Data Gathering Date and Location

Location: Lapu-Lapu City, Cebu, Philippines

Date: May 19, 2022

In [1]:

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Data Import and Display

In [2]:

```
#Importing and Displaying Data
headers = ["DateTime (dd/mm/yyyyy hh:mm:ss)", "Servo Lower Angle (°)","Servo Upper Angl
          "Roll Angle (°)", "Pitch Angle (°)", "Humidity (%)", "Temperature (°C)", "Hea
          "Irradiance (W/m2)", "Voltage (V)", "Current (mA)"]
data = pd.read_csv("DATA 19_05_22.txt", header=None, index_col=False,
    names=headers, parse_dates= ["DateTime (dd/mm/yyyyy hh:mm:ss)"])
display(data.head())
print("Data Total Rows: {}\nData Total Columns: {}".format(data.shape[0], data.shape[1]
```

	DateTime (dd/mm/yyyyy hh:mm:ss)	Servo Lower Angle (°)	Servo Upper Angle (°)	Roll Angle (°)	Pitch Angle (°)	Humidity (%)	Temperature (°C)	Heat Index (°C)	Irradiance (W/m2)	Voltag (V
0	2022-05-19 04:56:00	0	0	15.5504	-66.8076	93.0	27.2	32.2115	0.0	0
1	2022-05-19 04:56:05	0	23	18.0047	-47.4466	93.7	27.2	32.3448	0.0	0
2	2022-05-19 04:56:10	0	46	17.8869	-24.9693	93.8	27.2	32.3639	0.0	0
3	2022-05-19 04:56:15	0	69	20.1395	-6.8439	93.7	27.1	32.0455	0.0	0
4	2022-05-19 04:56:20	0	92	18.2653	16.4409	93.8	27.1	32.0641	0.0	0

```
Data Total Rows: 10246
Data Total Columns: 11
```

In [3]:

```
data.describe()
```

Out[3]:

	Servo Lower Angle (°)	Servo Upper Angle (°)	Roll Angle (°)	Pitch Angle (°)	Humidity (%)	Temperature (°C)	Heat Inde (°C
count	10246.000000	10246.000000	10246.000000	10246.000000	10100.000000	10100.000000	10100.00000
mean	89.705739	45.988776	2.932538	-27.246695	59.662436	38.137990	51.71731
std	63.702569	32.527704	12.180201	33.273733	21.728205	8.727385	16.12592
min	0.000000	0.000000	-15.914300	-88.719000	28.800000	25.600000	27.13560
25%	45.000000	23.000000	-7.956250	-58.117750	38.800000	30.400000	37.44500
50%	90.000000	46.000000	2.863500	-26.004200	57.600000	37.100000	48.98315
75%	135.000000	69.000000	14.529500	3.724350	75.100000	46.200000	67.53632
max	180.000000	92.000000	21.027800	27.078000	99.900000	56.100000	86.33320

In [4]:

```
display(pd.DataFrame(data.isna().sum(), columns=["Total Nan Values"]).T)
```

	DateTime (dd/mm/yyyyy hh:mm:ss)	Servo Lower Angle (°)	Servo Upper Angle (°)	Roll Angle (°)	Pitch Angle (°)	Humidity (%)	Temperature (°C)	Heat Index (°C)	Irradiance (W/m2)	Voltag (\
Total Nan Values	0	0	0	0	0	146	146	146	0	1

~ The nan/missing values from the "Humidity", "Temperature", and "Heat Index" columns are caused by the low sampling rate capability of the sensor and the library used which is expected.

~ The nan/missing values from the "Voltage" and "Current" columns mostly occur when the sun is down or no current was produced from the solar panel, the developers of the product ensures that quality data is being recorded so a Nan value was used to record instead for this scenario.

Data Cleaning

~ Imputation process for "Temperature", "Humidity", and "Heat Index" columns is to find the average of the nearest previous not-Nan value and nearest next not-Nan value since these values don't really much differ from each other within a minute of timeframe.

In [5]:

```
col_to_clean = ["Humidity (%)", "Temperature (°C)", "Heat Index (°C)"]
display(data[col_to_clean])
```

	Humidity (%)	Temperature (°C)	Heat Index (°C)
0	93.0	27.2	32.2115
1	93.7	27.2	32.3448
2	93.8	27.2	32.3639
3	93.7	27.1	32.0455
4	93.8	27.1	32.0641
...	...	...	...
10241	91.9	27.5	32.8921
10242	91.8	27.4	32.5728
10243	91.8	27.4	32.5728
10244	91.9	27.4	32.5927
10245	92.0	27.4	32.6126

10246 rows × 3 columns

In [6]:

```
#Find closest previous value that is not a Nan value relative to the row where a Nan va
def FindPreviousNotNan(current_row, pd_series): #
    i=current_row-1
    val = np.nan
    pd_series_min_pos = 0
    while i>=pd_series_min_pos:
        if np.isnan(pd_series.values[i]):
            i-=1
        else:
            val = pd_series.values[i]
            break
    return val

#Find closest next value that is not a Nan value relative to the row where a Nan value
def FindNextNotNan(current_row, pd_series):
    i=current_row+1
    val=np.nan
    pd_series_max_pos = len(pd_series)-1
```

```
    while i<=pd_series_max_pos:
        if np.isnan(pd_series.values[i]):
            i+=1
        else:
            val = pd_series.values[i]
            break
    return val
```

In [7]:

```
#Impute rows where Nan values are found
#It will get the average of previous and next row values that are not Nan
#If all of the previous row values are Nan but next row value is not Nan then the value
def ImputeColumn(pd_dataframe, column):
    for i in range(len(pd_dataframe)):
        current_val = pd_dataframe[column].values[i]
        previous_val = 0
        next_val = 0
        if np.isnan(current_val):
            previous_val = FindNextNotNan(i,pd_dataframe[column])
            next_val = FindPreviousNotNan(i,pd_dataframe[column])
            if not np.isnan(previous_val) and not np.isnan(next_val):
                pd_dataframe[column].values[i] = np.mean([previous_val,next_val])
            elif np.isnan(previous_val) and np.isnan(next_val):
                pd_dataframe[column].values[i] = np.nan
            else:
                if np.isnan(previous_val):
                    pd_dataframe[column].values[i] = next_val
                else:
                    pd_dataframe[column].values[i] = previous_val
```

In [8]:

```
print("Sample Rows Before Data Imputation")
display(data.loc[139:141,col_to_clean])
display(data.loc[199:201,col_to_clean])

for col in col_to_clean:
    ImputeColumn(data, col)

display(pd.DataFrame(data.isna().sum(), columns=["Total Nan Values"]).T)
print("Sample Rows After Data Imputation")
display(data.loc[139:141,col_to_clean])
display(data.loc[199:201,col_to_clean])
```

Sample Rows Before Data Imputation

	Humidity (%)	Temperature (°C)	Heat Index (°C)
139	97.5	25.8	27.7138
140	NaN	NaN	NaN
141	97.6	25.8	27.7161

	Humidity (%)	Temperature (°C)	Heat Index (°C)
199	98.4	25.7	27.4298
200	NaN	NaN	NaN
201	98.5	25.7	27.4313

	DateTime (dd/mm/yyyyy hh:mm:ss)	Servo Lower Angle (°)	Servo Upper Angle (°)	Roll Angle (°)	Pitch Angle (°)	Humidity (%)	Temperature (°C)	Heat Index (°C)	Irradiance (W/m2)	Voltag (\
Total Nan Values	0	0	0	0	0	0	0	0	0	1

Sample Rows After Data Imputation

	Humidity (%)	Temperature (°C)	Heat Index (°C)
139	97.50	25.8	27.71380

	Humidity (%)	Temperature (°C)	Heat Index (°C)
140	97.55	25.8	27.71495
141	97.60	25.8	27.71610

	Humidity (%)	Temperature (°C)	Heat Index (°C)
199	98.40	25.7	27.42980
200	98.45	25.7	27.43055
201	98.50	25.7	27.43130

~ Imputation process for "Voltage" and "Current" columns is to find the average of the nearest previous not-Nan value and nearest next not-Nan value (just like the same with the "Humidity", "Temperature", and "Heat Index" columns) since these values don't really much differ from when the sun is down and is more likely to be approximately equal to zero.

In [9]:

```
col_to_clean = ["Voltage (V)", "Current (mA)"]

print("Sample Rows Before Data Imputation")
display(data.loc[132:134,col_to_clean])
display(data.loc[9441:9445,col_to_clean])

for col in col_to_clean:
    ImputeColumn(data, col)

display(pd.DataFrame(data.isna().sum(), columns=["Total Nan Values"]).T)
print("Sample Rows After Data Imputation")
display(data.loc[132:134,col_to_clean])
display(data.loc[9441:9445,col_to_clean])
```

```
Sample Rows Before Data Imputation
```

	Voltage (V)	Current (mA)
132	0.0	0.1
133	NaN	NaN
134	0.0	0.0

	Voltage (V)	Current (mA)
9441	0.0	0.1
9442	NaN	NaN
9443	NaN	NaN
9444	NaN	NaN
9445	0.0	0.0

	DateTime (dd/mm/yyyyy hh:mm:ss)	Servo Lower Angle (°)	Servo Upper Angle (°)	Roll Angle (°)	Pitch Angle (°)	Humidity (%)	Temperature (°C)	Heat Index (°C)	Irradiance (W/m2)	Voltag (\
Total Nan Values	0	0	0	0	0	0	0	0	0	

```
Sample Rows After Data Imputation
```

	Voltage (V)	Current (mA)
132	0.0	0.10
133	0.0	0.05
134	0.0	0.00

	Voltage (V)	Current (mA)
9441	0.0	0.1000
9442	0.0	0.0500
9443	0.0	0.0250
9444	0.0	0.0125
9445	0.0	0.0000

Exploratory Data Analysis

In [10]:

```
data["Power (mW)"] = data["Voltage (V)"]*data["Current (mA)"]
display(data)
```

	DateTime (dd/mm/yyyyy hh:mm:ss)	Servo Lower Angle (°)	Servo Upper Angle (°)	Roll Angle (°)	Pitch Angle (°)	Humidity (%)	Temperature (°C)	Heat Index (°C)	Irradiance (W/m2)	V
0	2022-05-19 04:56:00	0	0	15.5504	-66.8076	93.0	27.2	32.2115	0.0	
1	2022-05-19 04:56:05	0	23	18.0047	-47.4466	93.7	27.2	32.3448	0.0	
2	2022-05-19 04:56:10	0	46	17.8869	-24.9693	93.8	27.2	32.3639	0.0	
3	2022-05-19 04:56:15	0	69	20.1395	-6.8439	93.7	27.1	32.0455	0.0	
4	2022-05-19 04:56:20	0	92	18.2653	16.4409	93.8	27.1	32.0641	0.0	
...	...	...	...	...	...	...	...	...	...	
10241	2022-05-19 19:09:47	0	92	20.5672	24.4719	91.9	27.5	32.8921	0.0	
10242	2022-05-19 19:09:52	45	0	12.3812	-74.3377	91.8	27.4	32.5728	0.0	
10243	2022-05-19 19:09:57	45	23	11.3962	-51.3019	91.8	27.4	32.5728	0.0	
10244	2022-05-19 19:10:02	45	46	14.6790	-24.8913	91.9	27.4	32.5927	0.0	
10245	2022-05-19 19:10:07	45	69	14.2961	-4.0299	92.0	27.4	32.6126	0.0	

10246 rows × 12 columns

In [11]:

```
power_pd = pd.pivot_table(data, values = "Power (mW)", index="Servo Lower Angle (°)", c
power_pd.style.background_gradient(cmap="viridis", axis=None).set_caption("Average Powe
```

Out[11]:

Average Power (mW) Delivered based on Servo Angle Positions

Servo Upper Angle (°)	0	23	46	69	92
Servo Lower Angle (°)					
0	18.339827	25.189814	27.510043	26.540156	22.891166
45	19.411268	25.378231	27.935678	26.228431	21.922363
90	18.139075	24.721523	27.536104	26.373132	22.013466
135	14.883955	23.227739	27.063587	27.014111	23.741312
180	10.729783	19.683897	25.531714	27.449549	26.176851

In [12]:

```
row,col = np.where(power_pd.values==power_pd.values.max())
print("According to data gathered (on the basis of maximum average power delivered),\nt
print("Servo Lower Angle: {}°".format(power_pd.index[row[0]]))
print("Servo Upper Angle: {}°".format(power_pd.columns[col[0]]))
print("Max Average Power: {:.4f}mW".format(power_pd.values.max()))

row,col = np.where(power_pd.values==power_pd.values.min())
print("\n\nMeanwhile the sets of Servo Angle that delivers the lowest average power:\n"
print("Servo Lower Angle: {}°".format(power_pd.index[row[0]]))
print("Servo Upper Angle: {}°".format(power_pd.columns[col[0]]))
print("Min Average Power: {:.4f}mW".format(power_pd.values.min()))
```

```
According to data gathered (on the basis of maximum average power delivered),
the solar panel holder can be designed on these following servo angles:

Servo Lower Angle: 45°
Servo Upper Angle: 46°
Max Average Power: 27.9357mW


Meanwhile the sets of Servo Angle that delivers the lowest average power:

Servo Lower Angle: 180°
Servo Upper Angle: 0°
Min Average Power: 10.7298mW
```

In [13]:

```
#To see what are the corresponding Pitch and Roll angles of each sets/pair of servo ang
# This can be used as the basis to design the inclination of the solar panel holders, bu
# must also take into consideration the inclincation of the base/roof/ground where the s
pitch_roll_pd = pd.pivot_table(data, values =["Roll Angle (°)", "Pitch Angle (°)"], ind
pitch_roll_pd.style.set_caption("Average Roll and Pitch Angles based on Servo Angle Pos
```

Out[13]:

Average Roll and Pitch Angles based on Servo Angle Positions

	Pitch Angle (°)								
Servo Upper Angle (°)	**0**	**23**	**46**	**69**	**92**	**0**	**23**	**46**	
Servo Lower Angle (°)									
0	-62.504728	-39.239285	-16.911226	4.182781	23.818237	16.916821	18.558999	19.309349	1
45	-75.494604	-53.981351	-27.755364	-5.923742	14.575541	11.685777	12.028940	13.852591	1
90	-84.652065	-60.413839	-33.468829	-11.329598	11.276907	1.419187	1.955682	2.859811	
135	-78.251013	-56.504445	-29.341165	-7.318470	14.308224	-8.691554	-7.836315	-6.472952	-
180	-67.439975	-44.399341	-18.919719	1.648204	22.664773	-14.872325	-14.438797	-13.957116	-1

In [14]:

```
print("Pitch Angle: {:.4f}°".format(pitch_roll_pd['Pitch Angle (°)'][46][45])) #Type of
print("Roll Angle: {:.4f}°".format(pitch_roll_pd['Roll Angle (°)'][46][45])) #Type of A
```

```
Pitch Angle: -27.7554°
Roll Angle: 13.8526°
```

In [15]:

```
#Now lets look if the servo angles which delivered the max average power will also have
irradiance_pd = pd.pivot_table(data, values = "Irradiance (W/m2)", index="Servo Lower A
irradiance_pd.style.background_gradient(cmap="viridis", axis=None).set_caption("Average
```

Out[15]:

Average Irradiance (W/m2) based on Servo Angle Positions

Servo Upper Angle (°)	**0**	**23**	**46**	**69**	**92**
Servo Lower Angle (°)					
0	87.708637	105.832895	113.995901	112.480507	102.923733

Servo Upper Angle (°)	0	23	46	69	92
Servo Lower Angle (°)					
45	90.463229	106.511786	114.941417	111.826011	99.273751
90	86.736455	104.551842	114.279525	112.167286	99.517700
135	78.990350	99.784179	112.267149	112.665229	103.919787
180	69.733758	91.900020	108.132492	112.527074	108.892678

In [16]:

```
row,col = np.where(irradiance_pd.values==irradiance_pd.values.max())
print("According to data gathered (on the basis of maximum average irradiance),\nthe so
print("Servo Lower Angle: {}°".format(irradiance_pd.index[row[0]]))
print("Servo Upper Angle: {}°".format(irradiance_pd.columns[col[0]]))
print("Max Irradiance: {:.4f}W/m2".format(irradiance_pd.values.max()))

row,col = np.where(irradiance_pd.values==irradiance_pd.values.min())
print("\n\nMeanwhile the sets of Servo Angle that has the lowest average irradiance:\n"
print("Servo Lower Angle: {}°".format(irradiance_pd.index[row[0]]))
print("Servo Upper Angle: {}°".format(irradiance_pd.columns[col[0]]))
print("Min Irradiance: {:.4f}W/m2".format(irradiance_pd.values.min()))
```

```
According to data gathered (on the basis of maximum average irradiance),
the solar panel holder can be designed on these following servo angles:

Servo Lower Angle: 45°
Servo Upper Angle: 46°
Max Irradiance: 114.9414W/m2


Meanwhile the sets of Servo Angle that has the lowest average irradiance:

Servo Lower Angle: 180°
Servo Upper Angle: 0°
Min Irradiance: 69.7338W/m2
```

In [17]:

```
corr_matrix = np.corrcoef(power_pd.values.flatten(),irradiance_pd.values.flatten())
display(pd.DataFrame(corr_matrix, columns=["Power","Irradiance"], index=["Power","Irrad
```

	Power	Irradiance
Power	1.00000	0.99509
Irradiance	0.99509	1.00000

This demonstrates that irradiance and power have a strong relationship. As a result, we must optimize the orientation of our solar panel holder so that it receives the most irradiance on average throughout the day.

In [18]:

```
#Let's visualize the power delivered all throughout the day by the module through movin
data_power_roll = data[["DateTime (dd/mm/yyyyy hh:mm:ss)","Power (mW)"]].copy()
data_power_roll["Power (mW) Roll Ave 25"] = data["Power (mW)"].rolling(25).mean()
data_power_roll["Power (mW) Roll Ave 500"] = data["Power (mW)"].rolling(500).mean()
data_power_roll["Power (mW) Roll Ave 1000"] = data["Power (mW)"].rolling(1000).mean()
data_power_roll.rename(columns={"DateTime (dd/mm/yyyyy hh:mm:ss)":"DateTime"}, inplace=
data_power_roll.set_index("DateTime", inplace=True)
display(data_power_roll.head())
```

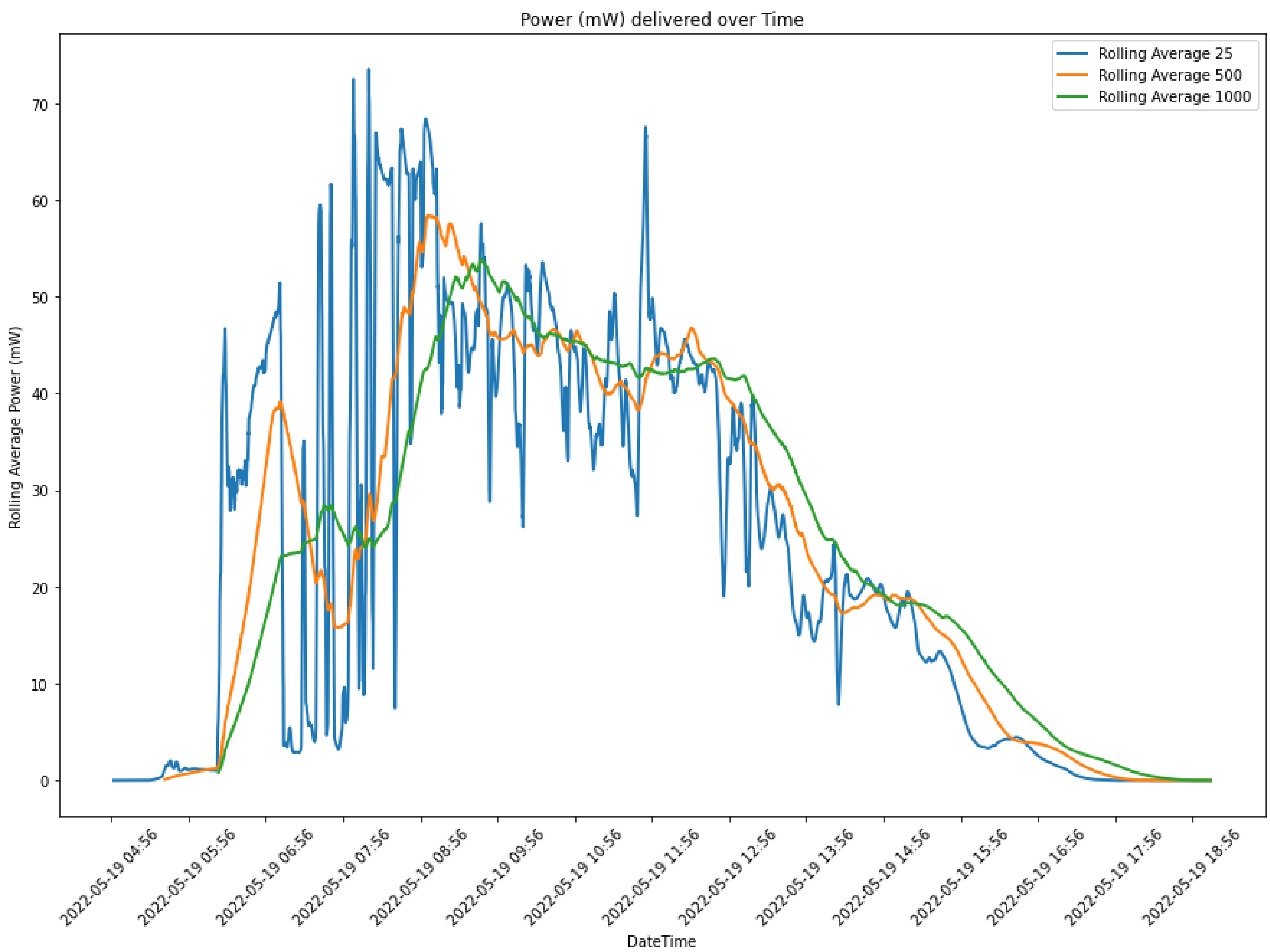
	Power (mW)	Power (mW) Roll Ave 25	Power (mW) Roll Ave 500	Power (mW) Roll Ave 1000
DateTime				
2022-05-19 04:56:00	0.0	NaN	NaN	NaN
2022-05-19 04:56:05	0.0	NaN	NaN	NaN
2022-05-19 04:56:10	0.0	NaN	NaN	NaN

DateTime	Power (mW)	Power (mW) Roll Ave 25	Power (mW) Roll Ave 500	Power (mW) Roll Ave 1000
2022-05-19 04:56:15	0.0	NaN	NaN	NaN
2022-05-19 04:56:20	0.0	NaN	NaN	NaN

In [19]:

```
plt.figure(figsize=(15,10))
x_ticks = pd.date_range(start=data_power_roll.index.min(), end=data_power_roll.index.ma
plt.xticks(x_ticks, x_ticks.strftime("%Y-%m-%d %H:%M"), rotation=45, ha="center")

plt.plot(data_power_roll["Power (mW) Roll Ave 25"], linewidth=2, label="Rolling Average
plt.plot(data_power_roll["Power (mW) Roll Ave 500"], linewidth=2, label="Rolling Averag
plt.plot(data_power_roll["Power (mW) Roll Ave 1000"], linewidth=2, label="Rolling Avera
plt.title("Power (mW) delivered over Time")
plt.xlabel("DateTime")
plt.ylabel("Rolling Average Power (mW)")
plt.legend(loc="best")
plt.show()
```



The fluctuation of data in the line plot of "Rolling Ave 25" higlights how different servo positions will deliver different amounts of power and the fluctuation is also caused by partial shading of the clouds. To visualize this properly, let's create a graph of different pairs of servo position (both from Lower and Upper).

In [20]:

```
col_to_use = ["DateTime (dd/mm/yyyyy hh:mm:ss)", "Servo Lower Angle (°)","Servo Upper A
servo_pair_all_list = list()
servo_lower = data["Servo Lower Angle (°)"].unique()
servo_upper = data["Servo Upper Angle (°)"].unique()
print("Servo Lower Angles: ", servo_lower)
print("Servo Upper Angles: ", servo_upper)

for servo_lower_angle in servo_lower:
    servo_pairings = list()
    for servo_upper_angle in servo_upper:
        servo_pair_filtered = data[(data["Servo Lower Angle (°)"]==servo_lower_angle) &
        servo_pair_filtered.rename(columns={"DateTime (dd/mm/yyyyy hh:mm:ss)":"DateTime
```

```
        servo_pair_filtered.set_index("DateTime", inplace=True)
        servo_pairings.append(servo_pair_filtered)
    servo_pair_all_list.append(servo_pairings)

display(servo_pair_all_list[0][0].head(5)) # Servo Lower: 0°, Servo Upper 0°
display(servo_pair_all_list[2][3].head(5)) # Servo Lower: 90°, Servo Upper 69°
```

```
Servo Lower Angles:  [  0  45  90 135 180]
Servo Upper Angles:  [ 0 23 46 69 92]
```

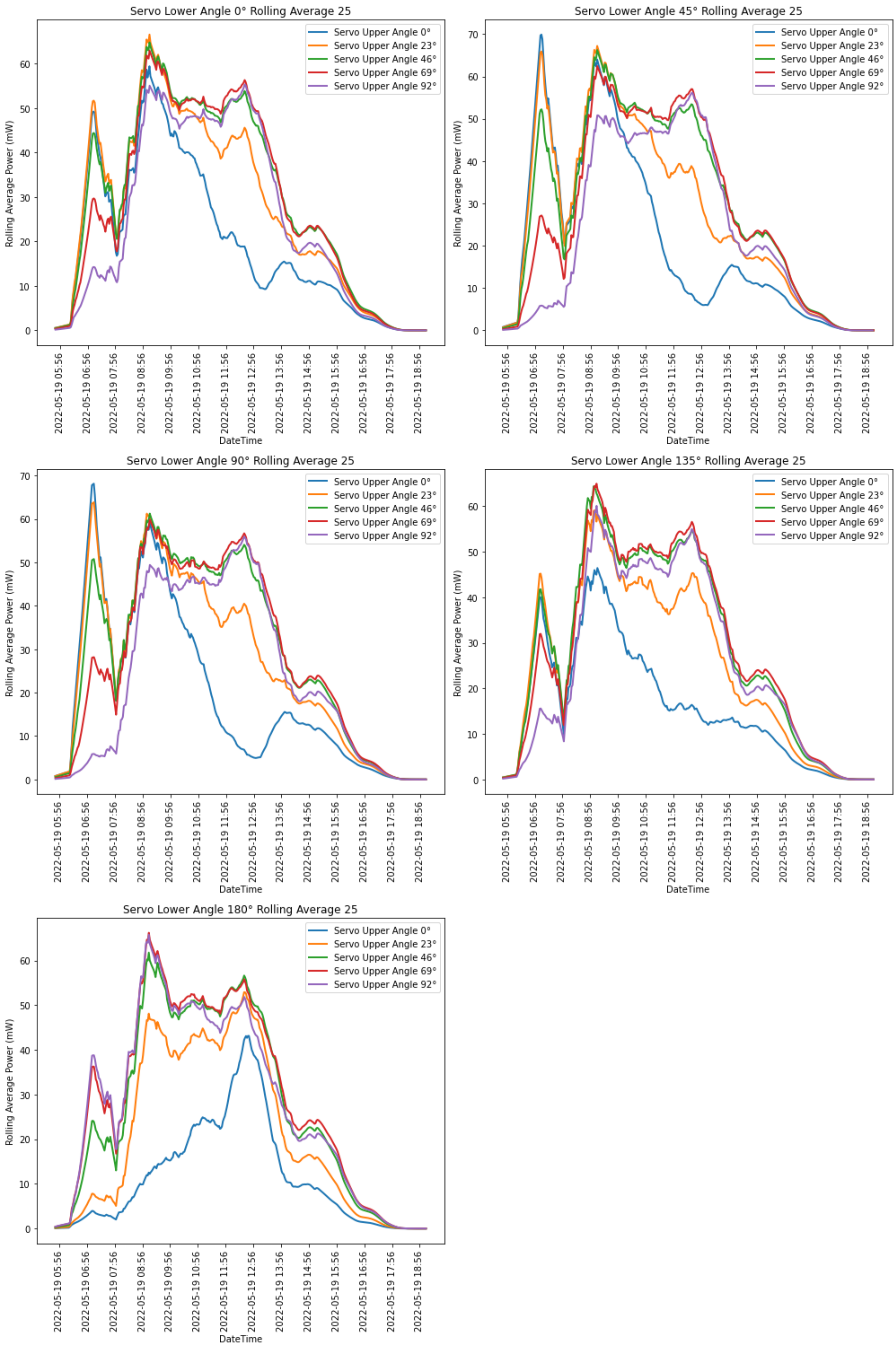
DateTime	Servo Lower Angle (°)	Servo Upper Angle (°)	Irradiance (W/m2)	Power (mW)
2022-05-19 04:56:00	0	0	0.0	0.0
2022-05-19 04:58:05	0	0	0.0	0.0
2022-05-19 05:00:10	0	0	0.0	0.0
2022-05-19 05:02:15	0	0	0.0	0.0
2022-05-19 05:04:20	0	0	0.0	0.0

DateTime	Servo Lower Angle (°)	Servo Upper Angle (°)	Irradiance (W/m2)	Power (mW)
2022-05-19 04:57:05	90	69	0.000	0.0
2022-05-19 04:59:10	90	69	0.000	-0.0
2022-05-19 05:01:15	90	69	0.000	-0.0
2022-05-19 05:03:20	90	69	0.000	0.0
2022-05-19 05:05:25	90	69	0.233	0.0

-- Graphing Rolling Average of Power (mW) of Different Servo Angle Pairs --

In [21]:

```
fig = plt.figure(figsize=(14, 21))
for i in range(len(servo_pair_all_list)):
    ax = fig.add_subplot(3,2,i+1)
    ax.set_title("Servo Lower Angle {}° Rolling Average 25".format(servo_lower[i]))
    ax.set_xlabel("DateTime")
    ax.set_ylabel("Rolling Average Power (mW)")
    ax.set_xticks(x_ticks)
    ax.set_xticklabels(x_ticks.strftime("%Y-%m-%d %H:%M"))
    plt.setp(ax.get_xticklabels(), rotation=90, horizontalalignment='center')
    for j in range(len(servo_pair_all_list[i])):
        ax.plot(servo_pair_all_list[i][j]["Power (mW)"].rolling(25).mean(), linewidth=2
    ax.legend(loc="best")
fig.tight_layout()
plt.show()
```

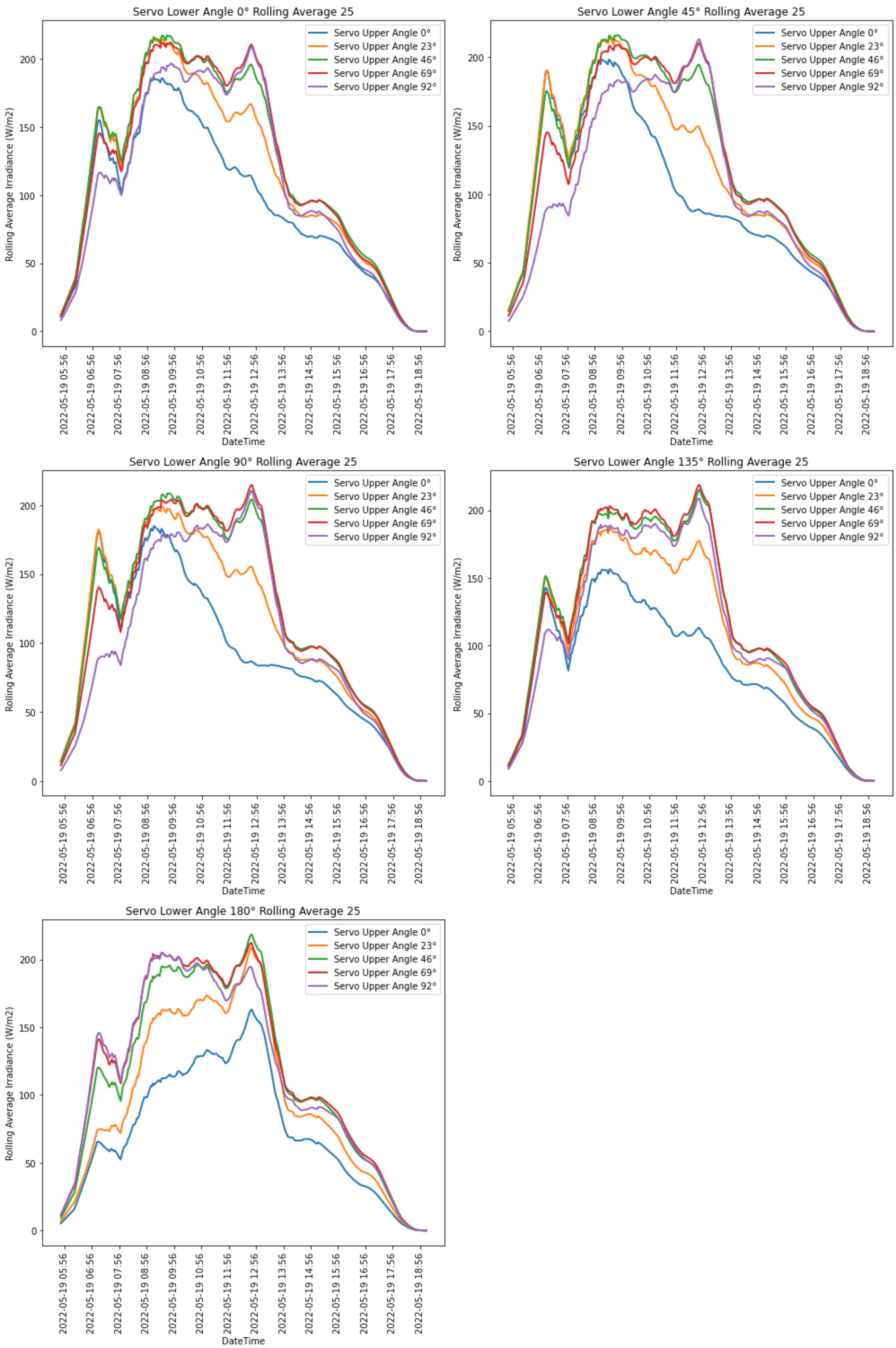


-- Graphing Rolling Average of Irradiance (W/m2) of Different Servo Angle Pairs --

In [22]:

```
fig = plt.figure(figsize=(14, 21))
for i in range(len(servo_pair_all_list)):
    ax = fig.add_subplot(3,2,i+1)
    ax.set_title("Servo Lower Angle {}° Rolling Average 25".format(servo_lower[i]))
    ax.set_xlabel("DateTime")
    ax.set_ylabel("Rolling Average Irradiance (W/m2)")
    ax.set_xticks(x_ticks)
    ax.set_xticklabels(x_ticks.strftime("%Y-%m-%d %H:%M"))
    plt.setp(ax.get_xticklabels(), rotation=90, horizontalalignment='center')
    for j in range(len(servo_pair_all_list[i])):
        ax.plot(servo_pair_all_list[i][j]["Irradiance (W/m2)"].rolling(25).mean(), linew
    ax.legend(loc="best")
```

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



Model Development

In [23]:

```
from sklearn.model_selection import train_test_split

columns_rename=["DateTime","Hour","Minute","Second","Servo Lower","Servo Upper", "Power
data_to_model =pd.concat([data["DateTime (dd/mm/yyyyy hh:mm:ss)"],
                          data["DateTime (dd/mm/yyyyy hh:mm:ss)"].dt.hour,
                          data["DateTime (dd/mm/yyyyy hh:mm:ss)"].dt.minute,
                          data["DateTime (dd/mm/yyyyy hh:mm:ss)"].dt.second,
```

```
                         data.iloc[:,1:3],data["Power (mW)"]], axis=1)
data_to_model.columns = columns_rename
data_to_model.set_index("DateTime", inplace=True)
display(data_to_model.iloc[:,:-1].head().style.set_caption("Predictor Variables to Use"

x_ticks = pd.date_range(start=data_to_model.index.min(), end=data_to_model.index.max(),
X_trainval, X_test, y_trainval, y_test = train_test_split(data_to_model.iloc[:,:-1], da
```

Predictor Variables to Use

DateTime	Hour	Minute	Second	Servo Lower	Servo Upper
2022-05-19 04:56:00	4	56	0	0	0
2022-05-19 04:56:05	4	56	5	0	23
2022-05-19 04:56:10	4	56	10	0	46
2022-05-19 04:56:15	4	56	15	0	69
2022-05-19 04:56:20	4	56	20	0	92

In [24]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error

def grid_search_report(regressor, param_grid, X_trainval, X_test, y_trainval, y_test, x
                       refit="r2", scoring=["r2","neg_mean_absolute_error"], cv=5):
                        #setting default values

    grid = GridSearchCV(regressor, param_grid=param_grid, scoring=scoring, refit=refit,
    grid.fit(X_trainval, y_trainval)
    print("Regression Model used: {}".format(regressor.__class__.__name__ ))
    print("Grid Search Scorer/s used: {}".format(scoring))
    print("Grid Search Scorer used to find best parameters: {}".format(refit))
    print("Best parameters: ", grid.best_params_)
    print("Best cross-validation score ({}): {:.4f}\n...".format(refit,grid.best_score_

    #Predict both X_trainval and X_test for scoring and plotting
    y_trainval_pred=grid.predict(X_trainval)
    y_test_pred=grid.predict(X_test)

    #Print Metrics
    print("R2 Score:")
    print("Train-Validation Set: {:.4f}".format(grid.score(X_trainval, y_trainval)))
    print("Test Set: {:.4f}".format(grid.score(X_test, y_test)))
    print("Mean Absolute Error:")
    print("Train-Validation Set: {:.4f}mW".format(mean_absolute_error(y_trainval, y_tra
    print("Test Set: {:.4f}mW".format(mean_absolute_error(y_test, y_test_pred)))

    #Create dataframe with columns y_trainval_actual and y_trainval_predict with index
        #from X_trainval.index which is the corresponding DateTime
    y_trainval_actual_pred = pd.DataFrame({"Power (mW) Actual":y_trainval.values, "Powe
    #Create dataframe with columns y_test_actual and y_test_predict with index
        #from X_test.index which is the corresponding DateTime
    y_test_actual_pred = pd.DataFrame({"Power (mW) Actual":y_test.values, "Power (mW) P

    #Display DataFrame both from TrainVal set and Test set
    display(y_trainval_actual_pred.head().style.background_gradient(cmap="viridis", axi
            .set_caption("TrainValidation Set: Actual vs. Predicted Power (mW)"))
    display(y_test_actual_pred.head().style.background_gradient(cmap="viridis", axis=0)
            .set_caption("Test Set: Actual vs. Predicted Predicted Power (mW)"))

    #Plot both TrainValidation and Test Set: Actual vs. Predicted
    fig = plt.figure(figsize=(14,7))
    for (subplot, y_set) in zip([1,2],[y_trainval_actual_pred,y_test_actual_pred]):
        ax = fig.add_subplot(1,2,subplot)
        if subplot==1 :
            ax.set_title("TrainValidation Set: Actual vs. Predicted (Rolling Average Pow
        elif subplot == 2:
            ax.set_title("Test Set: Actual vs. Predicted (Rolling Average Power (mW))")
        ax.set_xlabel("DateTime")
        ax.set_ylabel("Rolling Average Power (mW)")
        ax.set_xticks(x_ticks)
```

```python
            ax.set_xticklabels(x_ticks.strftime("%Y-%m-%d %H:%M"))
            plt.setp(ax.get_xticklabels(), rotation=90, horizontalalignment='center')
            #Plot Actual
            ax.plot(y_set.loc[:,"Power (mW) Actual"].sort_index().rolling(25).mean(), linew
            #Plot Predicted
            ax.plot(y_set.loc[:,"Power (mW) Predict"].sort_index().rolling(25).mean(), linew
            ax.legend(loc="best")
        fig.tight_layout()
        plt.show()

    return (grid.best_estimator_, grid.cv_results_) #return best estimator and cv resul
```

In [25]:

```python
from sklearn.ensemble import RandomForestRegressor

rf_param_grid = {"n_estimators":[200,500,700],
                "criterion":["squared_error", "absolute_error"],
                "max_depth":[5,None]}

tup_rf=grid_search_report(RandomForestRegressor(random_state=47, n_jobs=-1), rf_param_g
```

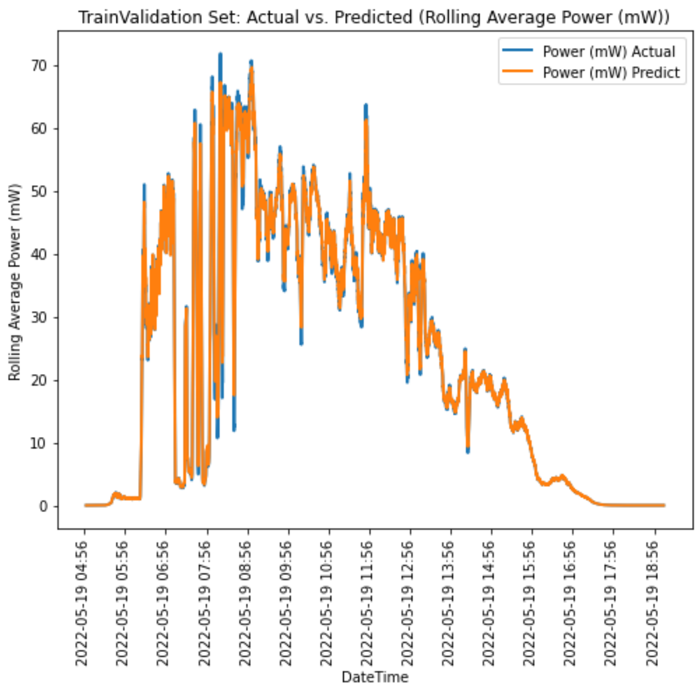
```
Regression Model used: RandomForestRegressor
Grid Search Scorer/s used: ['r2', 'neg_mean_absolute_error']
Grid Search Scorer used to find best parameters: r2
Best parameters:  {'criterion': 'squared_error', 'max_depth': None, 'n_estimators': 500}
Best cross-validation score (r2): 0.9340
...
R2 Score:
Train-Validation Set: 0.9920
Test Set: 0.9384
Mean Absolute Error:
Train-Validation Set: 0.8432mW
Test Set: 2.3082mW
```

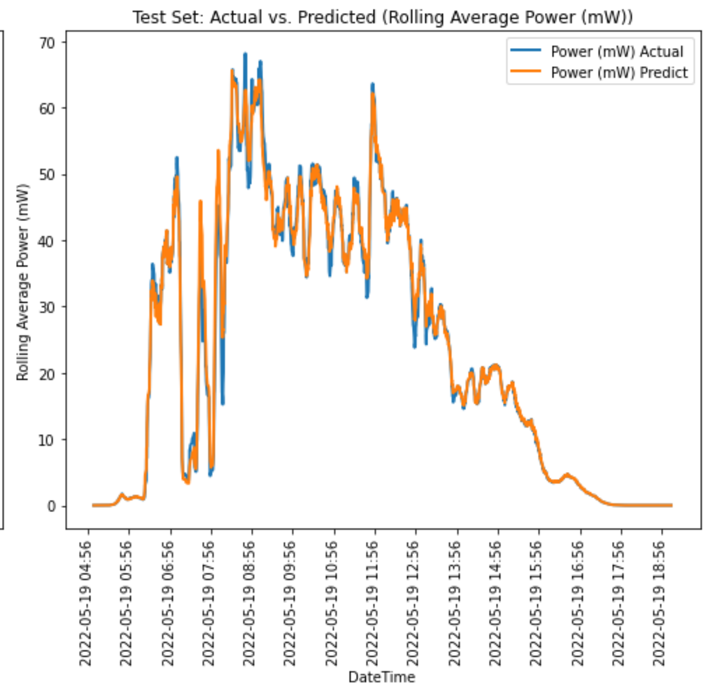
TrainValidation Set: Actual vs. Predicted Power (mW)

DateTime	Power (mW) Actual	Power (mW) Predict
2022-05-19 05:16:30	0.000050	0.000080
2022-05-19 08:30:50	60.797730	63.327226
2022-05-19 18:54:04	0.000000	0.000000
2022-05-19 06:54:15	19.418050	19.405335
2022-05-19 16:06:51	3.403700	3.257295

Test Set: Actual vs. Predicted Predicted Power (mW)

DateTime	Power (mW) Actual	Power (mW) Predict
2022-05-19 14:21:46	13.092480	11.600516
2022-05-19 08:23:10	72.742500	65.504031
2022-05-19 18:33:27	0.000000	0.000000
2022-05-19 16:09:31	4.313120	4.238830
2022-05-19 17:14:26	1.174860	1.192806





In [26]:

```
from sklearn.ensemble import GradientBoostingRegressor

gbr_param_grid = {"learning_rate":[0.5, 0.8],
                  "n_estimators":[500,1000],
                  "max_depth":[5,None]}

tup_gbr=grid_search_report(GradientBoostingRegressor(random_state=47), gbr_param_grid, 
```

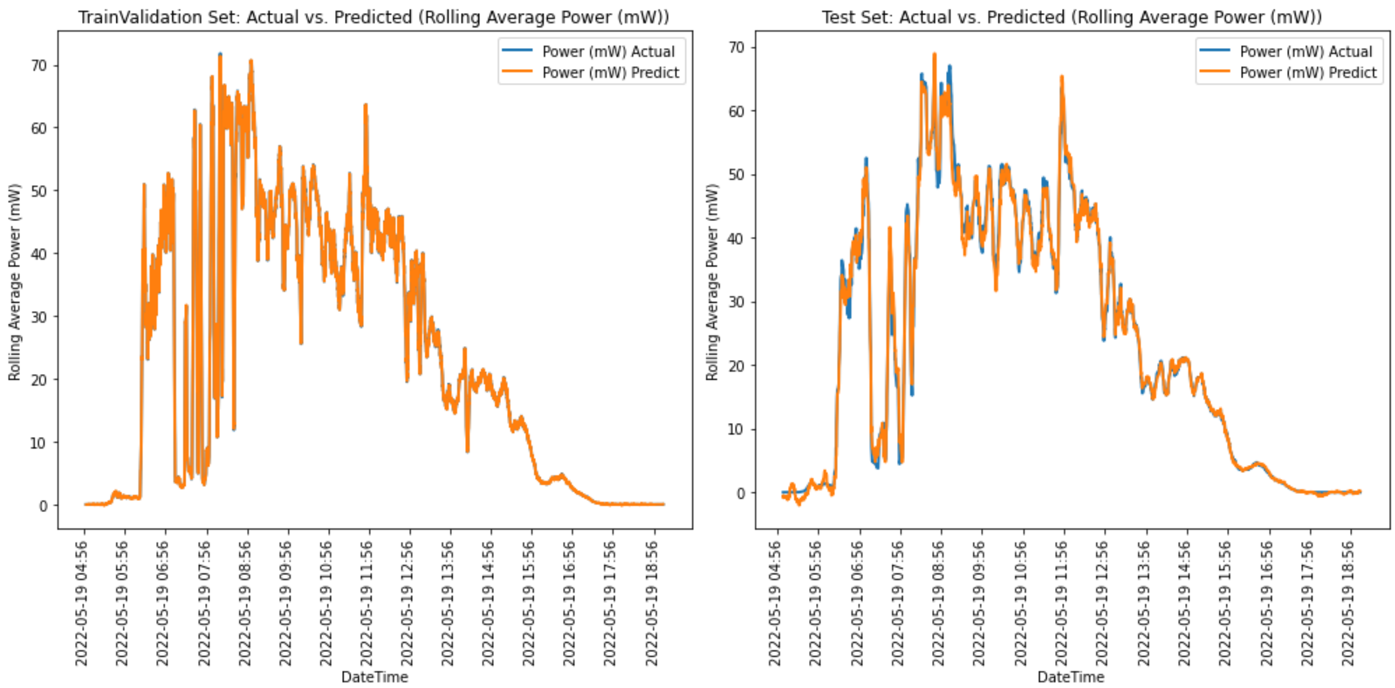
```
Regression Model used: GradientBoostingRegressor
Grid Search Scorer/s used: ['r2', 'neg_mean_absolute_error']
Grid Search Scorer used to find best parameters: r2
Best parameters:  {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 1000}
Best cross-validation score (r2): 0.9372
...
R2 Score:
Train-Validation Set: 0.9989
Test Set: 0.9524
Mean Absolute Error:
Train-Validation Set: 0.5160mW
Test Set: 2.7913mW
```

TrainValidation Set: Actual vs. Predicted Power (mW)

DateTime	Power (mW) Actual	Power (mW) Predict
2022-05-19 05:16:30	0.000050	-0.859534
2022-05-19 08:30:50	60.797730	62.765559
2022-05-19 18:54:04	0.000000	-0.238554
2022-05-19 06:54:15	19.418050	19.688915
2022-05-19 16:06:51	3.403700	3.540735

Test Set: Actual vs. Predicted Predicted Power (mW)

DateTime	Power (mW) Actual	Power (mW) Predict
2022-05-19 14:21:46	13.092480	13.224717
2022-05-19 08:23:10	72.742500	65.401428
2022-05-19 18:33:27	0.000000	-0.252414
2022-05-19 16:09:31	4.313120	4.906825
2022-05-19 17:14:26	1.174860	0.644425



Gradient Boosting Regressor has higher R2 score in the Cross Validation and Test Set compared to Random Forest Regressor but it also has a higher Mean Absolute Error compared to Random Forest Regresssor. Gradient Boosting Regressor also predict negative values while Random Forest only predicts zero as its lowest possible prediction value. To improve gradient boosting regressor's predicting performance and ability to consider zero/close to zero as its lower bounds of prediction, we can do logarithmic transformation but it would also increase/decrease the upper bounds of GBR's prediction and its MAE depending on the constant value applied to the transfromation. To remedy this situation, we will just transform GBR's negative prediction values to zero and see whether the MAE and R2 score of GBR will improve compared to Random Forest Regressor. If not, then we will use Random Forest Regressor as our final model for prediction.

In [27]:

```
#Create unclipped and clipped GBR lower bound prediction to zero
GBR_trainval_predict_clipped = tup_gbr[0].predict(X_trainval).clip(min=0)
GBR_trainval_predict_unclipped = tup_gbr[0].predict(X_trainval)
GBR_test_predict_clipped = tup_gbr[0].predict(X_test).clip(min=0)
GBR_test_predict_unclipped = tup_gbr[0].predict(X_test)
```

In [28]:

```
print("Mean Absolute Error (Clipped):")
MAE_trainval_clipped = mean_absolute_error(y_trainval, GBR_trainval_predict_clipped)
MAE_test_clipped = mean_absolute_error(y_test, GBR_test_predict_clipped)
print("Train-Validation Set: {:.4f}mW".format(MAE_trainval_clipped))
print("Test Set: {:.4f}mW\n".format(MAE_test_clipped))

print("Mean Absolute Error (Unclipped):")
MAE_trainval_unclipped = mean_absolute_error(y_trainval, GBR_trainval_predict_unclipped
MAE_test_unclipped = mean_absolute_error(y_test, GBR_test_predict_unclipped)
print("Train-Validation Set: {:.4f}mW".format(MAE_trainval_unclipped))
print("Test Set: {:.4f}mW\n".format(MAE_test_unclipped))

print("Based on the Test Set of Clipped and Unclipped Lower Bounds of GBR's Prediction.
if MAE_test_clipped < MAE_test_unclipped:
    print("Assessment: Clip GBR's Prediction Lower Bounds ")
else:
    print("Assessment: Don't Clip GBR's Prediction Lower Bounds ")
```

```
Mean Absolute Error (Clipped):
Train-Validation Set: 0.4960mW
Test Set: 2.5328mW

Mean Absolute Error (Unclipped):
Train-Validation Set: 0.5160mW
Test Set: 2.7913mW

Based on the Test Set of Clipped and Unclipped Lower Bounds of GBR's Prediction...
Assessment: Clip GBR's Prediction Lower Bounds
```

In [29]:

```
from sklearn.metrics import r2_score

print("R2 score (Clipped):")
```

```
R2_trainval_clipped = r2_score(y_trainval, GBR_trainval_predict_clipped)
R2_test_clipped = r2_score(y_test, GBR_test_predict_clipped)
print("Train-Validation Set: {:.4f}".format(R2_trainval_clipped))
print("Test Set: {:.4f}\n".format(R2_test_clipped))

print("R2 score (Unclipped):")
R2_trainval_unclipped = r2_score(y_trainval, GBR_trainval_predict_unclipped)
R2_test_unclipped = r2_score(y_test, GBR_test_predict_unclipped)
print("Train-Validation Set: {:.4f}".format(R2_trainval_unclipped))
print("Test Set: {:.4f}\n".format(R2_test_unclipped))

print("Based on the Test Set of Clipped and Unclipped Lower Bounds of GBR's Prediction.
if R2_test_clipped > R2_test_unclipped:
    print("Assessment: Clip GBR's Prediction Lower Bounds ")
else:
    print("Assessment: Don't Clip GBR's Prediction Lower Bounds ")
```

```
R2 score (Clipped):
Train-Validation Set: 0.9989
Test Set: 0.9571

R2 score (Unclipped):
Train-Validation Set: 0.9989
Test Set: 0.9524

Based on the Test Set of Clipped and Unclipped Lower Bounds of GBR's Prediction...
Assessment: Clip GBR's Prediction Lower Bounds
```

Based on the results of clipped and unclipped lower bounds of GBR's prediction, we can see that both MAE and R2 score improves and no scoring type became worse by clipping.

In [30]:

```
#Print CV results of GBR
display(pd.DataFrame(tup_gbr[1]))
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	param_max_depth
0	3.598007	0.093613	0.040397	0.018335	0.5	5
1	7.116014	0.286549	0.066905	0.006119	0.5	5
2	1.068741	0.271815	0.012510	0.006255	0.5	None
3	1.078115	0.019763	0.015633	0.000005	0.5	None
4	3.490990	0.032118	0.031247	0.000001	0.8	5
5	6.900923	0.032964	0.071872	0.012499	0.8	5
6	0.531246	0.066292	0.012505	0.006253	0.8	None
7	0.709361	0.025377	0.012511	0.006255	0.8	None

8 rows × 24 columns

Optimizing Servo Pair Angles

Based on the initial data gathered, we can already identify the best servo pair angles. But we can use the model that we developed to further gain insights or optimize the servo pair angles since the data we gathered only focused on specific sets of steps of servo angles.

In [31]:

```
#We previously identified that the pair of servo angles that has highest average power (
    #Servo Lower: 45
    #Servo Upper: 46
#Let's estimate the average power of other sets of servo pair angles (close to these se
#We will use the following data points
#Servo Lower Previous Bounds: 0,  #Servo Lower Next Bounds: 90,
#Servo Upper Previouse Bounds: 23, #Servo Upper Next Bounds: 69,

final_model = tup_gbr[0] #set our final model to GBR
servo_lower_set=np.linspace(0,90, num=11)
servo_upper_set=np.linspace(23,69, num=5)
print("Sets of Angle to Pair and Predict")
print("Servo lower set: ", servo_lower_set)
print("Servo upper set: ", servo_upper_set)

#Setup empty dataframe first filled with ones
servo_pairs_pred_ave_power = pd.DataFrame(1,index=servo_lower_set, columns=servo_upper_
servo_pairs_pred_ave_power.rename_axis("Servo Upper Angle", axis=1, inplace=True)
servo_pairs_pred_ave_power.rename_axis("Servo Lower Angle", axis=0, inplace=True)
for servo_lower_angle in servo_lower_set:
    for servo_upper_angle in servo_upper_set:
        data_to_predict=data_to_model.loc[:,:"Servo Upper"].copy() #get all x_predictor
        data_to_predict["Servo Lower"] = servo_lower_angle
        data_to_predict["Servo Upper"] = servo_upper_angle
        power_predicted = final_model.predict(data_to_predict).clip(min=0) #clip predic
        average_power=np.mean(power_predicted)
        servo_pairs_pred_ave_power.loc[servo_lower_angle,servo_upper_angle] = average_p


display(servo_pairs_pred_ave_power.style.background_gradient(cmap="viridis", axis=None)
```

```
Sets of Angle to Pair and Predict
Servo lower set:  [ 0.   9. 18. 27. 36. 45. 54. 63. 72. 81. 90.]
Servo upper set:  [23.  34.5 46.  57.5 69. ]
```

Average Power (mW) Delivered based on Servo Angle Positions

Servo Upper Angle	23.0	34.5	46.0	57.5	69.0
Servo Lower Angle					
0.0	25.593535	25.594966	27.957157	27.951605	26.719754
9.0	25.593535	25.594966	27.957157	27.951605	26.719754
18.0	25.593535	25.594966	27.957157	27.951605	26.719754
27.0	25.594358	25.594790	28.110292	28.107796	26.617448
36.0	25.594358	25.594790	28.110292	28.107796	26.617448
45.0	25.594358	25.594790	28.110292	28.107796	26.617448
54.0	25.587276	25.587709	28.103080	28.100583	26.609840
63.0	25.587276	25.587709	28.103080	28.100583	26.609840
72.0	24.593539	24.594033	27.609934	27.607437	26.364032
81.0	24.593539	24.594033	27.609934	27.607437	26.364032
90.0	24.593539	24.594033	27.609934	27.607437	26.364032

Conclusions

Based on the prediciton models and data gathered, we can conclude that as for the current physical setup (location, current roofing materials and coatings used) where the device/module gathered the data, we can optimize the average power output of the solar panels during installment phase if we design the inclinations of our panel holders - the same way how the device/module identified its optimal inclinations (with estimated amount of tolerance) through the servo angle pairs which are - Servo Lower: 27°-45° and Servo Upper: 46° - 57.5°. But how can we design it based on pitch and roll angles?

Method 1: We can do these through setting up the device again to the physical location were it gathered the data, and set it up to move towards the optimal servo angle pairs and record its corresponding pitch and roll angles which will then be used to design the solar panel holders.

Method 2: We can also do these by training another machine learning model where we use Servo Lower and Servo Upper Angles as the predictor/X variables and the Pitch and Roll angles as the response/Y variables and then used the trained model to predict the pitch and roll angles of the optimal servo angle pairs.

But, it is highly recommeded to use Method 1. The results of other servo angle pairs are useful to set the amount of tolerance of the inclination angles on designing our solar panel holders. We cen further improve our confidence of these optimization if we also create models where the variable we will be predicting is the Irradiance instead of Power.

Recommendations

If more data and experimentations are done to other location and datetime, we can use the temperature and humidity variables to create a more generalized machine learning model if our goal is to improve efficiency of solar panels by improving temperature conditions. We could use the results to identify whether we need to improve roofing materials, or coatings, or choose the location that already have the best temperature conditions. We also recommend to develop the device even further to also record the yaw angles, and if possible to record the longitude, latitude, and altitude where the device was gathering data. This might be useful if we are to pool the data from different users and locations.