Plant_1-Copy4

November 9, 2022

Table of Contents

1 Project exploration

In this notebook, we seek to understand the behaviour of two solar power plants through the data generated by the photovoltaic modules. To do so, we will talk about:

1. **Reminder on photovoltaic systems or PV systems**
2. **EDA on:**
   - ***DC and AC power***
   - ***Irradiation***
   - ***ambient and module temperature***
   - ***yield***
3. **Correlation of all features**
4. **Comparison of two power plants**

1.1 PV systems

`PV system` is a power system designed to supply usable solar power by means of photovoltaics.

`PV Cell` is an electrical device that converts the energy of light directly into electricity by the photovoltaic effect, which is a physical and chemical phenomenon. It is also the basics photovoltaic device that is the building block PV modules.

`Photovoltaic effect` is the generation of voltage and electric current in a material upon exposure to light.

`PV module` is a group of PV cell connected in serie and/or parallel and encapsulated in an environmentally protective laminate.

`PV panel` is a group of modules that is the basic building block of a PV array.

`PV array` is a group of panels that comprises the complete PV generating unit.

1.1.1 PV inverter

PV inverter convert battery or PV array DC power to AC power for use with conventional utility-powered appliances. It is heart of PV systems because PV array is a DC source, an inverter is required to convert the dc power to normal ac power that is used in our homes and offices.

PV systems are very influenced by weather condition, if the weather is good, we get a maximun yield but if the weather is bad, we get a minimun yield. That is why there is important to know how weather condition can impact on yield of the two solar power plants.

Source - Photovoltaic(PV) Tutorial

- PV Inverter
- PV Systems

According to the notion of PV systems, the important feature are:

- *DC power*
- *AC power*
- *Yield*
- *ambiant Temperature*
- *module temperature*
- *irradiation*

Okay, let's go to the next section.

2 Importing Required Libraries

[1]:
```
import matplotlib.dates as mdates
import cufflinks as cf
from scipy.stats import normaltest
import statsmodels.api as sm
import warnings
from scipy.optimize import curve_fit
import sklearn
import numpy as np  # linear algebra
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)
import plotly
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import datetime as dt
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
xformatter = mdates.DateFormatter('%H:%M')  # for time axis plots
```

```
import plotly.io as pio
pio.renderers.default = "png"
warnings.filterwarnings('ignore')
# import all package needed
```

3 Plant 1

[2]:
```
cf.set_config_file(offline=True)
sns.set(style="whitegrid")
```

3.1 Generation Data

3.1.1 Dataset Description

[3]:
```
# we take file for plant 1 Generation data
file = 'Plant_1_Generation_Data.csv'
```

[4]:
```
plant1_data = pd.read_csv(file)  # load data
```

[5]:
```
plant1_data.head()
```

[5]:

	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

[6]:
```
plant1_data.describe().style.background_gradient(cmap='Blues')
```

[6]:
```
<pandas.io.formats.style.Styler at 0x15a82bd0f70>
```

[7]:
```
print('The number of inverter for each day are {}'.format(
    plant1_data[plant1_data.DATE_TIME == '15-05-2020 23:00']['SOURCE_KEY'].
 ↪nunique()))
```

```
The number of inverter for each day are 22
```

Plant contains 22 inverters where each inverter are connected with several PV array. Every 15 min, each inverter records his data. So, if we want to know how many the plant has produced a power

in a hour, we just compute the contribution of 22 inverters.

[8]:
```
plant1_data.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
```

[9]:
```
plant1_data.shape  # our data reduced very well
```

[9]:
```
(68778, 7)
```

3.1.2 Data Pre-proccessing

Cleaning data Converting the column DATE_TIME object type to datetime type.Then spliting DATE_TIME to **Date** and **Time** respectively.

[10]:
```
# we compute a sum of 22 inverters
plant1_data = plant1_data.groupby(
    'DATE_TIME')[['DC_POWER', 'AC_POWER', 'DAILY_YIELD', 'TOTAL_YIELD']].
 ↪agg('sum')
```

[11]:
```
plant1_data = plant1_data.reset_index()
```

[12]:
```
plant1_data['DATE_TIME'] = pd.to_datetime(
    plant1_data['DATE_TIME'], format='%d-%m-%Y %H:%M')
```

[13]:
```
plant1_data['time'] = plant1_data['DATE_TIME'].dt.time

plant1_data['date'] = pd.to_datetime(plant1_data['DATE_TIME'].dt.date)
```

[14]:
```
plant1_data.head()
```

[14]:

	DATE_TIME	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD	time	\
0	2020-06-01 00:00:00	0.0	0.0	5407.25	153519480.0	00:00:00	
1	2020-06-01 00:15:00	0.0	0.0	0.00	153519480.0	00:15:00	
2	2020-06-01 00:30:00	0.0	0.0	0.00	153519480.0	00:30:00	
3	2020-06-01 00:45:00	0.0	0.0	0.00	153519480.0	00:45:00	

```
4 2020-06-01 01:00:00       0.0       0.0         0.00  153519480.0  01:00:00

        date
0 2020-06-01
1 2020-06-01
2 2020-06-01
3 2020-06-01
4 2020-06-01
```

[15]:
```
plant1_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3158 entries, 0 to 3157
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   DATE_TIME    3158 non-null   datetime64[ns]
 1   DC_POWER     3158 non-null   float64
 2   AC_POWER     3158 non-null   float64
 3   DAILY_YIELD  3158 non-null   float64
 4   TOTAL_YIELD  3158 non-null   float64
 5   time         3158 non-null   object
 6   date         3158 non-null   datetime64[ns]
dtypes: datetime64[ns](2), float64(4), object(1)
memory usage: 172.8+ KB
```

3.1.3 Exploratory Data Analysis

We will explore these columns `DC power`, `AC power` and `Yield` using libraries used plotly and seaborn.

Here, we use

1. Line or Scatter plot
2. Box Plot
3. Calendar Plot
4. Bar Plot.

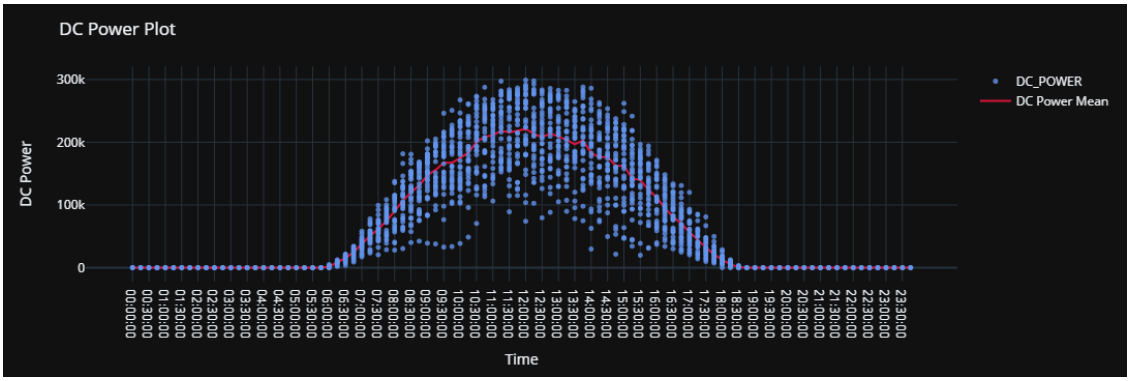
[16]:
```
plant1_data
```

[16]:
```
                DATE_TIME  DC_POWER  AC_POWER    DAILY_YIELD  TOTAL_YIELD  \
0    2020-06-01 00:00:00       0.0       0.0    5407.250000  153519480.0
1    2020-06-01 00:15:00       0.0       0.0       0.000000  153519480.0
2    2020-06-01 00:30:00       0.0       0.0       0.000000  153519480.0
3    2020-06-01 00:45:00       0.0       0.0       0.000000  153519480.0
4    2020-06-01 01:00:00       0.0       0.0       0.000000  153519480.0
…                    …       …         …              …            …
3153 2020-05-31 22:45:00       0.0       0.0  125291.000000  153519480.0
```

```
3154 2020-05-31 23:00:00       0.0       0.0  125291.000000  153519480.0
3155 2020-05-31 23:15:00       0.0       0.0  125291.000000  153519480.0
3156 2020-05-31 23:30:00       0.0       0.0  125291.000000  153519480.0
3157 2020-05-31 23:45:00       0.0       0.0  113737.142857  153519480.0

          time       date
0     00:00:00 2020-06-01
1     00:15:00 2020-06-01
2     00:30:00 2020-06-01
3     00:45:00 2020-06-01
4     01:00:00 2020-06-01
…        …        …
3153  22:45:00 2020-05-31
3154  23:00:00 2020-05-31
3155  23:15:00 2020-05-31
3156  23:30:00 2020-05-31
3157  23:45:00 2020-05-31

[3158 rows x 7 columns]
```

DC Power

[17]:
```
dc_mean = plant1_data.groupby('time')['DC_POWER'].agg('mean').reset_index()
#fig = go.Figure()
fig = px.scatter(data_frame=plant1_data, x='time', y=[
                 'DC_POWER'], hover_data=["DATE_TIME"], template="plotly_dark")

#fig.add_trace(go.Scatter(x=plant1_data["time"], y=plant1_data['DC_POWER'],␣
 ↪name='DC Power', text = plant1_data['date'],mode='markers'))
fig.update_traces(marker=dict(color="#6495ED", size=5,
                  opacity=0.8), selector=dict(mode='markers'))
fig.add_scatter(x=dc_mean["time"], y=dc_mean["DC_POWER"],
                name='DC Power Mean', line=dict(color="crimson"))
fig.update_layout(title="DC Power Plot",
                  xaxis_title="Time",
                  yaxis_title="DC Power", template="plotly_dark",␣
 ↪legend=dict(title=""), hovermode="y unified")

plotly.offline.iplot(fig)
```

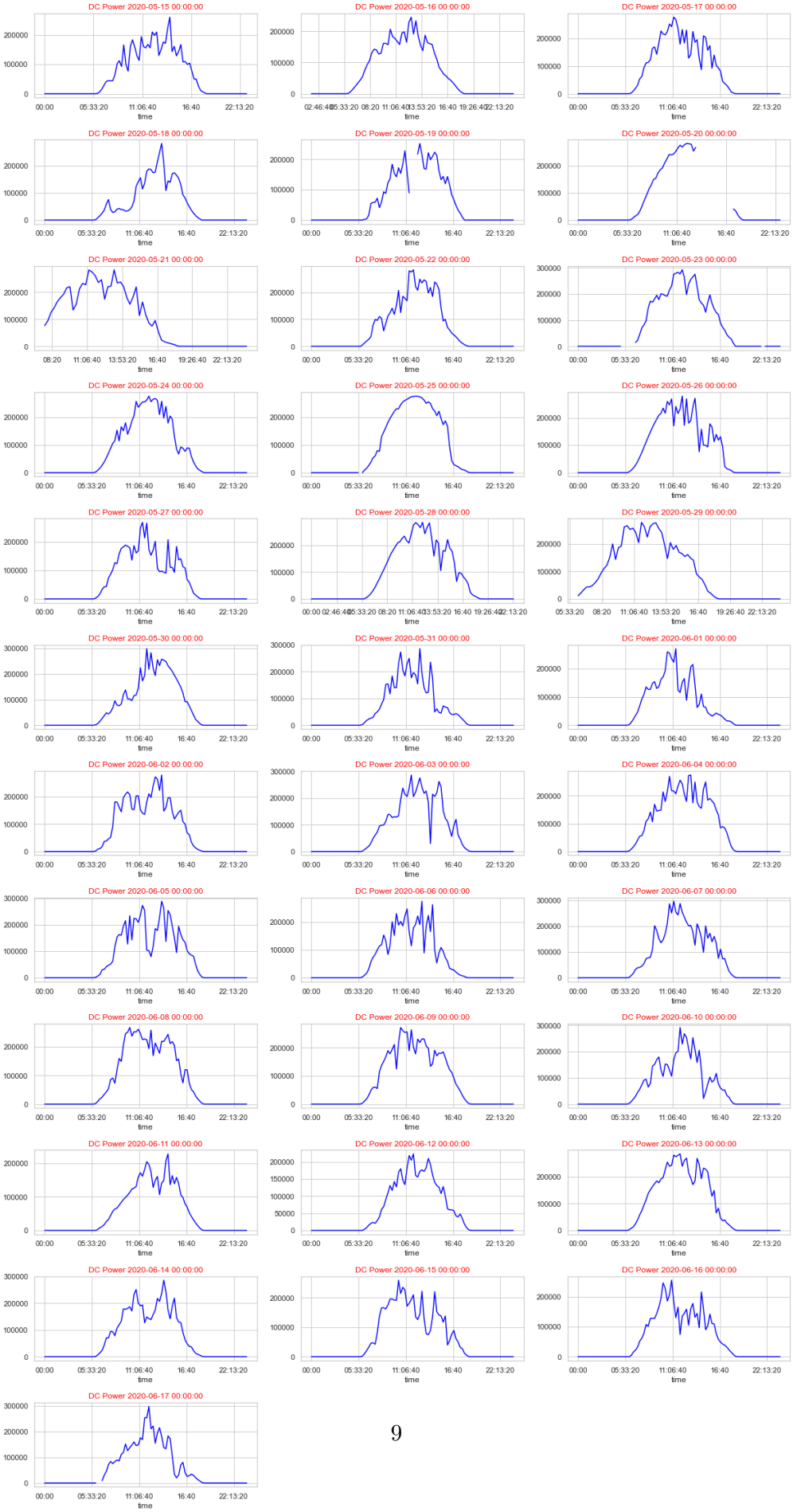


Between 05:33:20 and 18:00:00, the Plant generates a DC Power but where as during remaininng hours of the day there was no power generation. The reason is sunlight.

[18]:
```
# define function to multi plot

def multi_plot(data=None, row=None, col=None, title='DC Power'):
    cols = data.columns  # take all column
    gp = plt.figure(figsize=(20, 20))

    gp.subplots_adjust(wspace=0.2, hspace=0.8)
    for i in range(1, len(cols)+1):
        ax = gp.add_subplot(row, col, i)
        data[cols[i-1]].plot(ax=ax, style='k.')
        ax.set_title('{} {}'.format(title, cols[i-1]))
```

[19]:
```
def Daywise_plot(data=None, row=None, col=None, title='DC Power'):
    cols = data.columns  # take all column
    gp = plt.figure(figsize=(20, 40))

    gp.subplots_adjust(wspace=0.2, hspace=0.5)
    for i in range(1, len(cols)+1):
        ax = gp.add_subplot(row, col, i)
        data[cols[i-1]].plot(ax=ax, color='blue')
        ax.set_title('{} {}'.format(title, cols[i-1]), color='red')
```

[20]:
```
# Okay, we are going to see dc power in each day produced by Plant.
# we create calendar_dc data how in each day Plant produce a dc power in each␣
 ↪time.

calendar_dc = plant1_data.pivot_table(
    values='DC_POWER', index='time', columns='date')

Daywise_plot(data=calendar_dc, row=12, col=3)
```


DC Power 2020-05-15 00:00:00
DC Power 2020-05-16 00:00:00
DC Power 2020-05-17 00:00:00
DC Power 2020-05-18 00:00:00
DC Power 2020-05-19 00:00:00
DC Power 2020-05-20 00:00:00
DC Power 2020-05-21 00:00:00
DC Power 2020-05-22 00:00:00
DC Power 2020-05-23 00:00:00
DC Power 2020-05-24 00:00:00
DC Power 2020-05-25 00:00:00
DC Power 2020-05-26 00:00:00
DC Power 2020-05-27 00:00:00
DC Power 2020-05-28 00:00:00
DC Power 2020-05-29 00:00:00
DC Power 2020-05-30 00:00:00
DC Power 2020-05-31 00:00:00
DC Power 2020-06-01 00:00:00
DC Power 2020-06-02 00:00:00
DC Power 2020-06-03 00:00:00
DC Power 2020-06-04 00:00:00
DC Power 2020-06-05 00:00:00
DC Power 2020-06-06 00:00:00
DC Power 2020-06-07 00:00:00
DC Power 2020-06-08 00:00:00
DC Power 2020-06-09 00:00:00
DC Power 2020-06-10 00:00:00
DC Power 2020-06-11 00:00:00
DC Power 2020-06-12 00:00:00
DC Power 2020-06-13 00:00:00
DC Power 2020-06-14 00:00:00
DC Power 2020-06-15 00:00:00
DC Power 2020-06-16 00:00:00
DC Power 2020-06-17 00:00:00
time

Almost all the curves are same , fluctuation between 11 am and 2 pm. Except the curve of May 20 and 25 which gives a uniform shape.

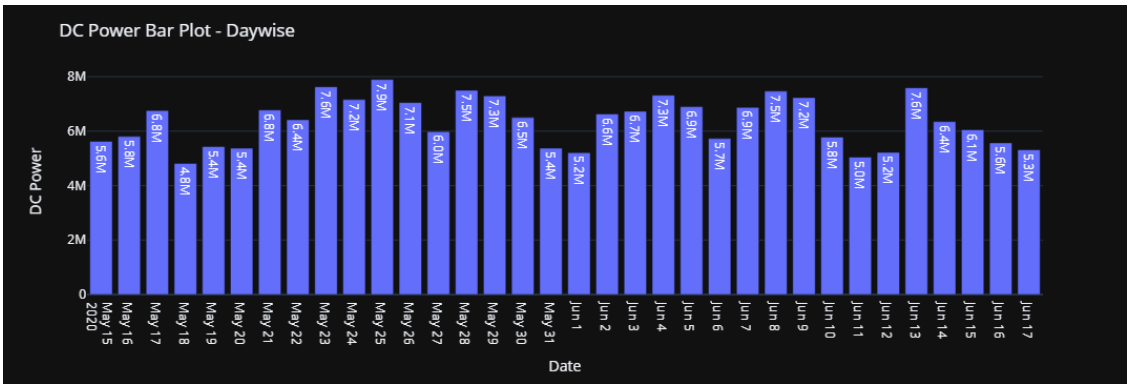
[21]:
```
daily_dc = plant1_data.groupby('date')['DC_POWER'].agg('sum').reset_index()
```

[22]:
```
daily_dc.head()
```

[22]:

	date	DC_POWER
0	2020-05-15	5.627239e+06
1	2020-05-16	5.806138e+06
2	2020-05-17	6.759595e+06
3	2020-05-18	4.812549e+06
4	2020-05-19	5.437955e+06

[23]:
```
fig = px.bar(data_frame=daily_dc, x='date', y='DC_POWER',
             text_auto='.2s', template="plotly_dark")
#fig.update_traces(textfont_size=12, textangle=0, textposition="outside",␣
 ↪cliponaxis=False)
fig.update_xaxes(dtick="d1")
fig.update_layout(title="DC Power Bar Plot - Daywise",
                  xaxis_title="Date",
                  yaxis_title="DC Power", hovermode="x unified")
plotly.offline.iplot(fig)
```

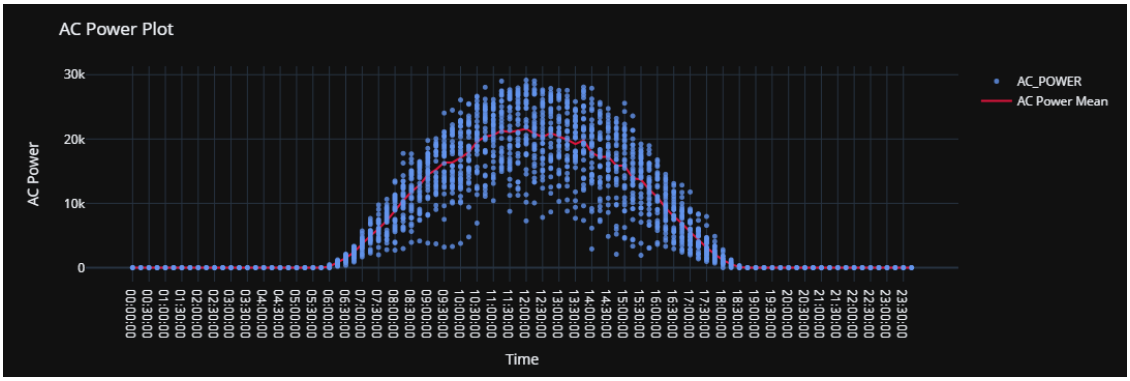


Dc Power is maximun on May 25 .

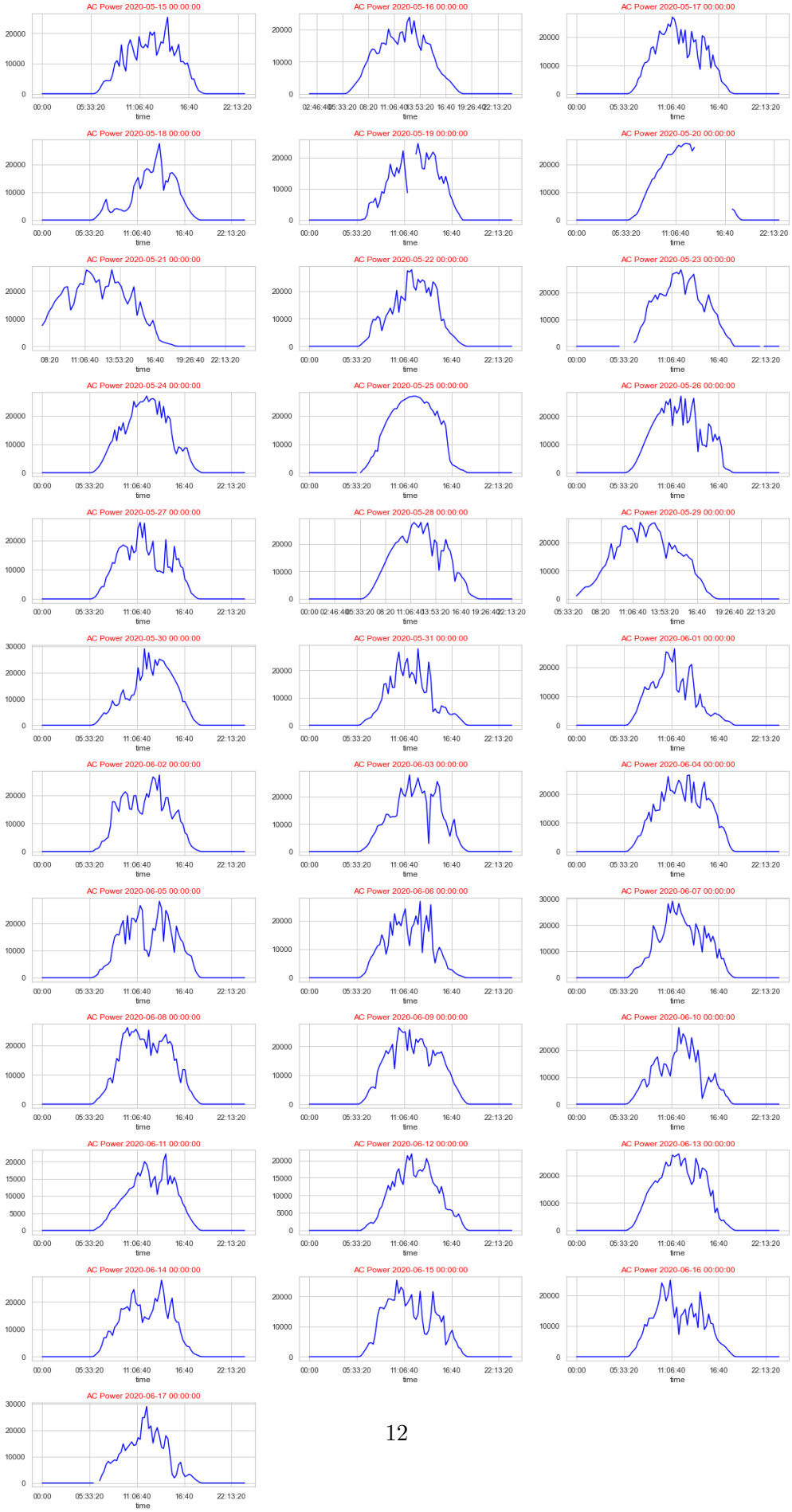
AC Power

[24]:
```
ac_mean = plant1_data.groupby('time')['AC_POWER'].agg('mean').reset_index()
#fig = go.Figure()
fig = px.scatter(data_frame=plant1_data, x='time', y=[
                 'AC_POWER'], hover_data=["DATE_TIME"], template="plotly_dark")
```

```
#fig.add_trace(go.Scatter(x=plant1_data["time"], y=plant1_data['DC_POWER'],␣
 ↪name='DC Power', text = plant1_data['date'],mode='markers'))
fig.update_traces(marker=dict(color="#6495ED", size=5,
                  opacity=0.8), selector=dict(mode='markers'))
fig.add_scatter(x=ac_mean["time"], y=ac_mean["AC_POWER"],
                name='AC Power Mean', line=dict(color="crimson"))
fig.update_layout(title="AC Power Plot",
                  xaxis_title="Time",
                  yaxis_title="AC Power", template="plotly_dark",␣
 ↪legend=dict(title=""), hovermode="y unified")

plotly.offline.iplot(fig)
```

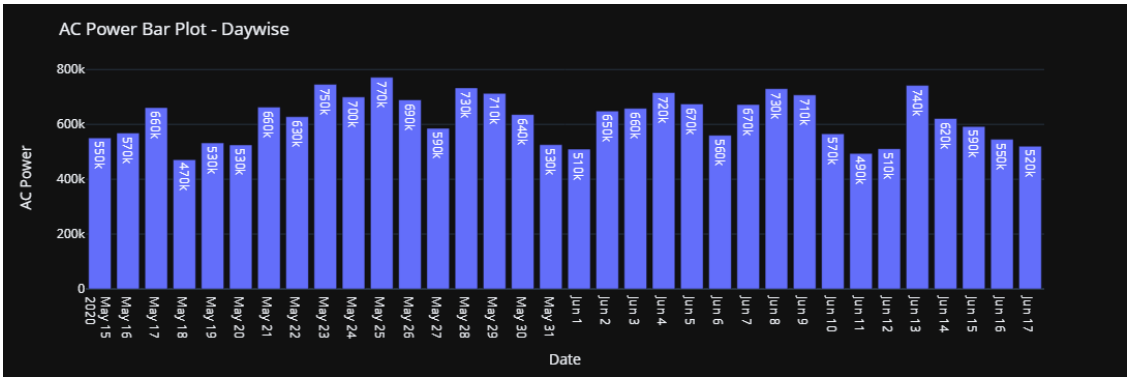


[25]:
```
# Okay, we are going to see dc power in each day produced by Plant.
# we create calendar_ac data how in each day Plant produce a ac power in  each␣
 ↪time.

calendar_ac = plant1_data.pivot_table(
    values='AC_POWER', index='time', columns='date')

Daywise_plot(data=calendar_ac, row=12, col=3, title='AC Power')
```


AC Power 2020-05-15 00:00:00
AC Power 2020-05-16 00:00:00
AC Power 2020-05-17 00:00:00
AC Power 2020-05-18 00:00:00
AC Power 2020-05-19 00:00:00
AC Power 2020-05-20 00:00:00
AC Power 2020-05-21 00:00:00
AC Power 2020-05-22 00:00:00
AC Power 2020-05-23 00:00:00
AC Power 2020-05-24 00:00:00
AC Power 2020-05-25 00:00:00
AC Power 2020-05-26 00:00:00
AC Power 2020-05-27 00:00:00
AC Power 2020-05-28 00:00:00
AC Power 2020-05-29 00:00:00
AC Power 2020-05-30 00:00:00
AC Power 2020-05-31 00:00:00
AC Power 2020-06-01 00:00:00
AC Power 2020-06-02 00:00:00
AC Power 2020-06-03 00:00:00
AC Power 2020-06-04 00:00:00
AC Power 2020-06-05 00:00:00
AC Power 2020-06-06 00:00:00
AC Power 2020-06-07 00:00:00
AC Power 2020-06-08 00:00:00
AC Power 2020-06-09 00:00:00
AC Power 2020-06-10 00:00:00
AC Power 2020-06-11 00:00:00
AC Power 2020-06-12 00:00:00
AC Power 2020-06-13 00:00:00
AC Power 2020-06-14 00:00:00
AC Power 2020-06-15 00:00:00
AC Power 2020-06-16 00:00:00
AC Power 2020-06-17 00:00:00
time

[26]:
```
daily_ac = plant1_data.groupby('date')['AC_POWER'].agg('sum').reset_index()
```

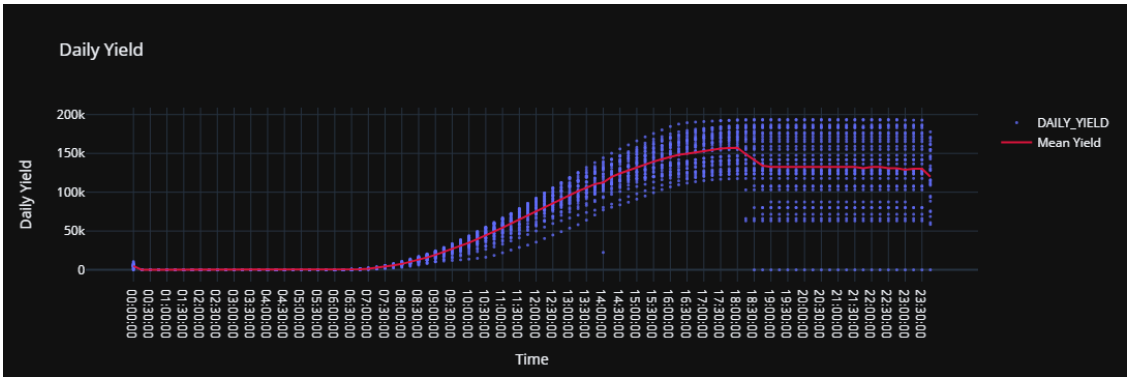
[27]:
```
fig = px.bar(data_frame=daily_ac, x='date', y='AC_POWER',
             text_auto='.2s', template="plotly_dark")
#fig.update_traces(textfont_size=12, textangle=0, textposition="outside",␣
 ↪cliponaxis=False)
fig.update_xaxes(dtick="d1")
fig.update_layout(title="AC Power Bar Plot - Daywise",
                  xaxis_title="Date",
                  yaxis_title="AC Power", hovermode="x unified")
plotly.offline.iplot(fig)
```



Daily Yield

[28]:
```
dy_mean = plant1_data.groupby('time')['DAILY_YIELD'].agg('mean').reset_index()
fig = go.Figure()

fig.add_trace(go.Scatter(x=plant1_data["time"], y=plant1_data['DAILY_YIELD'],
              name='DAILY_YIELD', mode='markers',␣
 ↪hovertext=plant1_data["DATE_TIME"]))

fig.add_scatter(x=dy_mean["time"], y=dy_mean["DAILY_YIELD"],
                name='Mean Yield', line=dict(color="crimson"))

fig.update_traces(marker=dict(size=3, opacity=0.8),
                  selector=dict(mode='markers'))
fig.update_layout(title="Daily Yield",
                  xaxis_title="Time",
                  yaxis_title="Daily Yield",
                  width=900,
                  height=360, template="plotly_dark", hovermode="y unified")
```

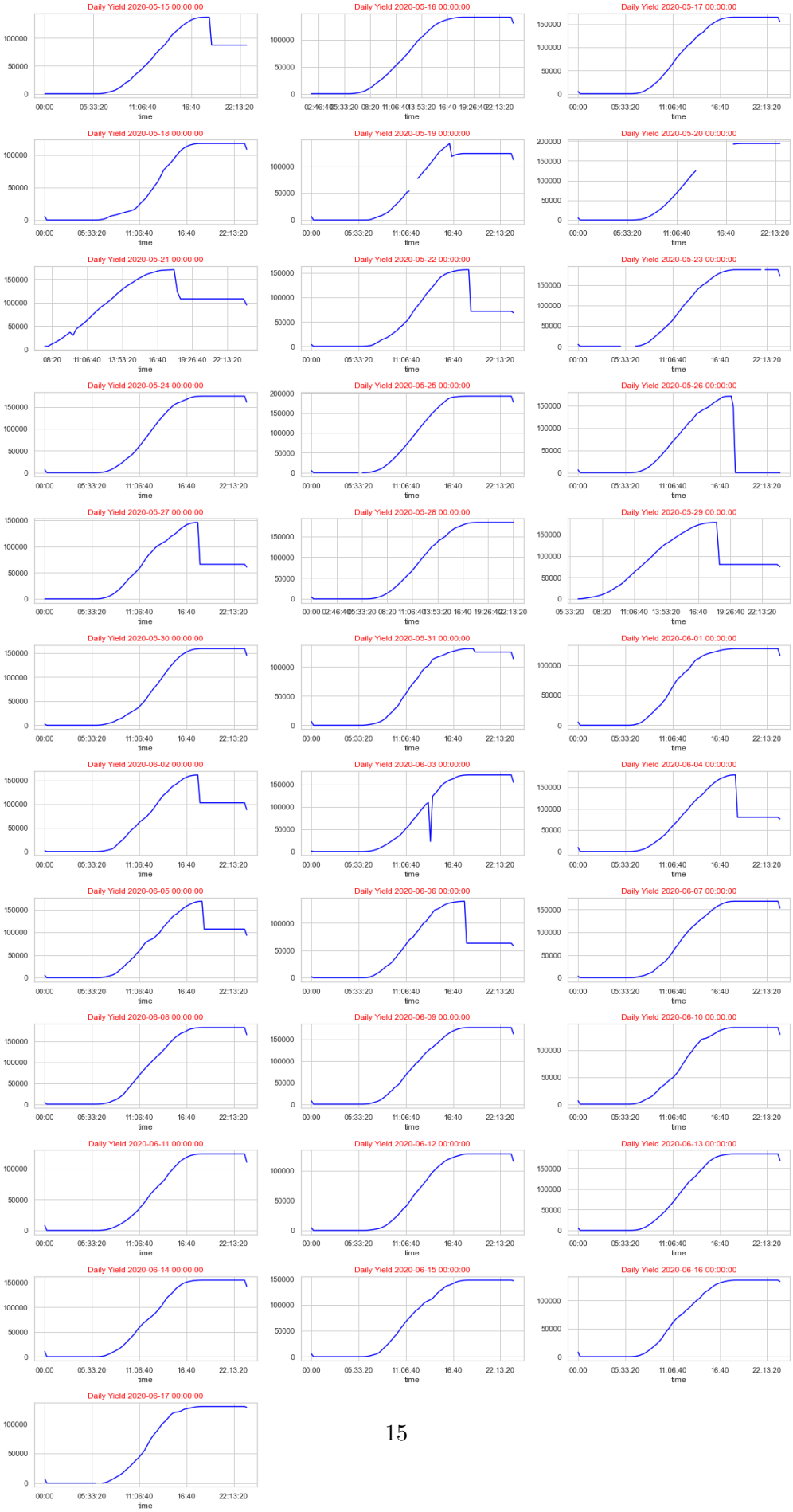
```
plotly.offline.iplot(fig)
```



The above logistics function like graph indicates that after 18:00 PM the power production decreases slowly, and usually at 00:00 AM the graph shows sudden breakdown indicating the production according to various time slots of the day.

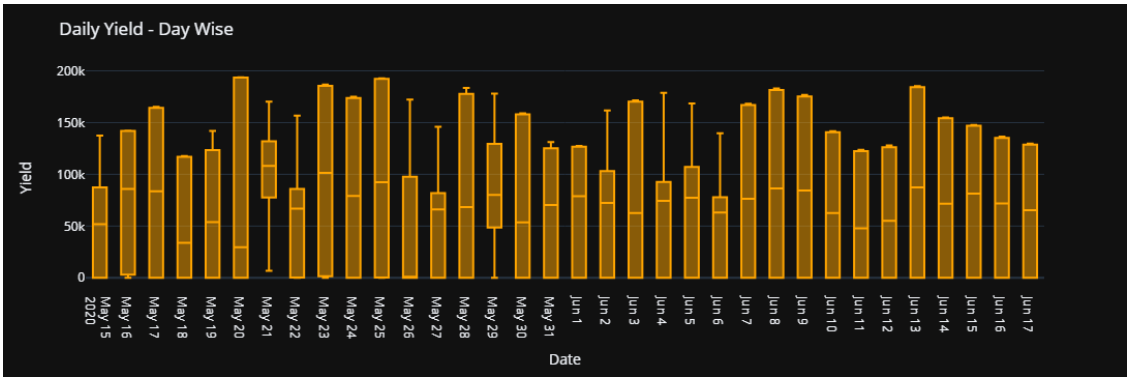
[29]:
```
# pivot table data
daily_yield = plant1_data.pivot_table(
    values='DAILY_YIELD', index='time', columns='date')
```

[30]:
```
# we plot all daily yield
Daywise_plot(data=daily_yield, row=12, col=3, title='Daily Yield')
```


Daily Yield 2020-05-15 00:00:00
Daily Yield 2020-05-16 00:00:00
Daily Yield 2020-05-17 00:00:00
Daily Yield 2020-05-18 00:00:00
Daily Yield 2020-05-19 00:00:00
Daily Yield 2020-05-20 00:00:00
Daily Yield 2020-05-21 00:00:00
Daily Yield 2020-05-22 00:00:00
Daily Yield 2020-05-23 00:00:00
Daily Yield 2020-05-24 00:00:00
Daily Yield 2020-05-25 00:00:00
Daily Yield 2020-05-26 00:00:00
Daily Yield 2020-05-27 00:00:00
Daily Yield 2020-05-28 00:00:00
Daily Yield 2020-05-29 00:00:00
Daily Yield 2020-05-30 00:00:00
Daily Yield 2020-05-31 00:00:00
Daily Yield 2020-06-01 00:00:00
Daily Yield 2020-06-02 00:00:00
Daily Yield 2020-06-03 00:00:00
Daily Yield 2020-06-04 00:00:00
Daily Yield 2020-06-05 00:00:00
Daily Yield 2020-06-06 00:00:00
Daily Yield 2020-06-07 00:00:00
Daily Yield 2020-06-08 00:00:00
Daily Yield 2020-06-09 00:00:00
Daily Yield 2020-06-10 00:00:00
Daily Yield 2020-06-11 00:00:00
Daily Yield 2020-06-12 00:00:00
Daily Yield 2020-06-13 00:00:00
Daily Yield 2020-06-14 00:00:00
Daily Yield 2020-06-15 00:00:00
Daily Yield 2020-06-16 00:00:00
Daily Yield 2020-06-17 00:00:00
time

As we can see some daily_yield date May 19,20,23,25 and June 17 have a logistic shape with missing values but others have not.

[31]:
```
fig = px.box(daily_yield)
fig.update_layout(title="Daily Yield - Day Wise",
                  xaxis_title="Date",
                  yaxis_title="Yield", template="plotly_dark")
fig.update_traces(marker=dict(opacity=1), marker_color='orange')
fig.update_xaxes(
    dtick="d1",
)
plotly.offline.iplot(fig)
```



For each day, the daily yield change. some day is high. The observation of all boxes is good, outliers does not exist.

[32]:
```
# we compute a daily yield for each date.
dyield = plant1_data.groupby('date')['DAILY_YIELD'].agg('sum').reset_index()
```

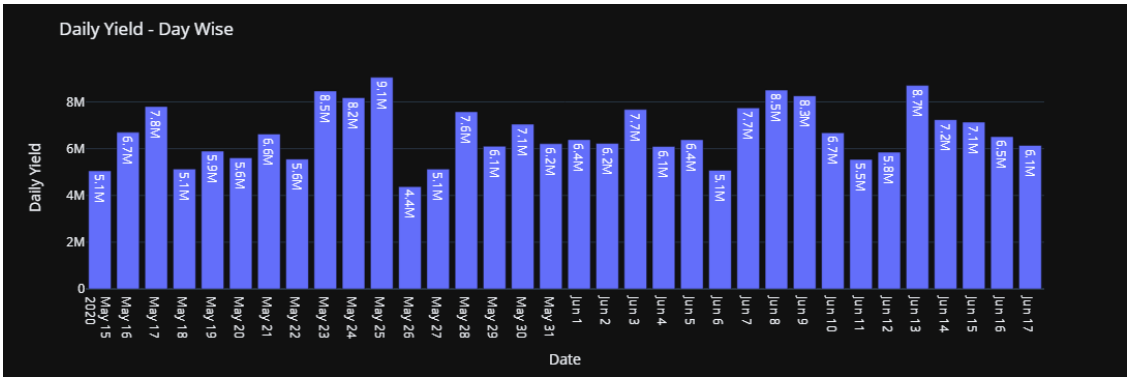
[33]:
```
dyield.head()
```

[33]:

	date	DAILY_YIELD
0	2020-05-15	5.053591e+06
1	2020-05-16	6.699905e+06
2	2020-05-17	7.804065e+06
3	2020-05-18	5.130290e+06
4	2020-05-19	5.896321e+06

[34]:
```
fig = px.bar(data_frame=dyield, x='date', y="DAILY_YIELD", text_auto='.2s')
fig.update_layout(title="Daily Yield - Day Wise",
                  xaxis_title="Date",
                  yaxis_title="Daily Yield", template="plotly_dark")
```

```
fig.update_xaxes(
    dtick="d1")

plotly.offline.iplot(fig)
```



Highest yield of Plant 1 was recorded on 25 May.

3.2 Weather Sensor Data

Data Pre-proccessing

[35]:
```
file1 = 'Plant_1_Weather_Sensor_Data.csv'
```

[36]:
```
plant1_sensor = pd.read_csv(file1)
```

[37]:
```
plant1_sensor.head()
```

[37]:

	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

[38]:
```
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
```

[39]:
```
plant1_sensor['DATE_TIME'] = pd.to_datetime(
    plant1_sensor['DATE_TIME'], errors='coerce')
```

[40]:
```
# same work cleaning data
plant1_sensor['date'] = pd.to_datetime(
    pd.to_datetime(plant1_sensor['DATE_TIME']).dt.date)
plant1_sensor['time'] = pd.to_datetime(plant1_sensor['DATE_TIME']).dt.time


del plant1_sensor['PLANT_ID']
del plant1_sensor['SOURCE_KEY']
```

[41]:
```
plant1_sensor.describe().style.background_gradient(cmap='coolwarm')
```

[41]:
```
<pandas.io.formats.style.Styler at 0x15a83d73160>
```

3.2.1 Exploratory Data Analysis

We will explore these columns `Ambient Temperature`, `Module Temperature` and `Irradiation` using libraries used plotly. and seaborn.

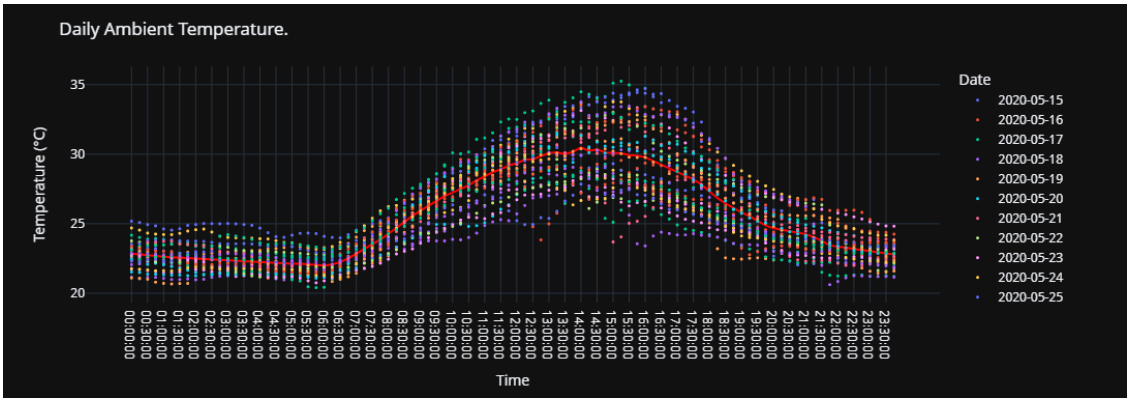
Here, we do

1. Line or scatter plot
2. %change
3. Box Plot
4. Calendar Plot
5. Bar Chart

Ambient Temperature

[42]:
```
plant1_sensor["date"] = plant1_sensor["date"].astype(str)
at_mean = plant1_sensor.groupby(
    'time')['AMBIENT_TEMPERATURE'].agg('mean').reset_index()

fig = px.scatter(data_frame=plant1_sensor, x='time',
```

```
                 y='AMBIENT_TEMPERATURE', color='date')
plant1_sensor['date'] = pd.to_datetime(plant1_sensor['DATE_TIME']).dt.date
fig.add_scatter(x=at_mean["time"], y=at_mean["AMBIENT_TEMPERATURE"],
                name='Mean Temperature', line=dict(color="red"))

fig.update_traces(marker=dict(size=3, opacity=1),
                  selector=dict(mode='markers'))
fig.update_layout(title="Daily Ambient Temperature.",
                  xaxis_title="Time",
                  yaxis_title="Temperature (°C)",
                  width=960,
                  height=380, template="plotly_dark",legend_title_text="Date")
plotly.offline.iplot(fig)
```



Ambient Temperature increases from 6:15 AM and then again starts decreasing around 2:00 PM for all the day .

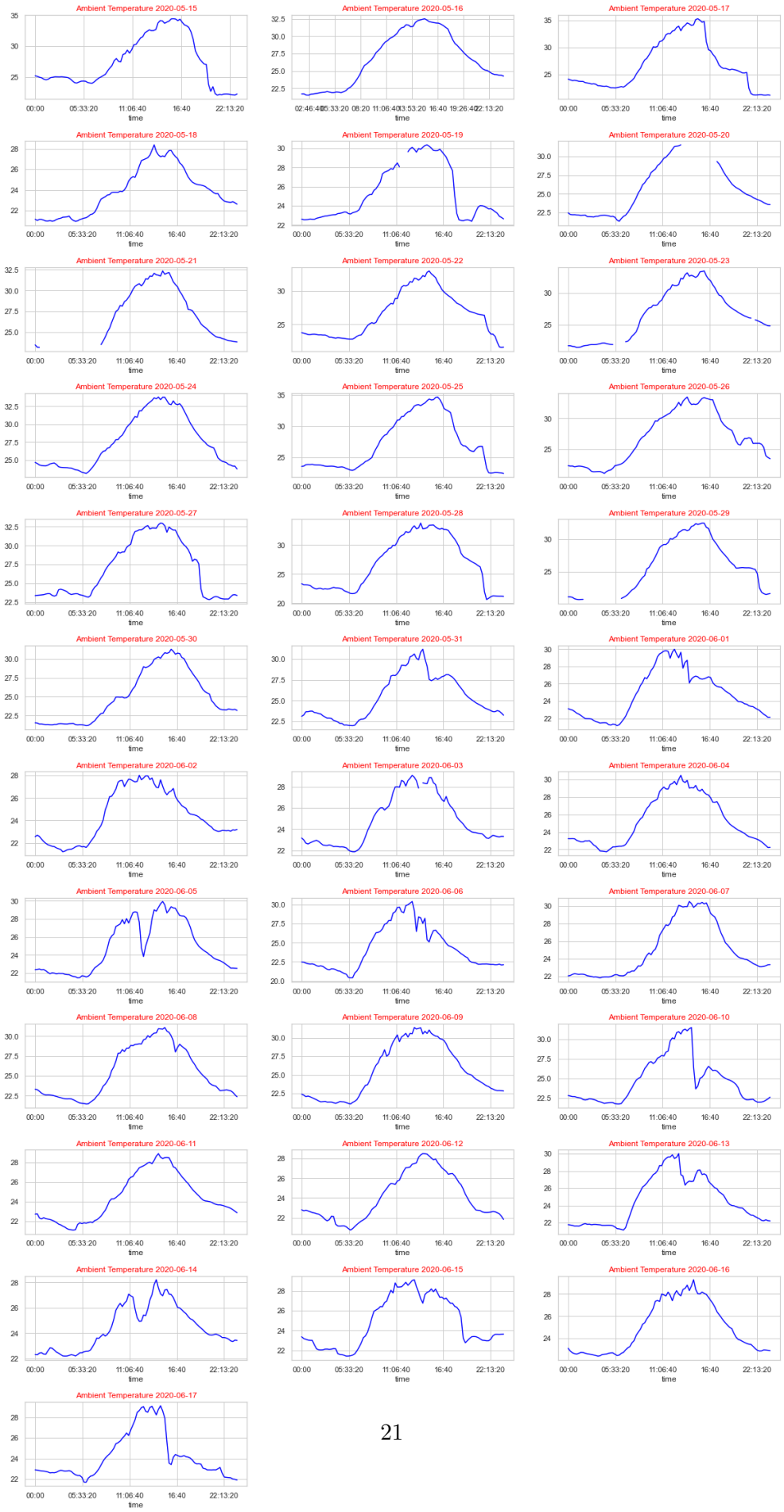
[43]:
```
ambient = plant1_sensor.pivot_table(
    values='AMBIENT_TEMPERATURE', index='time', columns='date')
```

[44]:
```
ambient.tail()
```

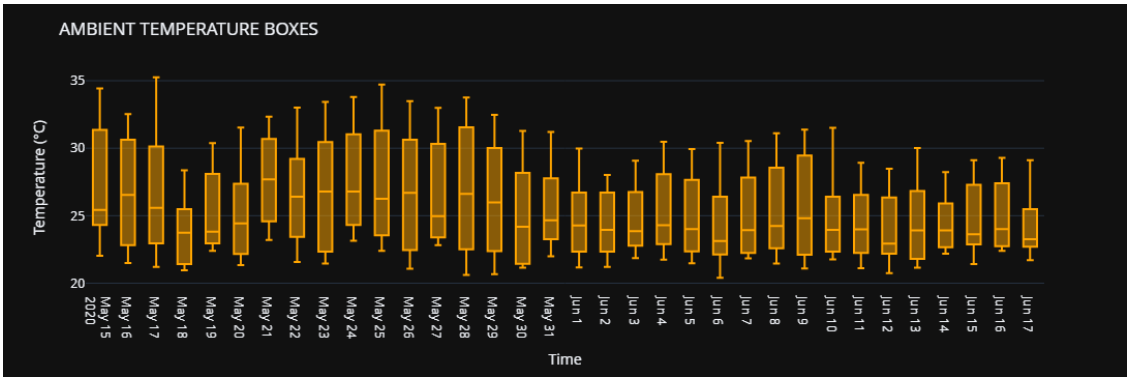
[44]:
```
date      2020-05-15  2020-05-16  2020-05-17  2020-05-18  2020-05-19  \
time
22:45:00   22.057080   24.492981   21.315892   22.815780   23.393146
23:00:00   22.236018   24.461625   21.220532   22.787469   23.230817
23:15:00         NaN   24.392568   21.216575   22.856671   22.916650
23:30:00         NaN   24.378021   21.273112   22.750576   22.787917
23:45:00         NaN   24.249347   21.209418   22.621004   22.638493

date      2020-05-20  2020-05-21  2020-05-22  2020-05-23  2020-05-24  …  \
time                                                                  …
22:45:00   24.003129   24.002574   23.214526   25.338855   24.403691  …
```

```
23:00:00   23.830852   23.978326   22.361055   25.133763   24.308990  …
23:15:00   23.701162   23.922381   21.641128   24.969698   24.175543  …
23:30:00   23.589626   23.892236   21.564903   24.838093   24.169597  …
23:45:00   23.569009   23.856805   21.605520   24.818829   23.777932  …

date      2020-06-08  2020-06-09  2020-06-10  2020-06-11  2020-06-12  \
time
22:45:00   23.170805   22.946915   22.014016   23.419015   22.599839
23:00:00   23.115568   22.883195   22.087756   23.328575   22.504606
23:15:00   22.929185   22.892314   22.247858   23.195113   22.398564
23:30:00   22.616367   22.866815   22.409064   23.027635   22.145038
23:45:00   22.369204   22.825907   22.643738   22.883351   21.820540

date      2020-06-13  2020-06-14  2020-06-15  2020-06-16  2020-06-17
time
22:45:00   22.265993   23.522461   23.612832   22.872359   22.150570
23:00:00   22.227169   23.386562   23.618848   22.962205   22.129816
23:15:00   22.334206   23.331081   23.602976   22.947974   22.008275
23:30:00   22.237394   23.444953   23.631051   22.925033   21.969495
23:45:00   22.205029   23.418154   23.641211   22.892004   21.909288

[5 rows x 34 columns]
```

[45]:
```
Daywise_plot(data=ambient, row=12, col=3, title='Ambient Temperature')
```


Ambient Temperature 2020-05-15
Ambient Temperature 2020-05-16
Ambient Temperature 2020-05-17
Ambient Temperature 2020-05-18
Ambient Temperature 2020-05-19
Ambient Temperature 2020-05-20
Ambient Temperature 2020-05-21
Ambient Temperature 2020-05-22
Ambient Temperature 2020-05-23
Ambient Temperature 2020-05-24
Ambient Temperature 2020-05-25
Ambient Temperature 2020-05-26
Ambient Temperature 2020-05-27
Ambient Temperature 2020-05-28
Ambient Temperature 2020-05-29
Ambient Temperature 2020-05-30
Ambient Temperature 2020-05-31
Ambient Temperature 2020-06-01
Ambient Temperature 2020-06-02
Ambient Temperature 2020-06-03
Ambient Temperature 2020-06-04
Ambient Temperature 2020-06-05
Ambient Temperature 2020-06-06
Ambient Temperature 2020-06-07
Ambient Temperature 2020-06-08
Ambient Temperature 2020-06-09
Ambient Temperature 2020-06-10
Ambient Temperature 2020-06-11
Ambient Temperature 2020-06-12
Ambient Temperature 2020-06-13
Ambient Temperature 2020-06-14
Ambient Temperature 2020-06-15
Ambient Temperature 2020-06-16
Ambient Temperature 2020-06-17
time

[46]:
```
fig = px.box(ambient)
fig.update_layout(title="AMBIENT TEMPERATURE BOXES",
                  xaxis_title="Time",
                  yaxis_title="Temperature (°C)", template="plotly_dark")
fig.update_traces(marker_color='orange')

fig.update_xaxes(
    dtick="d1",
)
plotly.offline.iplot(fig)
```



Which date ambient temperature mean is maximun?

[47]:
```
am_temp = plant1_sensor.groupby(
    'date')['AMBIENT_TEMPERATURE'].agg('mean').reset_index()
```

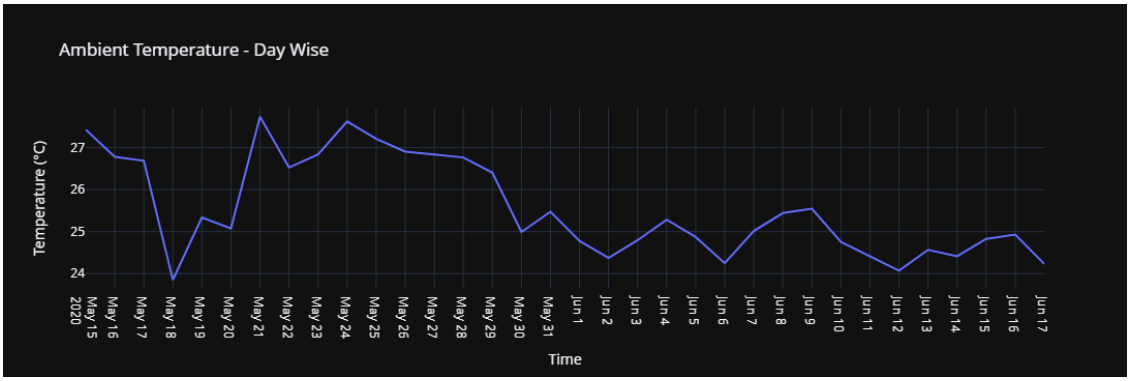
[48]:
```
am_temp.head()
```

[48]:

	date	AMBIENT_TEMPERATURE
0	2020-05-15	27.430823
1	2020-05-16	26.780538
2	2020-05-17	26.686727
3	2020-05-18	23.850938
4	2020-05-19	25.338021

[49]:
```
fig = px.line(am_temp, x="date", y="AMBIENT_TEMPERATURE",
              title='AMBIENT TEMPERATURE Day wise', template="plotly_dark")
fig.update_layout(title="Ambient Temperature - Day Wise",
                  xaxis_title="Time",
                  yaxis_title="Temperature (°C)", template="plotly_dark")
fig.update_xaxes(
```

```
    dtick="d1",
)
plotly.offline.iplot(fig)
```



Comment:

In May, ambiant Temperature in Plant 1 was between 24 and 30°C, this means that May was very hot. But in June ambiant Temperature decreases considerately between 24 and 26°C.

In the next cell, we will seek how % change of ambient Temperature is.

[50]:
```
am_temp = plant1_sensor.groupby('date')['AMBIENT_TEMPERATURE'].agg('mean')
```

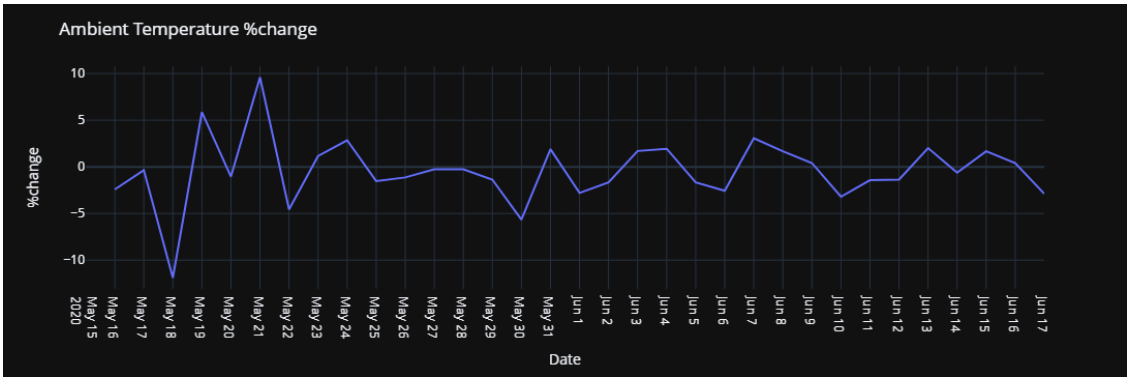
[51]:
```
am_change_temp = (am_temp.diff()/am_temp)*100
am_change_temp = am_change_temp.reset_index()
```

[52]:
```
am_change_temp.head()
```

[52]:

	date	AMBIENT_TEMPERATURE
0	2020-05-15	NaN
1	2020-05-16	-2.428198
2	2020-05-17	-0.351528
3	2020-05-18	-11.889635
4	2020-05-19	5.868980

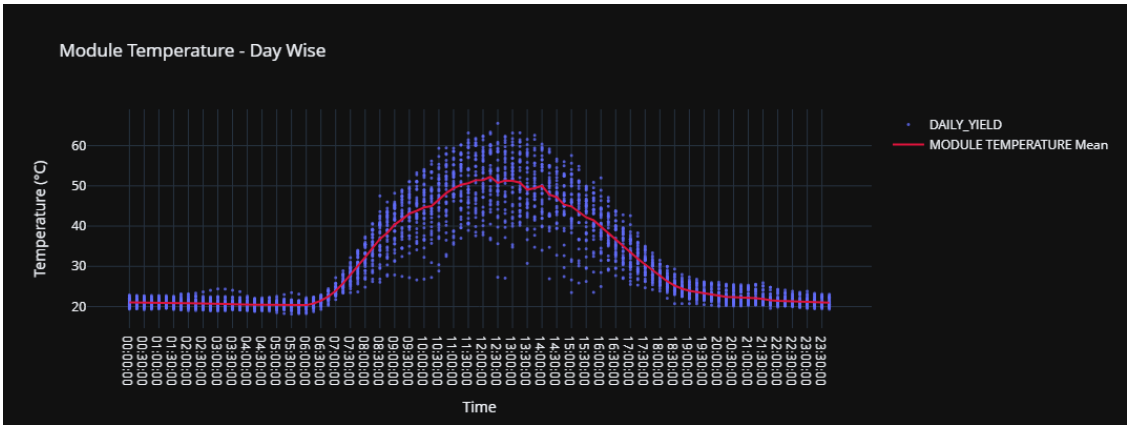
[53]:
```
fig = px.line(am_change_temp, x="date", y="AMBIENT_TEMPERATURE")
fig.update_layout(title="Ambient Temperature %change",
                  xaxis_title="Date",
                  yaxis_title="%change", template="plotly_dark")
fig.update_xaxes(
    dtick="d1",
)
plotly.offline.iplot(fig)
```



Insights : 1. Sunday 17 May 2020 to Monday 18 May 2020, the ambient Temperature decreases to 10%. 2. Monday 18 May 2020 to Tuesday 19 May 2020, the ambient Temperature increases to 15% and tomorrow decreases to 5%. 3. Wednesday 20 May 2020 to Thursday 21 May 2020, the ambient Temperature increases to 10% and tomorrow decreases to 15%. 4. June month's, the ambiant Temperature %change stabilize between -2.5 and 2.5%.

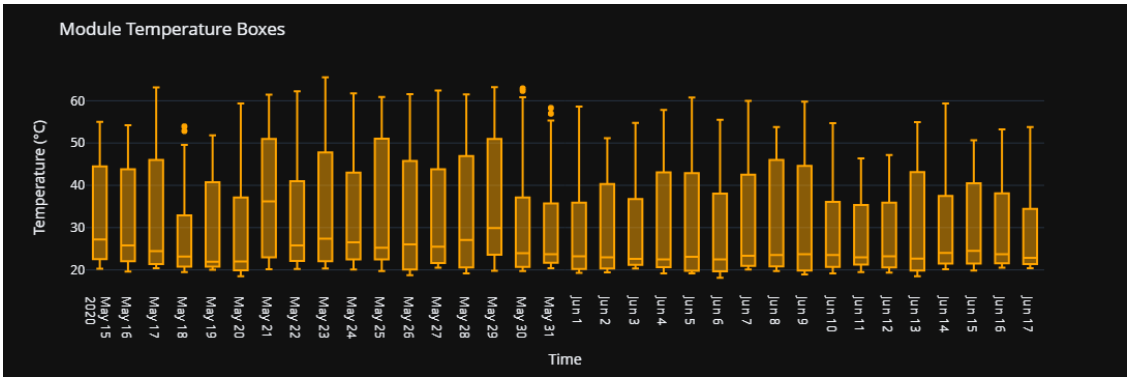
Module Temperature

[54]:
```
mt_mean = plant1_sensor.groupby(
    'time')['MODULE_TEMPERATURE'].agg('mean').reset_index()
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=plant1_sensor["time"], y=plant1_sensor['MODULE_TEMPERATURE'],␣
 ↪name='DAILY_YIELD', mode='markers'))

fig.add_scatter(x=mt_mean["time"], y=mt_mean["MODULE_TEMPERATURE"],
                name='MODULE TEMPERATURE Mean', line=dict(color="crimson"))

fig.update_traces(marker=dict(size=3, opacity=0.8),
                  selector=dict(mode='markers'))
fig.update_layout(title="Module Temperature - Day Wise",
                  xaxis_title="Time",
                  yaxis_title="Temperature (°C)",
                  width=1000,
                  height=400, template="plotly_dark")
plotly.offline.iplot(fig)
```



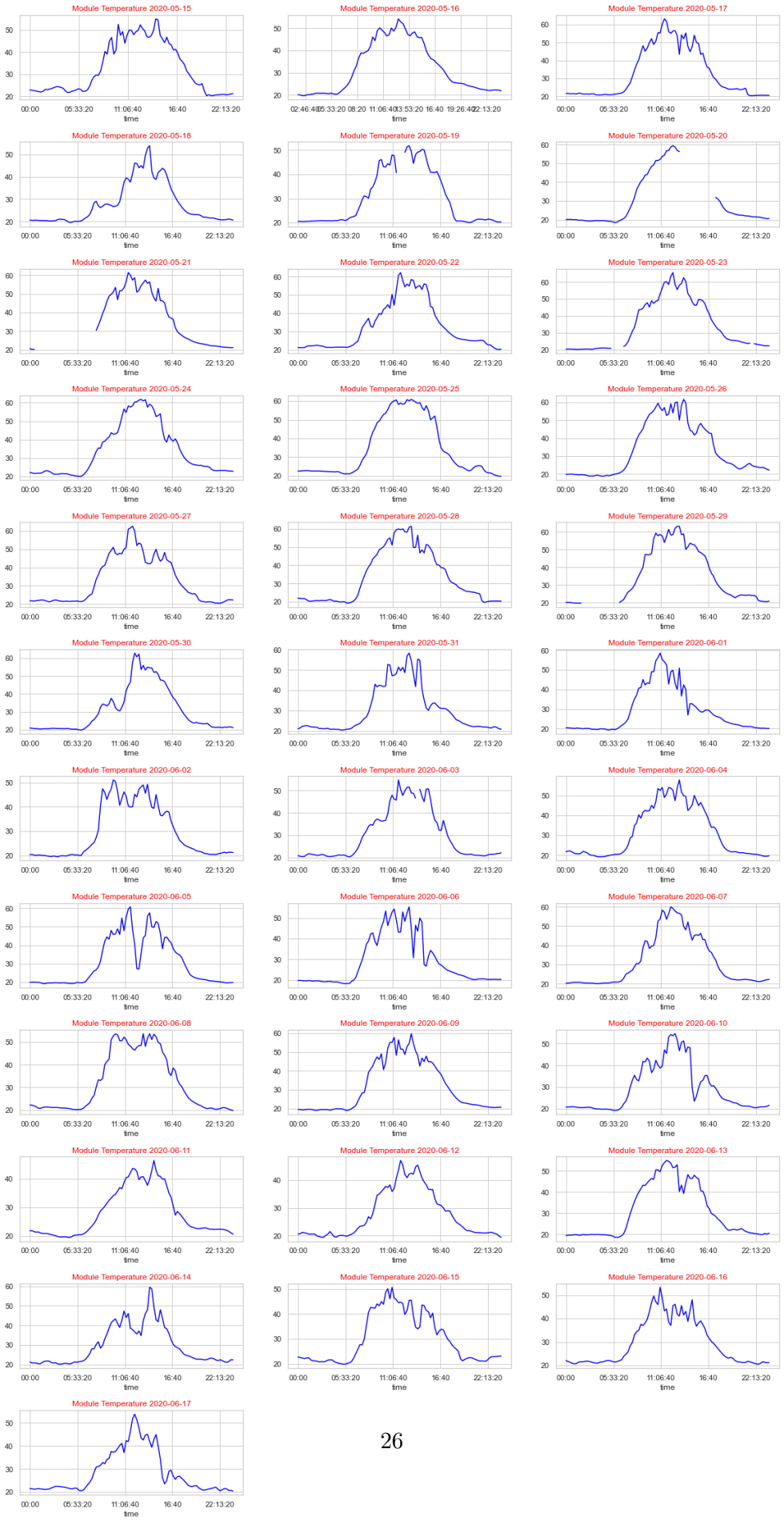
[55]:
```
module_temp = plant1_sensor.pivot_table(
    values='MODULE_TEMPERATURE', index='time', columns='date')
```

[56]:
```
fig = px.box(module_temp)
fig.update_layout(title="Module Temperature Boxes",
                  xaxis_title="Time",
                  yaxis_title="Temperature (°C)", template="plotly_dark")
fig.update_traces(marker_color='orange')
fig.update_xaxes(
    dtick="d1",
)
plotly.offline.iplot(fig)
```



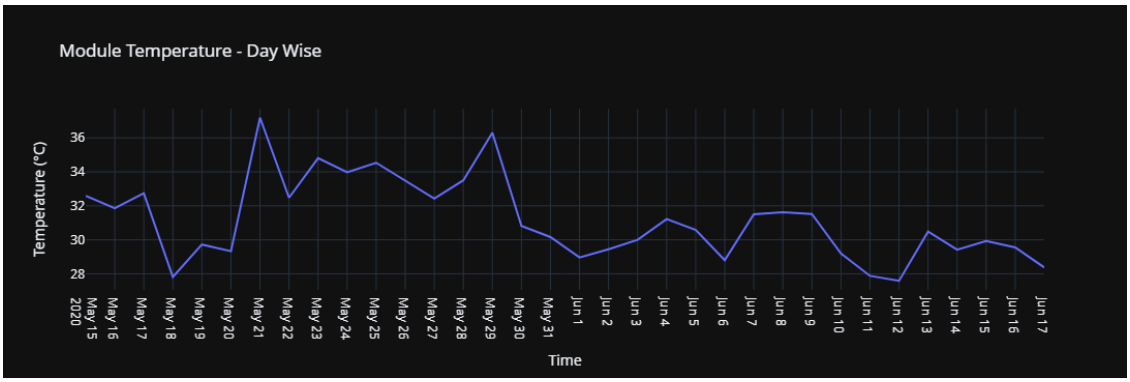
Insight: Four dates contains outliers: 18-05-2020, 30-05-2020, 31-05-2020, 01-06-2020. The outlier of these 3 dates occurs precisely at interval time [11:06:40, 16:40].

[57]:
```
Daywise_plot(data=module_temp, row=12, col=3, title="Module Temperature")
```


Module Temperature 2020-05-15
Module Temperature 2020-05-16
Module Temperature 2020-05-17
Module Temperature 2020-05-18
Module Temperature 2020-05-19
Module Temperature 2020-05-20
Module Temperature 2020-05-21
Module Temperature 2020-05-22
Module Temperature 2020-05-23
Module Temperature 2020-05-24
Module Temperature 2020-05-25
Module Temperature 2020-05-26
Module Temperature 2020-05-27
Module Temperature 2020-05-28
Module Temperature 2020-05-29
Module Temperature 2020-05-30
Module Temperature 2020-05-31
Module Temperature 2020-06-01
Module Temperature 2020-06-02
Module Temperature 2020-06-03
Module Temperature 2020-06-04
Module Temperature 2020-06-05
Module Temperature 2020-06-06
Module Temperature 2020-06-07
Module Temperature 2020-06-08
Module Temperature 2020-06-09
Module Temperature 2020-06-10
Module Temperature 2020-06-11
Module Temperature 2020-06-12
Module Temperature 2020-06-13
Module Temperature 2020-06-14
Module Temperature 2020-06-15
Module Temperature 2020-06-16
Module Temperature 2020-06-17
time

[58]:
```
# we can also see also calendar plot
mod_temp = plant1_sensor.groupby(
    'date')['MODULE_TEMPERATURE'].agg('mean').reset_index()
```

[59]:
```
fig = px.line(mod_temp, x="date", y="MODULE_TEMPERATURE",
              title='MODULE TEMPERATURE Day Wise')
fig.update_xaxes(
    dtick="d1",
)
fig.update_layout(title="Module Temperature - Day Wise",
                  xaxis_title="Time",
                  yaxis_title="Temperature (°C)", template="plotly_dark")
plotly.offline.iplot(fig)
```

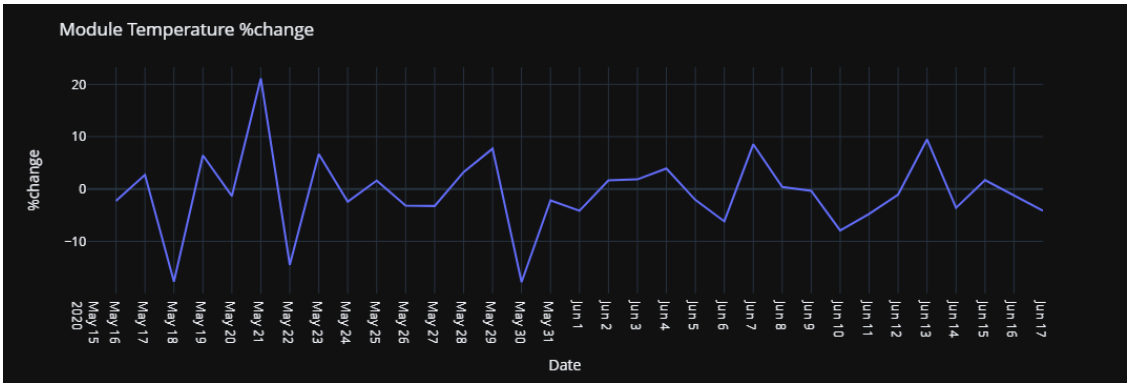


Insigths May month's have: 2 huges hot date 21 and 29.

[60]:
```
mod_temp = plant1_sensor.groupby('date')['MODULE_TEMPERATURE'].agg('mean')
```

[61]:
```
# we plot a %change of MODULE TEMPERATURE.
chan_mod_temp = (mod_temp.diff()/mod_temp)*100
chan_mod_temp = chan_mod_temp.reset_index()
```

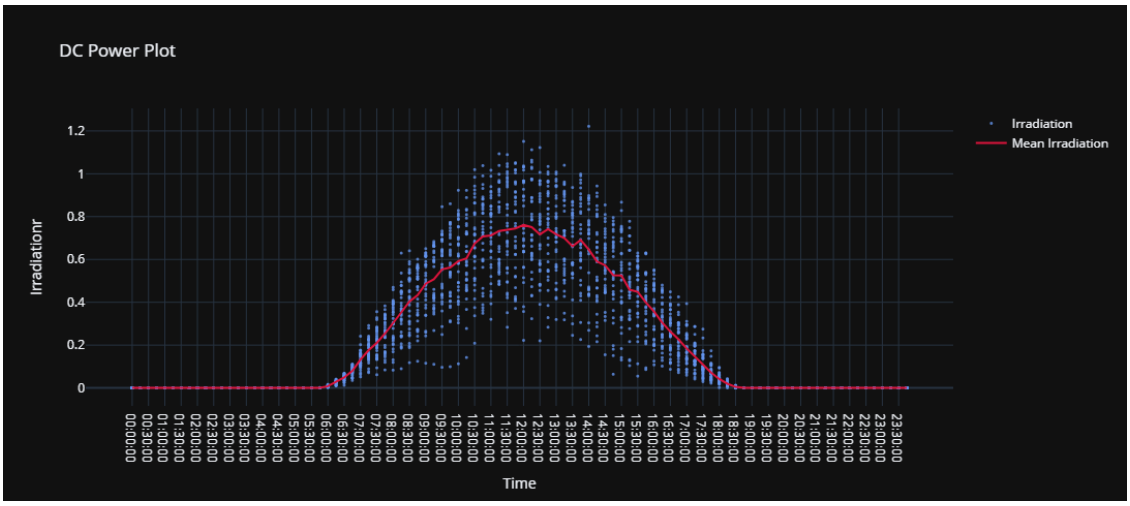
[62]:
```
fig = px.line(chan_mod_temp, x="date", y="MODULE_TEMPERATURE")
fig.update_layout(title="Module Temperature %change",
                  xaxis_title="Date",
                  yaxis_title="%change", template="plotly_dark")
fig.update_xaxes(
    dtick="d1",
)
plotly.offline.iplot(fig)
```



Insights : 1. Sunday 17 May 2020 to Monday 18 May 2020, the ambient Temperature decreases to 20%. 2. Monday 18 May 2020 to Tuesday 19 May 2020, the ambient Temperature increases to 24% and tomorrow decreases to 8%. 3. Wednesday 20 May 2020 to Thursday 21 May 2020, the ambient Temperature increases to 33% and tomorrow decreases to 35%. 4. June month's, the ambiant Temperature %change stabilize between -5 and 7% (mean).

Irradiation

[63]:
```
i_mean = plant1_sensor.groupby('time')['IRRADIATION'].agg('mean').reset_index()
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=plant1_sensor["time"], y=plant1_sensor['IRRADIATION'],␣
 ↪name='Irradiation', mode='markers'))
fig.update_traces(marker=dict(color="#6495ED", size=3,
                  opacity=0.8), selector=dict(mode='markers'))
fig.add_scatter(x=i_mean["time"], y=i_mean["IRRADIATION"],
                name='Mean Irradiation', line=dict(color="crimson"))
fig.update_layout(title="DC Power Plot",
                  xaxis_title="Time",
                  yaxis_title="Irradiationr", template="plotly_dark",
                  width=960,
                  height=480)

plotly.offline.iplot(fig)
```



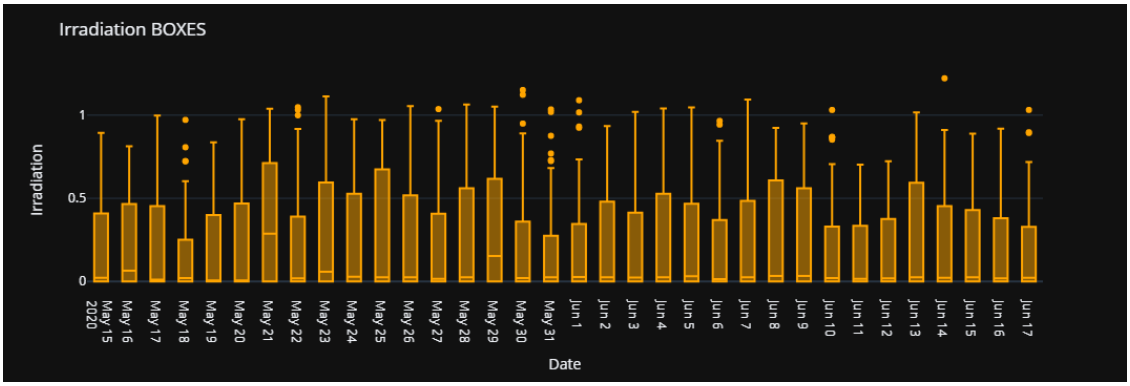
[64]:
```
irra = plant1_sensor.pivot_table(
    values='IRRADIATION', index='time', columns='date')
```

[65]:
```
irra.tail()
```

[65]:
```
date      2020-05-15  2020-05-16  2020-05-17  2020-05-18  2020-05-19  \
time
22:45:00         0.0         0.0         0.0         0.0         0.0
23:00:00         0.0         0.0         0.0         0.0         0.0
23:15:00         NaN         0.0         0.0         0.0         0.0
23:30:00         NaN         0.0         0.0         0.0         0.0
23:45:00         NaN         0.0         0.0         0.0         0.0

date      2020-05-20  2020-05-21  2020-05-22  2020-05-23  2020-05-24  …  \
time                                                                  …
22:45:00         0.0         0.0         0.0         0.0         0.0  …
23:00:00         0.0         0.0         0.0         0.0         0.0  …
23:15:00         0.0         0.0         0.0         0.0         0.0  …
23:30:00         0.0         0.0         0.0         0.0         0.0  …
23:45:00         0.0         0.0         0.0         0.0         0.0  …

date      2020-06-08  2020-06-09  2020-06-10  2020-06-11  2020-06-12  \
time
22:45:00         0.0         0.0         0.0         0.0         0.0
23:00:00         0.0         0.0         0.0         0.0         0.0
23:15:00         0.0         0.0         0.0         0.0         0.0
23:30:00         0.0         0.0         0.0         0.0         0.0
23:45:00         0.0         0.0         0.0         0.0         0.0

date      2020-06-13  2020-06-14  2020-06-15  2020-06-16  2020-06-17
```

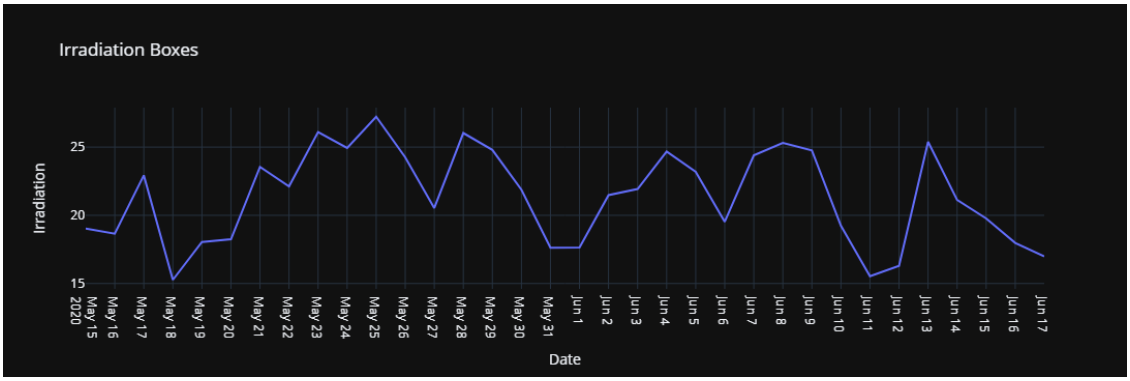
```
time
22:45:00          0.0          0.0          0.0          0.0          0.0
23:00:00          0.0          0.0          0.0          0.0          0.0
23:15:00          0.0          0.0          0.0          0.0          0.0
23:30:00          0.0          0.0          0.0          0.0          0.0
23:45:00          0.0          0.0          0.0          0.0          0.0

[5 rows x 34 columns]
```

[66]:
```
fig = px.box(irra)
fig.update_layout(title="Irradiation BOXES",
                  xaxis_title="Date",
                  yaxis_title="Irradiation", template="plotly_dark")
fig.update_traces(marker_color='orange')
fig.update_xaxes(
    dtick="d1",
)
plotly.offline.iplot(fig)
```



[67]:
```
rad = plant1_sensor.groupby('date')['IRRADIATION'].agg('sum').reset_index()
```

[68]:
```
fig = px.line(rad, x="date", y="IRRADIATION",
              title='IRRADIATION TEMPERATURE Day Wise')
fig.update_layout(title="Irradiation Boxes",
                  xaxis_title="Date",
                  yaxis_title="Irradiation", template="plotly_dark")
fig.update_xaxes(
    dtick="d1",
)
plotly.offline.iplot(fig)
```



[ ]:

Insights : Thursday 21 May 2020 is a date where plant 1: 1. ambient temperature, module temperature are maximun.

3.3 Correlation

In this part, we are making correlation between feature to see how some feature can explain another feature. or see relation between them.

[69]:
```
# we are merge our solar power generation data and weather sensor data
power_sensor = plant1_sensor.merge(
    plant1_data, left_on='DATE_TIME', right_on='DATE_TIME')
```

[70]:
```
power_sensor.tail(3)
```

[70]:
```
              DATE_TIME  AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  \
3154 2020-06-17 23:15:00            22.008275           20.709211
3155 2020-06-17 23:30:00            21.969495           20.734963
3156 2020-06-17 23:45:00            21.909288           20.427972

      IRRADIATION      date_x    time_x  DC_POWER  AC_POWER    DAILY_YIELD  \
3154          0.0  2020-06-17  23:15:00       0.0       0.0  129571.000000
3155          0.0  2020-06-17  23:30:00       0.0       0.0  129571.000000
3156          0.0  2020-06-17  23:45:00       0.0       0.0  127962.767857

      TOTAL_YIELD    time_y     date_y
3154  156142755.0  23:15:00 2020-06-17
3155  156142755.0  23:30:00 2020-06-17
3156  156142755.0  23:45:00 2020-06-17
```

[71]:
```
# we remove the columns that we do not need
del power_sensor['date_x']
del power_sensor['date_y']
del power_sensor['time_x']
```

```
del power_sensor['time_y']
```

[72]:
```
power_sensor.tail(3)
```

[72]:
```
                DATE_TIME  AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  \
3154 2020-06-17 23:15:00            22.008275           20.709211
3155 2020-06-17 23:30:00            21.969495           20.734963
3156 2020-06-17 23:45:00            21.909288           20.427972

      IRRADIATION  DC_POWER  AC_POWER    DAILY_YIELD  TOTAL_YIELD
3154          0.0       0.0       0.0  129571.000000  156142755.0
3155          0.0       0.0       0.0  129571.000000  156142755.0
3156          0.0       0.0       0.0  127962.767857  156142755.0
```

[73]:
```
power_sensor.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3157 entries, 0 to 3156
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   DATE_TIME            3157 non-null   datetime64[ns]
 1   AMBIENT_TEMPERATURE  3157 non-null   float64
 2   MODULE_TEMPERATURE   3157 non-null   float64
 3   IRRADIATION          3157 non-null   float64
 4   DC_POWER             3157 non-null   float64
 5   AC_POWER             3157 non-null   float64
 6   DAILY_YIELD          3157 non-null   float64
 7   TOTAL_YIELD          3157 non-null   float64
dtypes: datetime64[ns](1), float64(7)
memory usage: 222.0 KB
```

[74]:
```
# we start correlation
power_sensor.corr(method='spearman')
```

[74]:
```
                     AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  IRRADIATION  \
AMBIENT_TEMPERATURE             1.000000            0.909535     0.742414
MODULE_TEMPERATURE              0.909535            1.000000     0.899598
IRRADIATION                     0.742414            0.899598     1.000000
DC_POWER                        0.742144            0.901076     0.991855
AC_POWER                        0.742110            0.901058     0.991850
DAILY_YIELD                     0.569439            0.398906     0.213424
TOTAL_YIELD                    -0.181517           -0.082369    -0.003114

                     DC_POWER  AC_POWER  DAILY_YIELD  TOTAL_YIELD
AMBIENT_TEMPERATURE  0.742144  0.742110     0.569439    -0.181517
MODULE_TEMPERATURE   0.901076  0.901058     0.398906    -0.082369
IRRADIATION          0.991855  0.991850     0.213424    -0.003114
```

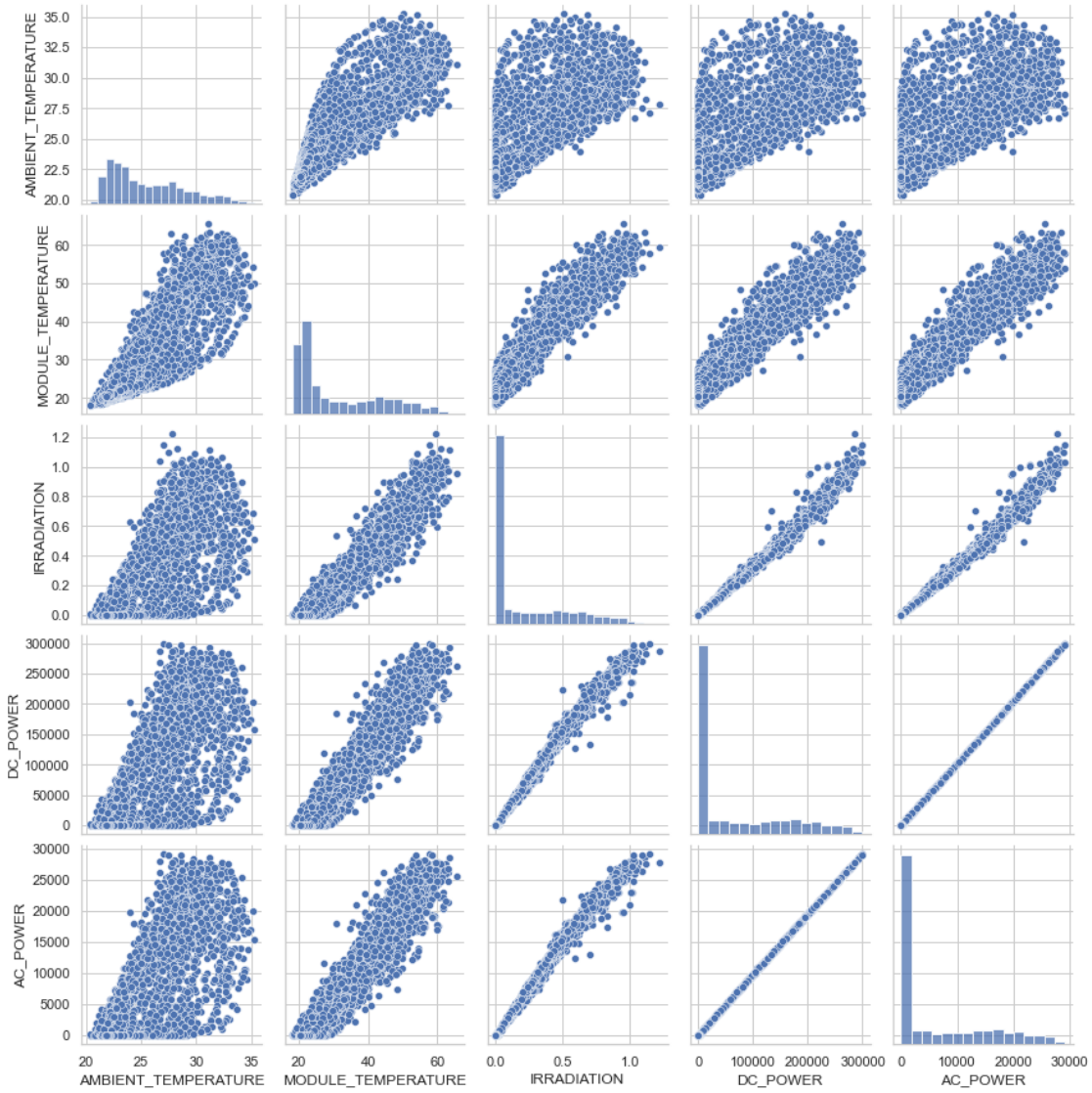
DC_POWER	1.000000	1.000000	0.215463	0.004527
AC_POWER	1.000000	1.000000	0.215442	0.004526
DAILY_YIELD	0.215463	0.215442	1.000000	0.028781
TOTAL_YIELD	0.004527	0.004526	0.028781	1.000000

Insights Daily yield is not correlated with all feature but Ambient Temperature is moreless correlated. Daily yield is also not correlated with all feature. I remove it in the correlation matrix.

[75]:
```
corr = power_sensor.drop(
    columns=['DAILY_YIELD', 'TOTAL_YIELD']).corr(method='spearman')
```

[76]:
```
corrMatrix = corr.corr()
fig = px.imshow(corrMatrix, text_auto=True, aspect="auto")
fig.update_xaxes(side="bottom")
plotly.offline.iplot(fig)
```

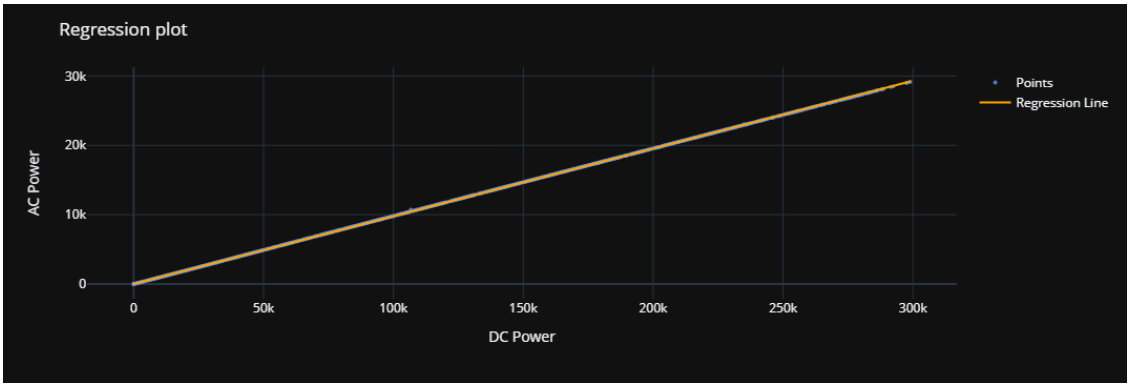
[77]:
```
# we make pairplot
sns.pairplot(power_sensor.drop(columns=['DAILY_YIELD', 'TOTAL_YIELD']))
plt.show()
```



[78]:
```
# we plot dc power vs ac power
```

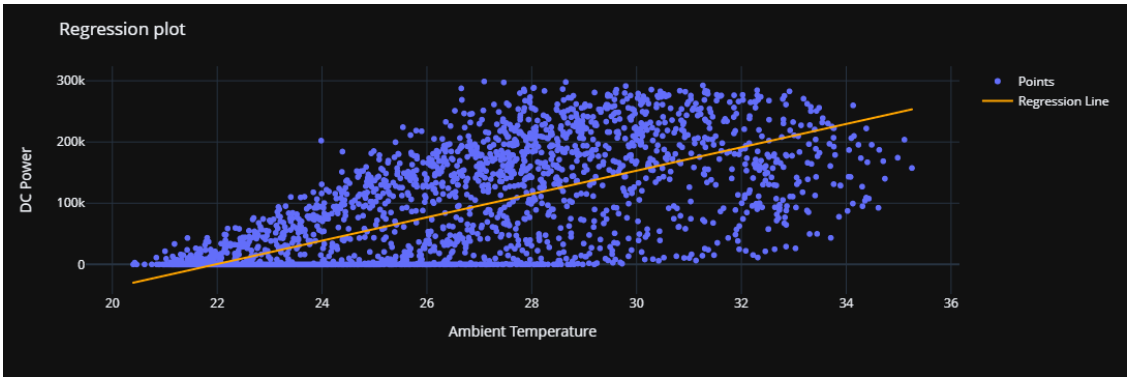
[79]:
```
fig = px.scatter(power_sensor, x="DC_POWER", y="AC_POWER", trendline="ols",
                 trendline_scope="trace", trendline_color_override="orange")
fig.update_traces(marker=dict(color="#6495ED", size=4,
                  opacity=0.8), selector=dict(mode='markers'))
fig.update_layout(title="Regression plot",
                  xaxis_title="DC Power",
                  yaxis_title="AC Power", template="plotly_dark")

fig['data'][0]['showlegend'] = True
fig['data'][0]['name'] = 'Points'
fig['data'][1]['showlegend'] = True
```

```
fig['data'][1]['name'] = 'Regression Line'


plotly.offline.iplot(fig)
```



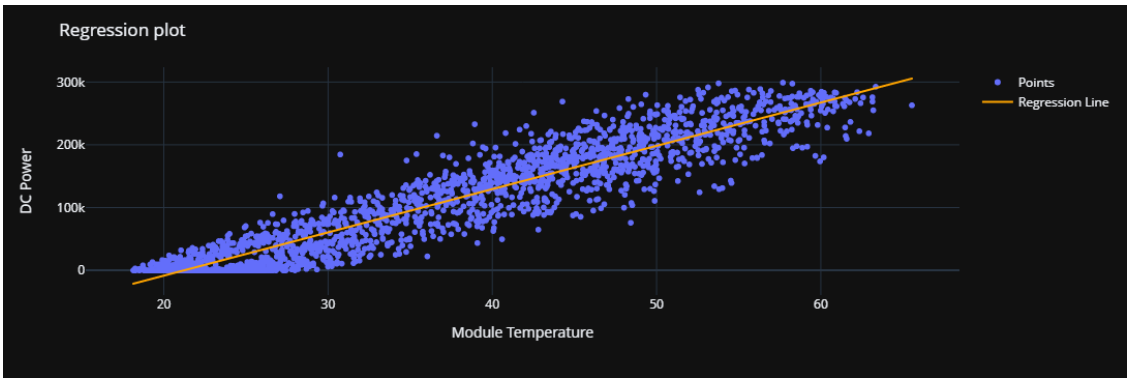
Insights This graph said that inverter convert dc power to ac power linearly. $dcpower = 10 * acpower$ inverter lost 90% of their power when it convert.

[80]:
```
fig = px.scatter(power_sensor, x="AMBIENT_TEMPERATURE", y="DC_POWER",
                 trendline="ols", trendline_color_override="orange")
fig.update_layout(title="Regression plot",
                  xaxis_title="Ambient Temperature",
                  yaxis_title="DC Power", template="plotly_dark")

fig['data'][0]['showlegend'] = True
fig['data'][0]['name'] = 'Points'
fig['data'][1]['showlegend'] = True
fig['data'][1]['name'] = 'Regression Line'

plotly.offline.iplot(fig)
```



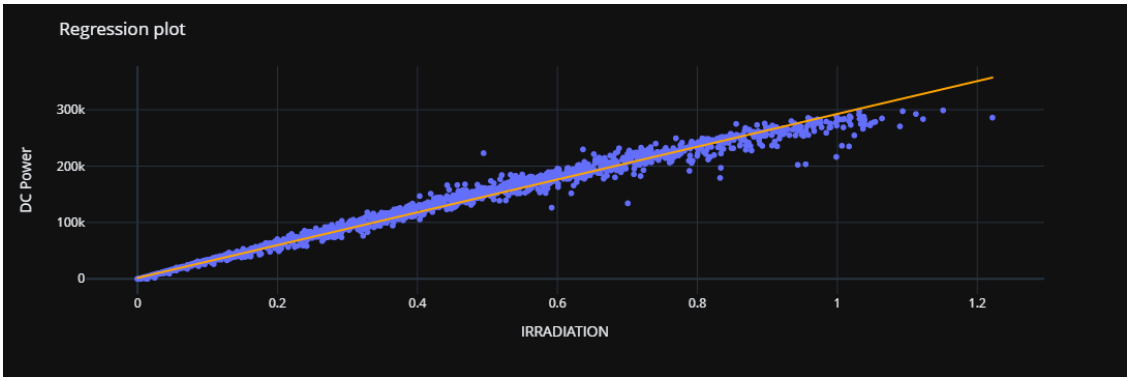
DC_power increases non linearly with an Ambient_Temperature.

[81]:
```
fig = px.scatter(power_sensor, x="MODULE_TEMPERATURE", y="DC_POWER",
                 trendline="ols", trendline_color_override="orange")
fig.update_layout(title="Regression plot",
                  xaxis_title="Module Temperature",
                  yaxis_title="DC Power", template="plotly_dark")

fig['data'][0]['showlegend'] = True
fig['data'][0]['name'] = 'Points'
fig['data'][1]['showlegend'] = True
fig['data'][1]['name'] = 'Regression Line'


plotly.offline.iplot(fig)
```



DC_POWER is produced linearly by `MODULE_TEMPERATURE` with some variability.

[82]:
```
fig = px.scatter(power_sensor, x="IRRADIATION", y="DC_POWER",
                 trendline="ols", trendline_color_override="orange")
fig.update_layout(title="Regression plot",
                  xaxis_title="IRRADIATION",
                  yaxis_title="DC Power", template="plotly_dark")
plotly.offline.iplot(fig)
```



DC Power increase with Irradiation.

What happens if I introuduce a difference Temperature between Ambient temperature and Module temperature?

[83]:
```
# we introduce DELTA_TEMPERATURE
power_sensor['DELTA_TEMPERATURE'] = abs(
    power_sensor.AMBIENT_TEMPERATURE - power_sensor.MODULE_TEMPERATURE)
```

[84]:
```
# we check if all is ok
power_sensor.tail(3)
```

[84]:
```
                DATE_TIME  AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  \
3154 2020-06-17 23:15:00            22.008275           20.709211
3155 2020-06-17 23:30:00            21.969495           20.734963
3156 2020-06-17 23:45:00            21.909288           20.427972

      IRRADIATION  DC_POWER  AC_POWER    DAILY_YIELD  TOTAL_YIELD  \
3154          0.0       0.0       0.0  129571.000000  156142755.0
3155          0.0       0.0       0.0  129571.000000  156142755.0
3156          0.0       0.0       0.0  127962.767857  156142755.0

      DELTA_TEMPERATURE
3154           1.299063
3155           1.234532
3156           1.481315
```

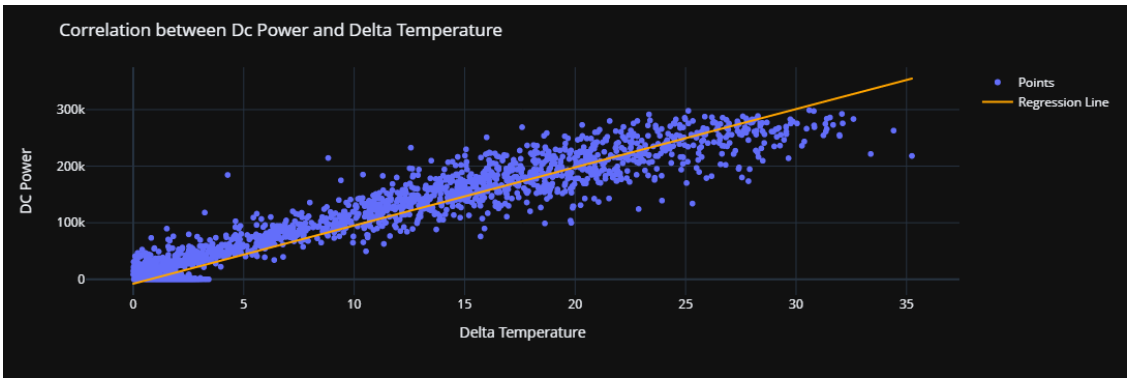
[85]:
```
# now we use correlation
power_sensor.corr(method='spearman')['DELTA_TEMPERATURE']
```

[85]:
```
AMBIENT_TEMPERATURE    0.674240
MODULE_TEMPERATURE     0.734092
IRRADIATION            0.785624
DC_POWER               0.794645
AC_POWER               0.794631
```

```
DAILY_YIELD            0.190184
TOTAL_YIELD           -0.051676
DELTA_TEMPERATURE      1.000000
Name: DELTA_TEMPERATURE, dtype: float64
```

We remark that yield does not depend on Delta Temperature also.

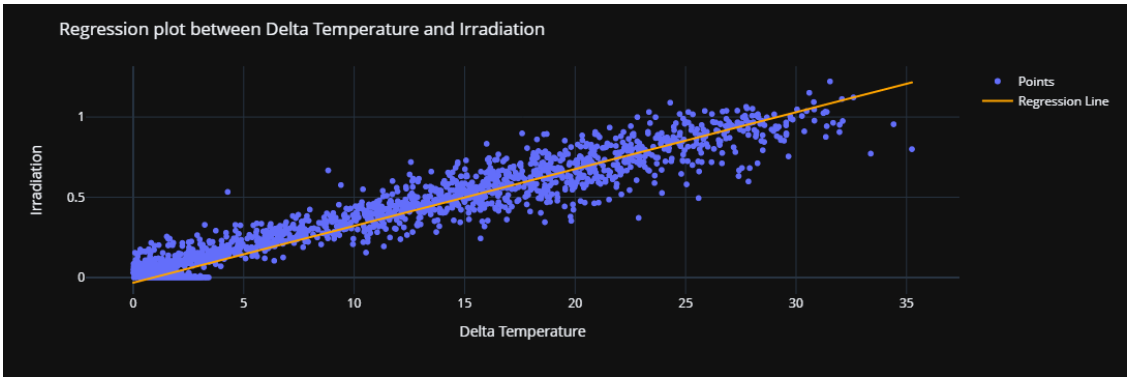
[86]:
```
fig = px.scatter(power_sensor, x="DELTA_TEMPERATURE", y="DC_POWER",
                 trendline="ols", trendline_color_override="orange")
fig.update_layout(title="Correlation between Dc Power and Delta Temperature",
                  xaxis_title="Delta Temperature",
                  yaxis_title="DC Power", template="plotly_dark")

fig['data'][0]['showlegend'] = True
fig['data'][0]['name'] = 'Points'
fig['data'][1]['showlegend'] = True
fig['data'][1]['name'] = 'Regression Line'

plotly.offline.iplot(fig)
```



We know that $\dot{Q} \propto \Delta T$.So, we could say that DC Power is influenced by heat transfer.

[87]:
```
fig = px.scatter(power_sensor, x="DELTA_TEMPERATURE", y="IRRADIATION",
                 trendline="ols", trendline_color_override="orange")
fig.update_layout(title="Regression plot between Delta Temperature and␣
 ↪Irradiation",
                  xaxis_title="Delta Temperature",
                  yaxis_title="Irradiation",
                  template="plotly_dark")

fig['data'][0]['showlegend'] = True
fig['data'][0]['name'] = 'Points'
fig['data'][1]['showlegend'] = True
fig['data'][1]['name'] = 'Regression Line'
```

```
plotly.offline.iplot(fig)
```



Irradiation of Module and Heat Transfert between ambient air and Module are very well correlated.

Short conclusion In this section, we conclude that: 1. Yield does not depend on the Temperature, the dc/ac power and irradiation. 2. the transfert function between dc and ac power is linear. 3. dc power is indeed influenced by the ambient temperature, by the temperature of the module, by the irradiation and finally by the heat transfer between the module and the air. 4. all 22 Inverters of Plant I lost 90% of their dc power when it convert.

3.4 Machine Leaning

3.4.1 2.1 Check for issues

[88]:
```
plant1_data = pd.read_csv("Plant_1_Generation_Data.csv")
plant2_data = pd.read_csv("Plant_2_Generation_Data.csv")

plant1_sensor = pd.read_csv("Plant_1_Weather_Sensor_Data.csv")
plant2_sensor = pd.read_csv("Plant_2_Weather_Sensor_Data.csv")
```

[89]:
```
# Check missing values
plant1_data.isnull().sum().sort_values(ascending=False)
```

[89]:
```
DATE_TIME      0
PLANT_ID       0
SOURCE_KEY     0
DC_POWER       0
AC_POWER       0
DAILY_YIELD    0
TOTAL_YIELD    0
dtype: int64
```

[90]:
```
# Check missing values
plant1_data.isnull().sum().sort_values(ascending=False)
```

[90]:
```
DATE_TIME      0
PLANT_ID       0
SOURCE_KEY     0
DC_POWER       0
AC_POWER       0
DAILY_YIELD    0
TOTAL_YIELD    0
dtype: int64
```

It looks like we have no missing values. We can use the PLANT_ID column to check if our data only contains information of one power plant:

[91]:
```
plant1_data['PLANT_ID'].nunique()
```

[91]:
```
1
```

[92]:
```
plant1_sensor['PLANT_ID'].nunique()
```

[92]:
```
1
```

As expected, we only have data from one plant in this database. How many inverters are in the database and how many measurements are there per inverter?

[93]:
```
plant1_data.SOURCE_KEY.value_counts()
```

[93]:
```
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: SOURCE_KEY, dtype: int64
```

[94]:
```
plant1_data.SOURCE_KEY.value_counts().sum()
```

[94]:
```
68778
```

[95]:
```
print('There are {} different inverters. Number of measurements per inverter␣
 ↪range from {} to {}.' .format(
    plant1_data.SOURCE_KEY.nunique(), plant1_data.SOURCE_KEY.value_counts().
 ↪min(), plant1_data.SOURCE_KEY.value_counts().max()))
```

```
There are 22 different inverters. Number of measurements per inverter range from
3104 to 3155.
```

As we can see there are 22 different inverters with between 3104 and 3155 measurements. This difference may cause an issue with prediction models and should be taken into account. Since one entry corresponds to a 15 min measurement, the maximum difference of 51 entries corresponds to a difference of almost 13 hours.

3.4.2 2.2 Preprocess and merge datasets

We want to merge both datasets. To do this we adjust the DATE_TIME formats, drop unnecessary columns and merge along DATE_TIME. In addition, we are going to add separate date and time columns as well as name our inverters from 1 to 22.

[96]:
```
# adjust datetime format
plant1_data['DATE_TIME'] = pd.to_datetime(
    plant1_data['DATE_TIME'], format='%d-%m-%Y %H:%M')
plant1_sensor['DATE_TIME'] = pd.to_datetime(
    plant1_sensor['DATE_TIME'], format='%Y-%m-%d %H:%M:%S')

# drop unnecessary columns and merge both dataframes along DATE_TIME
df_plant1 = pd.merge(plant1_data.drop(columns=['PLANT_ID']), plant1_sensor.drop(
    columns=['PLANT_ID', 'SOURCE_KEY']), on='DATE_TIME')
```

[97]:
```
# add inverter number column to dataframe
sensorkeys = df_plant1.SOURCE_KEY.unique().tolist()  # unique sensor keys
sensornumbers = list(range(1, len(sensorkeys)+1))  # sensor number
# dictionary of sensor numbers and corresponding keys
dict_sensor = dict(zip(sensorkeys, sensornumbers))

# add column
df_plant1['SENSOR_NUM'] = 0
for i in range(plant1_data.shape[0]):
    df_plant1['SENSOR_NUM'][i] = dict_sensor[plant1_data["SOURCE_KEY"][i]]

# add Sensor Number as string
df_plant1["SENSOR_NAME"] = df_plant1["SENSOR_NUM"].apply(
    str)  # add string column of sensor name
```

[98]:
```
# adding separate time and date columns
df_plant1["DATE"] = pd.to_datetime(
    df_plant1["DATE_TIME"]).dt.date  # add new column with date
df_plant1["TIME"] = pd.to_datetime(
    df_plant1["DATE_TIME"]).dt.time  # add new column with time

# add hours and minutes for ml models
df_plant1['HOURS'] = pd.to_datetime(
    df_plant1['TIME'], format='%H:%M:%S').dt.hour
df_plant1['MINUTES'] = pd.to_datetime(
    df_plant1['TIME'], format='%H:%M:%S').dt.minute
df_plant1['MINUTES_PASS'] = df_plant1['MINUTES'] + df_plant1['HOURS']*60

# add date as string column
df_plant1["DATE_STR"] = df_plant1["DATE"].astype(
    str)  # add column with date as string
```

[99]:
```
df_plant1.head()
```

[99]:

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

	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION	SENSOR_NUM	\
0	25.184316	22.857507	0.0	1	
1	25.184316	22.857507	0.0	2	
2	25.184316	22.857507	0.0	3	
3	25.184316	22.857507	0.0	4	
4	25.184316	22.857507	0.0	5	

	SENSOR_NAME	DATE	TIME	HOURS	MINUTES	MINUTES_PASS	DATE_STR
0	1	2020-05-15	00:00:00	0	0	0	2020-05-15
1	2	2020-05-15	00:00:00	0	0	0	2020-05-15
2	3	2020-05-15	00:00:00	0	0	0	2020-05-15
3	4	2020-05-15	00:00:00	0	0	0	2020-05-15
4	5	2020-05-15	00:00:00	0	0	0	2020-05-15

3. Data Exploration & Failure Detection

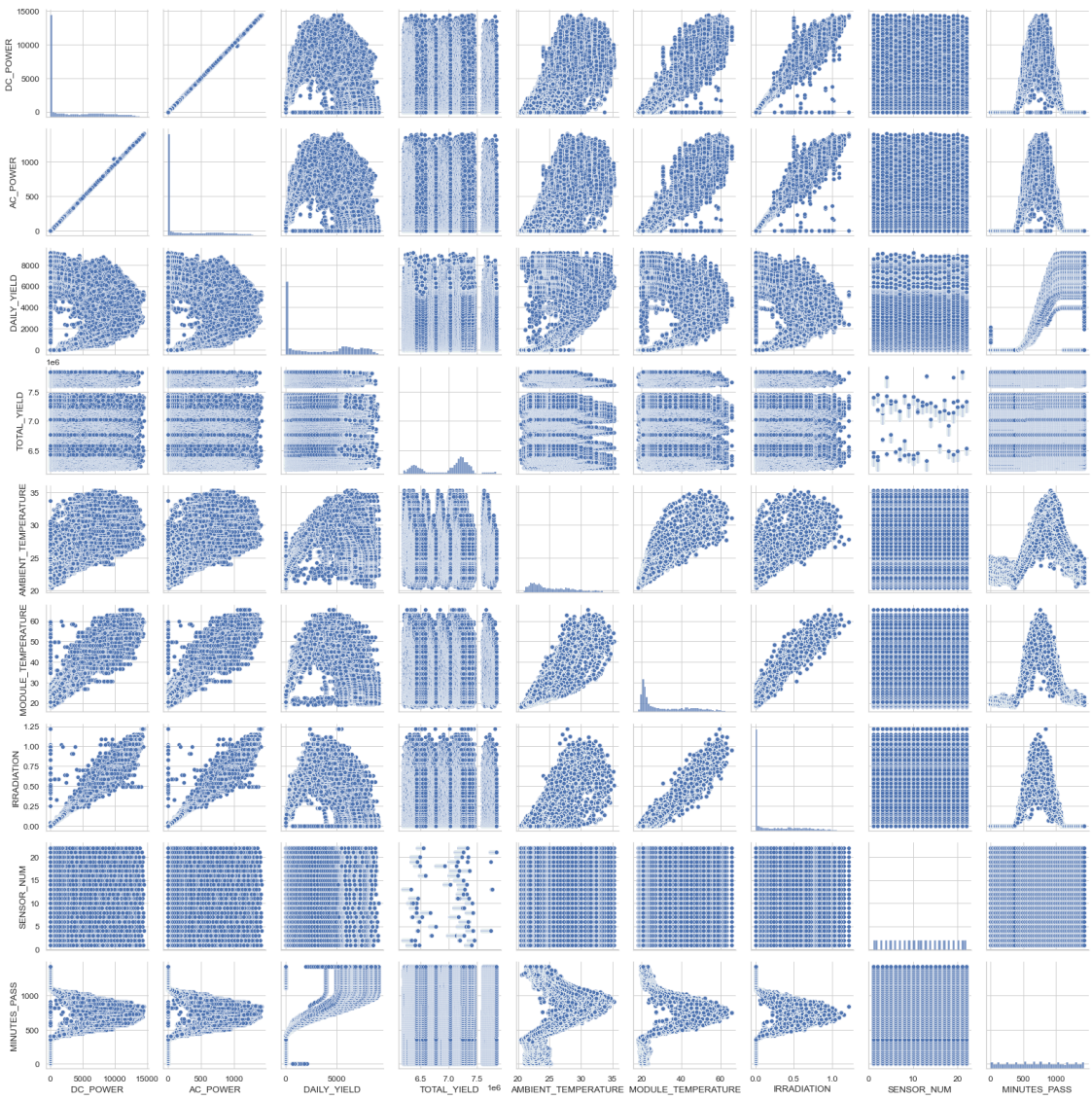
Now that we have a merged dataset we can take a closer look at data distributions and correlations.

3.1 Data Distribution and Correlations

[100]:
```
cols_corr = ["DC_POWER", "AC_POWER", "DAILY_YIELD", "TOTAL_YIELD",␣
 ↪"AMBIENT_TEMPERATURE",
```

```
            "MODULE_TEMPERATURE", "IRRADIATION", "SENSOR_NUM", "HOURS",␣
 ↪"MINUTES", "MINUTES_PASS"]
corrMatrix = df_plant1[cols_corr].corr()
fig = px.imshow(corrMatrix, text_auto=True, aspect="auto")
fig.update_xaxes(side="bottom")
plotly.offline.iplot(fig)
```

[101]:
```
sns.set(style='whitegrid', palette="deep",
        font_scale=1.1, rc={"figure.figsize": [15, 5]})
cols_pair = ["DC_POWER", "AC_POWER", "DAILY_YIELD", "TOTAL_YIELD",␣
 ↪"AMBIENT_TEMPERATURE",
             "MODULE_TEMPERATURE", "IRRADIATION", "SENSOR_NUM", "MINUTES_PASS"]
fig_pair = sns.pairplot(df_plant1[cols_pair])
plt.show()
```



From the last two figures we already can gain a lot of insight:

High correlation between DC Power and AC Power

High correlation between Power and Irradiation

Correlation between DC Power, AC Power and Module Temperature and Ambient Temperature

Correlation between Daily Yield and Ambient Temperature

and that there seem to be outliers in

- AC+DC Power - Irradiation
- DC Power - AC Power (very few)

We can use these outliers for equipment fault and need for maintenance detection!

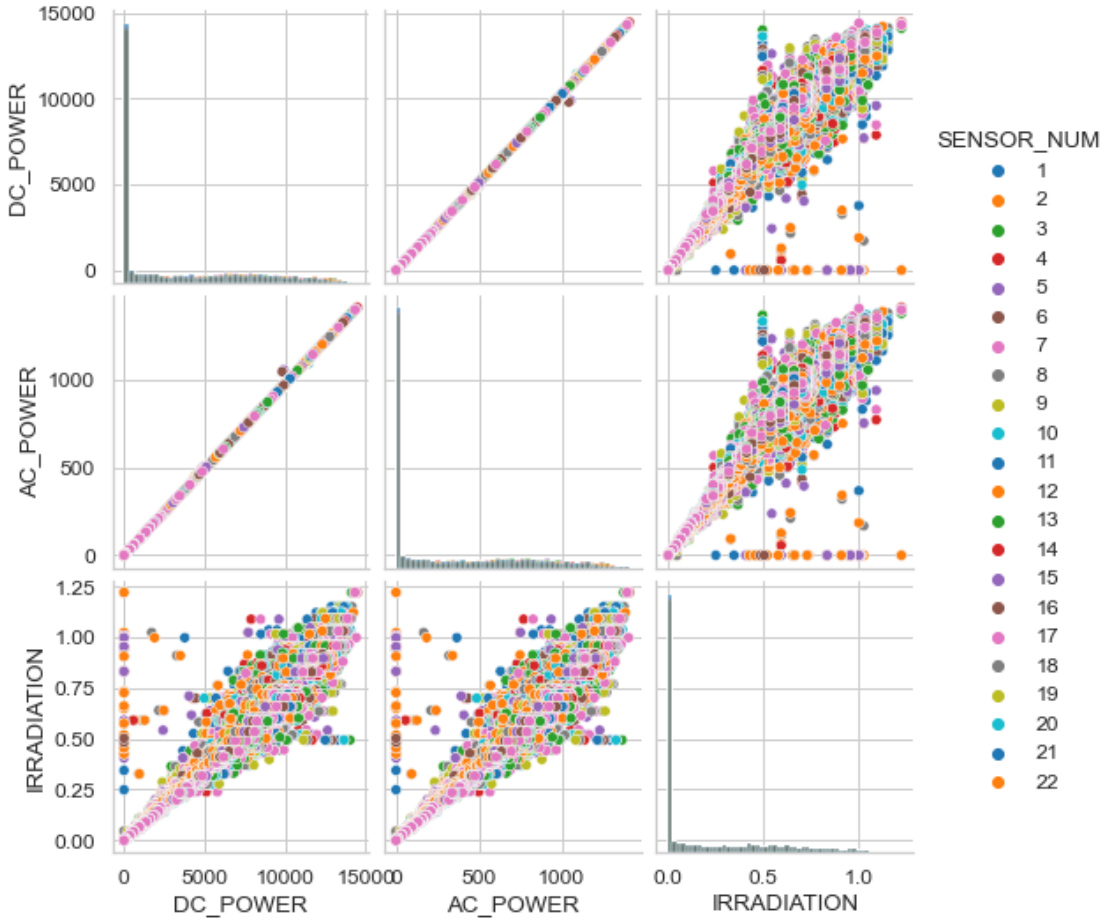
- Outliers in Power-Irradiation indicate failure of the panel lines. If there is enough sunlight but no power is generated, this points to faulty photovoltaic cells.
- Outliers in DC-AC conversion indicate failure at the inverter. If there is DC power delivered but less AC power generated than expected the inverter may be malfunctioning.

If we take a close look at TOTAL_YIELD vs SENSOR_NUM, we see that there are two groups of inverters. One group starts with a higher total yield than the other one. This is most likely because this group was installed earlier than the other group.

NOTE: There are multiple entries where DAILY_YIELD decreased during a day, which should not be possible according to the definition of this column. Since Daily_YIELD is calculated from measured DC_POWER, there seems to be an issue with how this data was generated.

Let's look a bit more closer at the pairplots where we identified outliers and see if these are spread out evenly across all inverters or if we can identify specific inverters.

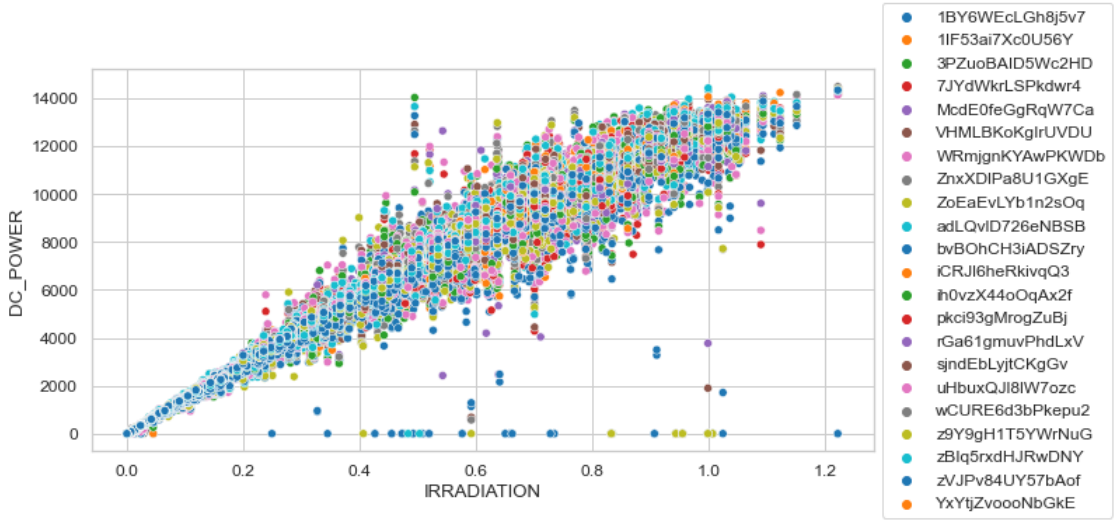
[102]:
```
cols_out = ["DC_POWER", "AC_POWER", "IRRADIATION", "SENSOR_NUM"]
sns.pairplot(df_plant1[cols_out], hue="SENSOR_NUM",
             diag_kind="hist", palette="tab10")
plt.show()
```



Most of the outliers seem to come from a small group of inverters.

3.2 AC Power vs DC Power This indicates how well our inverter is converting DC power to AC Power. Any outliers in this power conversion process can be used to detect faulty inverters.

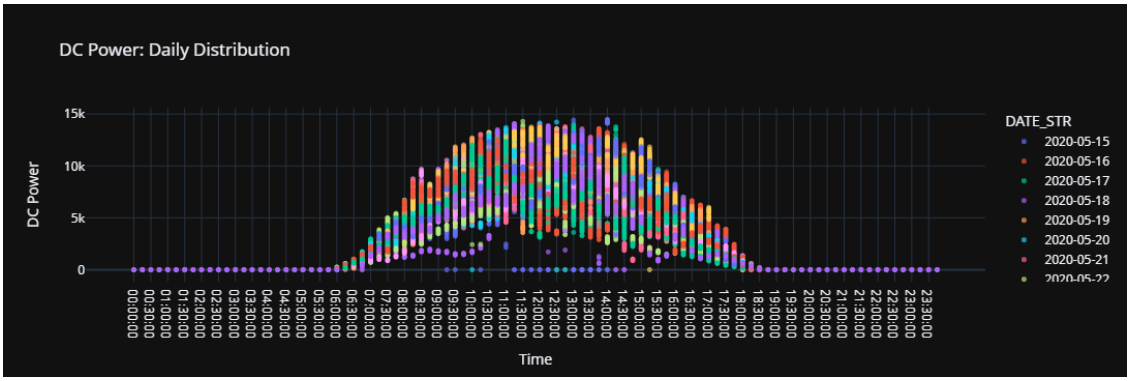
[103]:
```
plt.figure(figsize=(10, 5))
fig_irr = sns.scatterplot(data=df_plant1, x="IRRADIATION",
                          y="DC_POWER", hue="SOURCE_KEY", palette="tab10")
fig_irr.legend(loc='center left', bbox_to_anchor=(1, 0.5), ncol=1)
plt.show()
```



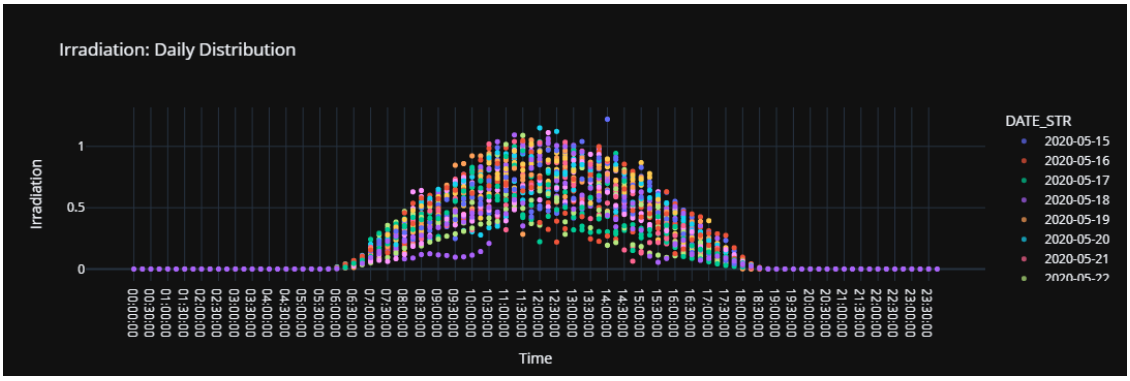
Our data clearly shows events where some inverters received no DC power even though there was more than enough sunlight to generate power. We clearly have some equipment malfunction in our data.

To illustrate this, we can take a closer look at the daily distribution of the generated power and the measured irradiation:

[104]:
```
fig = px.scatter(df_plant1, x="TIME", y="DC_POWER",
                 title="DC Power: Daily Distribution", color="DATE_STR")
fig.update_traces(marker=dict(size=5, opacity=0.7),
                  selector=dict(mode='markers'))
fig.update_layout(title="DC Power: Daily Distribution",
                  xaxis_title="Time",
                  yaxis_title="DC Power", template="plotly_dark")
plotly.offline.iplot(fig)
```



[105]:
```
fig = px.scatter(df_plant1, x="TIME", y="IRRADIATION",
                 title="Irradiation: Daily Distribution", color="DATE_STR")
fig.update_traces(marker=dict(size=5, opacity=0.7),
                  selector=dict(mode='markers'))
fig.update_layout(xaxis_title="Time",
                  yaxis_title="Irradiation", template="plotly_dark")
plotly.offline.iplot(fig)
```



The first figure shows multiple occasions where there was no power generated during daytime. The second figure shows that the irradiation level never dropped low enough during the day to explain this loss of power. This indicates equipment failure! FYI: These plot are interactive! To choose any specific days, double click on the day in the legend on the right.

4. Condition Monitoring

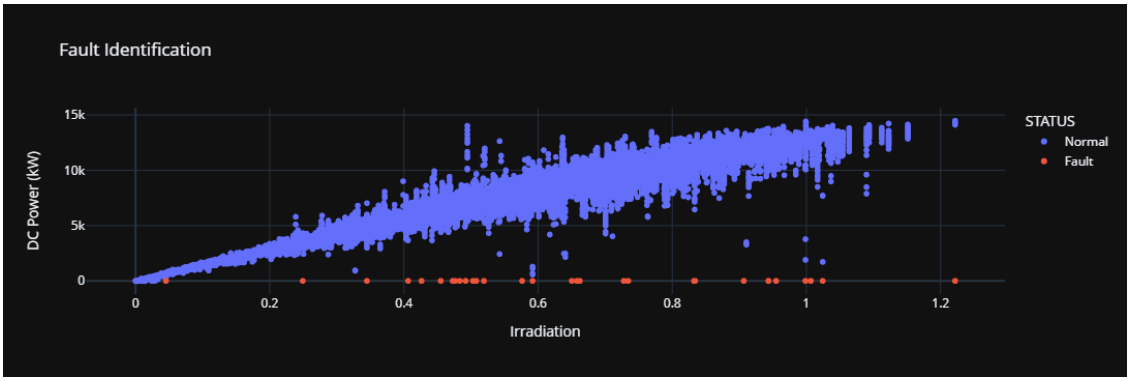
In section 3 we found evidence that point towards equipment failure. Now we are going to build models to detect equipment failure automatically.

4.1 Rule-based Fault Detection During the data exploration we found a simple way to identify faulty equipment: If there is no power measured at the inverter during normal daytime operation, we can assume/identify equipment failure. Let's create a new column ("STATUS") that identifies

faulty operation:

[106]:
```
# Function to check if time is during daytime operation
def time_in_range(start, end, x):
    """Return true if x is in the range [start, end]"""
    if start <= end:
        return start <= x <= end
    else:
        return start <= x or x <= end


# set normal daytime operation range
start = dt.time(6, 30, 0)  # sunrise
end = dt.time(17, 30, 0)  # sunset

# Create new column to check proper operation
# Return "Normal" if operation is normal and "Fault" if operation is faulty
df_plant1["STATUS"] = 0
for index in df_plant1.index:
    if time_in_range(start, end, df_plant1["TIME"][index]) and␣
 ↪df_plant1["DC_POWER"][index] == 0:
        df_plant1["STATUS"][index] = "Fault"
    else:
        df_plant1["STATUS"][index] = "Normal"
```

[107]:
```
fig = px.scatter(df_plant1, x="IRRADIATION", y="DC_POWER", title="Fault␣
 ↪Identification",
                 color="STATUS", labels={"DC_POWER": "DC Power (kW)",␣
 ↪"IRRADIATION": "Irradiation"})
fig.update_traces(marker=dict(size=3, opacity=0.8),
                  selector=dict(mode='marker'))
fig.update_layout(template="plotly_dark")
plotly.offline.iplot(fig)
```

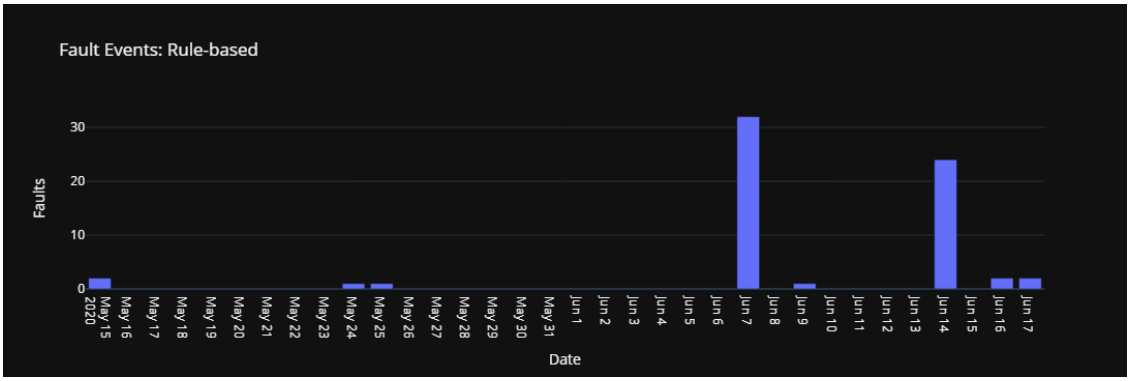


4.1.1 Days with faults

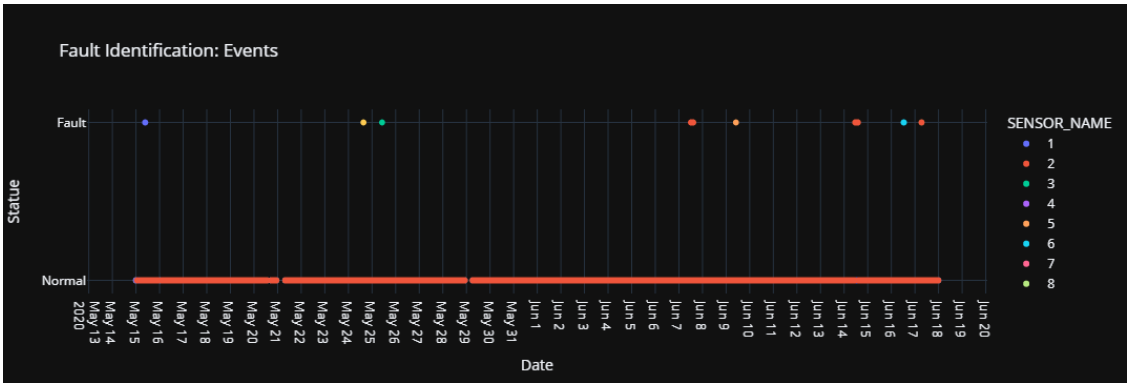
[108]:
```
df_plant1[df_plant1["STATUS"] == "Fault"]["DATE"].value_counts()
```

[108]:
```
2020-06-07    32
2020-06-14    24
2020-05-15     2
2020-06-16     2
2020-06-17     2
2020-05-24     1
2020-05-25     1
2020-06-09     1
Name: DATE, dtype: int64
```

[109]:
```
fig = px.bar(df_plant1[df_plant1["STATUS"] == "Fault"]["DATE"].value_counts(
), title="Fault Events: Rule-based", labels={"value": "Faults", "index":␣
 ↪"Date", "SENSOR_NAME": "Inverter"})
fig.update_xaxes(
    dtick="d1",
)
fig.update_layout(template="plotly_dark")
fig.update(layout_showlegend=False)
```



[110]:
```
# Date & Inverter Time series
fig10 = px.scatter(df_plant1, x="DATE_TIME", y="STATUS",
                   title="Fault Identification: Events", color="SENSOR_NAME")
fig10.update_traces(marker=dict(size=1, opacity=0.8),
                    selector=dict(mode='marker'))
fig10.update_xaxes(
    dtick="d1")
fig10.update_layout(xaxis_title="Date",
                    yaxis_title="Statue", template="plotly_dark")
fig10.show()
```



[111]:
```
# uncomment to see faulty days
#fig = px.scatter(df_plant1[df_plant1.DATE_STR=="2020-06-07"], x="DATE_TIME",␣
 ↪y="DC_POWER", title="2020-06-07", color="SENSOR_NAME", labels={"DC_POWER":
 ↪"DC Power", "DATE_TIME":"Time"})
#fig.update_traces(marker=dict(size=3, opacity=0.7),␣
 ↪selector=dict(mode='marker'))
# fig.update_layout(template="plotly_dark")
# plotly.offline.iplot(fig)
```

4.1.2 Number of recorded faults

[112]:
```
df_plant1.STATUS.value_counts()
```

[112]:
```
Normal    68709
Fault        65
Name: STATUS, dtype: int64
```

[113]:
```
print("There are {} records of faulty operation!" .format(
    df_plant1.STATUS.value_counts().Fault))
```

```
There are 65 records of faulty operation!
```

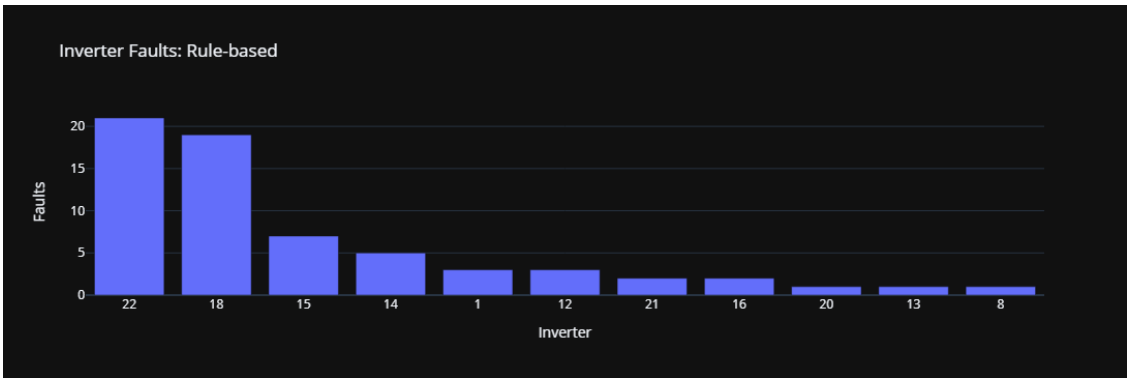
4.1.3 Inverters with faults

[114]:
```
df_plant1[df_plant1["STATUS"] == "Fault"]["SENSOR_NAME"].value_counts()
```

[114]:
```
22    21
18    19
15     7
14     5
1      3
12     3
21     2
16     2
20     1
```

```
13     1
8      1
Name: SENSOR_NAME, dtype: int64
```

[115]:
```
fig = px.bar(df_plant1[df_plant1["STATUS"] == "Fault"]["SENSOR_NAME"].
 ↪value_counts(
), title="Inverter Faults: Rule-based", labels={"value": "Faults", "index":␣
 ↪"Inverter", "SENSOR_NAME": "Inverter"})
fig.update_layout(template="plotly_dark")
fig.update(layout_showlegend=False)
```



4.1.4 Summary

[116]:
```
print("The most faults were recorded on {} and {}." .
 ↪format(df_plant1[df_plant1["STATUS"] == "Fault"]["DATE"].value_counts(
).index[0], df_plant1[df_plant1["STATUS"] == "Fault"]["DATE"].value_counts().
 ↪index[1]))
print("Inverter {} and {} had the most failures." .
 ↪format(df_plant1[df_plant1["STATUS"] == "Fault"]["SOURCE_KEY"].value_counts(
).index[0], df_plant1[df_plant1["STATUS"] == "Fault"]["SOURCE_KEY"].
 ↪value_counts().index[1]))
```

```
The most faults were recorded on 2020-06-07 and 2020-06-14.
Inverter bvBOhCH3iADSZry and 1BY6WEcLGh8j5v7 had the most failures.
```

3.4.3 4.2 Fault Detection with Regression Models

While the simple rule-based approach in the previous chapter was already successful at detecting severe failure, finding less obvious anomalies and more subtle indicators for equipment failure (or cleaning/maintenance need) require more effort.

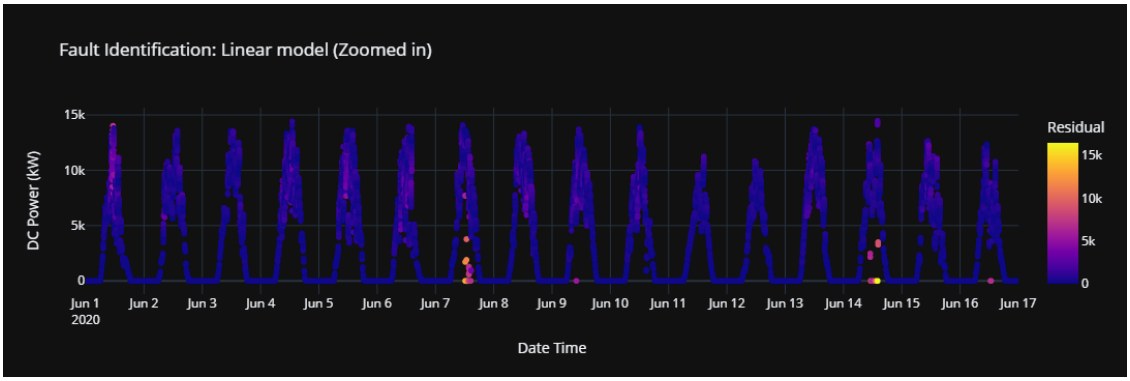
4.2.1 Linear Model A first attempt at predicting DC power from irradiance by assuming a linear relationship

$$P(t) = a + b \cdot E(t) \tag{1}$$

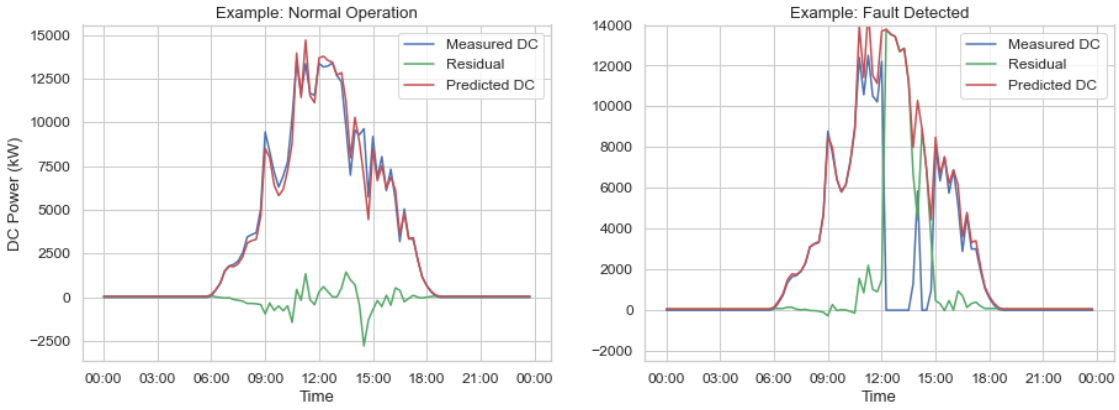
with the generated DC power $P(t)$, irradiance $E(t)$ and coefficients a, b. Note: PV module panels can reach up to 65°C. The efficiency of PV cells is usually lower at high temperatures. This leads to a non-linear relationship between irradiance and generated DC power. We are going to model this nonlinearity in the next section.

[117]:
```
from sklearn.linear_model import LinearRegression

# Model
reg = LinearRegression()

# choose training data
train_dates = ["2020-05-16", "2020-05-17", "2020-05-18",
               "2020-05-19", "2020-05-20", "2020-05-21"]
df_train = df_plant1[df_plant1["DATE_STR"].isin(train_dates)]

#fit & predict
reg.fit(df_train[["IRRADIATION"]], df_train.DC_POWER)
prediction = reg.predict(df_plant1[["IRRADIATION"]])

# save prediction, residual, and absolute residual
df_train["Prediction"] = reg.predict(df_train[["IRRADIATION"]])
df_train["Residual"] = df_train["Prediction"] - df_train["DC_POWER"]
df_plant1["Prediction"] = reg.predict(df_plant1[["IRRADIATION"]])
df_plant1["Residual"] = df_plant1["Prediction"] - df_plant1["DC_POWER"]
df_plant1["Residual_abs"] = df_plant1["Residual"].abs()
```

[118]:
```
fig = px.scatter(df_plant1, x="DATE_TIME", y="DC_POWER", title="Fault␣
 ↪Identification: Linear model (Zoomed in)", color="Residual_abs", labels={
                "DC_POWER": "DC Power (kW)", "DATE_TIME": "Date Time",␣
 ↪"Residual_abs": "Residual"}, range_x=[dt.date(2020, 6, 1), dt.date(2020, 6,␣
 ↪17)])
fig.update_traces(marker=dict(size=3, opacity=0.7),
                  selector=dict(mode='marker'))
fig.update_xaxes(
    dtick="d1",
)
fig.update_layout(template="plotly_dark")
plotly.offline.iplot(fig)
```



[119]:
```
# choose data
day = "2020-06-07"
inverter1 = "2"
inverter2 = "22"
df_pred = df_plant1[(df_plant1["DATE_STR"] == day)].copy()

fig, axes = plt.subplots(1, 2)

sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter1].DC_POWER, label="Measured DC", color="b", ax=axes[0])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter1].Residual, label="Residual", color="g", ax=axes[0])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter1].Prediction, label="Predicted DC", color="r", ax=axes[0])

sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter2].DC_POWER, label="Measured DC", color="b", ax=axes[1])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter2].Residual, label="Residual", color="g", ax=axes[1])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter2].Prediction, label="Predicted DC", color="r", ax=axes[1])

plt.gcf().axes[0].xaxis.set_major_formatter(xformatter)  # set xaxis format
plt.gcf().axes[1].xaxis.set_major_formatter(xformatter)  # set xaxis format
axes[0].set_xlabel("Time")
axes[1].set_xlabel("Time")
axes[0].set_ylabel("DC Power (kW)")
axes[1].set_ylabel("")
axes[1].set_ylim(-2500, 14000)
axes[0].set_title("Example: Normal Operation")
axes[1].set_title("Example: Fault Detected")
plt.show()
```



[120]:
```
inverter2 = "22"
df_pred2 = df_plant1[df_plant1["SENSOR_NAME"] == inverter2].copy()
sns.lineplot(df_pred2.DATE_TIME, df_pred2.Prediction,
             label="Predicted DC", color="r")
sns.lineplot(df_pred2.DATE_TIME, df_pred2.DC_POWER,
             label="Measured DC", color="b")
sns.lineplot(df_pred2.DATE_TIME, df_pred2.Residual -
             5000, label="Residual (Offset)", color="g")
plt.xlabel("Date")
plt.ylabel("Power (kW)")
plt.title("Fault Detection: Inverter 22")
plt.show()
```

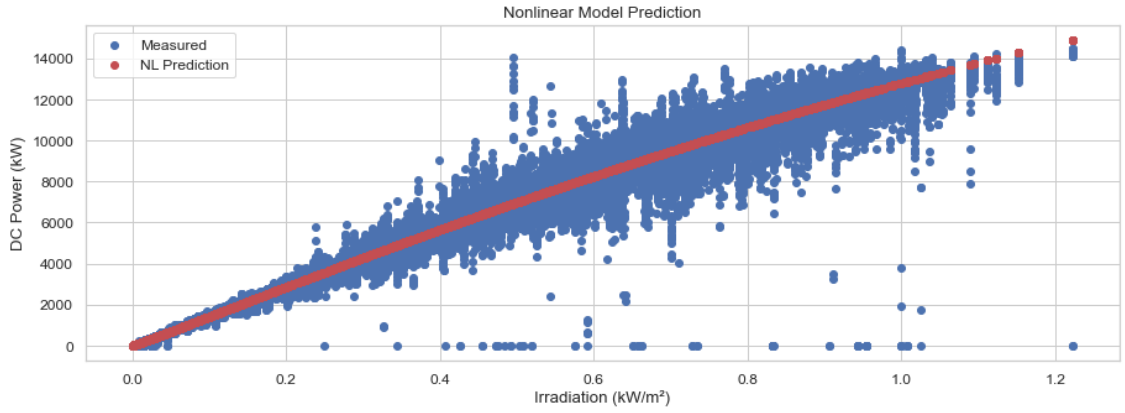
4.2.1 Nonlinear Model According to Hooda et al. (2018) the generated power of a photovoltaic cell can be modeled by the nonlinear equation

$$P(t) = aE(t)\left(1 - b(T(t) + \frac{E(t)}{800}(c - 20) - 25) - d\ln(E(t))\right) \tag{2}$$

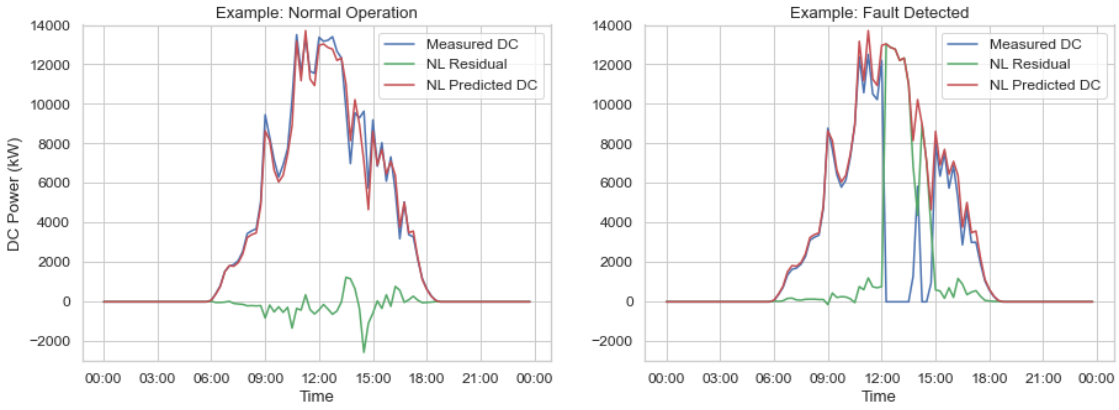
with irradiance $E(t)$, Temperature $T(t)$ and coefficients a, b, c, d.

[121]:
```
def func(X, a, b, c, d):
    '''Nonlinear function to predict DC power output from Irradiation and␣
 ↪Temperature.'''
    x, y = X
    x = x*1000
    y = y*1000
    return a*x*(1-b*(y+x/800*(c-20)-25)-d*np.log(x+1e-10))


# fit function
p0 = [1., 0., -1.e4, -1.e-1]  # starting values
popt, pcov = curve_fit(func, (df_train.IRRADIATION,
                        df_train.MODULE_TEMPERATURE), df_train.DC_POWER, p0,␣
 ↪maxfev=5000)
sigma_abcd = np.sqrt(np.diagonal(pcov))

# predict & save
df_train["Prediction_NL"] = func(
    (df_train.IRRADIATION, df_train.MODULE_TEMPERATURE), *popt)
df_train["Residual_NL"] = df_train["Prediction_NL"] - df_train["DC_POWER"]

df_plant1["Prediction_NL"] = func(
    (df_plant1.IRRADIATION, df_plant1.MODULE_TEMPERATURE), *popt)
df_plant1["Residual_NL"] = df_plant1["Prediction_NL"] - df_plant1["DC_POWER"]
```

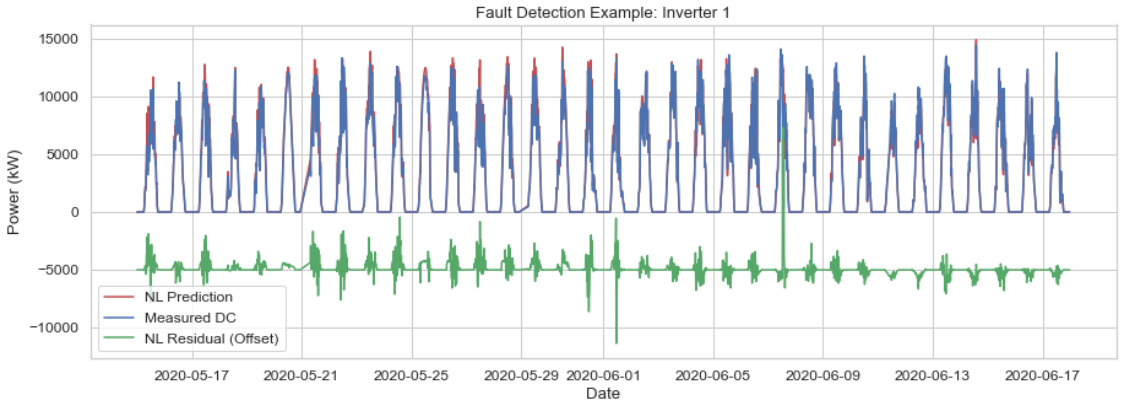
[122]:
```
plt.scatter(df_plant1.IRRADIATION, df_plant1.DC_POWER, label="Measured")
plt.scatter(df_plant1.IRRADIATION, df_plant1.Prediction_NL,
            color="r", label="NL Prediction")
plt.legend()
plt.xlabel("Irradiation (kW/m²)")
plt.ylabel("DC Power (kW)")
plt.title("Nonlinear Model Prediction")
plt.show()
```



[123]:
```
# choose data
day = "2020-06-07"
inverter1 = "2"
inverter2 = "22"
df_pred = df_plant1[(df_plant1["DATE_STR"] == day)].copy()

fig, axes = plt.subplots(1, 2)

sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter1].DC_POWER, label="Measured DC", color="b", ax=axes[0])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter1].Residual_NL, label="NL Residual", color="g", ax=axes[0])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter1].Prediction_NL, label="NL Predicted DC", color="r",␣
 ↪ax=axes[0])

sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter2].DC_POWER, label="Measured DC", color="b", ax=axes[1])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter2].Residual_NL, label="NL Residual", color="g", ax=axes[1])
sns.lineplot(df_pred.DATE_TIME, df_pred[df_pred["SENSOR_NAME"] ==
             inverter2].Prediction_NL, label="NL Predicted DC", color="r",␣
 ↪ax=axes[1])

plt.gcf().axes[0].xaxis.set_major_formatter(xformatter)  # set xaxis format
plt.gcf().axes[1].xaxis.set_major_formatter(xformatter)  # set xaxis format
axes[0].set_xlabel("Time")
axes[1].set_xlabel("Time")
axes[0].set_ylabel("DC Power (kW)")
axes[1].set_ylabel("")
axes[0].set_ylim(-3000, 14000)
axes[1].set_ylim(-3000, 14000)
axes[0].set_title("Example: Normal Operation")
axes[1].set_title("Example: Fault Detected")
plt.show()
```

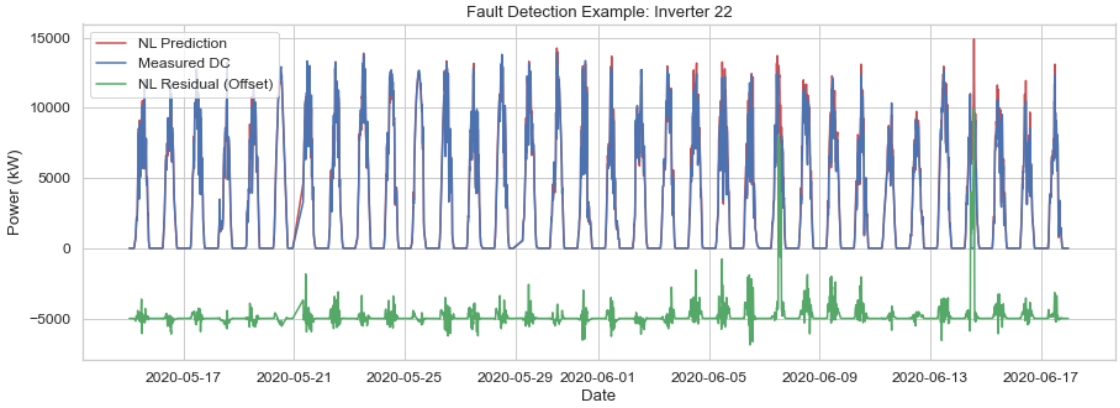


[124]:
```
inverter2 = "1"
df_pred2 = df_plant1[df_plant1["SENSOR_NAME"] == inverter2].copy()
sns.lineplot(df_pred2.DATE_TIME, df_pred2.Prediction_NL,
             label="NL Prediction", color="r")
sns.lineplot(df_pred2.DATE_TIME, df_pred2.DC_POWER,
             label="Measured DC", color="b")
sns.lineplot(df_pred2.DATE_TIME, df_pred2.Residual_NL -
             5000, label="NL Residual (Offset)", color="g")
plt.xlabel("Date")
plt.ylabel("Power (kW)")
plt.title("Fault Detection Example: Inverter {}".format(inverter2))
plt.show()
```



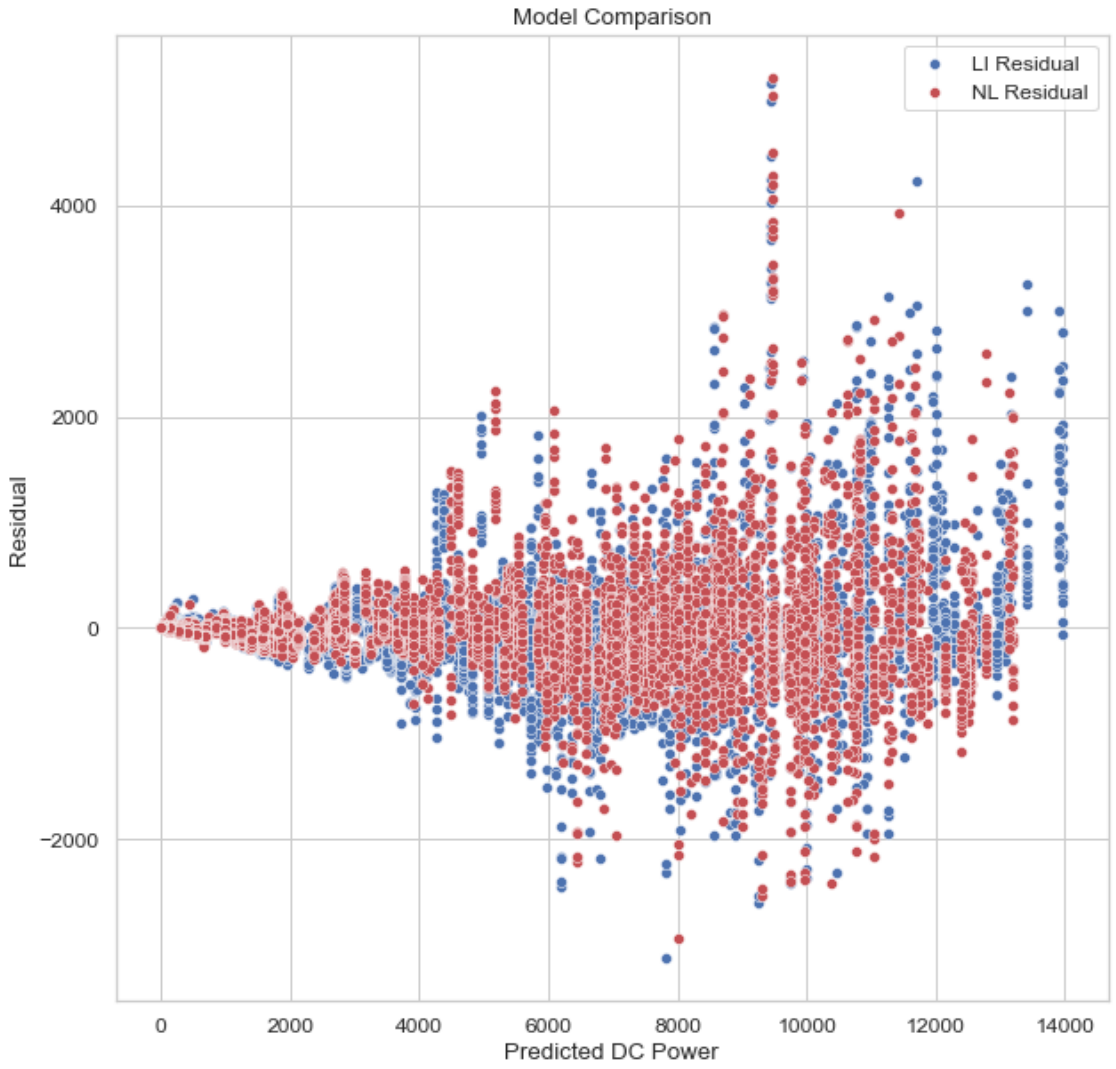
[125]:
```
inverter2 = "22"
df_pred2 = df_plant1[df_plant1["SENSOR_NAME"] == inverter2].copy()
sns.lineplot(df_pred2.DATE_TIME, df_pred2.Prediction_NL,
             label="NL Prediction", color="r")
```

```
sns.lineplot(df_pred2.DATE_TIME, df_pred2.DC_POWER,
             label="Measured DC", color="b")
sns.lineplot(df_pred2.DATE_TIME, df_pred2.Residual_NL -
             5000, label="NL Residual (Offset)", color="g")
plt.xlabel("Date")
plt.ylabel("Power (kW)")
plt.title("Fault Detection Example: Inverter {}".format(inverter2))
plt.show()
```



4.2.3 Model Comparison To compare the two models we can take a look at their respective residuals. The nonlinear model seems to perform slightly better than the linear model, especially at times of high irradiance.

[126]:
```
# plot model comparison residual
plt.figure(figsize=(10, 10))
sns.scatterplot(df_train.Prediction, df_train.Residual,
                color="b", label="LI Residual")
sns.scatterplot(df_train.Prediction_NL, df_train.Residual_NL,
                color="r", label="NL Residual")
axes = plt.gca()
plt.ylabel("Residual")
plt.xlabel("Predicted DC Power")
plt.title("Model Comparison")
plt.show()
```



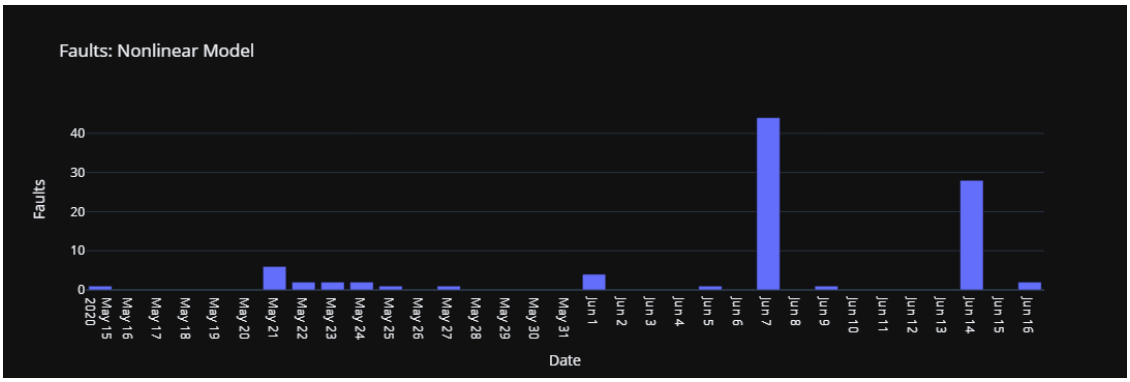
3.4.4 4.2.4 NL Fault Detection

Let's now use the irradiance and temperature data to predict the expected DC power with the nonlinear model. This allows us to identify addtional anomalies by comparing the measured DC power with the prediction. The additional anomalies indicate equipment underperformance or need for maintenance.

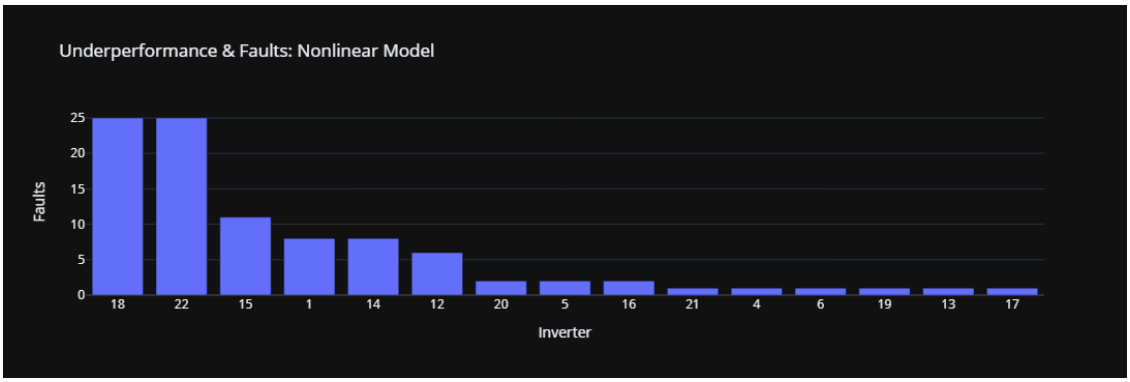
[127]:
```
# set confidence range for residual for fault
limit_fault = 4000

# Create new column to check proper operation
# Return "Normal" if operation is normal and "Fault" if operation is faulty
df_plant1["STATUS_NL"] = 0
for index in df_plant1.index:
```

```
    if df_plant1["Residual_NL"][index] > limit_fault:
        df_plant1["STATUS_NL"][index] = "Fault"
    else:
        df_plant1["STATUS_NL"][index] = "Normal"
```

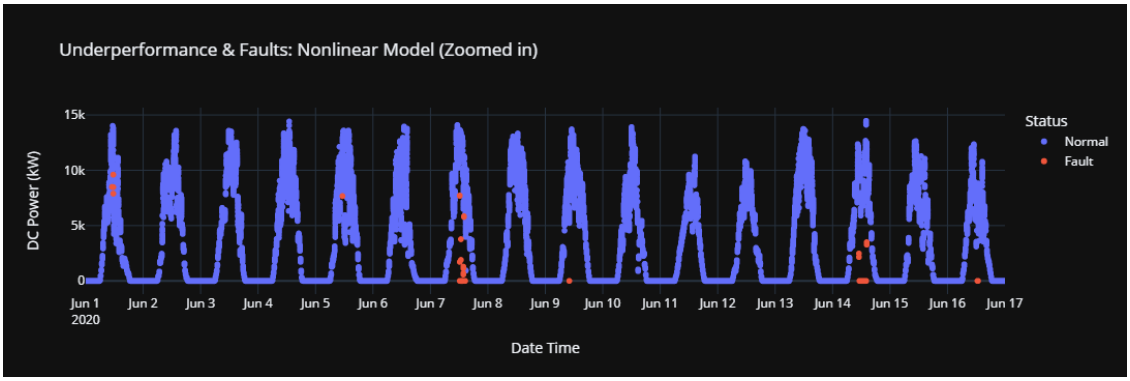
[128]:
```
fig = px.bar(df_plant1[df_plant1["STATUS_NL"] == "Fault"]["DATE"].value_counts(
), title="Faults: Nonlinear Model", labels={"value": "Faults", "index": "Date",␣
 ↪"SENSOR_NAME": "Inverter"}, )
fig.update(layout_showlegend=False)
fig.update_xaxes(
    dtick="d1",
)
fig.update_layout(template="plotly_dark")
plotly.offline.iplot(fig)
```



[129]:
```
fig = px.bar(df_plant1[df_plant1["STATUS_NL"] == "Fault"]["SENSOR_NAME"].
 ↪value_counts(
), title="Underperformance & Faults: Nonlinear Model", labels={"value":␣
 ↪"Faults", "index": "Inverter", "SENSOR_NAME": "Inverter"})
fig.update(layout_showlegend=False)
fig.update_layout(template="plotly_dark")
plotly.offline.iplot(fig)
```



[130]:
```
fig = px.scatter(df_plant1, x="DATE_TIME", y="DC_POWER",␣
 ↪title="Underperformance & Faults: Nonlinear Model (Zoomed in)",␣
 ↪color="STATUS_NL", labels={
                 "DC_POWER": "DC Power (kW)", "DATE_TIME": "Date Time",␣
 ↪"STATUS_NL": "Status"}, range_x=[dt.date(2020, 6, 1), dt.date(2020, 6, 17)])
fig.update_traces(marker=dict(size=3, opacity=0.7),
                  selector=dict(mode='marker'))
fig.update_xaxes(
    dtick="d1",
)
fig.update_layout(template="plotly_dark")
plotly.offline.iplot(fig)
```



[131]:
```
print("The most anomalies were recorded on {} and {}." .
 ↪format(df_plant1[df_plant1["STATUS_NL"] == "Fault"]["DATE"].value_counts(
).index[0], df_plant1[df_plant1["STATUS_NL"] == "Fault"]["DATE"].value_counts().
 ↪index[1]))
print("Inverter {} and {} had the most events of failure/underperformance." .
 ↪format(df_plant1[df_plant1["STATUS_NL"] == "Fault"]["SENSOR_NAME"].
 ↪value_counts(
).index[0], df_plant1[df_plant1["STATUS_NL"] == "Fault"]["SENSOR_NAME"].
 ↪value_counts().index[1]))
```

```
The most anomalies were recorded on 2020-06-07 and 2020-06-14.
Inverter 18 and 22 had the most events of failure/underperformance.
```

5. Implementing Machine Learning for power prediction.

[132]:
```
df2 = df_plant1.copy()
```

[133]:
```
X = df2[['AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE', 'IRRADIATION']]
y = df2['AC_POWER']
```

[134]:
```
from sklearn.model_selection import train_test_split
import sklearn.metrics as sm
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=.2, random_state=21)
```

1.

[135]:
```
def acc(y_test, y_test_pred):
    print("Mean absolute error =", round(
        sm.mean_absolute_error(y_test, y_test_pred), 2))
    print("Mean squared error =", round(
        sm.mean_squared_error(y_test, y_test_pred), 2))
    print("Median absolute error =", round(
        sm.median_absolute_error(y_test, y_test_pred), 2))
    print("Explain variance score =", round(
        sm.explained_variance_score(y_test, y_test_pred), 2))
    print("R2 score =", round(sm.r2_score(y_test, y_test_pred) * 100, 2), "%")
```

[136]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
R2_Score_lr = round(r2_score(y_pred_lr, y_test) * 100, 3)

#score_lr = 100*lr_clf.score(X_test,y_test)
acc(y_test, y_pred_lr)
```

```
Mean absolute error = 25.9
Mean squared error = 3219.04
Median absolute error = 8.41
Explain variance score = 0.98
R2 score = 97.92 %
```

Random Forest Regressor

[137]:
```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(X_train, y_train)
y_pred_rfr = rfr.predict(X_test)
```

```
acc(y_test, y_pred_rfr)
```

```
Mean absolute error = 15.99
Mean squared error = 2175.2
Median absolute error = 0.45
Explain variance score = 0.99
R2 score = 98.59 %
```

Decision Tree Regressor

[138]:
```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)

y_pred_dtr = dtr.predict(X_test)

acc(y_test, y_pred_dtr)
```

```
Mean absolute error = 15.99
Mean squared error = 2177.17
Median absolute error = 0.47
Explain variance score = 0.99
R2 score = 98.59 %
```

3.4.5 5.1 Result Prediction

[139]:
```
forecast_prediction = rfr.predict(X_test)
print(forecast_prediction)
```

```
[   0.         1031.95001986  305.17519585 …  749.34667708  375.25946882
  114.86280891]
```

[140]:
```
cross_check = pd.DataFrame(
    {'Actual': y_test, 'Predicted': forecast_prediction})
cross_check.head()
```

[140]:

	Actual	Predicted
43819	0.0000	0.000000
2949	1072.3250	1031.950020
33769	299.8125	305.175196
47825	0.0000	0.000000
29370	0.0000	0.000000

[141]:
```
cross_check['Error'] = cross_check['Actual'] - cross_check['Predicted']
cross_check.head()
```

[141]:

	Actual	Predicted	Error
43819	0.0000	0.000000	0.000000
2949	1072.3250	1031.950020	40.374980
33769	299.8125	305.175196	-5.362696

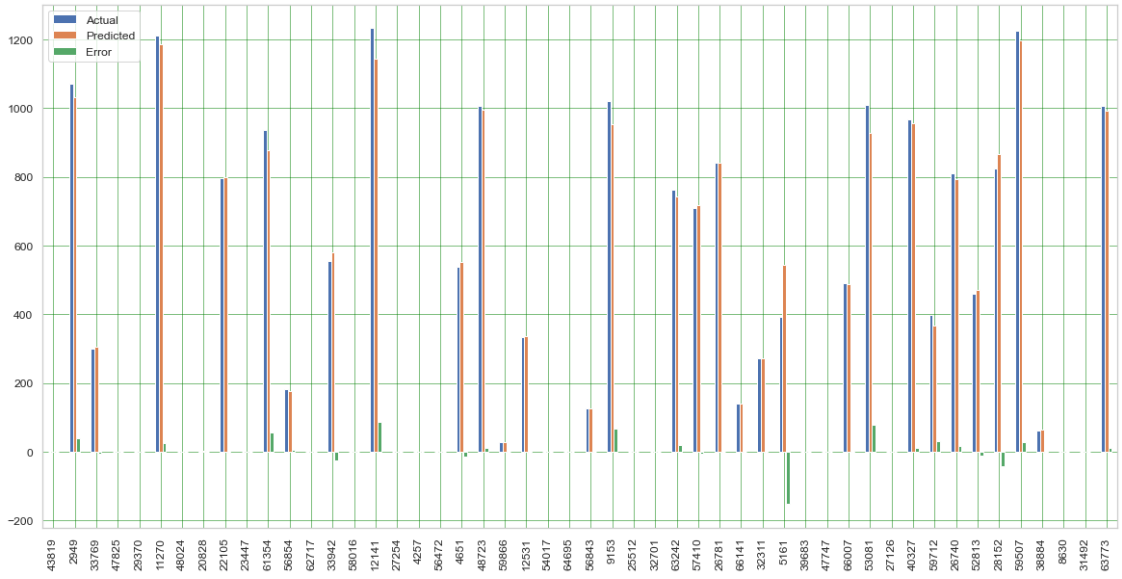
```
47825      0.0000      0.000000    0.000000
29370      0.0000      0.000000    0.000000
```

[142]:
```
cross_check_final = cross_check[cross_check['Error'] <= 20]
cross_check_final.sample(25).style.background_gradient(
    cmap='coolwarm').set_properties(**{
        'font-family': 'Times',
        'color': 'LigntGreen',
        'font-size': '13px'
    })
```

[142]:
```
<pandas.io.formats.style.Styler at 0x15a94dd9660>
```

[143]:
```
plt.scatter(y_test, forecast_prediction)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```

[144]:
```
df3 = cross_check.head(50)
df3.plot(kind='bar', figsize=(20, 10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



3.5 Summary

- **EDA:** Noticed some potential issues with the data: DAILY_YIELD is decreasing on some days even though this should not be possible by definition. Inverters have different number of data points
- **Challenge 1:** Even though this notebook did not focus on forecasting, the power-irradiance models may be helpful in combination with external data from local weather forecasts to predict the generated power for the next couple days.
- **Challenge 2&3:** Successfully identified **events of equipment failure and underperformance** with a rule-based method and linear/nonlinear modeling of the relationship between irradiance, temperature and DC power. This approach can be useful for real-time condition monitoring and fault detection.

[ ]: