Solar Panel Analysis

January 18, 2021

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.

0.0.1 Provenance

Sources

Power generation and sensor data gathered from two solar power plants

Collection methodology

Power generation and sensor data gathered at 15 minutes intervals over a 34 day period. Generation data collected at inverter level, while the sensor data is at the plant level. ***

0.0.2 Columns

Plant 1&2 Generation data @Inverter level

```
DATE_TIME- Date and time for each observation.
           Observations recorded at 15 minute intervals.

PLANT_ID - this will be common for the entire file.

SOURCE_KEY - Source key in this file stands for the inverter id.
             changed to Inverter id)

DC_POWER - Amount of DC power generated by the inverter (source_key)
           in this 15 minute interval. Units - kW.

AC_POWER - Amount of AC power generated by the inverter (source_key)
           in this 15 minute interval. Units - kW.

DAILY_YIELD - Daily yield is a cumulative sum of power generated
              on that day, till that point in time.

TOTAL_YIELD - This is the total yield for the inverter till that
```

```
                    point in time.
```

Plant 1&2 Weather sensor data @Plant level

```
DATE_TIME- Date and time for each observation.
           Observations recorded at 15 minute intervals.

PLANT_ID - this will be common for the entire file.

SOURCE_KEY - Stands for the sensor panel id. This will be
             common for the entire file because there's
             only one sensor panel for the plant.

AMBIENT_TEMPERATURE - This is the ambient temperature at
                      the plant.

MODULE_TEMPERATURE - There's a module (solar panel) attached
                     to the sensor panel. This is the
                     temperature reading for that module.
```

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mlp
import seaborn as sns
import numpy as np
import stat as st
import datetime as dt
```

[2]:
```
gen_1 = pd.read_csv('Plant_1_Generation_Data.csv',delimiter=',')
gen_2 = pd.read_csv('Plant_2_Generation_Data.csv',delimiter=',')

s1 = pd.read_csv('Plant_1_Weather_Sensor_Data.csv',delimiter=',')
s2 = pd.read_csv('Plant_2_Weather_Sensor_Data.csv',delimiter=',')

gen_1.rename(columns={'SOURCE_KEY':'INVERTER_ID'},inplace =True)
gen_2.rename(columns={'SOURCE_KEY':'INVERTER_ID'},inplace =True)
```

1 Functions

[3]:
```
def slice_df(columns,data=[gen_1,gen_2]):
    df1= data[0].copy()
    df2=data[1].copy()
    df1 = df1[columns]
    df2 = df2[columns]

    return (df1),(df2)
```

[4]:
```
def split_date(df,h=False,**kwargs):
    reset = kwargs.get('reset',False)

    if reset == True:
        df = df.reset_index()

    df['TIME'] = df["DATE_TIME"].dt.time
    df['DATE'] = df['DATE_TIME'].dt.date

    #convert to hour
    if h ==True:
        df['HOUR'] = df['DATE_TIME'].apply(lambda t : t.hour)

    return df
```

[5]:
```
def Generate_sd_mean(df,df2,column,rows=1,cols=2):
    #agg as list.
    #column as str
    results={}
    results[0]= df.groupby('TIME')[column].agg(['mean','std'])
    results[1] = df2.groupby('TIME')[column].agg(['mean','std'])
    fig,axes = plt.subplots(rows,cols,figsize=(13,6))

    ax={}
    for i in range(0,rows*cols):
        ax[i]= results[i]['mean'].plot(ax=axes[i])
        ax[i].fill_between(results[i].
 ↪index,results[i]['mean']-results[i]['std'],results[i]['mean']+results[i]['std'],color='b',a
 ↪3)
```

[6]:
```
def groupby_inv_date(df,freq,fillna=False,agg_m =␣
 ↪'count',multi_index=True,**kwargs):

    col = kwargs.get('col','INVERTER_ID')
    if multi_index ==False:
        gb = df.groupby(pd.Grouper(freq=freq,key='DATE_TIME'))[col].agg([agg_m])
        gb_org = gb.unstack().transpose()
    else:
        gb = df.groupby(['INVERTER_ID',pd.
 ↪Grouper(freq=freq,key='DATE_TIME')])[col].agg(agg_m)
        gb_org = gb.unstack().transpose()

    if fillna == True:
        gb_org_cleaned = gb_org.fillna(0)
        return gb_org_cleaned

    return gb_org
```

[7]:
```python
def␣
↪groupby_power(df1,df2=None,freq='15t',cols=['AC_POWER','DC_POWER'],agg_m='mean',multi_index
↪

    r = kwargs.get('reset_i',False)

    gb1 = df1.groupby(pd.Grouper(freq=freq,key='DATE_TIME'))[cols].agg(agg_m)
    if df2 is not None:
        gb2 = df2.groupby(pd.Grouper(freq=freq,key='DATE_TIME'))[cols].
↪agg(agg_m)
    if r ==True:
        gb1 = gb1.reset_index()
        gb2 = gb2.reset_index()

    if df2 is not None:
        return gb1,gb2
    return gb1
```

2 Understanding the data

2.1 Generation data

[8]:
```python
gen_1.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   INVERTER_ID  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]:
```python
print('gen1 # of Inverters:',gen_1['INVERTER_ID'].nunique())
print('gen2 # of Inverters:',gen_2['INVERTER_ID'].nunique())
```

```
gen1 # of Inverters: 22
gen2 # of Inverters: 22
```

[10]:
```python
gen_1[['DATE_TIME','PLANT_ID','INVERTER_ID','DC_POWER']].head(23)
```

[10]:

	DATE_TIME	PLANT_ID	INVERTER_ID	DC_POWER
0	15-05-2020 00:00	4135001	1BY6WEcLGh8j5v7	0.0
1	15-05-2020 00:00	4135001	1IF53ai7Xc0U56Y	0.0
2	15-05-2020 00:00	4135001	3PZuoBAID5Wc2HD	0.0
3	15-05-2020 00:00	4135001	7JYdWkrLSPkdwr4	0.0
4	15-05-2020 00:00	4135001	McdE0feGgRqW7Ca	0.0
5	15-05-2020 00:00	4135001	VHMLBKoKgIrUVDU	0.0
6	15-05-2020 00:00	4135001	WRmjgnKYAwPKWDb	0.0
7	15-05-2020 00:00	4135001	ZnxXDlPa8U1GXgE	0.0
8	15-05-2020 00:00	4135001	ZoEaEvLYb1n2sOq	0.0
9	15-05-2020 00:00	4135001	adLQvlD726eNBSB	0.0
10	15-05-2020 00:00	4135001	bvBOhCH3iADSZry	0.0
11	15-05-2020 00:00	4135001	iCRJl6heRkivqQ3	0.0
12	15-05-2020 00:00	4135001	ih0vzX44oOqAx2f	0.0
13	15-05-2020 00:00	4135001	pkci93gMrogZuBj	0.0
14	15-05-2020 00:00	4135001	rGa61gmuvPhdLxV	0.0
15	15-05-2020 00:00	4135001	sjndEbLyjtCKgGv	0.0
16	15-05-2020 00:00	4135001	uHbuxQJl8lW7ozc	0.0
17	15-05-2020 00:00	4135001	wCURE6d3bPkepu2	0.0
18	15-05-2020 00:00	4135001	z9Y9gH1T5YWrNuG	0.0
19	15-05-2020 00:00	4135001	zBIq5rxdHJRwDNY	0.0
20	15-05-2020 00:00	4135001	zVJPv84UY57bAof	0.0
21	15-05-2020 00:15	4135001	1BY6WEcLGh8j5v7	0.0
22	15-05-2020 00:15	4135001	1IF53ai7Xc0U56Y	0.0

[11]:

```
gen_2[['DATE_TIME','PLANT_ID','INVERTER_ID','DC_POWER']].head(23)
```

[11]:

	DATE_TIME	PLANT_ID	INVERTER_ID	DC_POWER
0	2020-05-15 00:00:00	4136001	4UPUqMRk7TRMgml	0.0
1	2020-05-15 00:00:00	4136001	81aHJ1q11NBPMrL	0.0
2	2020-05-15 00:00:00	4136001	9kRcWv60rDACzjR	0.0
3	2020-05-15 00:00:00	4136001	Et9kgGMDl729KT4	0.0
4	2020-05-15 00:00:00	4136001	IQ2d7wF4YD8zU1Q	0.0
5	2020-05-15 00:00:00	4136001	LYwnQax7tkwH5Cb	0.0
6	2020-05-15 00:00:00	4136001	LlT2YUhhzqhg5Sw	0.0
7	2020-05-15 00:00:00	4136001	Mx2yZCDsyf6DPfv	0.0
8	2020-05-15 00:00:00	4136001	NgDl19wMapZy17u	0.0
9	2020-05-15 00:00:00	4136001	PeE6FRyGXUgsRhN	0.0
10	2020-05-15 00:00:00	4136001	Qf4GUc1pJu5T6c6	0.0
11	2020-05-15 00:00:00	4136001	Quc1TzYxW2pYoWX	0.0
12	2020-05-15 00:00:00	4136001	V94E5Ben1TlhnDV	0.0
13	2020-05-15 00:00:00	4136001	WcxssY2VbP4hApt	0.0
14	2020-05-15 00:00:00	4136001	mqwcsP2rE7J0TFp	0.0
15	2020-05-15 00:00:00	4136001	oZ35aAeoifZaQzV	0.0
16	2020-05-15 00:00:00	4136001	oZZkBaNadn6DNKz	0.0
17	2020-05-15 00:00:00	4136001	q49J1IKaHRwDQnt	0.0
18	2020-05-15 00:00:00	4136001	rrq4fwE8jgrTyWY	0.0

```
19   2020-05-15 00:00:00   4136001  vOuJvMaM2sgwLmb       0.0
20   2020-05-15 00:00:00   4136001  xMbIugepa2P7lBB       0.0
21   2020-05-15 00:00:00   4136001  xoJJ8DcxJEcupym       0.0
22   2020-05-15 00:15:00   4136001  4UPUqMRk7TRMgml       0.0
```

- There are 22 inverters active inverters for each plant.
- After an initial inspection of both plant data, there seem to be missing rows. For plant1 at '15-05-2020 00:00' there are only 21 rows out of the expected 22, indicating that there is a missing inverter.
- The total number of data entries for plant1 and plant2 do not match. Considering the data has been collected over the same period (34 days), and that both plants have 22 inverters, this should not be the case.

[12]:
```
#Formating DATE_TIME from object to datetime.
gen_1['DATE_TIME'] = pd.to_datetime(gen_1['DATE_TIME'],format ='%d-%m-%Y %H:%M')
gen_2['DATE_TIME'] = pd.to_datetime(gen_2['DATE_TIME'],format ='%Y-%m-%d %H:%M:
 ↪%S')
start_date =gen_1['DATE_TIME'].min()
end_date = gen_1['DATE_TIME'].max()
```

3 Missing inverter data

[13]:
```
print('Gen_1 unique inverters')
print('\n')
inv_freq1 = gen_1['INVERTER_ID'].value_counts()
print(inv_freq1.tail())
m_pct = (1-(inv_freq1/3264))*100
print('Mean % missing data per inverter',round(m_pct.mean(),1))
```

```
Gen_1 unique inverters


zBIq5rxdHJRwDNY    3119
1IF53ai7Xc0U56Y    3119
WRmjgnKYAwPKWDb    3118
3PZuoBAID5Wc2HD    3118
YxYtjZvoooNbGkE    3104
Name: INVERTER_ID, dtype: int64
Mean % missing data per inverter 4.2
```

[14]:
```
print('Gen_2 unique inverters')
print('\n')
inv_freq2 = gen_2['INVERTER_ID'].value_counts()
print(inv_freq2.tail())
m_pct2 = (1-(inv_freq2/3264))*100
print('Mean % missing data per inverter',round(m_pct2.mean(),1))
```

```
print('Mean % missing data per inverter w/ lowest four',round((m_pct2.head(18)).
 ↪mean(),1))
```

```
Gen_2 unique inverters


Quc1TzYxW2pYoWX    3195
mqwcsP2rE7J0TFp    2355
IQ2d7wF4YD8zU1Q    2355
NgDl19wMapZy17u    2355
xMbIugepa2P7lBB    2355
Name: INVERTER_ID, dtype: int64
Mean % missing data per inverter 5.7
Mean % missing data per inverter w/ lowest four 0.8
```

- It seems that the amount of missing inverter data is much larger than I had initially thought. My initial thought was that a few culprit inverters were not functioning properly, causing the disparity in data. However, it seems that most if not all the inverters are missing at least some data.
- To understand the extent of the problem i want to know how many data entries there should be for each inverter.

"*Collection methodology Power generation and sensor data gathered at 15 minutes intervals over 34 days*"

There should be 4 data entries per hour for each inverter. With 24 hours in a day for 34 days, equals a total of 816 hours.

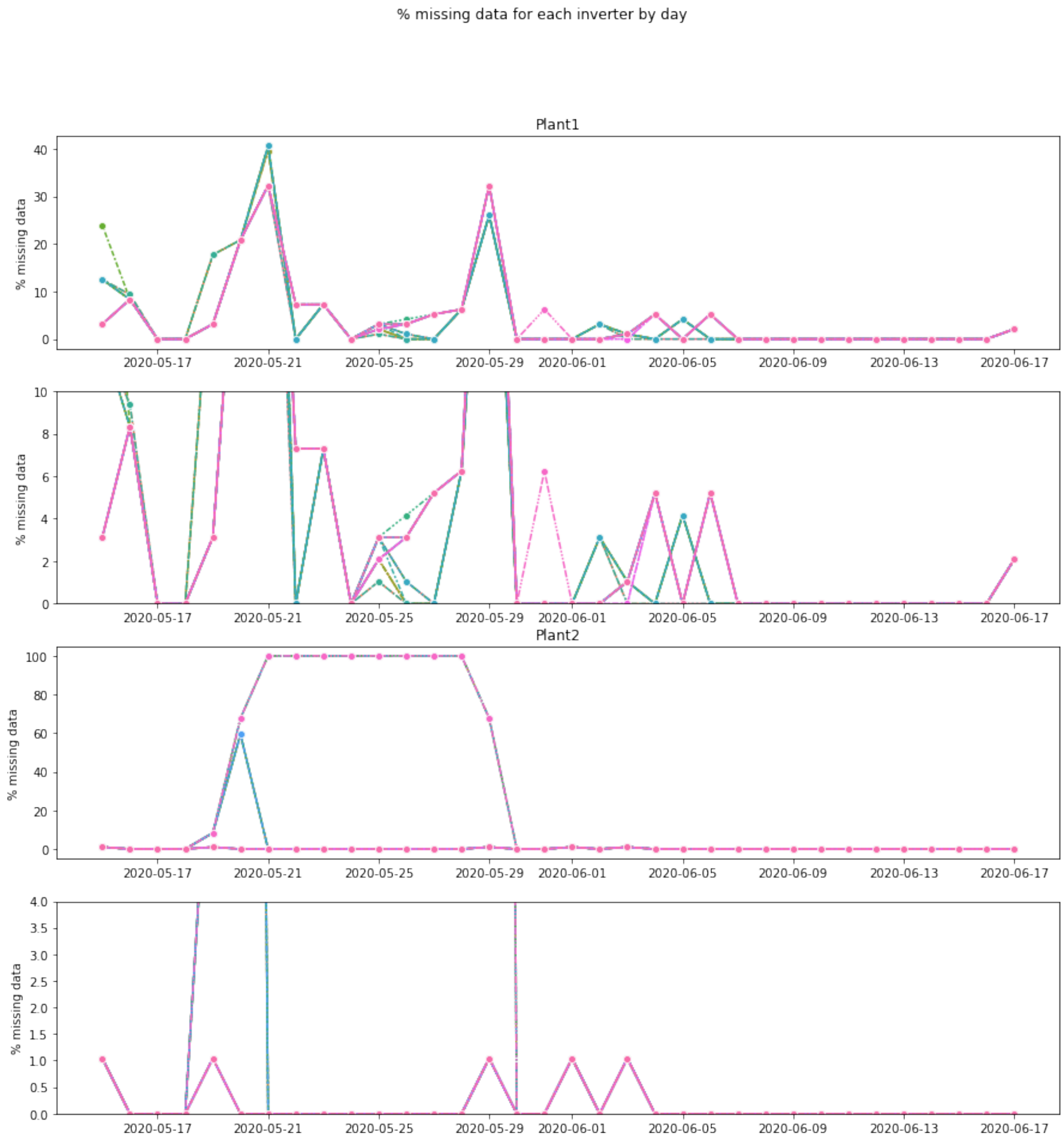
$816 * 4 = 3264$

- None of the inverters matches this number, However, most are close enough except for 4. these four inverters from **gen_2** are far below 3,264.

```
mqwcsP2rE7J0TFp    2355
NgDl19wMapZy17u    2355
IQ2d7wF4YD8zU1Q    2355
xMbIugepa2P7lBB    2355
```

[16]:
```
r = {}
ax = {}
pos = [0,0,1,1]
n_rows = 4
n_cols = 1

r[0] = groupby_inv_date(gen_1,'24h',True)
r[1] = groupby_inv_date(gen_2,'24h',True)
r[0].index.rename('',inplace=True)
r[1].index.rename('',inplace=True)
```

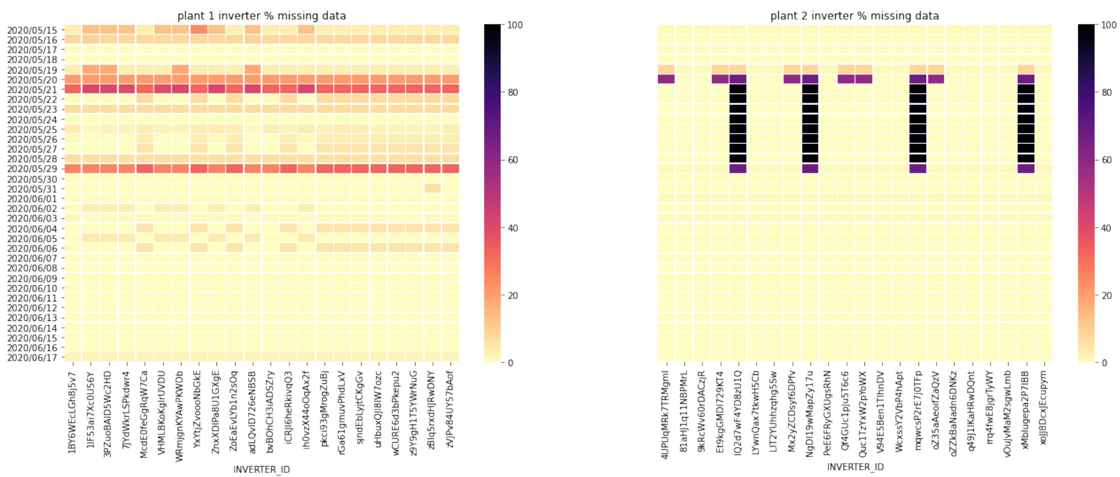
```
r[0]=(1-(r[0]/96))*100
r[1]=(1-(r[1]/96))*100

fig,axes = plt.subplots(n_rows,n_cols,figsize=(15,15))
fig.suptitle('% missing data for each inverter by day')

for x,a in enumerate(range(0,n_rows*n_cols)):
    x=x%2
    ax[a] =sns.lineplot(ax=axes[a],data=r[pos[a]],legend=False,marker='o')
    ax[a].set_ylabel('% missing data')

    if x ==False:
        #ax[a].tick_params(axis='x',labelbottom=False,bottom=False)
        ax[a].set_title('Plant{}'.format(pos[a]+1))

ax[3].set_ylim(0,4)
ax[3].margins(x=0.05,y=-0.25)
ax[1].set_ylim(0,10)

plt.show()
```



The above graphs each plot all 22 inveters

- What's interesting is that within each plant, the inverters seem to follow a very similar pattern of missing data. Initially, I plotted each plant in groups of 4 inverters so that each inverter could be seen and identified. However, this seemed redundant after seeing how similar the pattern of missing data was between them.
- In Fig1 there are two days in particular, where all the inverters had a significantly lower inverter count for that day than usual. I wonder if these low count days could be due to scheduled maintenance.
- For **plant 2** there are 7 days where some inverters do not have any data. I suspect that these inverters are the four that I flagged earlier for missing data.

```
[17]: h = groupby_inv_date(gen_1,'1d',True)
h2 =groupby_inv_date(gen_2,'1d',True)

fig, axes = plt.subplots(1,2,figsize=(22,7))
ax = sns.heatmap(r[0],ax=axes[0],cbar=True,linewidth=0.
↪5,vmax=100,cmap='magma_r')
ax2 = sns.heatmap(r[1],ax=axes[1],yticklabels=False,linewidth=0.
↪5,cmap='magma_r')

ax.set_title('plant 1 inverter % missing data')
ax2.set_title('plant 2 inverter % missing data')

ax.set_yticklabels(pd.date_range(start_date,end_date,freq='d').strftime('%Y/%m/
↪%d'))

plt.show()
```



Using a heat map the difference in missing data between plant 1 and plant 2 is more comparable.

- Plant 1 has a higher occurrence of missing data but at lower levels. The two valleys in the graph can be seen here too by the two horizontal red lines.
- Plant_2 has fewer occurrences of significant missing data but at a much higher level. When data is missing it is very structured in its time and levels.
- the same 4 inverters from **plant 2** with the lowest count did not record any data between the same 7 days period from the 21st to the 28th of may.

After looking at both the line graphs and the heat map, It could be possible especially for plant 2 that the missing data could be due to maintenance, as opposed to error or hardware malfunction. For plant_1 I am more uncertain due to the low-level spread of missing data. Despite this, there are still patterns of missing data where large quantities of inverters are missing substantial amounts of data.

4 Power Output

4.1 AC/DC power

[18]:
```
gen_1[['AC_POWER','DC_POWER']].describe()
```

[18]:

	AC_POWER	DC_POWER
count	68778.000000	68778.000000
mean	307.802752	3147.426211
std	394.396439	4036.457169
min	0.000000	0.000000
25%	0.000000	0.000000
50%	41.493750	429.000000
75%	623.618750	6366.964286
max	1410.950000	14471.125000

[19]:
```
gen_2[['AC_POWER','DC_POWER']].describe()
```

[19]:

	AC_POWER	DC_POWER
count	67698.000000	67698.000000
mean	241.277825	246.701961
std	362.112118	370.569597
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	438.215000	446.591667
max	1385.420000	1420.933333

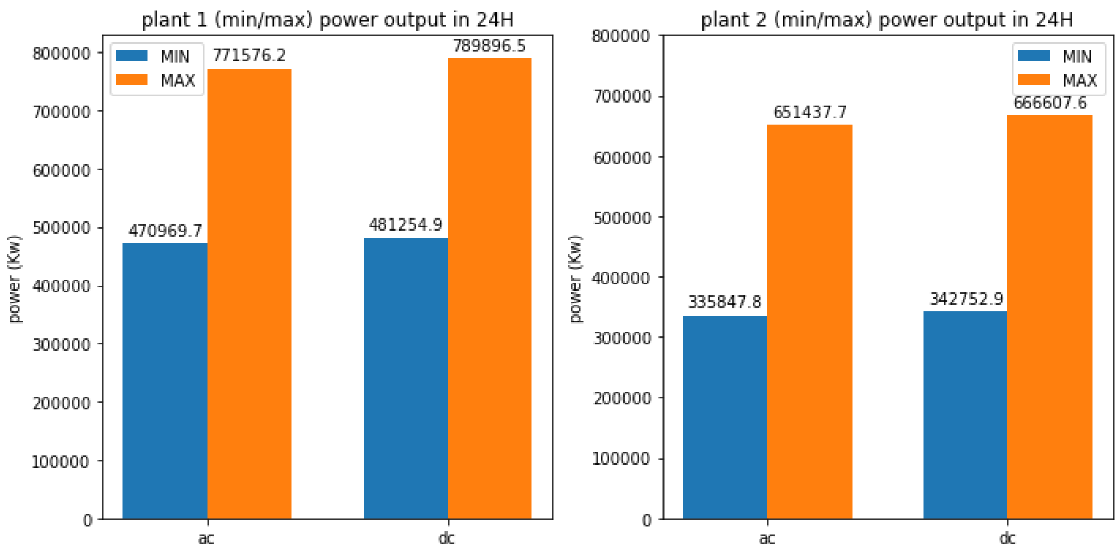
- **Plant 1** DC_POWER appears to be roughly 10x that of AC_POWER. After confirming that plant_2 DC_POWER did not have show similar results i feel confidant that this is due to error.
- *talk about how ac and dc are meant to be comparable*

[20]:
```
gen_1['DC_POWER'] = gen_1['DC_POWER']/10
```

4.1.1 Maximum/Minimum power generated in 24 hours

[21]:
```
c = ['AC_POWER','DC_POWER']
pwr1,pwr2 =␣
 ↪groupby_power(gen_1,freq='1d',cols=c,agg_m='sum',multi_index=False,df2=gen_2,reset_i=False)
m1 = pwr1
m2 = pwr2

pwr1 = pwr1.agg(['max','min'])
pwr2 = pwr2.agg(['max','min'])
pwr = pwr1.append(pwr2)
```

[22]:
```python
labels = ['ac', 'dc']

title = 'plant 1'
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(1,2,figsize=(10,5))
rects1 = ax[0].bar(x - width/2, pwr1.iloc[1], width, label='MIN')
rects2 = ax[0].bar(x + width/2,  pwr1.iloc[0], width, label='MAX')

rects3 = ax[1].bar(x - width/2, pwr2.iloc[1], width, label='MIN')
rects4 = ax[1].bar(x + width/2,  pwr2.iloc[0], width, label='MAX')


# Add some text for labels, title and custom x-axis tick labels, etc.
ax[1].set_ylim(0,800000)

for i in range(0,2):
    ax[i].set_ylabel('power (Kw)')
    ax[i].set_title('plant {}'.format(i+1)+' (min/max) power output in 24H')
    ax[i].set_xticks(x)
    ax[i].set_xticklabels(labels)
    ax[i].legend()

def autolabel(rects,i=0):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax[i].annotate('{}'.format(round(height,1)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)
autolabel(rects3,i=1)
autolabel(rects4,i=1)
fig.tight_layout()

plt.show()
autolabel(rects2)
v = round(m1.mean()/m2.mean(),1)
print('plant_1 ac/dc =',v[0],'* Daily output of plant 2 on average.')
```

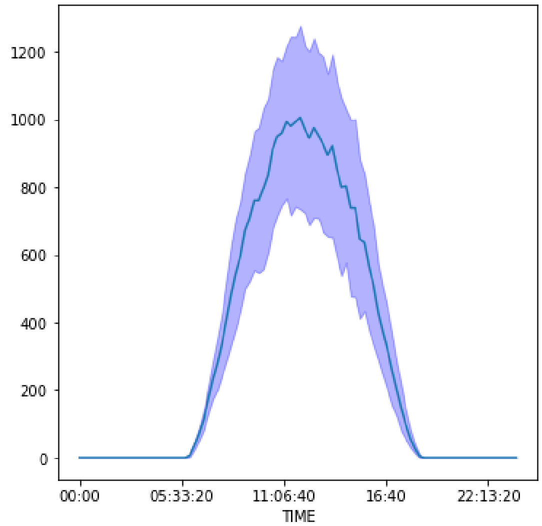


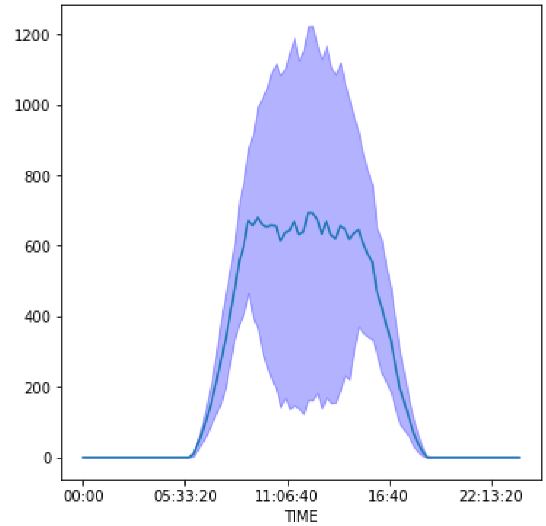

```
plant_1 ac/dc = 1.3 * Daily output of plant 2 on average.
```

- Over the 34 days, it seems that **plant 1** outperformed **plant 2**, Having both a higher min 24h power output and higher max 24h power output for both ac power and dc power.
- **Plant 1** on average produced 1.3* that of **plant 2's** power output by day.

4.1.2 Power output over the course of a day.

[23]:
```
g1,g2 = slice_df(['DATE_TIME','INVERTER_ID','DC_POWER'])

dc1 = split_date(g1)
dc2 = split_date(g2)
Generate_sd_mean(dc1,dc2,'DC_POWER')
plt.show()
```

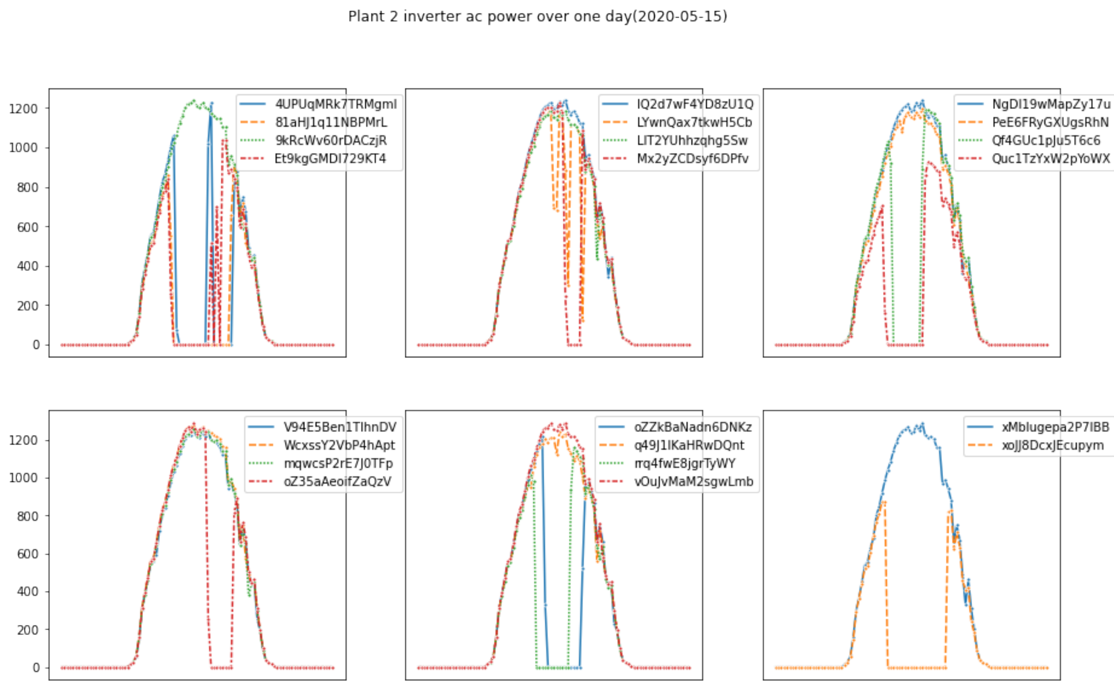




- Max power output is reached at around mid day for both plant 1 and and plant 2.
- The deivation in power output is around mid day is far more significant in **PLANT 2**, this gives plot 2 the cut off apperance seen by the blue line **average**.
- **plant 2** may be having hardware issues converting power, whats interesting is that is huge deviation only seems to happend at certain period of time.
- This gives more information on why **plant 2** had a lower 24 hour max and min output compared to **plant 1**

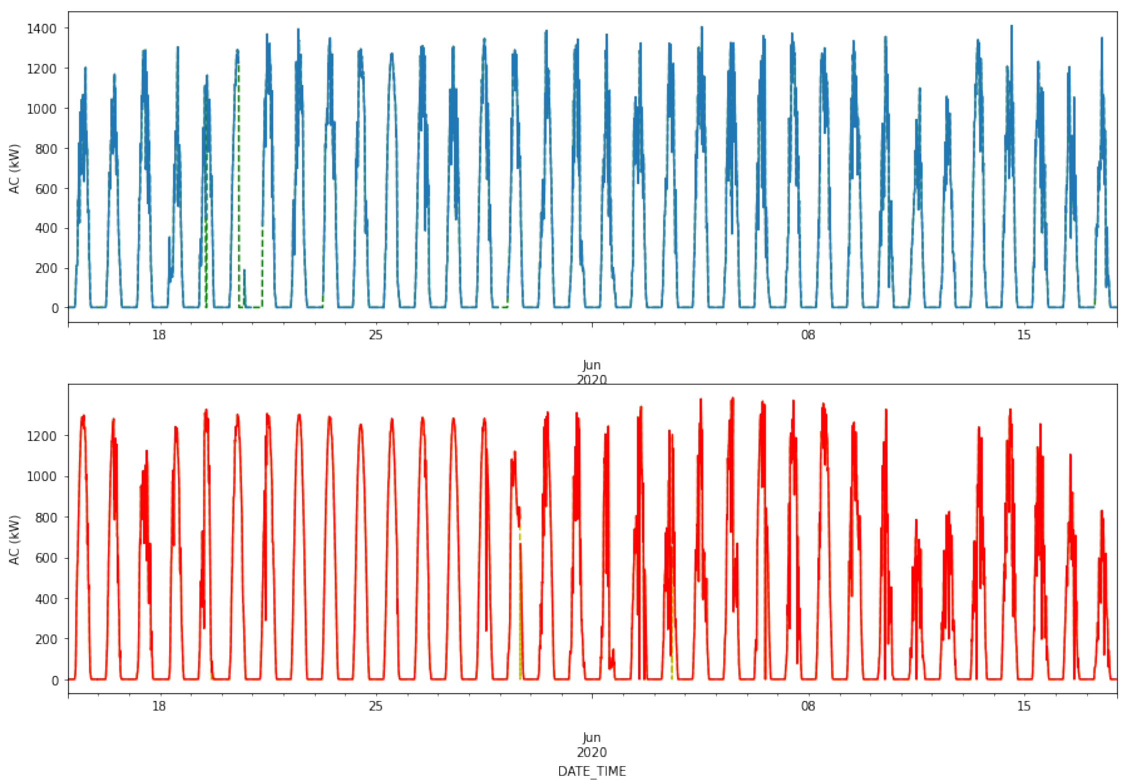
[24]:
```
x =groupby_inv_date(gen_2,freq='15T',col='AC_POWER',agg_m='mean',fillna=True)
x = x[x.index <= '2020-05-15 23:45']
x =split_date(x,h=False,reset=True)

fig,axes = plt.subplots(2,3,figsize=(15,9))
fig.suptitle('Plant 2 inverter ac power over one day(2020-05-15)')

ax0 = sns.lineplot(ax=axes[0,0],data=x.iloc[:,1:5],legend=True,marker='.')
ax1 = sns.lineplot(ax=axes[0,1],data=x.iloc[:,5:9],legend=True,marker='.')
ax2 = sns.lineplot(ax=axes[0,2],data=x.iloc[:,9:13],legend=True,marker='.')

ax3= sns.lineplot(ax=axes[1,0],data=x.iloc[:,13:17],legend=True,marker='.')
ax4= sns.lineplot(ax=axes[1,1],data=x.iloc[:,17:21],legend=True,marker='.')
ax5= sns.lineplot(ax=axes[1,2],data=x.iloc[:,21:],legend=True,marker='.')

ax_list = [ax1,ax2,ax4,ax5]
xaxis = [ax0,ax1,ax2,ax3,ax4,ax5]
for i in xaxis:
    i.tick_params(axis='x',labelbottom=False,bottom=False)
    i.legend(loc='upper right',bbox_to_anchor=(1.21,1))
```

```python
for i in ax_list:
    i.tick_params(axis='y',labelleft=False,left=False)
```



[25]:
```python
g1_t15= gen_1.groupby(pd.Grouper(freq='15T',key='DATE_TIME'))
g2_t15= gen_2.groupby(pd.Grouper(freq='15T',key='DATE_TIME'))

gen1_t15_ac = g1_t15['AC_POWER'].max()
gen1_t15_dc = g1_t15['DC_POWER'].max()

gen2_t15_ac = g2_t15['AC_POWER'].max()
gen2_t15_dc = g2_t15['DC_POWER'].max()

g1_day_ac = gen1_t15_ac[(gen1_t15_ac.index >='15-05-2020 00:00')&(gen1_t15_ac.
 ↪index <'2020-06-17 23:45:00')]
g1_day_ac_smoothed = g1_day_ac.fillna(0)

#g1_day_dc = gen1_t15_dc[(gen1_t15_dc.index >='15-05-2020 00:00')&(gen1_t15_dc.
 ↪index <'17-05-2020 23:45')]
#g1_day_dc_smoothed = g1_day_dc.fillna(0)

g2_day_ac = gen2_t15_ac[(gen1_t15_ac.index >='15-05-2020 00:00')&(gen1_t15_ac.
 ↪index <'2020-06-17 23:45:00')]
g2_day_ac_smoothed = g2_day_ac.fillna(0)
```

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


ax1 = g1_day_ac_smoothed.plot(ax=axes[0],figsize=(15,10),c='g',ls='--')
ax1 = g1_day_ac.plot(ax=axes[0],figsize=(15,10))

ax2 = g2_day_ac_smoothed.plot(ax=axes[1],figsize=(15,10),c='y',ls='--')
ax2 = g2_day_ac.plot(ax=axes[1],figsize=(15,10),c='r')
plt.plot()

ax2.set_yticks([0,200,400,600,800,1000,1200])

ax1.set_ylabel('AC (kW)')
ax2.set_ylabel('AC (kW)')
ax1.set_xlabel('')
plt.show()
#ax2 = g1_day_dc_smoothed.plot(ax=axes[1],figsize=(15,10),c='g',ls='--')
#ax2 = g1_day_dc.plot(ax=axes[1],figsize=(15,10))
```



4.2 MTFB

If we assume that the missing data is because of malfunctioning hardware we can assign a score to each inverter.

MTBF is a basic measure of an asset's reliability. It is calculated by dividing the total operating time of the asset by the number of failures over a given period of time. Taking the example of the AHU above, the calculation to determine MTBF is: 3,600 hours divided by 12 failures. The result is 300 operating hours.

p = [] c=-1 for v in mtfb: c=c+1 inv = [] for i in range(0,24): inv.append(np.exp(-((1/int(mtfb[c]))*i))) p.append(inv)

for i in p: plt.plot(i)

handles, labels = plt.gca().get_legend_handles_labels() h = [] h.append(handles[0]) h.append(handles[-1])