Department of Computer Science and Engineering (Data Science)

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.

Department of Computer Science and Engineering (Data Science)

Dataset Link: https://www.kaggle.com/datasets/anikannal/solar-power-generationdata?resource=download&select=Plant_1_Generation_Data.csv

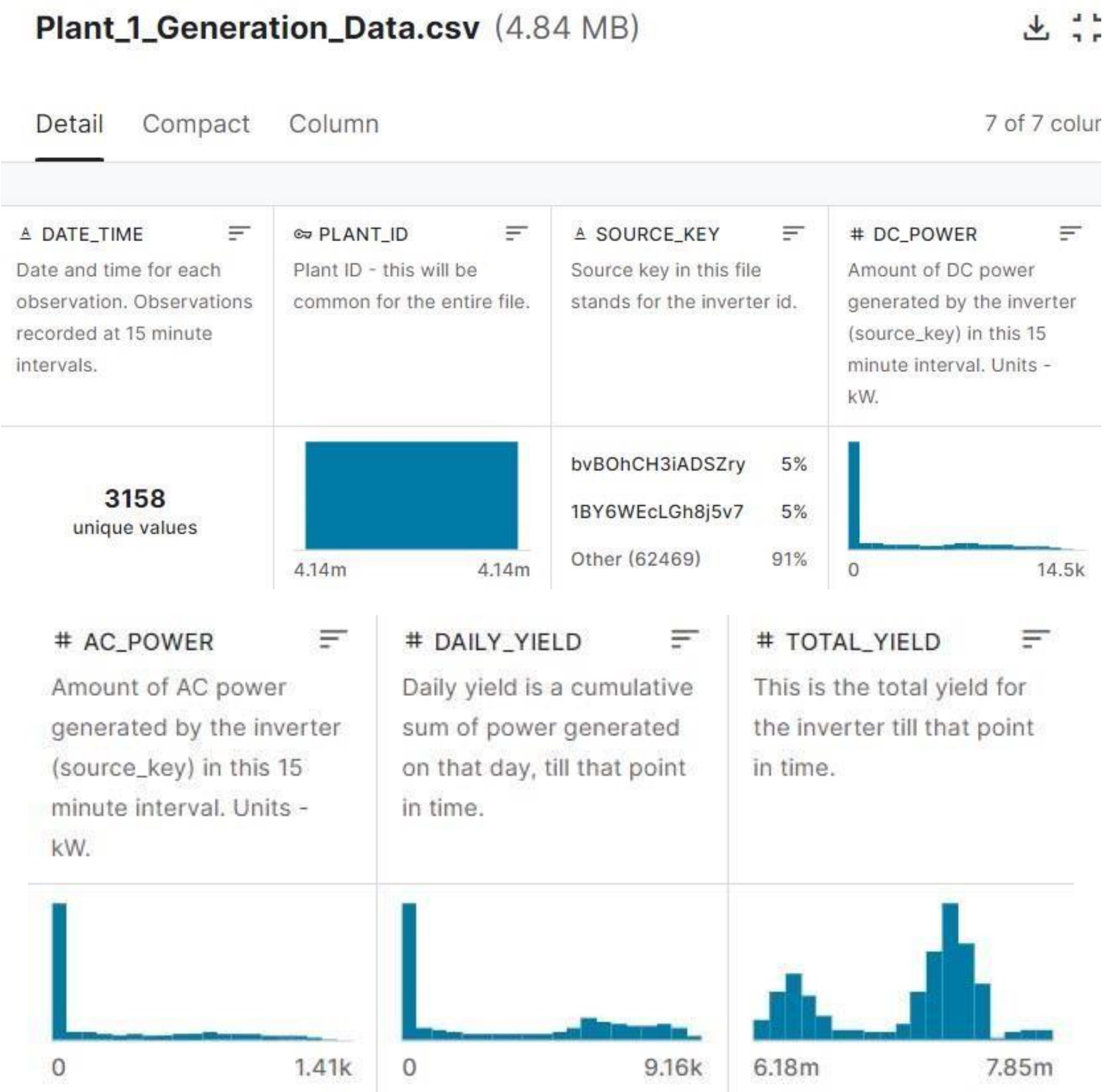
This data has been gathered at two solar power plants in India over a 34-day period. It has two pairs of files - each pair has one power generation dataset and one sensor readings dataset. The power generation datasets are gathered at the inverter level - each inverter has multiple lines of solar panels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

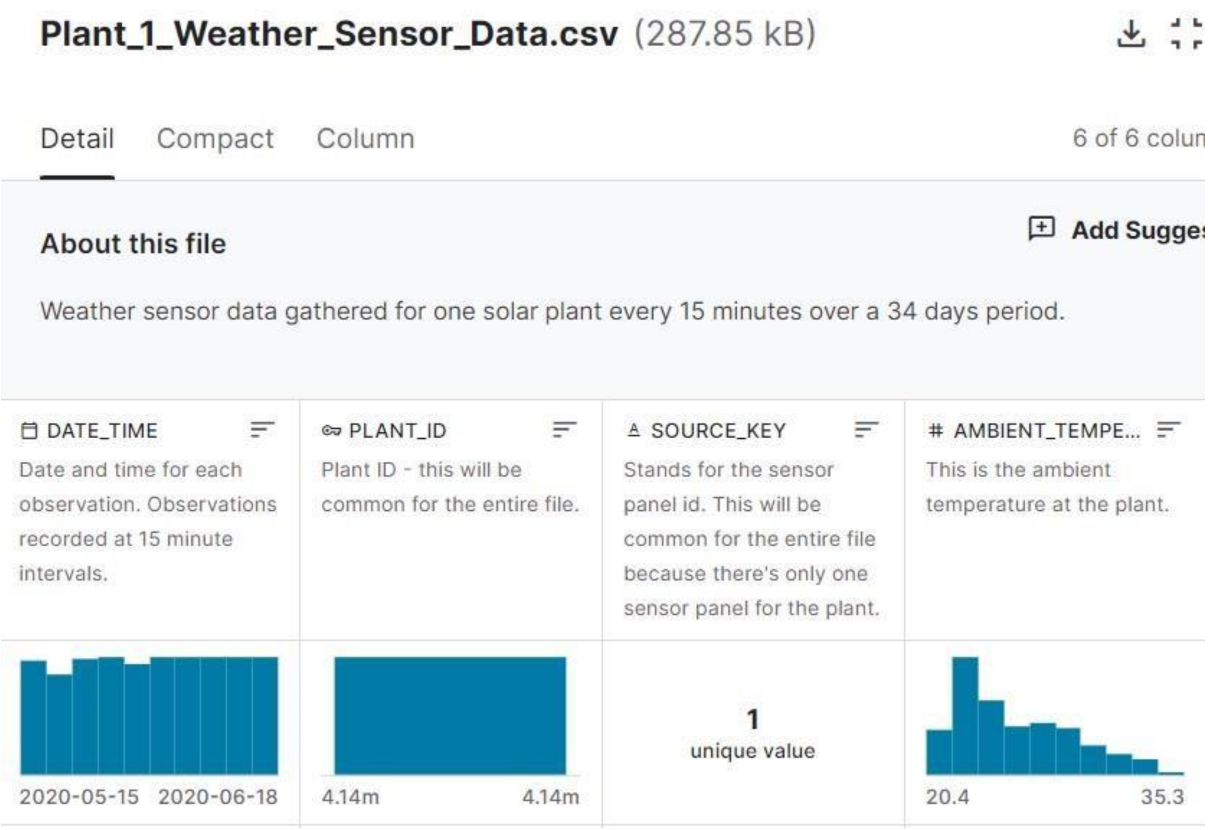
Department of Computer Science and Engineering (Data Science)

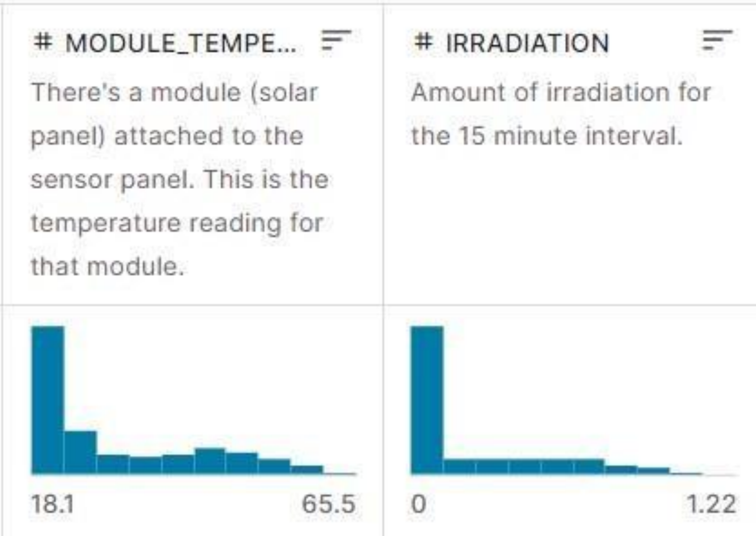
Data Description:

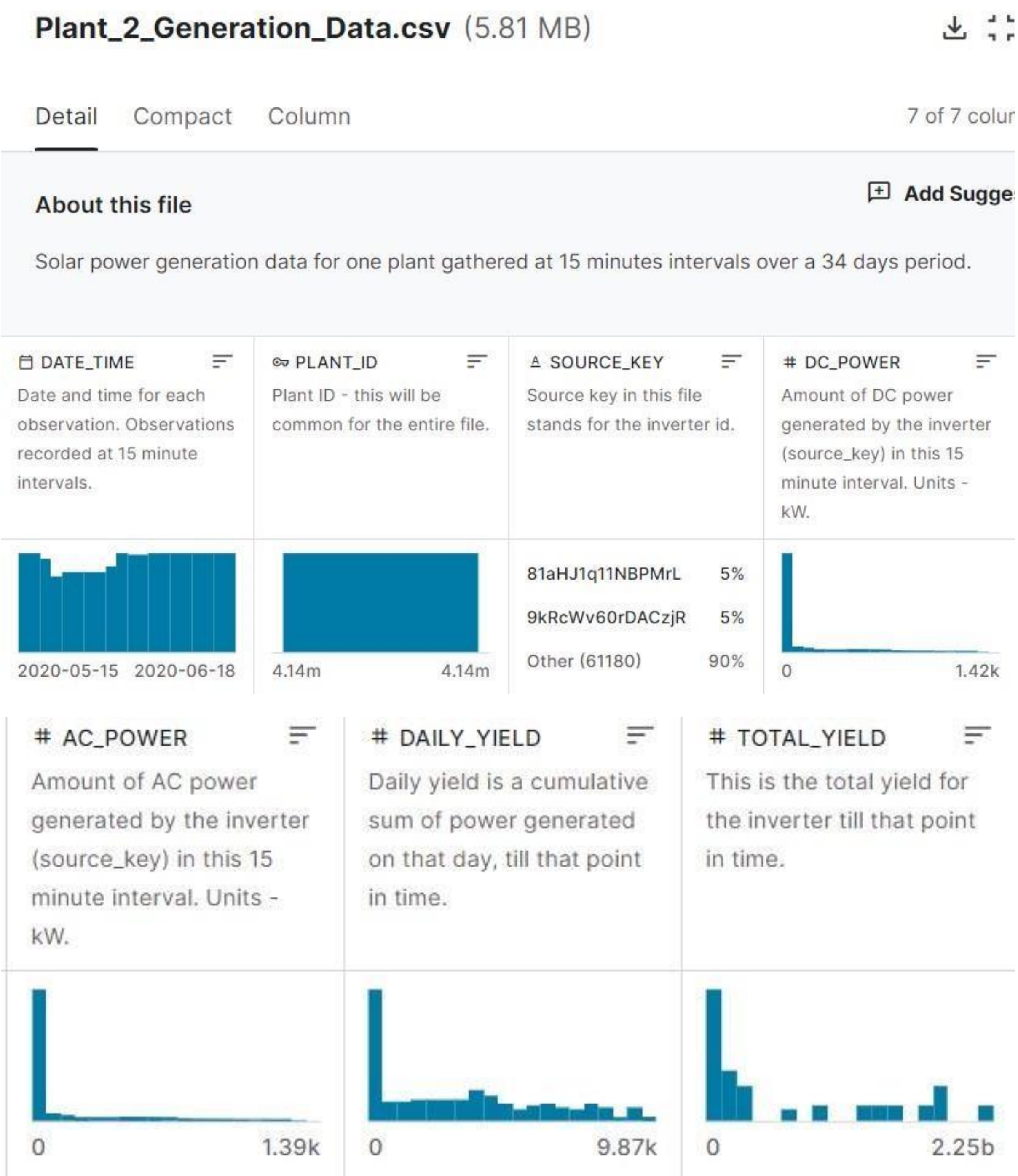
This data has been gathered at two solar power plants in India over a 34-day period. It has two pairs of files - each pair has one power generation dataset and one sensor readings dataset. The power generation datasets are gathered at the inverter level - each inverter has multiple lines of solar panels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



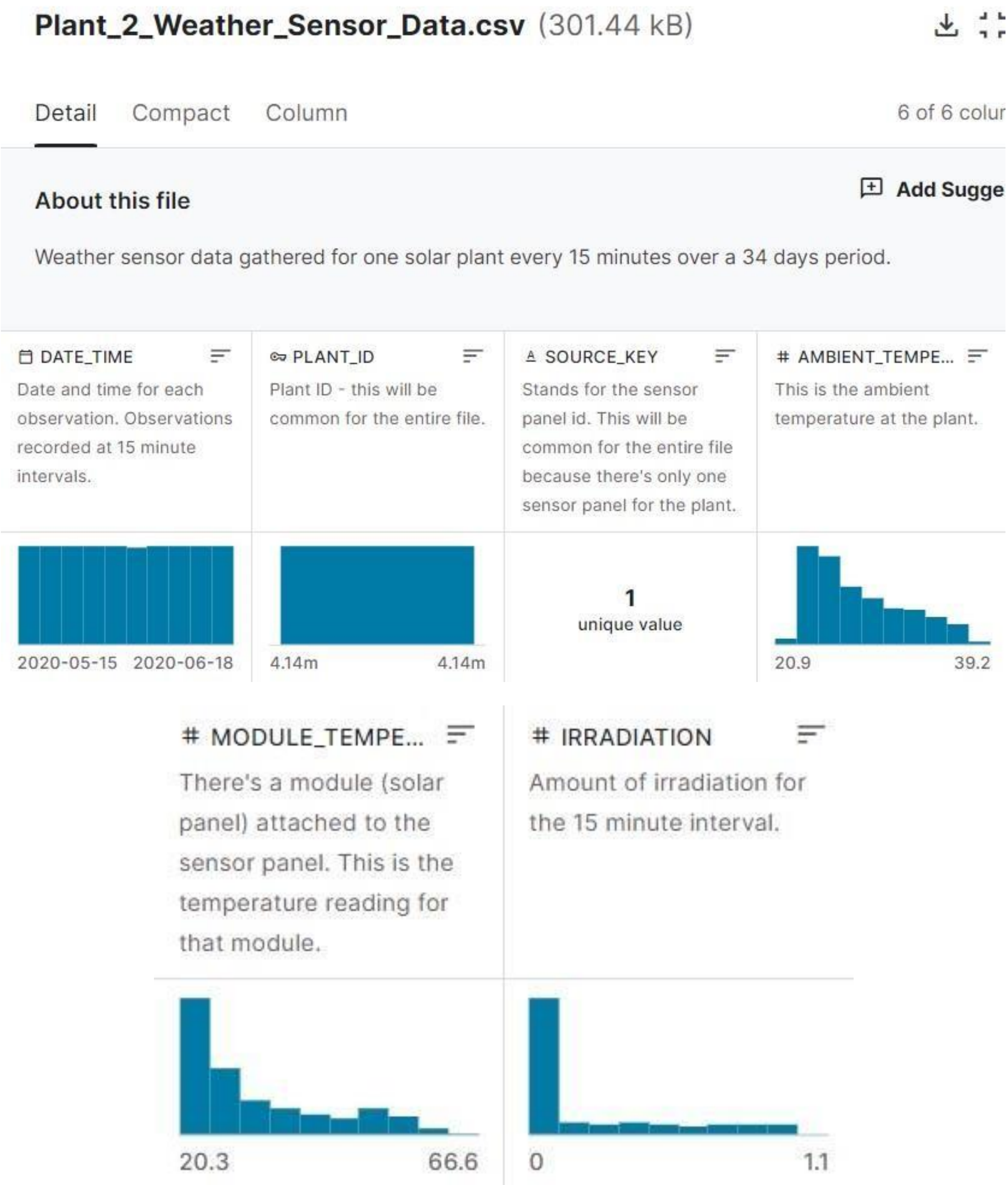
Department of Computer Science and Engineering (Data Science)






Plant_2_Generation_Data.csv (5.81 MB)
Detail
Compact
Column
7 of 7 colur
About this file
Add Sugge
Solar power generation data for one plant gathered at 15 minutes intervals over a 34 days period.
DATE_TIME
Date and time for each observation. Observations recorded at 15 minute intervals.
2020-05-15
2020-06-18
PLANT_ID
Plant ID - this will be common for the entire file.
4.14m
4.14m
SOURCE_KEY
Source key in this file stands for the inverter id.
81aHJ1q11NBPMrL 5%
9kRcWv60rDACzjR 5%
Other (61180) 90%
DC_POWER
Amount of DC power generated by the inverter (source_key) in this 15 minute interval. Units - kW.
0
1.42k
AC_POWER
Amount of AC power generated by the inverter (source_key) in this 15 minute interval. Units - kW.
0
1.39k
DAILY_YIELD
Daily yield is a cumulative sum of power generated on that day, till that point in time.
0
9.87k
TOTAL_YIELD
This is the total yield for the inverter till that point in time.
0
2.25b

Department of Computer Science and Engineering (Data Science)



Exploratory Data Analysis & Pre-Processing:

Name: Ananya Godse SAP ID: 60009220161

Importing the necessary libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline import seaborn
as sns from datetime import
datetime
```

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

In [2]:
```
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)
```

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
68777	17-06-2020 23:45	4135001	zVJPv84UY57bAof	0.0	0.0	5910.000	7363272.0

68778 rows × 7 columns PLANT 1 WEATHER SENSOR DATA

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

```
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
0	2020-05-15 00:00:00	4135001	HmiyD2TTLFNqkNe	25.184316	22.857507	0.0
1	2020-05-15 00:15:00	4135001	HmiyD2TTLFNqkNe	25.084589	22.761668	0.0
2	2020-05-15 00:30:00	4135001	HmiyD2TTLFNqkNe	24.935753	22.592306	0.0
3	2020-05-15 00:45:00	4135001	HmiyD2TTLFNqkNe	24.846130	22.360852	0.0
4	2020-05-15 01:00:00	4135001	HmiyD2TTLFNqkNe	24.621525	22.165423	0.0
...	...	...	...	...	...	...
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
0	2020-05-15 00:00:00	4135001	HmiyD2TTLFNqkNe	25.184316	22.857507	0.0
1	2020-05-15 00:15:00	4135001	HmiyD2TTLFNqkNe	25.084589	22.761668	0.0
2	2020-05-15 00:30:00	4135001	HmiyD2TTLFNqkNe	24.935753	22.592306	0.0
3	2020-05-15 00:45:00	4135001	HmiyD2TTLFNqkNe	24.846130	22.360852	0.0
4	2020-05-15 01:00:00	4135001	HmiyD2TTLFNqkNe	24.621525	22.165423	0.0

```
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
0	2020-05-15 00:00:00	4136001	iq8k7ZNt4Mwm3w0	27.004764	25.060789	0.0
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
---  ------       --------------  -----
0   DATE_TIME    68778 non-null  object
1   PLANT_ID     68778 non-null  int64
2   SOURCE_KEY   68778 non-null  object
3   DC_POWER     68778 non-null  float64
4   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
---  ------               --------------  -----
0   DATE_TIME            3182 non-null   object
1   PLANT_ID             3182 non-null   int64
2   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
---  ------       --------------  -----
0   DATE_TIME    67698 non-null  object
1   PLANT_ID     67698 non-null  int64
2   SOURCE_KEY   67698 non-null  object
3   DC_POWER     67698 non-null  float64
4   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
---  ------               --------------  -----
0   DATE_TIME            3259 non-null   object
1   PLANT_ID             3259 non-null   int64
2   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()
```

Out[12]:
```
SOURCE_KEY
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()
```

Out[17]:
```
SOURCE_KEY
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
xMbIugepa2P71BB    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()
```

Out[20]:
```
SOURCE_KEY
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

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
68777	17-06-2020 23:45	4135001	zVJPv84UY57bAof	0.0	0.0	5910.000	7363272.0

68778 rows × 7 columns

In [22]:

```
plant1_generation.drop("PLANT_ID", axis=1, inplace=True) plant1_generation
```

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

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
67697	2020-06-17 23:45:00	xoJJ8DcxJEcupym	0.0	0.0	4316.000000	2.093357e+08

67698 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.
2. 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.
3. 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	20:05:55.968088064	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:

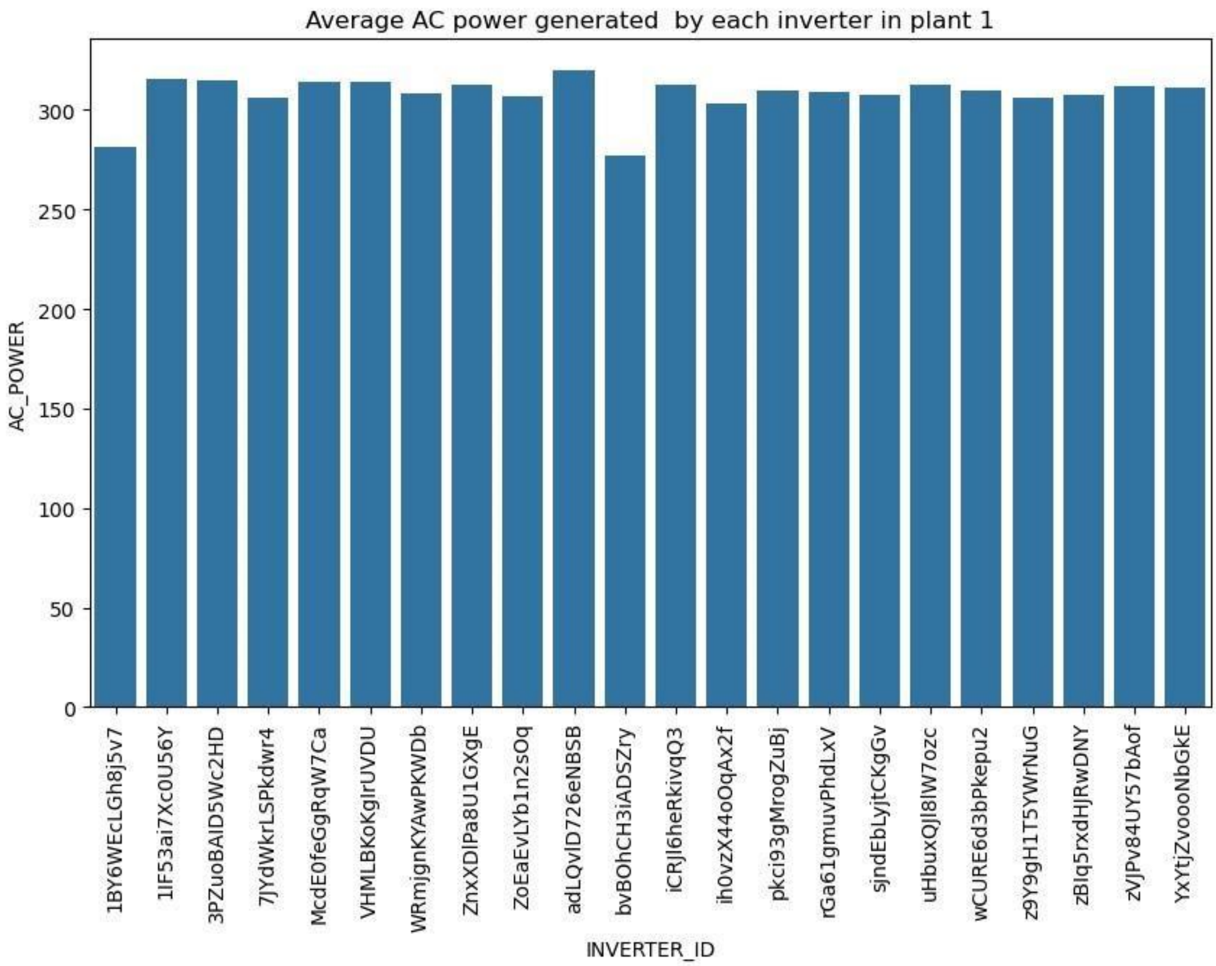
1. There isn't much difference in the avg, Q1, Q2 & Q3 values of AMBIENT_TEMPERATURE & MODULE_TEMPERATURE.
2. The max value of MODULE_TEMPERATURE is much higher than the max value of AMBIENT_TEMPERATURE.

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

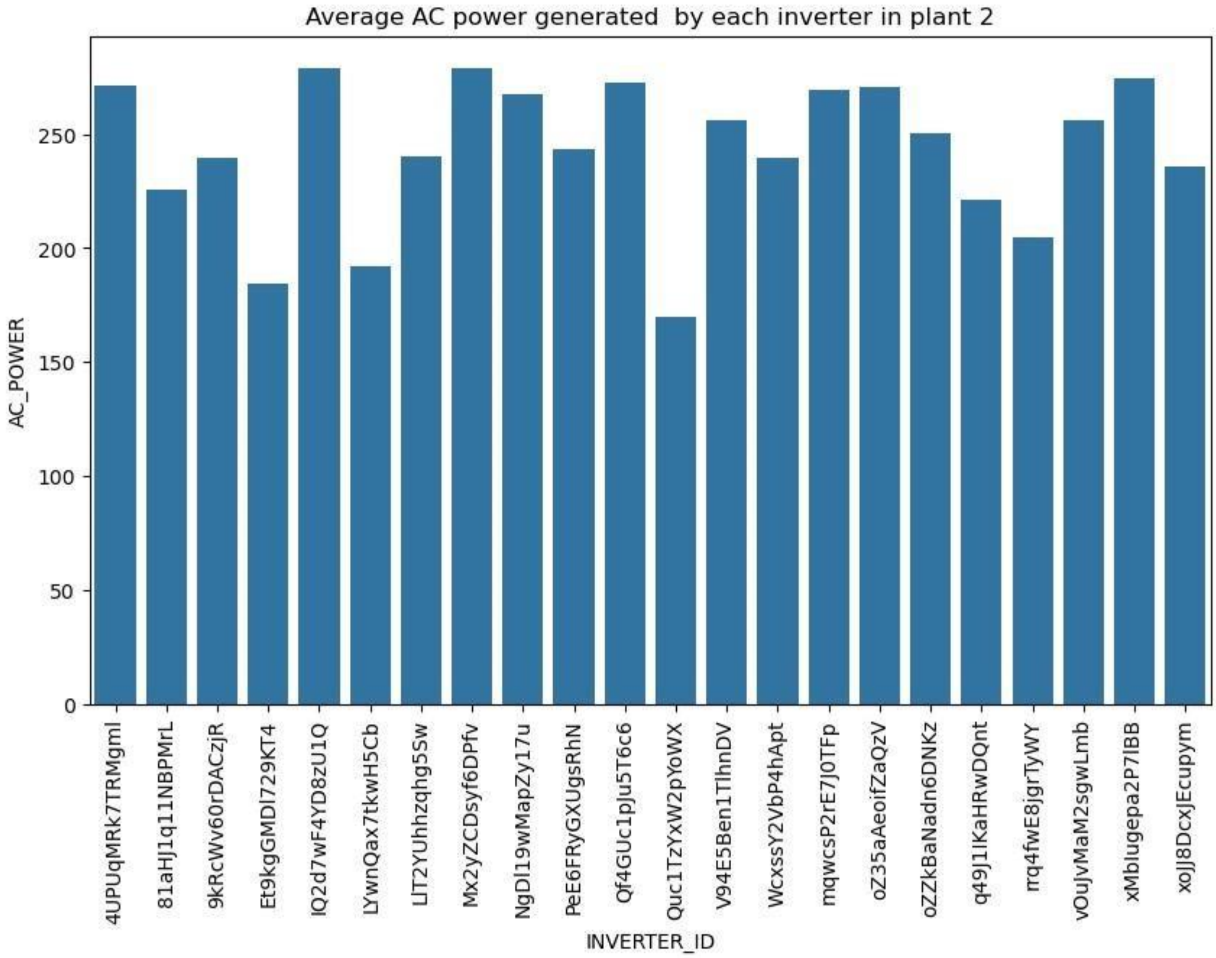
```
In [47]: plt.figure(figsize=(10, 6)) sns.barplot(data=plant1_generation, x="INVERTER_ID", y="AC_POWER",
         errorbar=None).set(title="Average plt.xticks(rotation=90) plt.show()
```



Observation:

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

In [48]:
```
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()
```

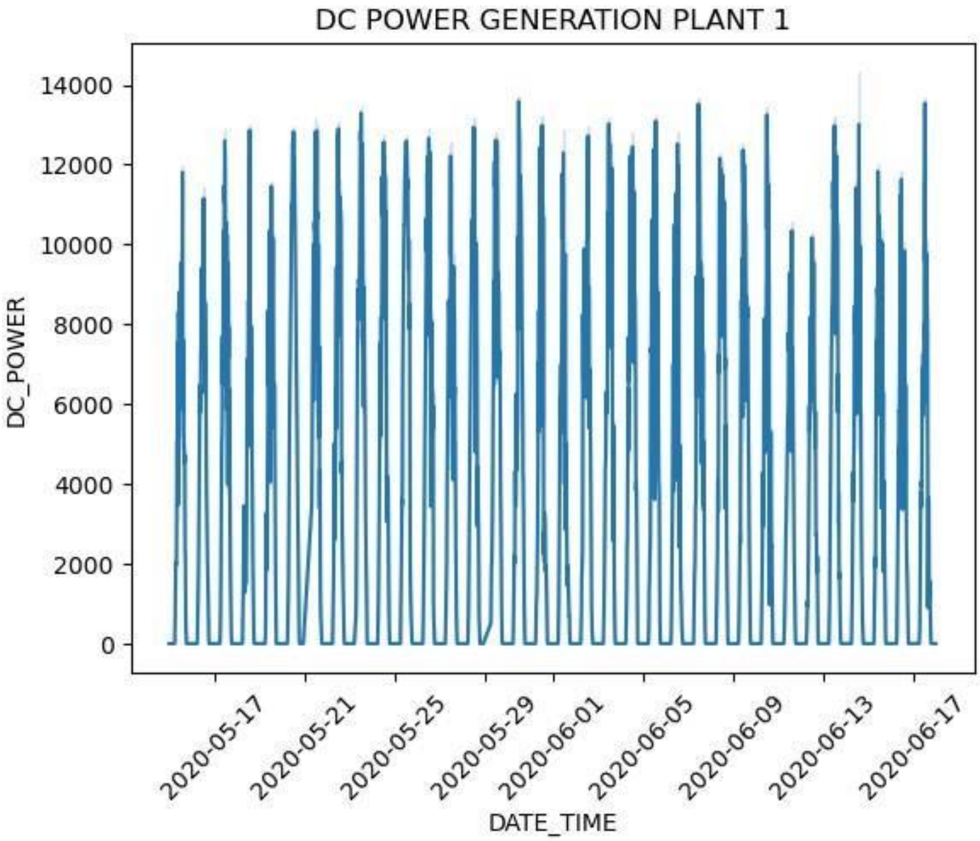


Observation:

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

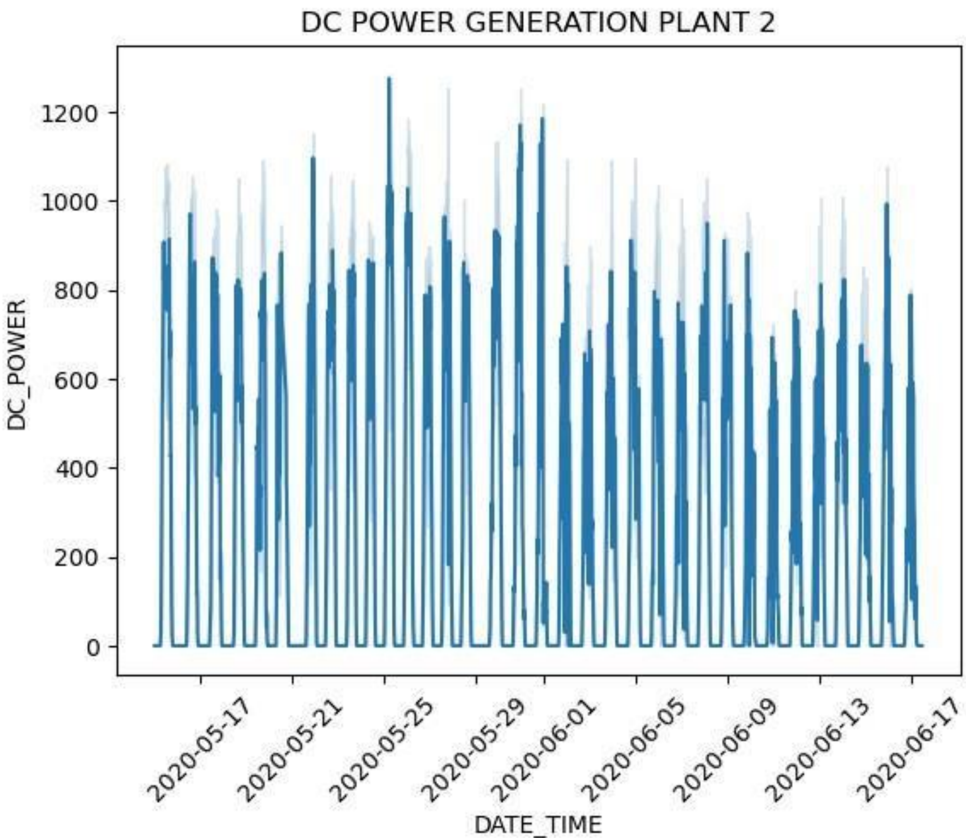
DC POWER Generation in the plants

In [49]:
```
sns.lineplot(data=plant1_generation, x='DATE_TIME', y='DC_POWER').set(title="DC POWER GENERATION
PLAN plt.xticks(rotation=45) plt.show()
```



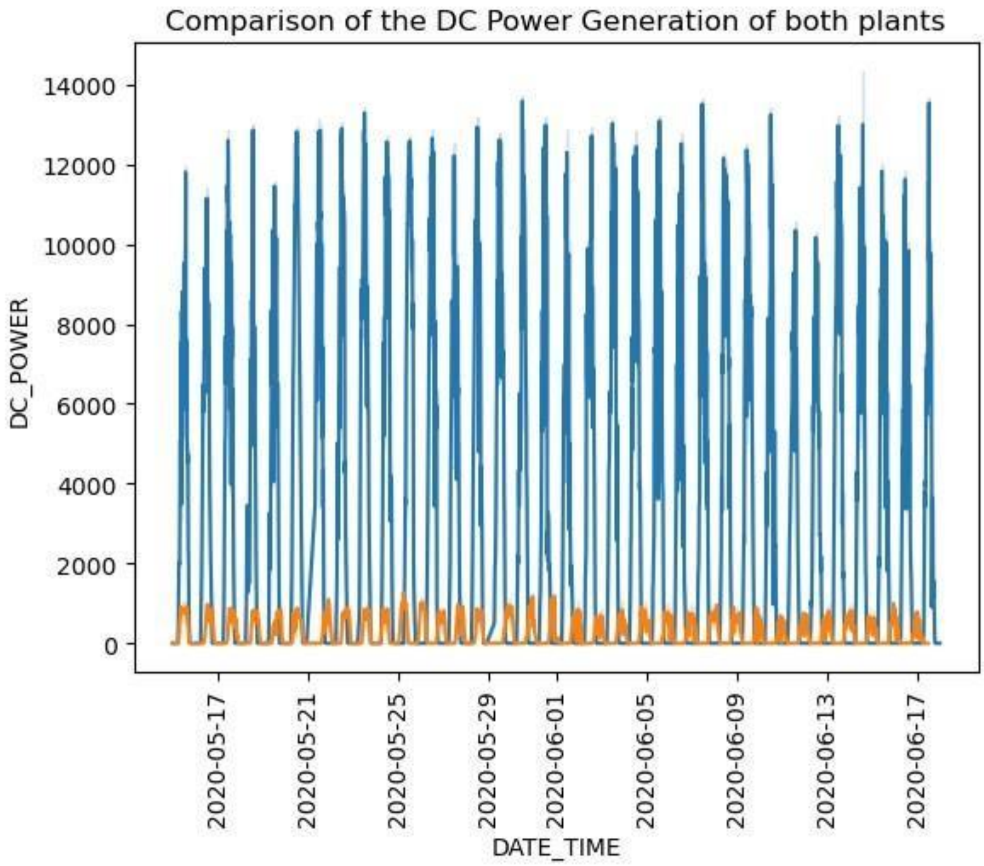
In [50]:

```
sns.lineplot(data=plant2_generation, x='DATE_TIME', y='DC_POWER').set(title="DC POWER GENERATION
PLAN plt.xticks(rotation=45) plt.show()
```



In [51]:

```
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()
```



Observations:

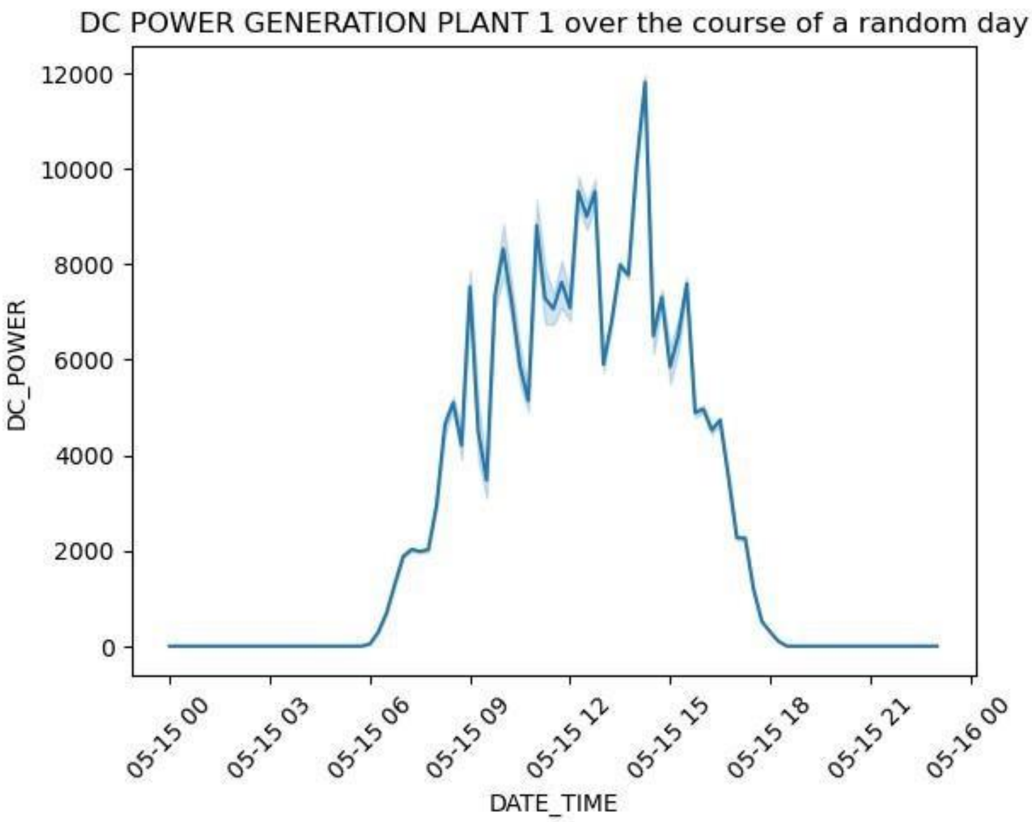
The DC power produced by Plant 1 (blue) is way higher than plant 2 (orange) In

[52]:

```
==  selected_day = '2020-05-15'
df_single_day  =     plant1_generation[plant1_generation['DATE_TIME'].dt.date
pd.to_datetime(selected_d
```

In [53]:

```
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()
```



Observation:

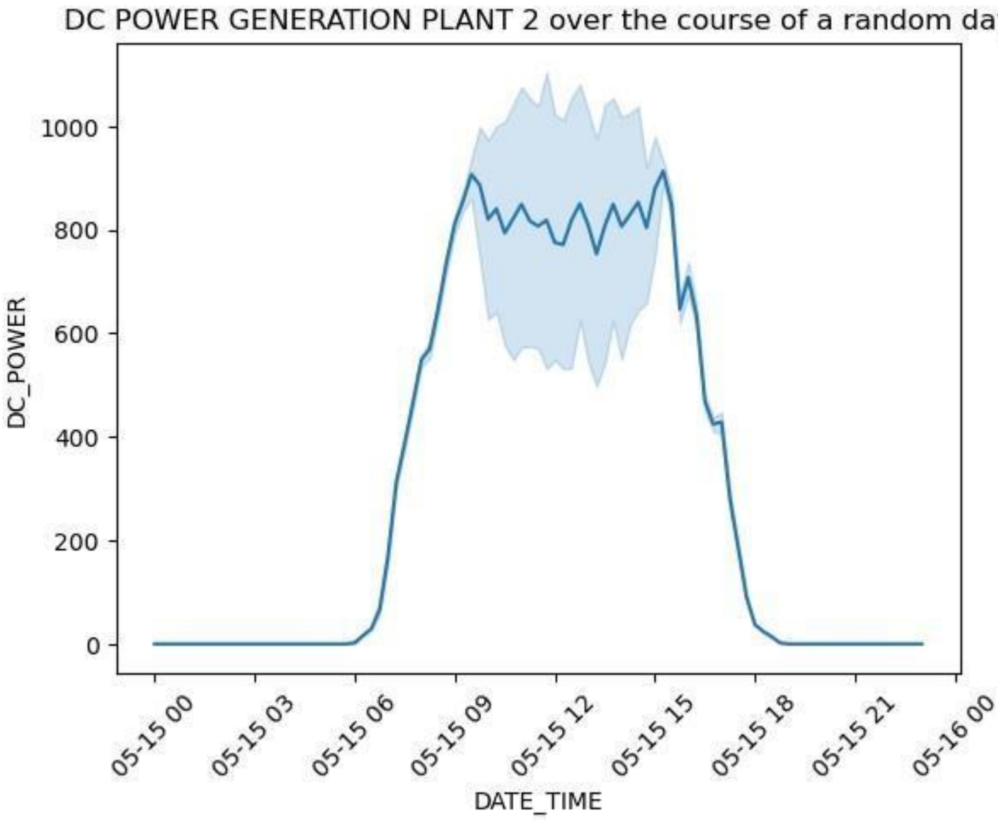
DC power is generated only between 6 am to 6 pm which makes sense since those are the daylight hours. Most power was produced between the hours of 10 am to 3 pm.

In [54]:

```
df_single_day2 = plant2_generation[plant2_generation['DATE_TIME'].dt.date
== pd.to_datetime(selected_
```

In [55]:

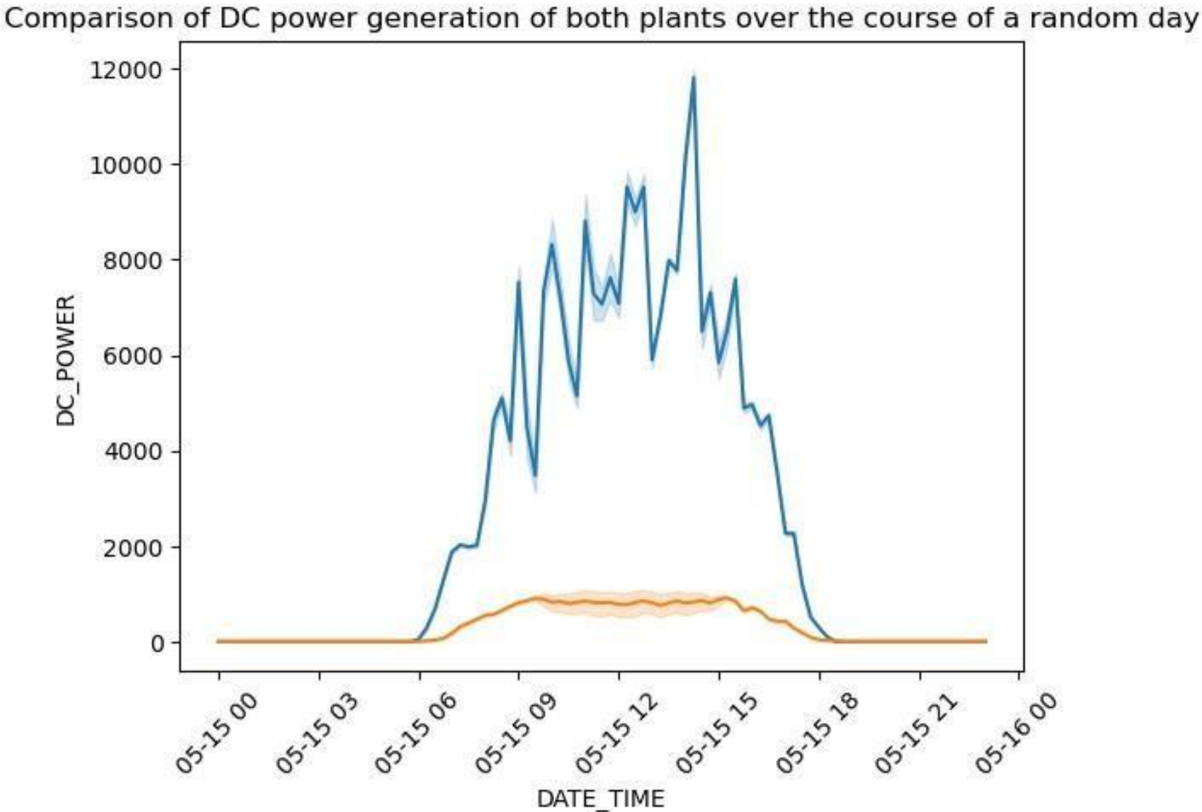
```
sns.lineplot(data=df_single_day2, x='DATE_TIME', y='DC_POWER').set(title="DC POWER GENERATION
PLANT plt.xticks(rotation=45) plt.show()
```



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.

In [56]:
```
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()
```



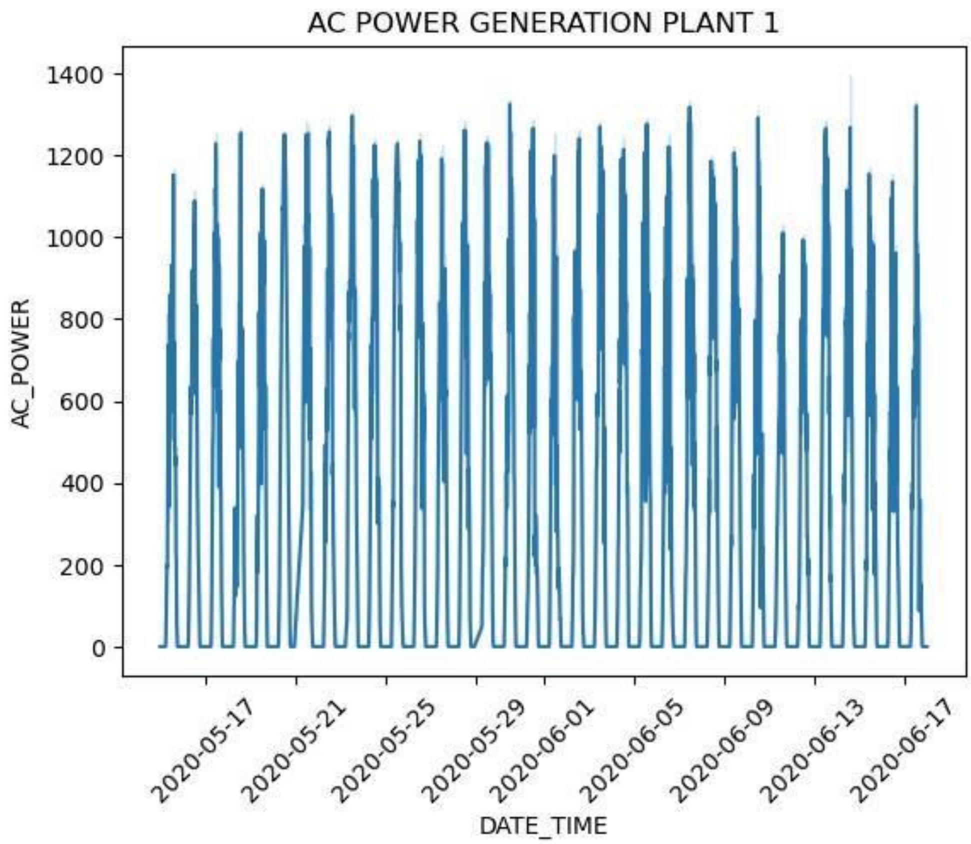
Observations:

Plant 1 produces more DC power than plant 2

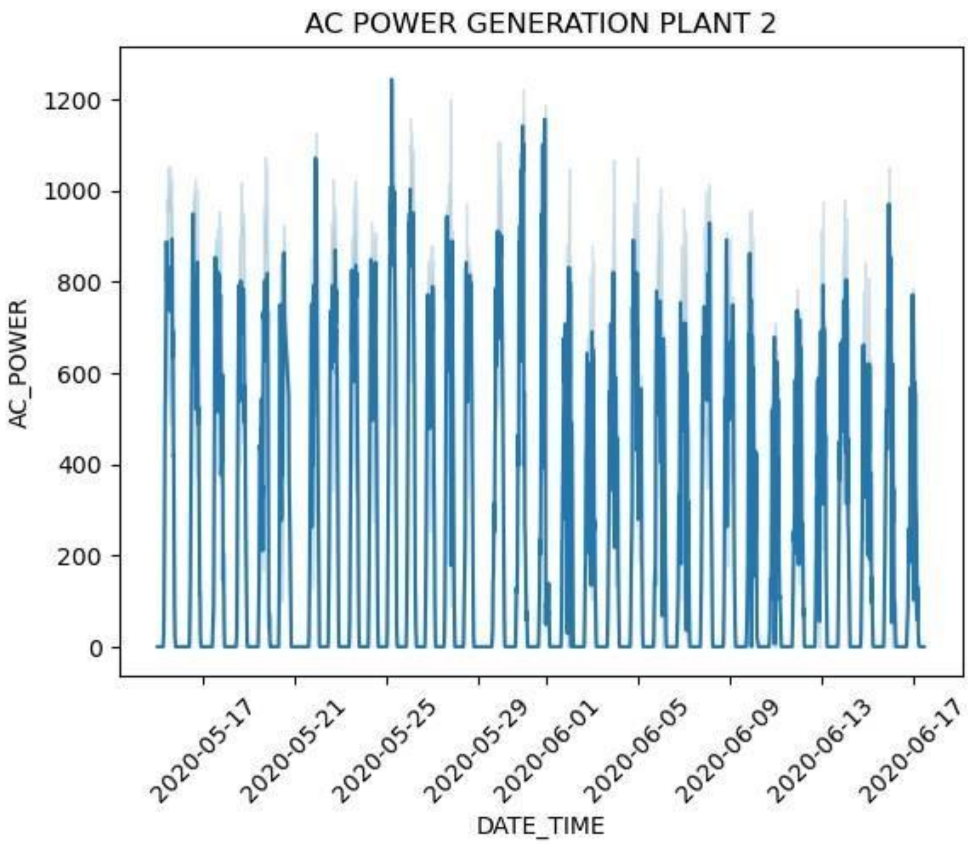
AC Power generation in the plants

In [57]:
```
sns.lineplot(data=plant1_generation, x='DATE_TIME', y='AC_POWER').set(title="AC POWER GENERATION
PLAN plt.xticks(rotation=45) plt.show()
```

```
sns.lineplot(data=plant2_generation, x='DATE_TIME', y='AC_POWER').set(title="AC POWER GENERATION
PLAN plt.xticks(rotation=45) plt.show()
```

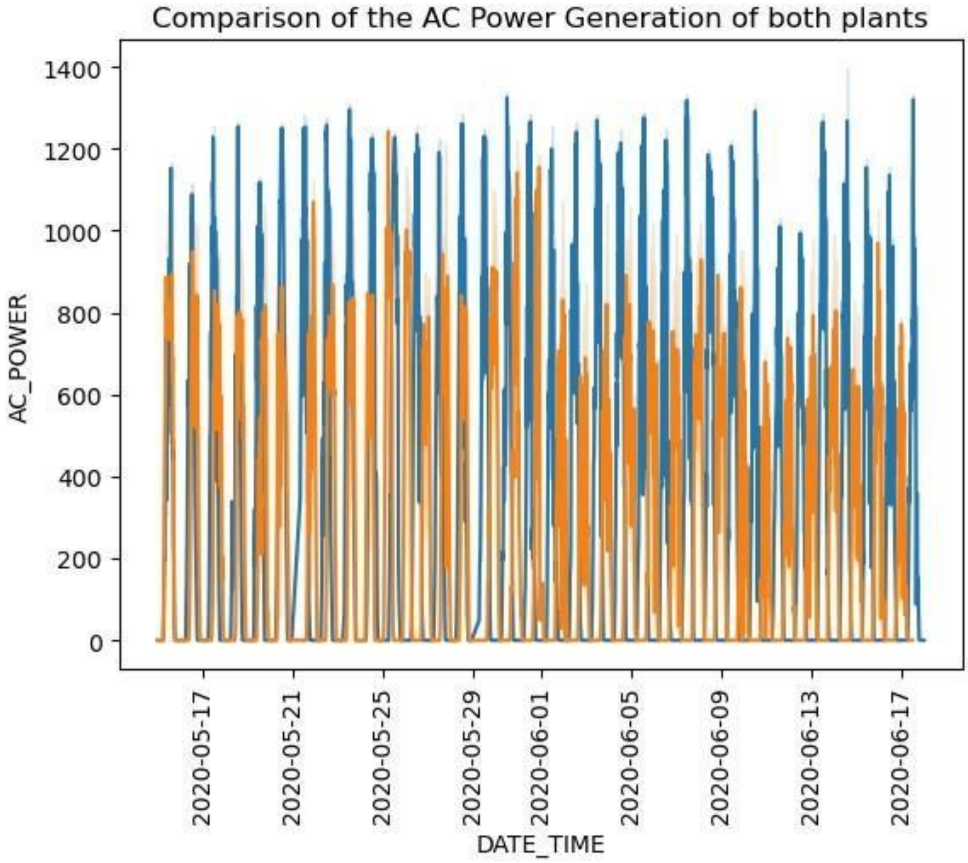


In [58]:



In [59]:

```
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()
```

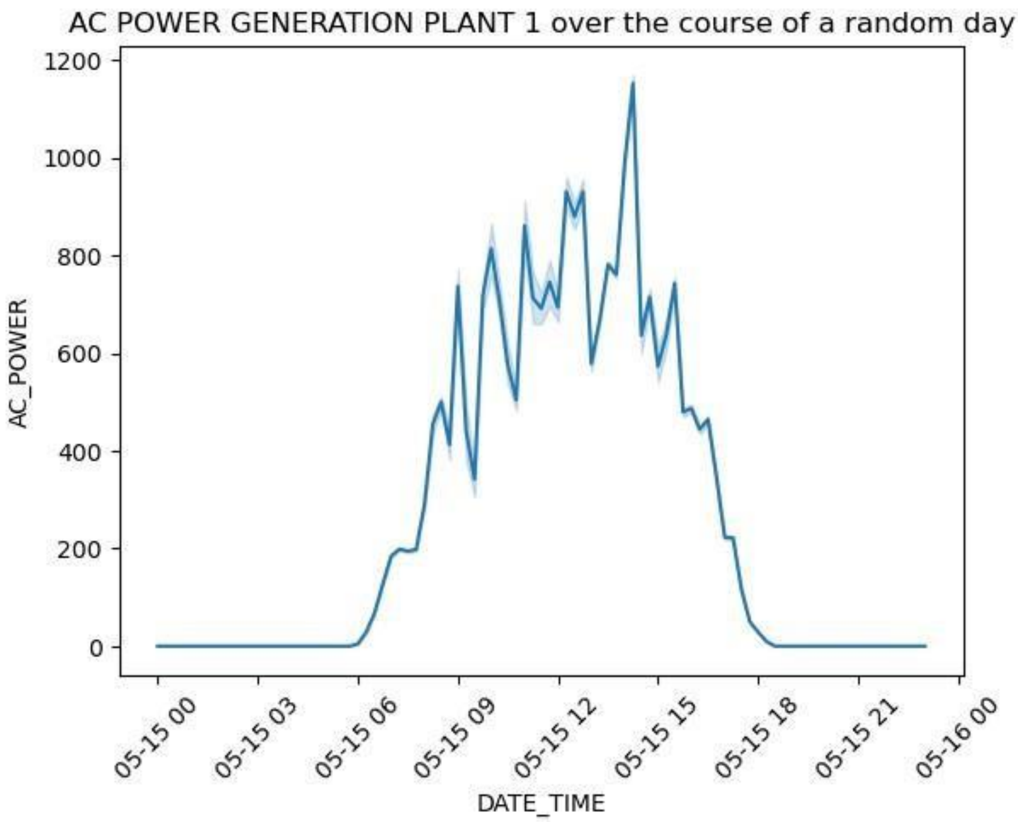


Observations:

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

In [60]:
```
df_single_day3 = plant1_generation[plant1_generation['DATE_TIME'].dt.date
== pd.to_datetime(selected_
```

In [61]:
```
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()
```



Observation:

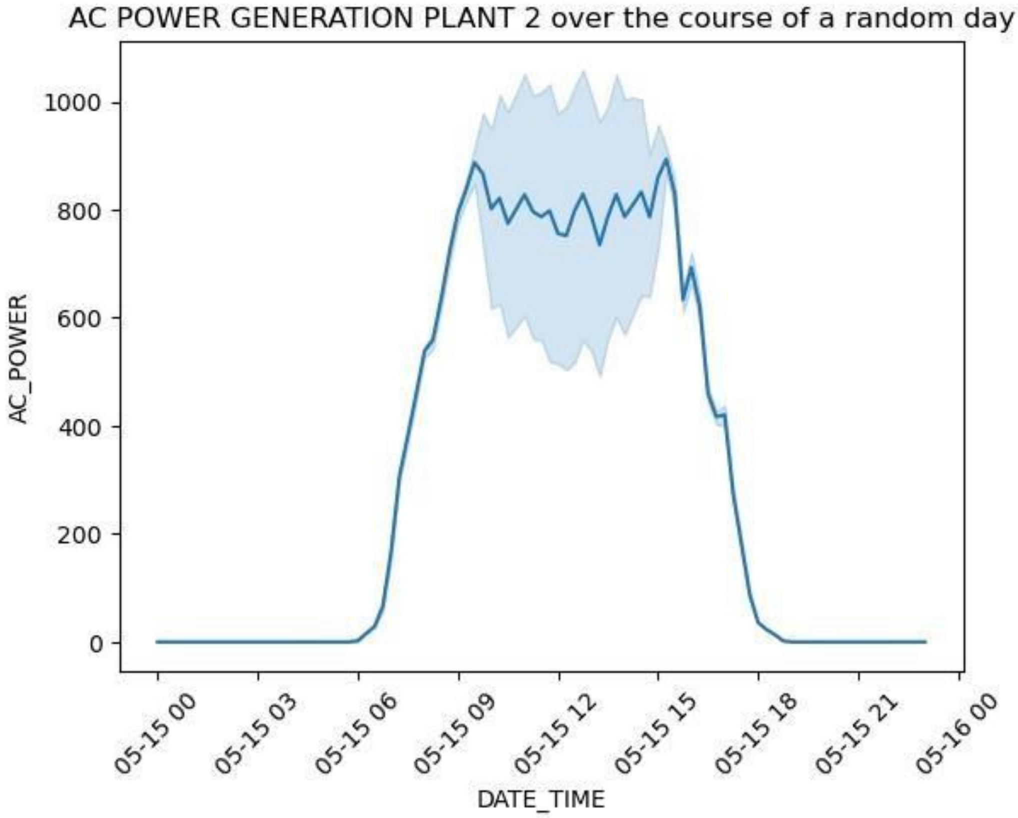
DC power is generated only between 6 am to 6 pm which makes sense since those are the daylight hours. Most power was produced between the hours of 10 am to 3 pm.

In [62]:

```
df_single_day4 = plant2_generation[plant2_generation['DATE_TIME'].dt.date
== pd.to_datetime(selected_
```

In [63]:

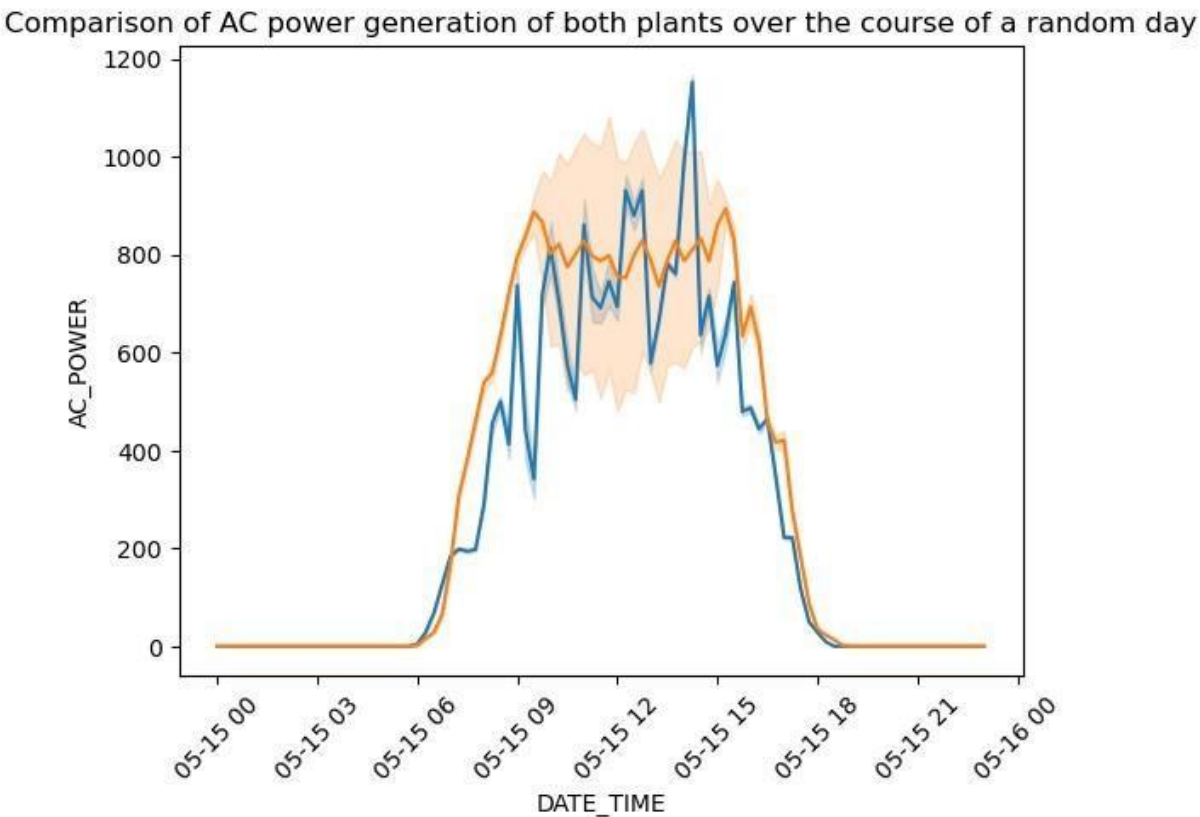
```
sns.lineplot(data=df_single_day4, x='DATE_TIME', y='AC_POWER').set(title="AC POWER GENERATION
PLANT plt.xticks(rotation=45) plt.show()
```



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.

In [64]:
```
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()
```

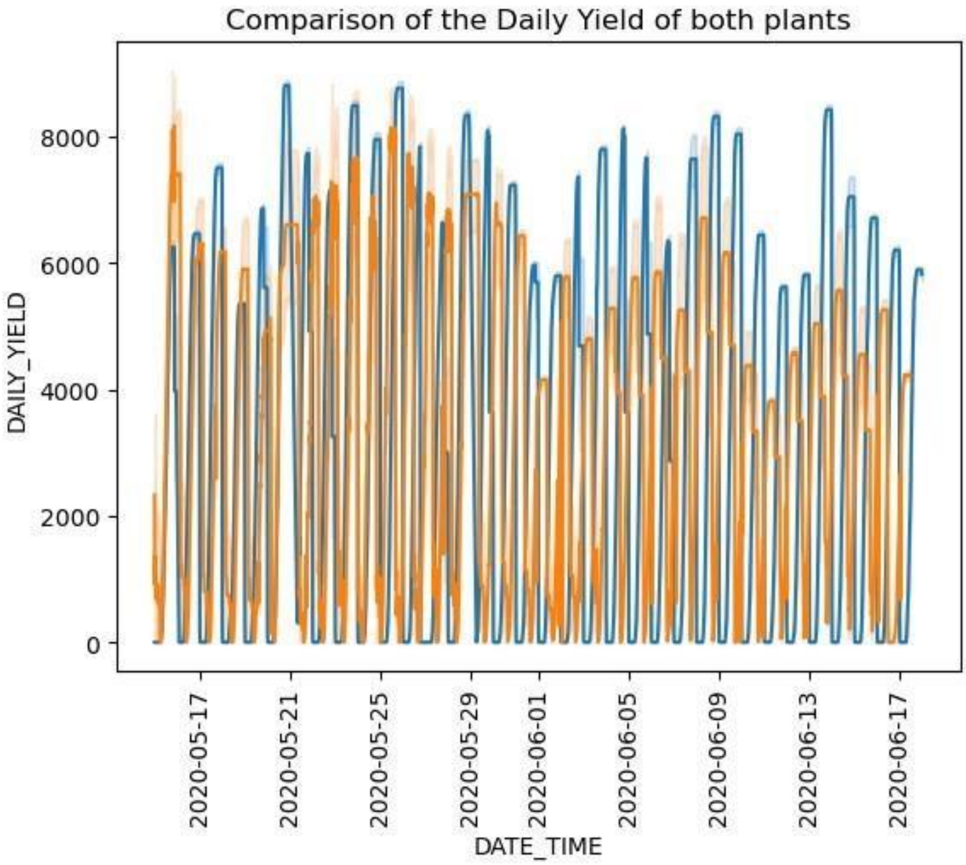


Observations:

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()
```

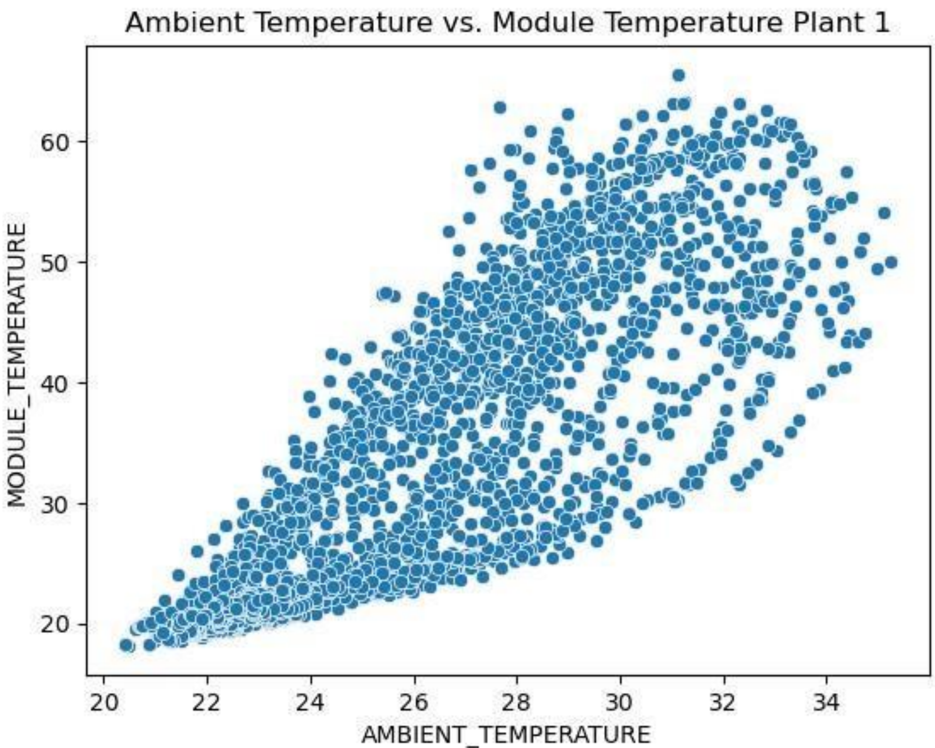


Observations:

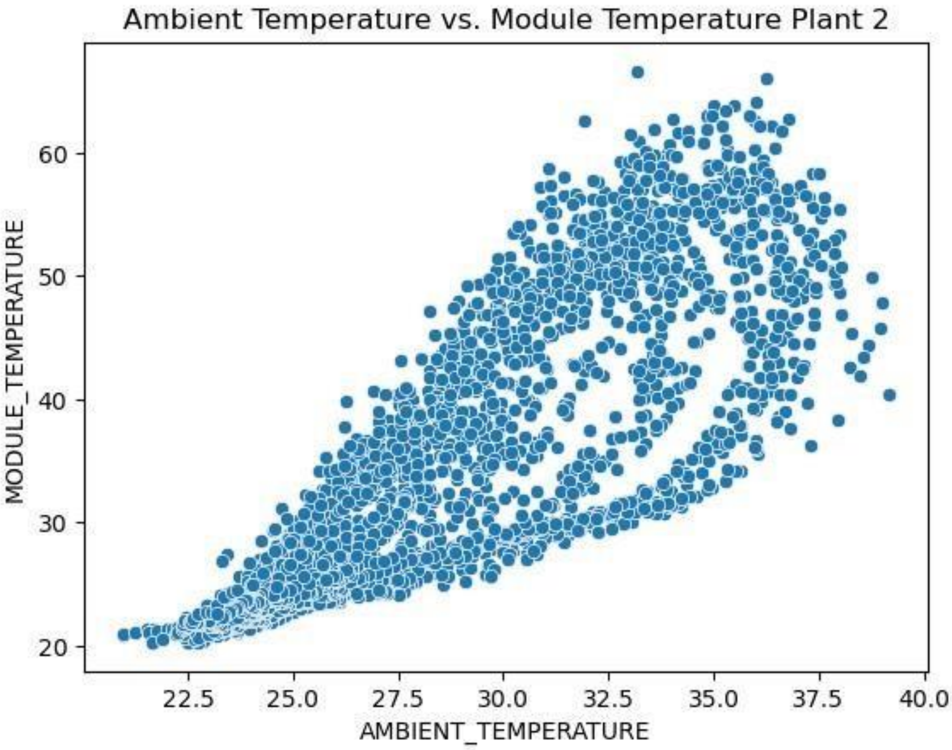
1. On average, the daily yield of plant 1 (blue) is much greater than plant 2 (orange).
2. The daily yield of plant 2 dropped after June 1. We can only assume because of monsoon.

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()
```

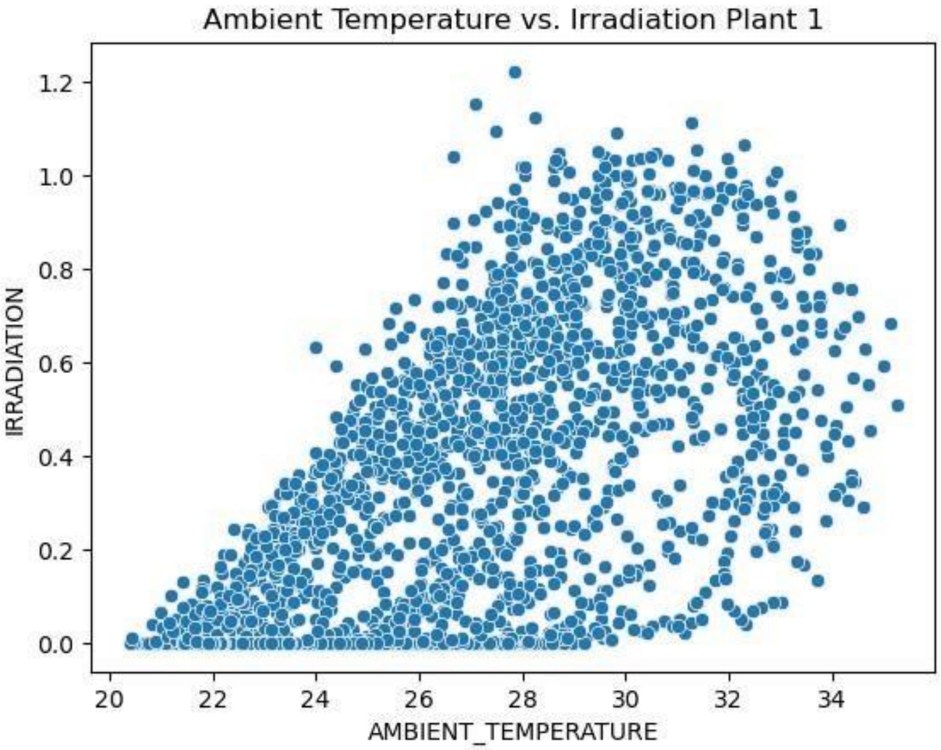


Observation:

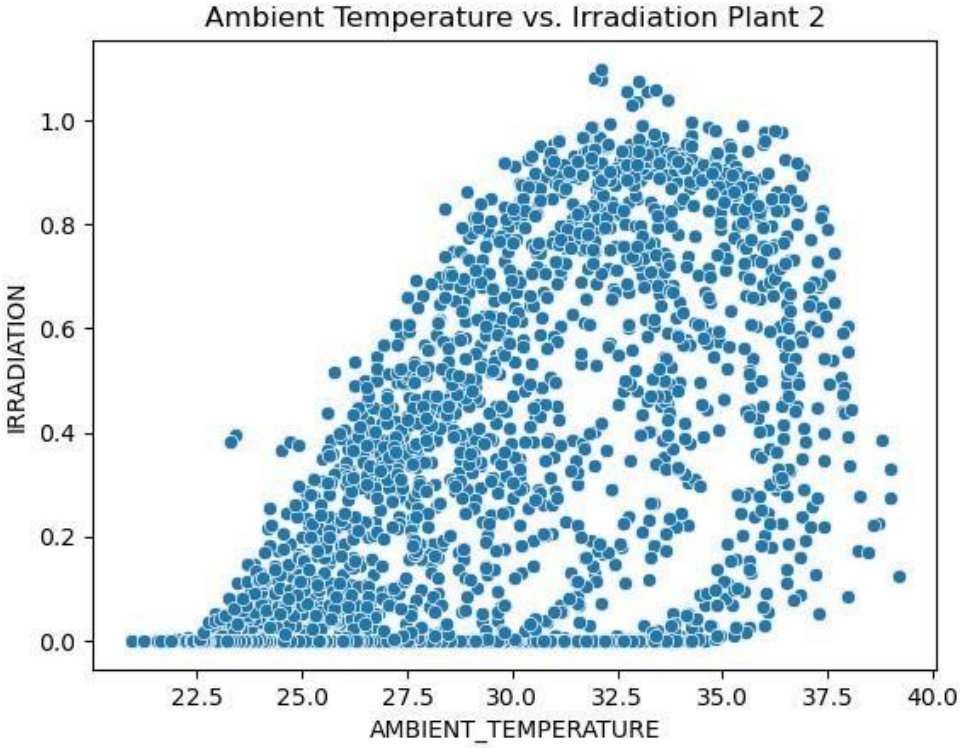
There is a Positive Correlation between Module Temperature and Ambient Temperature. The Module Temperature increases as the Ambient Temperature increases.

Relationship between Ambient Temperature & Irradiation:

```
In [68]: sns.scatterplot(data=plant1_sensor, x="AMBIENT_TEMPERATURE", y="IRRADIATION").set(title="Ambient
         Tem plt.show()
```

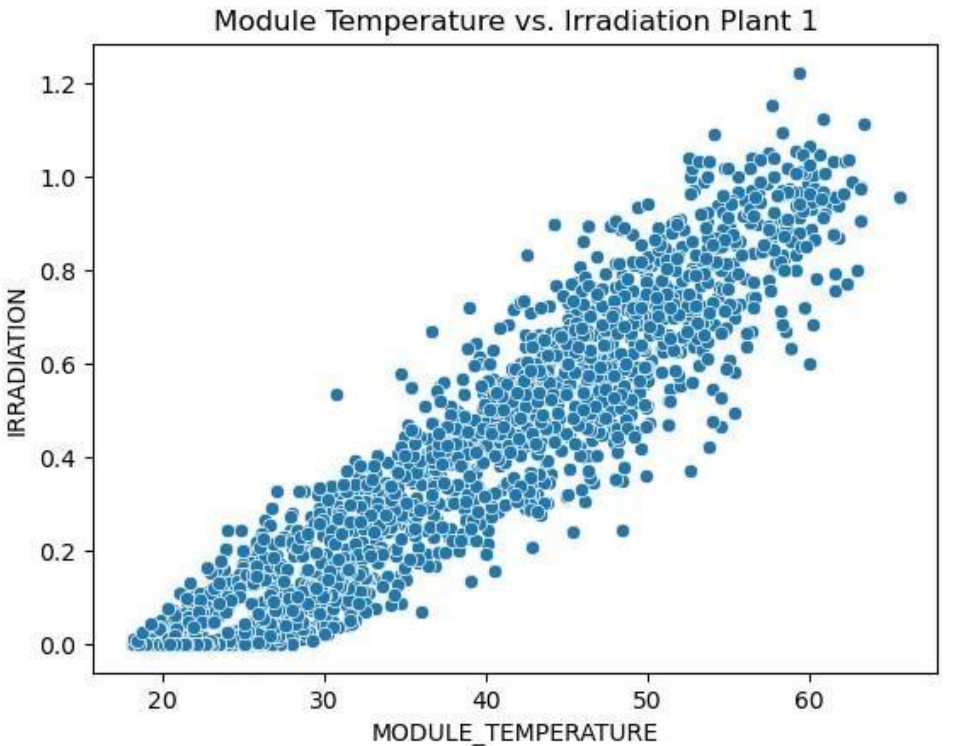


```
In [69]: sns.scatterplot(data=plant2_sensor, x="AMBIENT_TEMPERATURE", y="IRRADIATION").set(title="Ambient Tem plt.show()
```

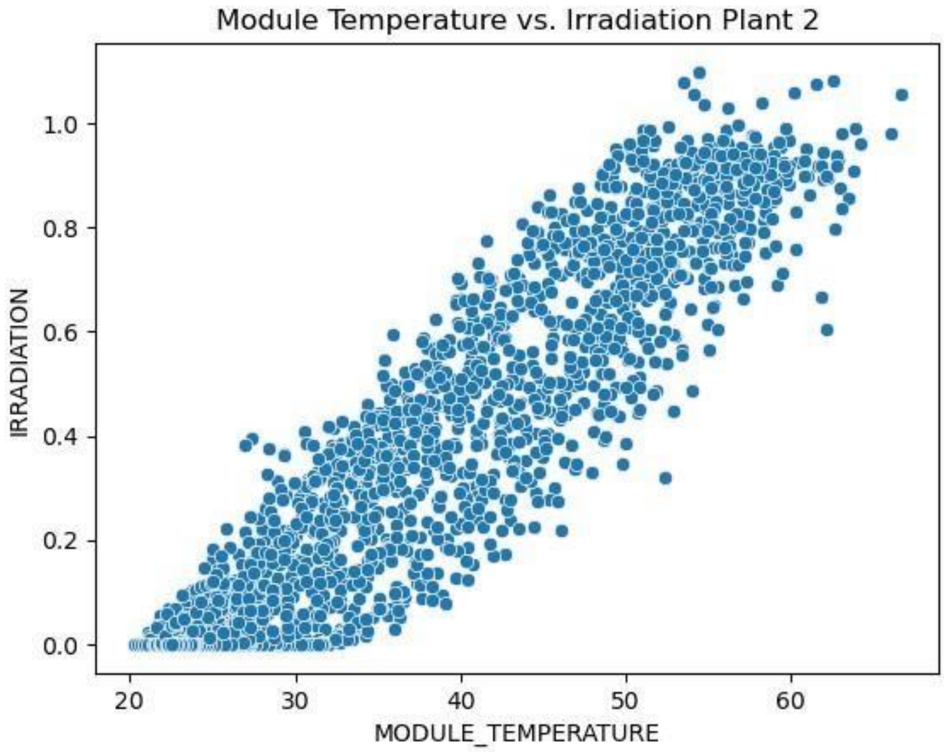


Relationship between Module Temperature & Irradiation:

```
In [70]: sns.scatterplot(data=plant1_sensor, x="MODULE_TEMPERATURE", y="IRRADIATION").set(title="Module Tempe plt.show()
```




```
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
0	2020-05-15 00:00:00	1BY6WEcLGh8j5v7	0.0	0.0	0.000	6259559.0	25.184316
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

In [82]:

```
plant2 = pd.merge(plant2_generation, plant2_sensor, on="DATE_TIME", how="left")
plant2.reset_index(drop=True)
```

Out[82]:

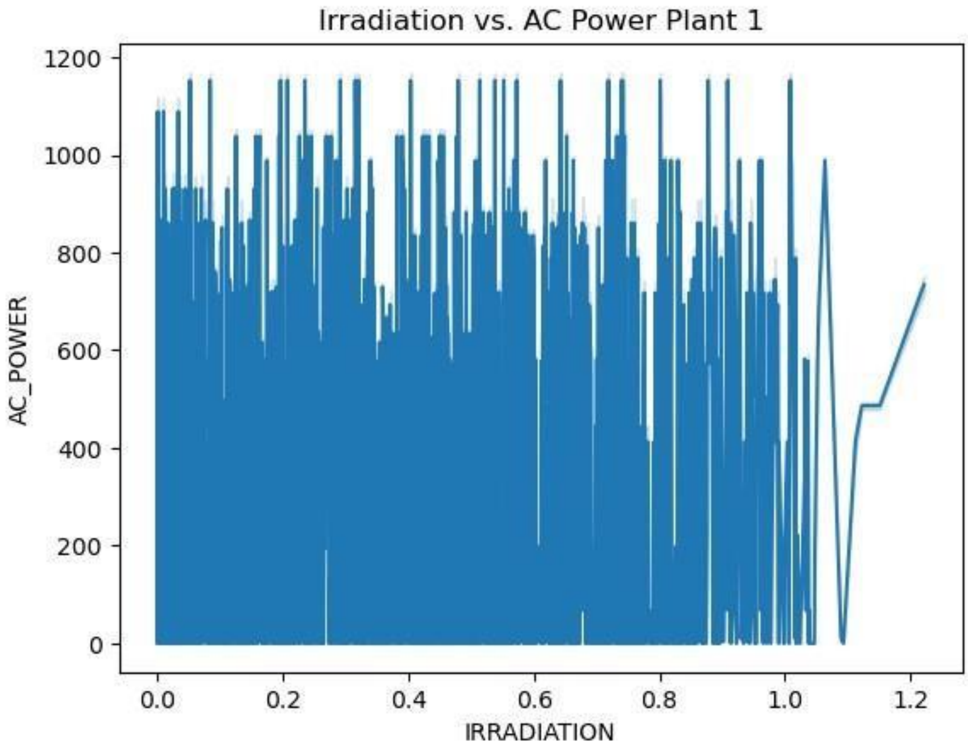
	DATE_TIME	INVERTER_ID	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD	AMBIENT_TEMPERATURE
0	2020-05-15 00:00:00	4UPUqMRk7TRMgml	0.0	0.0	9425.0	2429011.0	27.004764
1	2020-05-15 00:00:00	4UPUqMRk7TRMgml	0.0	0.0	9425.0	2429011.0	26.880811
2	2020-05-15 00:00:00	4UPUqMRk7TRMgml	0.0	0.0	9425.0	2429011.0	26.682055
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

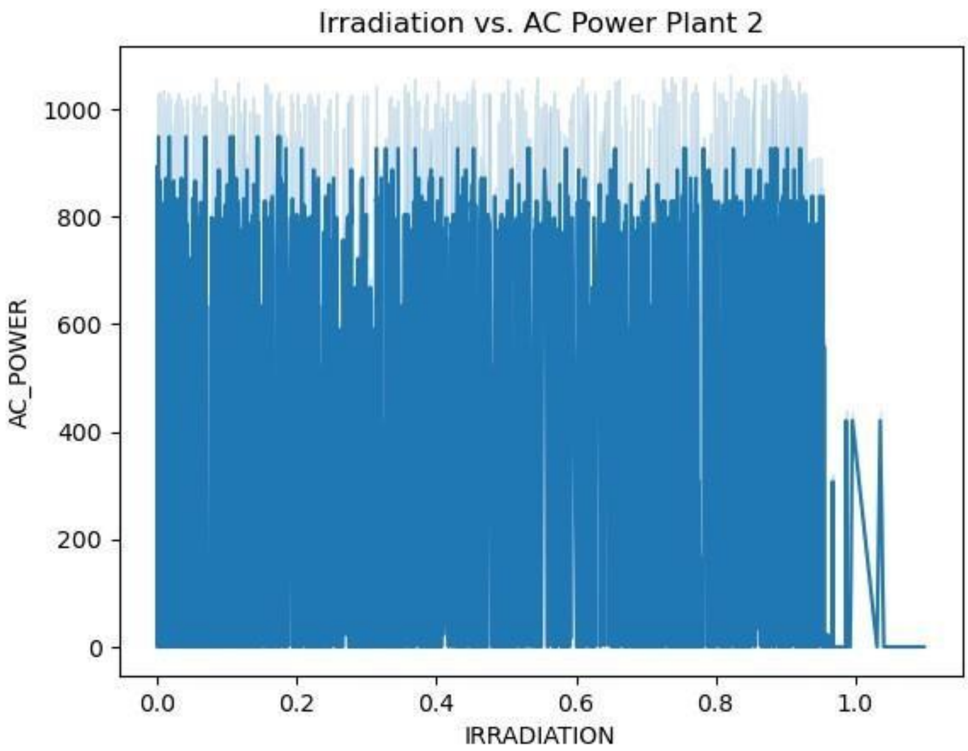
134656 rows × 9 columns

Exploring the Relationship between AC Power and Irradiation

In [83]:
```
sns.lineplot(data=plant1, x="IRRADIATION", y="AC_POWER").set(title="Irradiation vs. AC Power
Plant 1 plt.show()
```

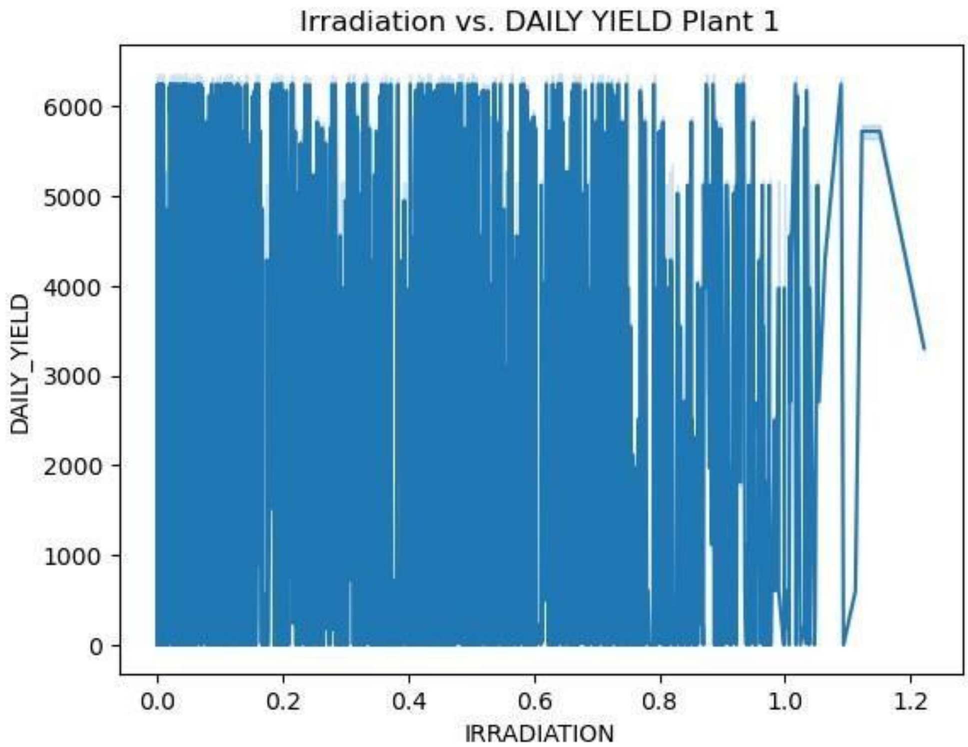


In [84]:
```
sns.lineplot(data=plant2, x="IRRADIATION", y="AC_POWER").set(title="Irradiation vs. AC Power
Plant 2 plt.show()
```

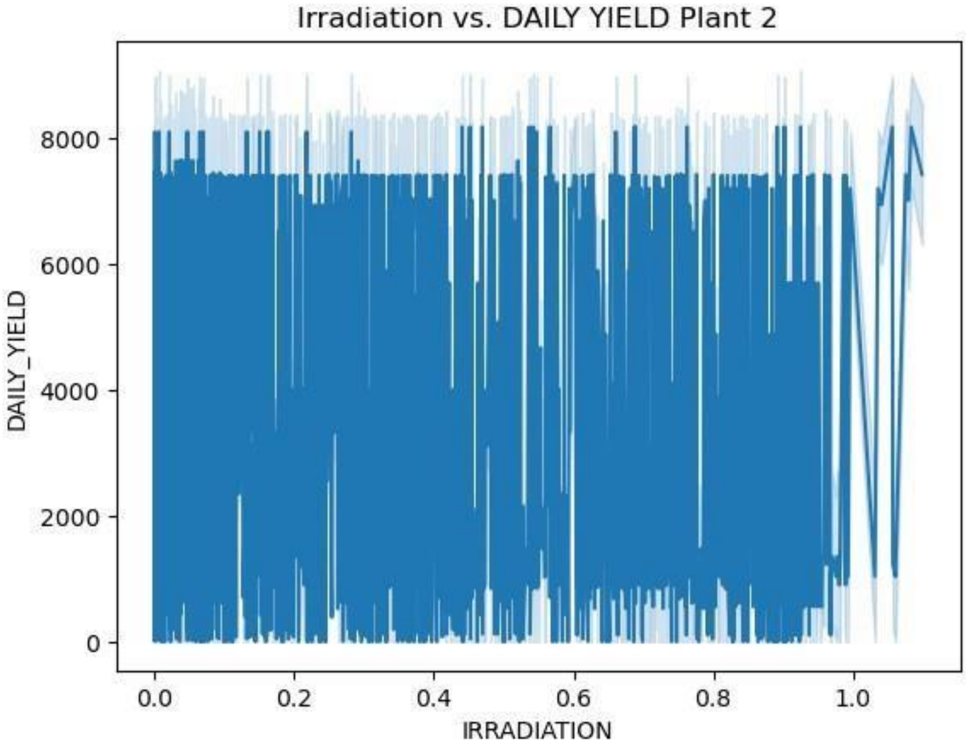


Exploring the Relationship between Daily Yield and Irradiation

In [85]:
```
sns.lineplot(data=plant1, x="IRRADIATION", y="DAILY_YIELD").set(title="Irradiation vs. DAILY
YIELD P plt.show()
```

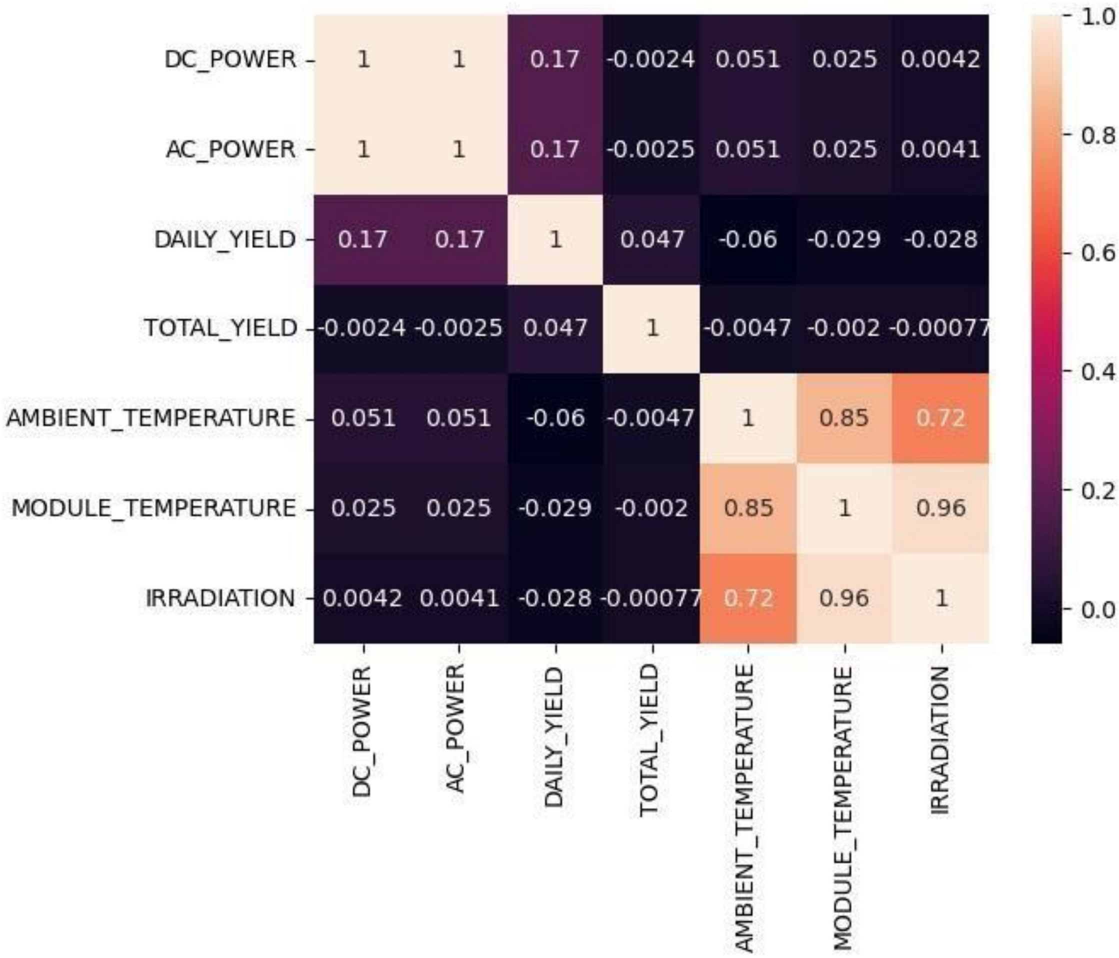


In [86]:
```
sns.lineplot(data=plant2, x="IRRADIATION", y="DAILY_YIELD").set(title="Irradiation vs. DAILY
YIELD P plt.show()
```



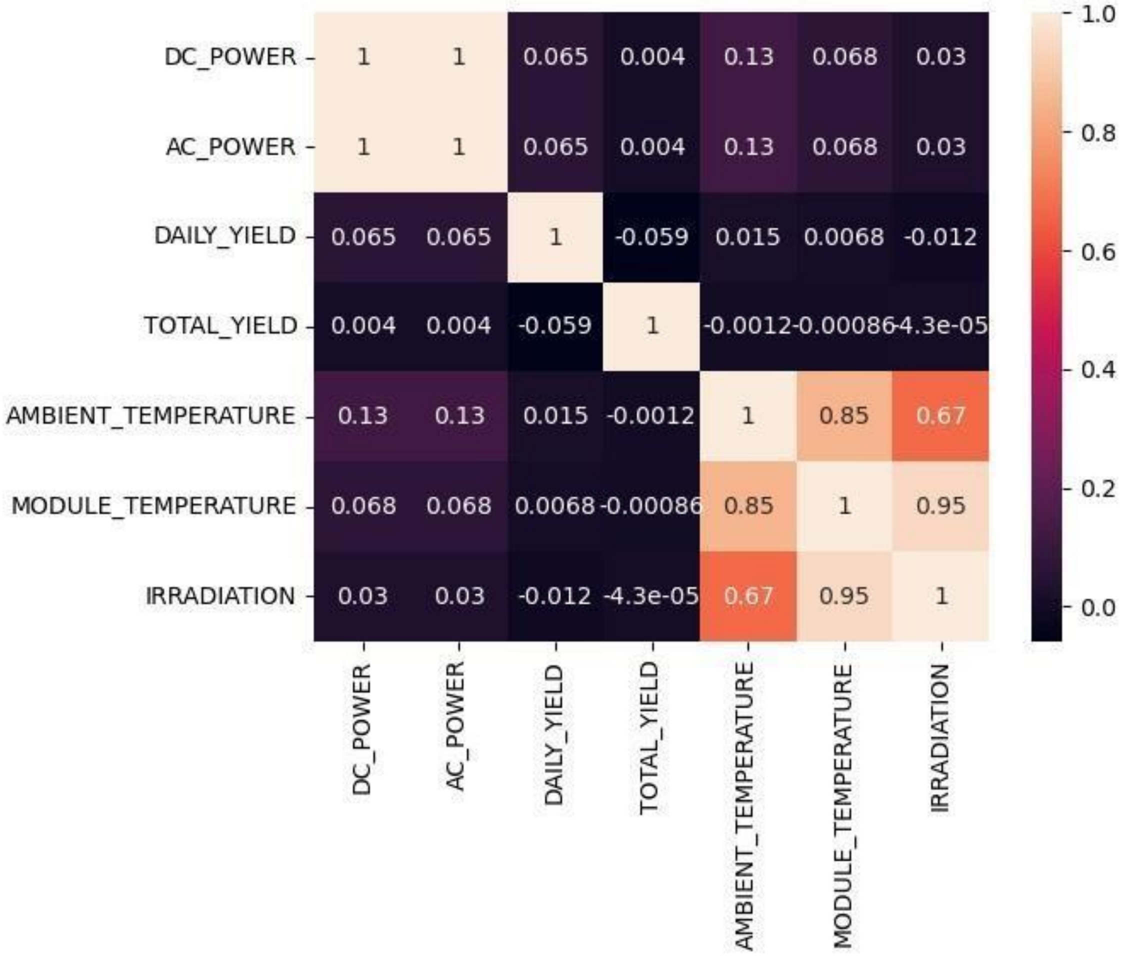
Exploring Correlation between features:

In [87]:
```
sns.heatmap(plant1.corr(numeric_only=True), annot=True, fmt='.2g') plt.show()
```



In [88]:

```
sns.heatmap(plant2.corr(numeric_only=True), annot=True, fmt='.2g') plt.show()
```



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

In [89]:

```
plant1.to_csv('plant1_merged.csv') plant2.to_csv('plant2_merged.csv')
```

Model Building:

Name: Ananya Godse SAP ID: 60009220161

Importing the basic libraries

In [1]:
```
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
```

Importing the datasets

In [2]:
```
plant1 = pd.read_csv("plant1_merged.csv")
plant1
```

Out[2]:

	Unnamed: 0	DATE_TIME	INVERTER_ID	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD	AMBIENT_TEM
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

In [3]:
```
plant2 = pd.read_csv("plant2_merged.csv")
plant2
```

Out[3]:

	Unnamed: 0	DATE_TIME	INVERTER_ID	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD	AMBIENT_TEM
		2020-05-15						

0	0	00:00:00	4UPUqMRk7TRMgml	0.0	0.0	9425.0	2429011.0
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

In [4]:

```
plant1.drop(["Unnamed: 0", "INVERTER_ID"],axis=1, inplace=True)
plant2.drop(["Unnamed: 0", "INVERTER_ID"],axis=1, inplace=True)
```

In [5]:

```
plant1.dtypes
```

Out[5]:

```
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
```

In [6]:

```
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")
```

In [7]:

```
plant1.dtypes
```

Out[7]:

```
DATE_TIME              datetime64[ns]
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

Out[8]:
```
DATE_TIME              datetime64[ns]
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)
```

In [13]:
```
train_features = ["AMBIENT_TEMPERATURE", "MODULE_TEMPERATURE", "IRRADIATION"]
X_p1 = reduced_plant1[train_features]

y_p1 = reduced_plant1["DAILY_YIELD"]
```

In [14]:
```
X_p2 = reduced_plant2[train_features]

y_p2 = reduced_plant2["DAILY_YIELD"]
```

In [15]:
```
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
```

In [16]:
```
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
```

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)
```

```
Out[19]: 7887607.970225091
```

```
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)
```

```
Out[21]: 9666845.977933416
```

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.
```

```
In [24]: 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 [25]: mean_squared_error(y_p1_test, pred)
```

```
Out[25]: 8109974.187599193
```

Okay the mse is still too high.

using GridSearchCV to find the best parameters

```
In [26]: 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)
```

```
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

In [27]:
```
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 [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)
```

```
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)
```

Out[30]: 6589861.20479132

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)
```

Out[33]:
```
7887634.380124176
```

In [34]:
```
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)
```

```
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)
```

Out[36]:
```
7887612.985658665
```

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
```

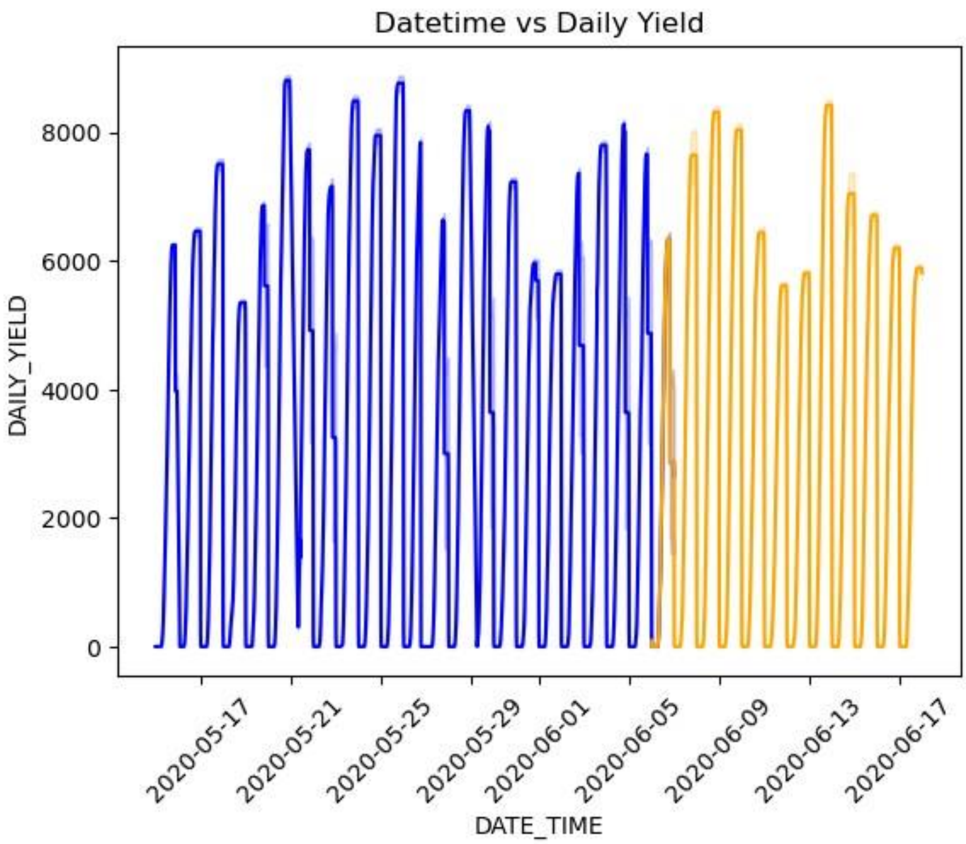
Out[39]:

	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

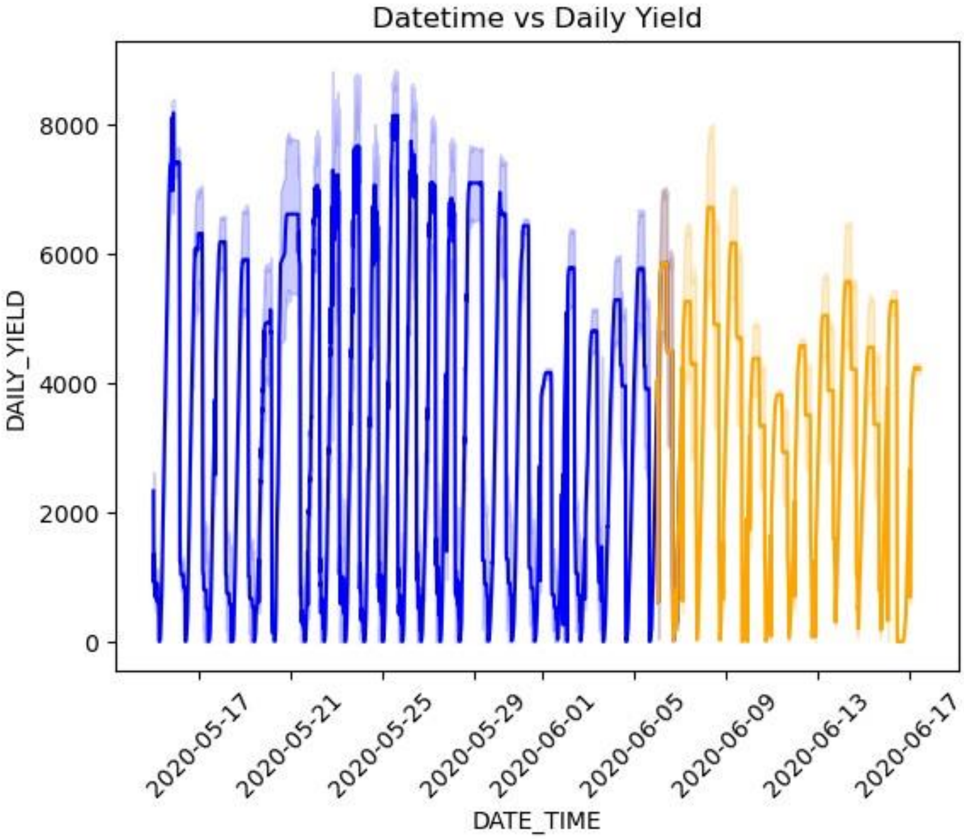
```
In [40]: split_date = '2020-06-06' plant1_train =
         t_reduced_plant1.loc[:split_date] plant1_test
         = t_reduced_plant1.loc[split_date:]
```

```
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()
```



In [42]:
```
plant2_train = t_reduced_plant2.loc[:split_date]
plant2_test = t_reduced_plant2.loc[split_date:]
```

In [43]:
```
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()
```



Datetime vs Daily Yield
DAILY_YIELD
0
2000
4000
6000
8000
2020-05-17
2020-05-21
2020-05-25
2020-05-29
2020-06-01
2020-06-05
2020-06-09
2020-06-13
2020-06-17
DATE_TIME

In [44]:
```
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 [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, ...)
```

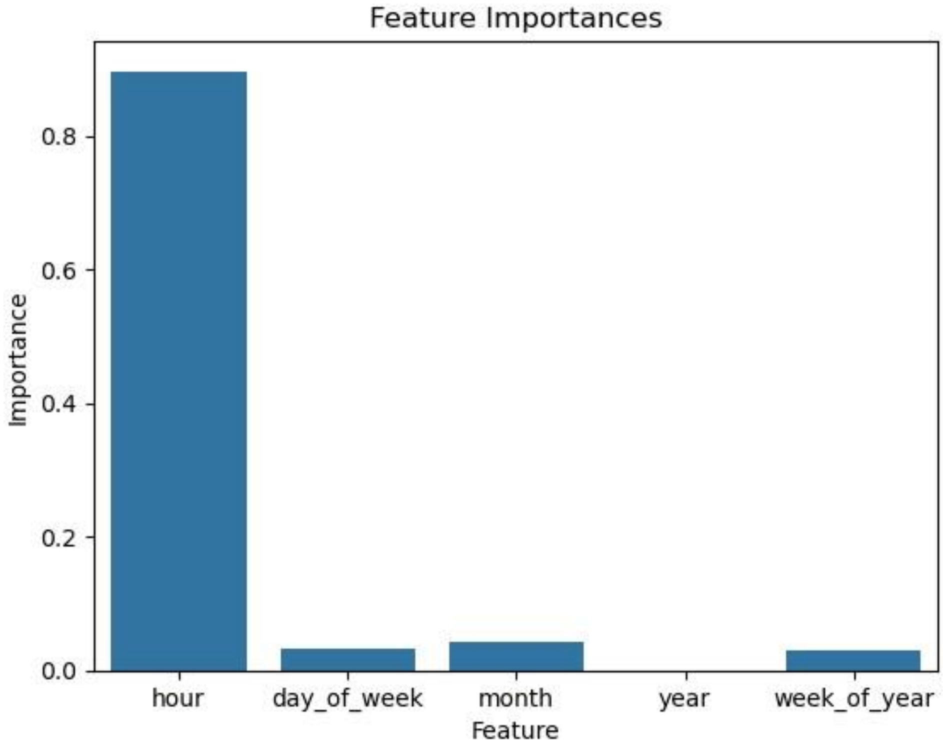
In [48]:
```
predictions = reg.predict(X_p1_test)
predictions
```

Out[48]:
```
array([  47.04413,   47.04413,   47.04413, ..., 7617.5845 , 7617.5845 ,
       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}")
```

RMSE = 1128.25

In [57]:
```
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_)
```

```
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:

In [58]:
```
plant2_train
```

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, ...)
```

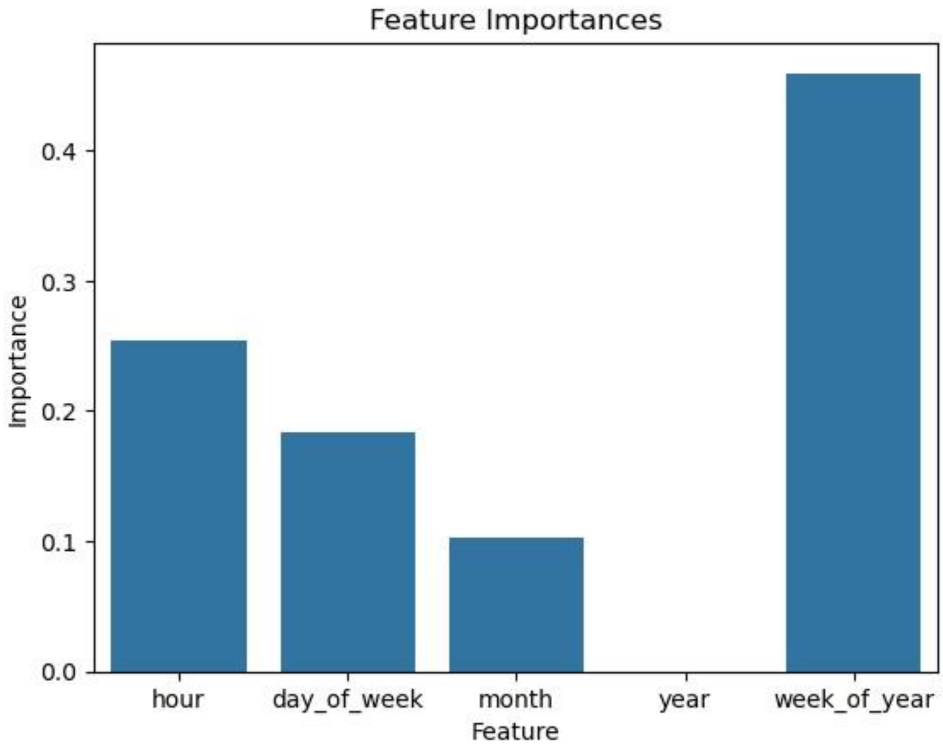
In [61]:
```
predictions = reg.predict(X_p2_test)
predictions
```

Out[61]:
```
array([3644.4348, 3644.4348, 3644.4348, ...,  975.2371,  975.2371,
        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
```

```
Out[66]: array([1067.047 , 5523.4956, 5374.6084, ..., 4315.8247, 4209.1377,
                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:

```
X_p1_test_final = plant1_test.iloc[:,1]
y_p1_test_final = plant1_test.iloc[:,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)
```

PLANT 1:

```
In [68]: X_p1_train_final = plant1_train.iloc[:, 1] # only hour data
          y_p1_train_final = plant1_train.iloc[:, 0]
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
```

```
Out[70]: array([  25.496717,   25.496717,   25.496717, ..., 4983.6963  ,
        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:

```
In [72]: 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 [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
```

Out[74]:
```
array([1067.047 , 5523.4956, 5374.6084, ..., 4315.8247, 4209.1377,
       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:

In [77]:
```
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 [ ]:

Performance Evaluation:

Plant 1:

```
In [70]: redictions_final_p1 = reg_final_p1.predict(X_p1_test_final)p
         predictions_final_p1

Out[70]: array([  25.496717,   25.496717,   25.496717, ..., 4983.6963  ,
                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:

```
In [74]: predictions_final_p2 = reg.predict(X_p2_test_final)
         predictions_final_p2

Out[74]: array([1067.047 , 5523.4956, 5374.6084, ..., 4315.8247, 4209.1377,
                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
```

Model Deployment:

Code:

models.py × index.html server.py

Model > models.py

```python
# Importing the libraries
import numpy as np
import pandas as pd
import xgboost as xgb
import pickle

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


#PLANT 1
plant1 = pd.read_csv("plant1_merged.csv")
plant1["DATE_TIME"] = pd.to_datetime(plant1["DATE_TIME"], format="%Y-%m-%d %H:%M:%S")
t_reduced_plant1 = plant1[["DATE_TIME","DAILY_YIELD"]]
t_reduced_plant1.set_index("DATE_TIME", inplace=True)

split_date = '2020-06-06'
plant1_train = t_reduced_plant1.loc[:split_date]
plant1_test = t_reduced_plant1.loc[split_date:]

t_reduced_plant1 = create_features(t_reduced_plant1)
plant1_train = create_features(plant1_train)
plant1_test = create_features(plant1_test)
```

Department of Computer Science and Engineering (Data Science)

models.py × | index.html | server.py

Model > models.py > ...

```python
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[:, 1]
y_p1_test_final = plant1_test.iloc[:, 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)

predictions_final_p1 = reg_final_p1.predict(X_p1_test_final)
pickle.dump(reg_final_p1, open('model1.pkl','wb'))


#PLANT 2
plant2 = pd.read_csv("plant2_merged.csv")
plant2["DATE_TIME"] = pd.to_datetime(plant2["DATE_TIME"], format="%Y-%m-%d %H:%M:%S")

t_reduced_plant2 = plant2[["DATE_TIME","DAILY_YIELD"]]
t_reduced_plant2.set_index("DATE_TIME", inplace=True)

plant2_train = t_reduced_plant2.loc[:split_date]
plant2_test = t_reduced_plant2.loc[split_date:]


t_reduced_plant2 = create_features(t_reduced_plant2)
plant2_train = create_features(plant2_train)
plant2_test = create_features(plant2_test)


X_p2_train_final = plant2_train.drop(['year', 'month', 'DAILY_YIELD'], axis=1) # using week_of_year, day_of_week, hour
y_p2_train_final = plant2_train.iloc[:, 0]
```

```python
X_p2_test_final = plant2_test.drop(['year', 'month', 'DAILY_YIELD'], axis=1)
y_p2_test_final = plant2_test.iloc[:, 0]


reg_final_p2 = xgb.XGBRegressor(n_estimators=1000, learning_rate=0.01)
reg_final_p2.fit(X_p2_train_final, y_p2_train_final)

predictions_final_p2 = reg_final_p2.predict(X_p2_test_final)
pickle.dump(reg_final_p2, open('model2.pkl','wb'))
```

Department of Computer Science and Engineering (Data Science)

models.py | index.html | server.py

Model > server.py > ...

```python
from flask import Flask, render_template, request, jsonify
import pickle
import numpy as np
import os

app = Flask(__name__)

# Get the directory of the current script
current_dir = os.path.dirname(os.path.abspath(__file__))

# Load the models
model1_path = os.path.join(current_dir, 'model1.pkl')
model2_path = os.path.join(current_dir, 'model2.pkl')

model1 = pickle.load(open(model1_path, 'rb'))
model2 = pickle.load(open(model2_path, 'rb'))

# Define route for rendering the index.html template
@app.route('/')
def index():
    return render_template('index.html')

# Define prediction endpoint for Model 1
@app.route('/predict_model1', methods=['POST'])
def predict_model1():
    # Get the JSON data from the request
    data = request.get_json()

    print("Received data for Model 1:", data)
    # Extract hour from the JSON data
    hour = int(data['hour'])
```

models.py | index.html | server.py

Model > server.py > ...

```python
        # Predict for model 1
        prediction = model1.predict(np.array([[hour]]))[0]

        # Create response JSON
        response = {
            'prediction': prediction.tolist()
        }

        return jsonify(response)

    # Define prediction endpoint for Model 2
    @app.route('/predict_model2', methods=['POST'])
    def predict_model2():
        # Get the JSON data from the request
        data = request.get_json()

        print("Received data for Model 2:", data)
        # Extract features from the JSON data
        hour = int(data['hour'])
        dayOfWeek = int(data['dayOfWeek'])
        weekOfYear = int(data['weekOfYear'])

        # Predict for model 2
        prediction = model2.predict(np.array([[weekOfYear, dayOfWeek, hour]]))[0]

        # Create response JSON
        response = {
            'prediction': prediction.tolist()
        }

        return jsonify(response)
```

Department of Computer Science and Engineering (Data Science)

models.py | index.html | server.py

Model > templates > index.html > html > body > script > submit() callback > success

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Solar Power Generation Prediction</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

    <style>
      body {
        font-family: sans-serif;
        margin: 2rem;
        background-color: #f0f0f0;
        background-image: url("static/images/Photovolatic-solar-system.webp");
        background-size: cover;
        opacity: 0.8;
      }

      h1 {
        text-align: center;
        color: white;
      }

      .form-container {
        display: flex;
        justify-content: center;
      }

```

Department of Computer Science and Engineering (Data Science)

models.py | index.html × | server.py

Model > templates > index.html > html > body > script > submit() callback > succe

```
        .prediction-form {
          width: 25%;
          height: 65vh;
          border: none;
          border-radius: 4px;
          padding: 2rem;
          padding-top: 1rem;
          margin: 1rem;
          margin-left: 2rem;
          background-color: #fff;
          box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
          text-align: center;
        }

        .prediction-form h2 {
          margin-bottom: 0.5rem;
          color: #3498db;
        }

        .prediction-form label {
          display: block;
          margin-bottom: 0.2rem;
          color: #3498db;
        }

        .prediction-form button {
          background-color: #3498db;
          color: #fff;
          border: none;
          border-radius: 4px;
          padding: 0.5rem 1rem;
```

Department of Computer Science and Engineering (Data Science)

models.py | index.html | server.py

Model > templates > index.html > html > body > script > submit() callback > success

```css
      .prediction-form button {
        background-color: #3498db;
        color: #fff;
        border: none;
        border-radius: 4px;
        padding: 0.5rem 1rem;
        cursor: pointer;
        transition: background-color 0.2s ease-in-out;
      }

      .prediction-form button:hover {
        background-color: #2980b9;
      }

      .prediction-form p {
        margin-top: 0.5rem;
        font-weight: bold;
        color: #3498db;
      }
    </style>
  </head>
```

models.py | index.html | server.py

Model > templates > index.html > html > body > script > submit() callback > success

```html
  <body>
    <h1>Solar Power Generation Prediction</h1>

    <div class="form-container">
      <form id="prediction-form1" class="prediction-form">
        <h2>Plant 1</h2> <br>
        <label for="feature1">Hour:</label>
        <input type="number" id="feature1" name="feature1" required /> <br> <br>
        <button type="submit">Predict Daily Yield</button> <br> <br>
        <p id="prediction1"></p>
      </form>

      <form id="prediction-form2" class="prediction-form">
        <h2>Plant 2</h2> <br>
        <label for="feature2">Hour:</label>
        <input type="number" id="feature2" name="feature2" required /> <br> <br>
        <label for="feature3">Day of Week:</label>
        <input type="number" id="feature3" name="feature3" required /> <br> <br>
        <label for="feature4">Week of Year:</label>
        <input type="number" id="feature4" name="feature4" required /> <br> <br>
        <button type="submit">Predict Daily Yield</button> <br> <br>
        <p id="prediction2"></p>
      </form>
    </div>
```

Department of Computer Science and Engineering (Data Science)

models.py | index.html | server.py

Model > templates > index.html > html > body > script > submit() callback > success

```
<script>
  $("#prediction-form1").submit(function (event) {
    event.preventDefault();
    const hour = $("#feature1").val();
    $.ajax({
      url: "/predict_model1",
      method: "POST",
      contentType: "application/json",
      data: JSON.stringify({ hour: hour }),
      success: function (response) {
        const roundedPrediction = Math.round(response.prediction * 100) / 100;
        $("#prediction1").text(
          "Daily Yield for Plant 1: " + roundedPrediction + " kW"
        );
      },
      error: function (jqXHR, textStatus, errorThrown) {
        console.error("Error:", textStatus, errorThrown);
        $("#prediction1").text(
          "An error occurred. Please try again later."
        );
      },
    });
  });
```

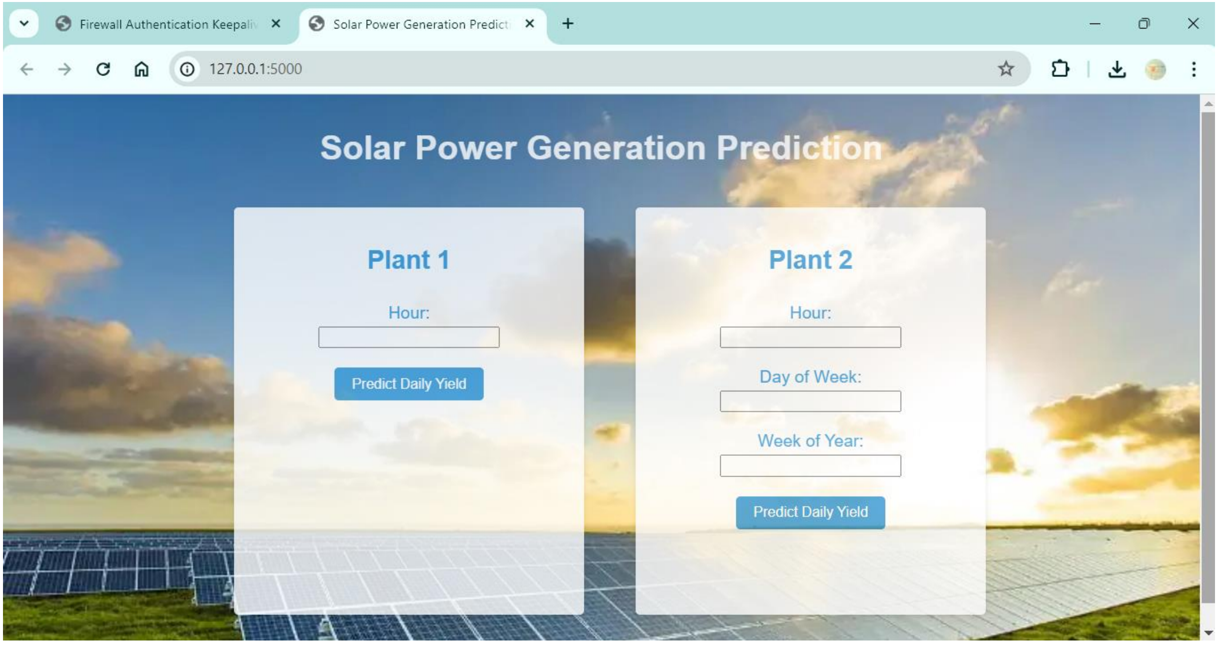
Department of Computer Science and Engineering (Data Science)

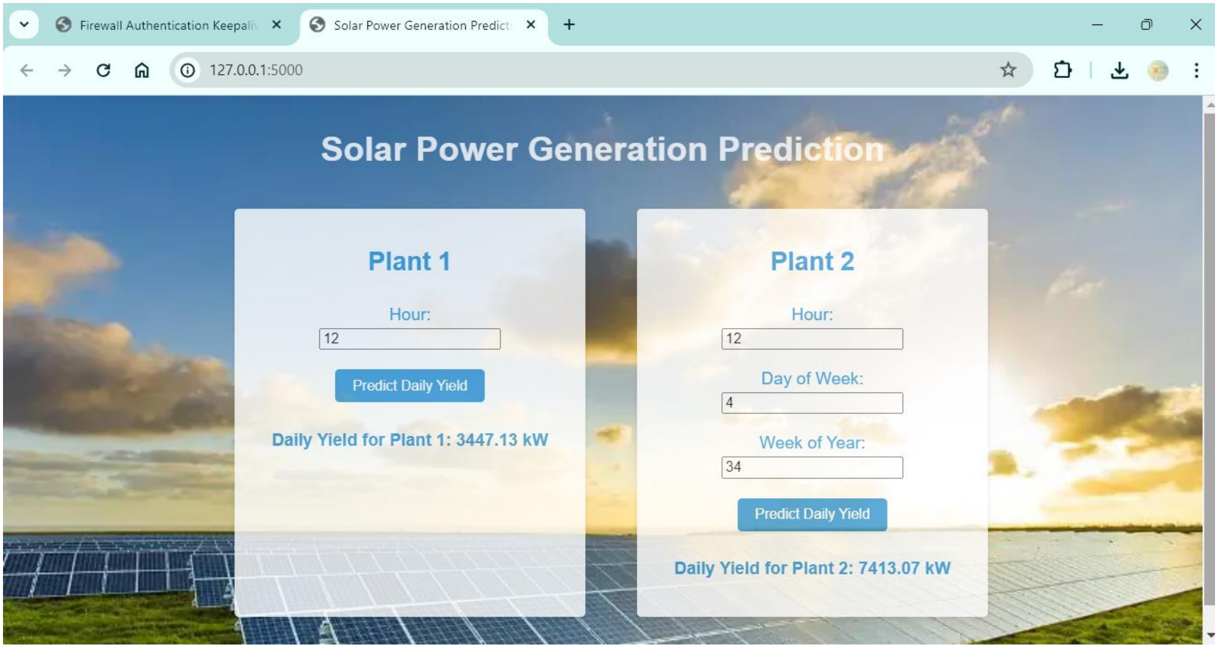
models.py | index.html | server.py

Model > templates > index.html > html > body > script > submit() callback > success

```
      $("#prediction-form2").submit(function (event) {
        event.preventDefault();
        const hour = $("#feature2").val();
        const dayOfWeek = parseInt($("#feature3").val());
        const weekOfYear = parseInt($("#feature4").val());
        $.ajax({
          url: "/predict_model2",
          method: "POST",
          contentType: "application/json",
          data: JSON.stringify({
            hour: hour,
            dayOfWeek: dayOfWeek,
            weekOfYear: weekOfYear,
          }),
          success: function (response) {
            const roundedPrediction = Math.round(response.prediction * 100) / 100;
            $("#prediction2").text(
              "Daily Yield for Plant 2: " + roundedPrediction + " kW"
            );
          },
          error: function (jqXHR, textStatus, errorThrown) {
            console.error("Error:", textStatus, errorThrown);
            $("#prediction2").text(
              "An error occurred. Please try again later."
            );
          },
        });
      });
    </script>
  </body>
</html>
```

Flask App:


Firewall Authentication Keepali
Solar Power Generation Predict
127.0.0.1:5000
Solar Power Generation Prediction
Plant 1
Hour:
Predict Daily Yield
Plant 2
Hour:
Day of Week:
Week of Year:
Predict Daily Yield


Firewall Authentication Keepali
Solar Power Generation Predict
127.0.0.1:5000
Solar Power Generation Prediction
Plant 1
Hour:
12
Predict Daily Yield
Daily Yield for Plant 1: 3447.13 kW
Plant 2
Hour:
12
Day of Week:
4
Week of Year:
34
Predict Daily Yield
Daily Yield for Plant 2: 7413.07 kW

Department of Computer Science and Engineering (Data Science)