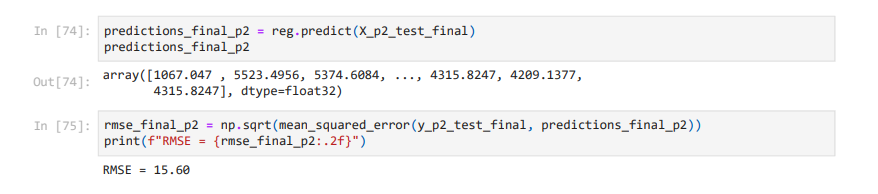
## 

## Performance Evaluation:

**Plant 1:**

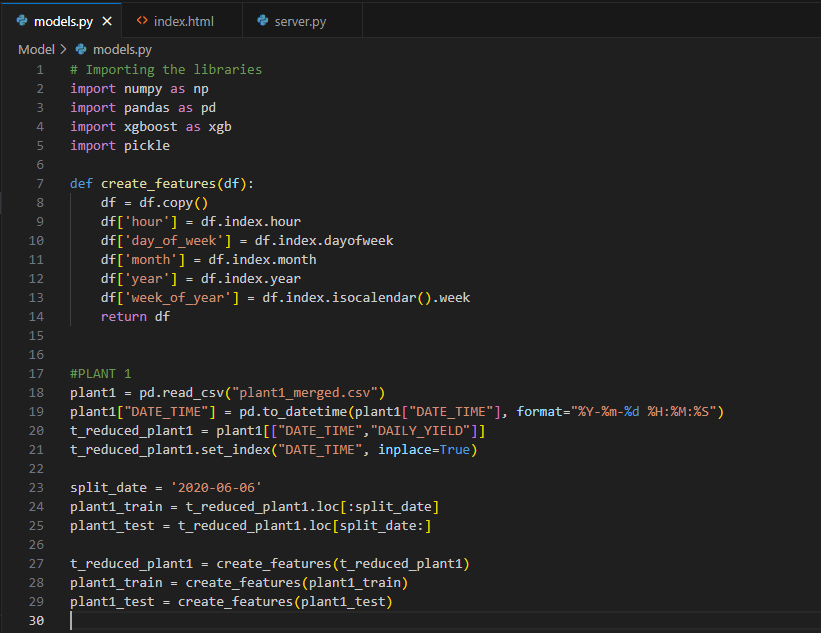


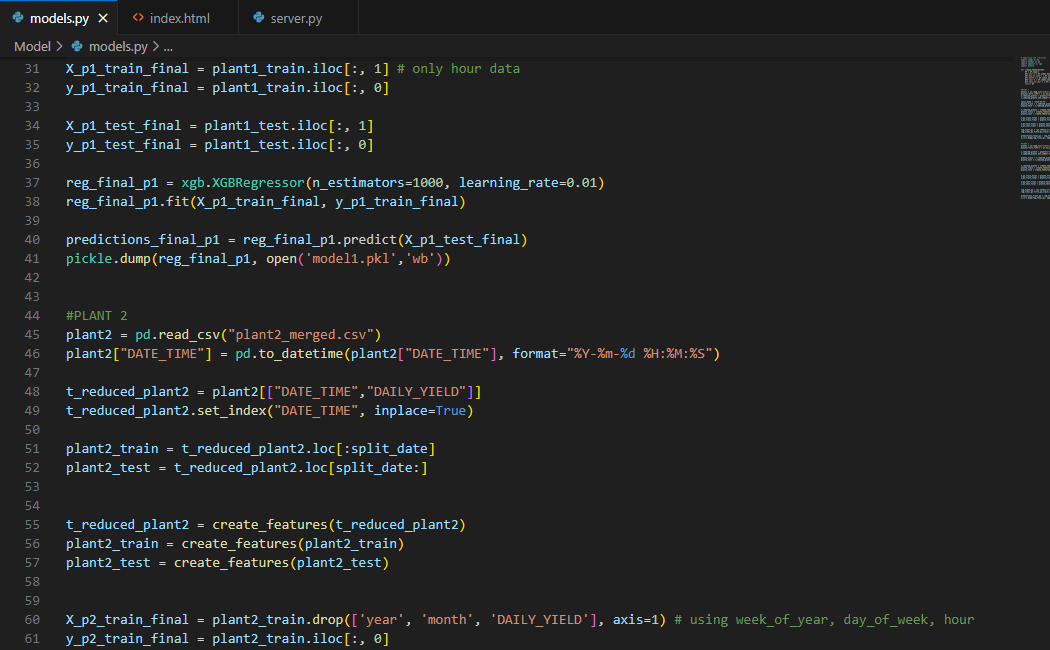
**Plant 2:**

****

## Model Deployment:

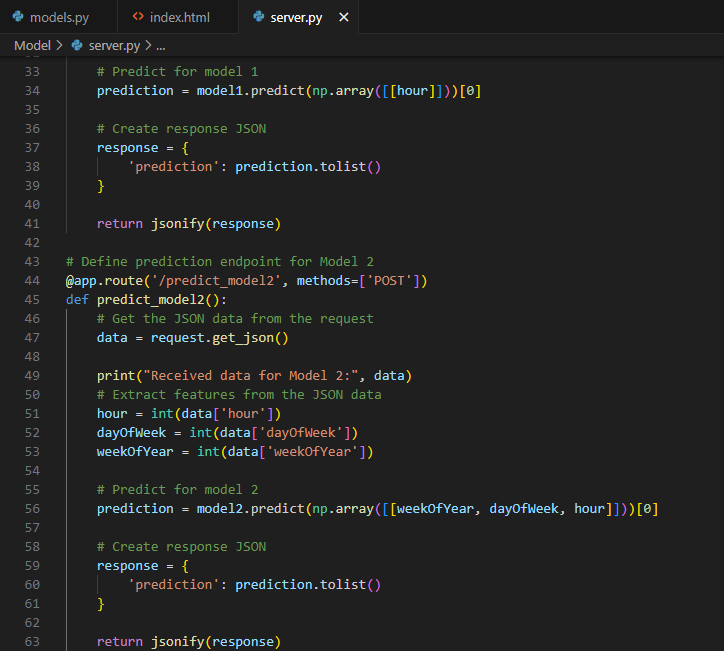
**Code:**

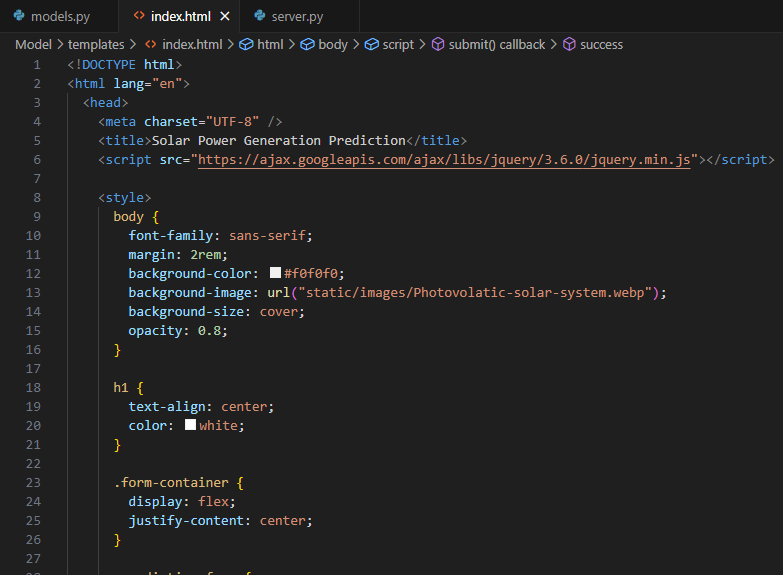




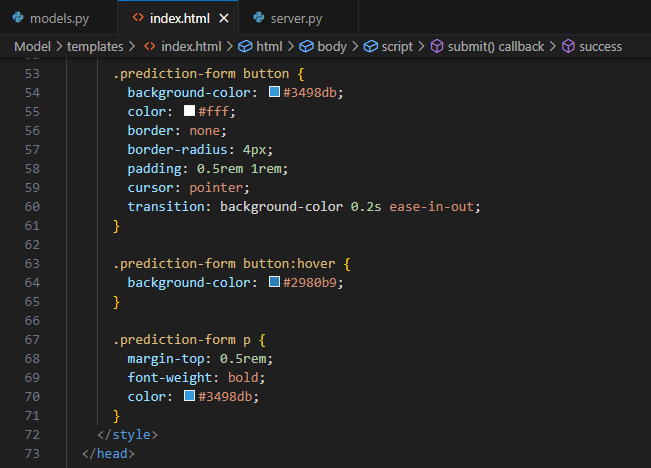


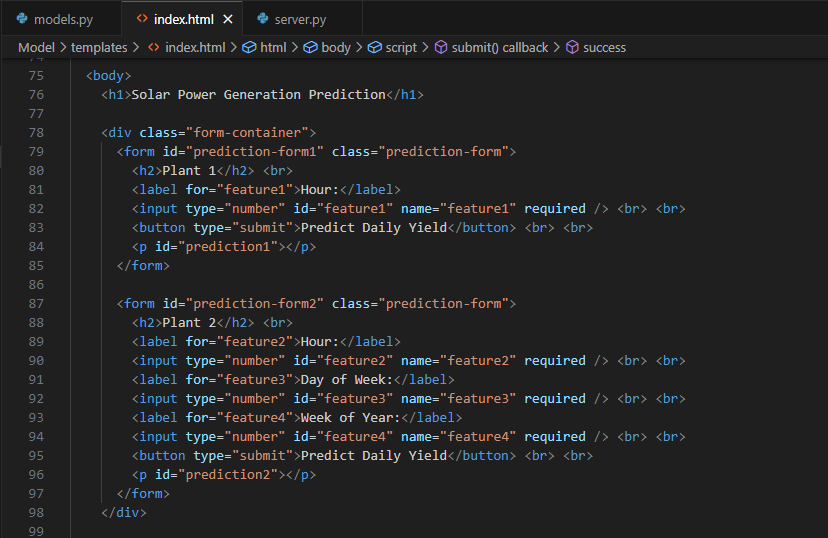
















**Flask App:**



