Project Report for INF 2179 - Machine Learning with Applications in Python

A Study of Solar Power Output in a Given Plant in India

December 24, 2020

Prof. Arik Senderovich

Group 2

Mansi Jain

Miao Xi

Table of Content

1 Introduction

1.1 Project Motivation

Even in this modern age, energy production in several developing countries often falls short of energy requirements which results in frequent power cuts. As the world economy continues to grow, energy consumption is expected to continue to grow. Fossil fuel is limited, so it is important to consider other sources of renewable energy, especially solar, to meet the energy demands in the future. In the solar energy industry, there is a general motto of “install it and forget it”. This notion is derived from much of the research and reliability studies around the photovoltaic (PV) panels or cells. This implies that maintenance and regular monitoring is not needed. Yet many things may go wrong to cause the actual system performance to deviate from the expected performance. This study sheds light on how we can optimize energy generation in solar power plants.

1.2 Solar Energy Generation

Solar panels work by absorbing sunlight with photovoltaic cells, generating direct current (DC) energy and then converting it to usable alternating current (AC) energy with the help of inverter technology. The science of generating electricity with solar panels boils down to the photovoltaic effect. The photovoltaic process works through the following simplified steps:

- The silicon photovoltaic solar cell absorbs solar radiation
- When the sun’s rays interact with the silicon cell, electrons begin to move, creating a flow of electric current
- Wires capture and feed this direct current (DC) electricity to a solar inverter to be converted to alternating current (AC) electricity

During this process many things can go wrong and lead to compromised energy generation, like:

wrong installation of equipment and inverter, accumulation or dirt or debris on solar cells, short circuits, broken wires etc. As the solar plants are very large in scale, manual inspection of the plants to identify faults and maintenance needs is inefficient. These issues lead us to our research questions, which look for ML solutions to identify them in an automatic and real-time manner.

2 Data Description

The data is solar power generation and sensor data gathered from two solar power plants in India at 15 minutes intervals over a 34-day period. It has two pairs of files - each pair has one power generation dataset and one weather sensor readings dataset. The power generation datasets are gathered at the inverter level - each inverter has multiple lines of solar panels attached to it, while the weather sensor data is gathered at a plant level - single array of sensors optimally placed at the plant. In this project we investigate the data from plant 2 as we suspect data gathered from plant 1 has collection error. In plant 1 data, DC to AC power conversion rate is extremely low (less than 10% vs. 90%) compared with that from plant 2. This raises concerns about the usability of plant 1 data as this level of inefficiency should not appear in a commercial plant if the data itself is collected correctly.

There are 2 csv files and 13 columns in total. Plant_2_Generation_Data.csv, containing power generation data, has columns DATE_TIME, PLANT_ID, SOURCE_KEY(inverter id), DC_POWER, AC_POWER, DAILY_YIELD(cumulative sum of power generated on that day, till that point in time), and TOTAL_YIELD(total yield for the inverter till that point in time). Plant_2_Weather_Sensor_Data.csv, contain weather sensor data, has columns DATE_TIME, PLANT_ID, SOURCE_KEY (inverter id), 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) and IRRADIATION. The observations for DATE_TIME, DC_POWER and AC_POWER are recorded at 15-minute intervals. The columns PLANT_ID in both the csv files and SOURCE_KEY in weather sensor data is the same for the whole data set, thus these columns are irrelevant and were dropped. Both the power generation data and weather data were merged to a single data frame to study the effect of weather on inverter outputs. This merged data has been used for all study and analysis in the project. For code please refer to Appendix A.1.

Power Generation Data

	DATE_TIME	PLANT_ID	SOURCE_KEY	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD
0	2020-05-15	4136001	4UPUqMRk7TRMgml	0.0	0.0	9425.000000	2.429011e+06
1	2020-05-15	4136001	81aHJ1q11NBPMrL	0.0	0.0	0.000000	1.215279e+09
2	2020-05-15	4136001	9kRcWv60rDACzjR	0.0	0.0	3075.333333	2.247720e+09
3	2020-05-15	4136001	Et9kgGMDl729KT4	0.0	0.0	269.933333	1.704250e+06
4	2020-05-15	4136001	IQ2d7wF4YD8zU1Q	0.0	0.0	3177.000000	1.994153e+07

Weather Sensor Data

	DATE_TIME	PLANT_ID	SOURCE_KEY	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION
0	2020-05-15 00:00:00	4136001	iq8k7ZNt4Mwm3w0	27.004764	25.060789	0.0
1	2020-05-15 00:15:00	4136001	iq8k7ZNt4Mwm3w0	26.880811	24.421869	0.0
2	2020-05-15 00:30:00	4136001	iq8k7ZNt4Mwm3w0	26.682055	24.427290	0.0
3	2020-05-15 00:45:00	4136001	iq8k7ZNt4Mwm3w0	26.500589	24.420678	0.0
4	2020-05-15 01:00:00	4136001	iq8k7ZNt4Mwm3w0	26.596148	25.088210	0.0

Table 1. Original Data Provided

Merged Data

	DATE_TIME	PLANT_ID	Inverter_ID	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION
0	2020-05-15 00:00:00	4136001	4UPUqMRk7TRMgml	0.0	0.0	9425.000000	2.429011e+06	27.004764	25.060789	0.0
1	2020-05-15 00:00:00	4136001	81aHJ1q11NBPMrL	0.0	0.0	0.000000	1.215279e+09	27.004764	25.060789	0.0
2	2020-05-15 00:00:00	4136001	9kRcWv60rDACzjR	0.0	0.0	3075.333333	2.247720e+09	27.004764	25.060789	0.0
3	2020-05-15 00:00:00	4136001	Et9kgGMDl729KT4	0.0	0.0	269.933333	1.704250e+06	27.004764	25.060789	0.0
4	2020-05-15 00:00:00	4136001	IQ2d7wF4YD8zU1Q	0.0	0.0	3177.000000	1.994153e+07	27.004764	25.060789	0.0
...	...	...	...	...	...	...	...	...	...	...
67693	2020-06-17 23:45:00	4136001	q49J1IKaHRwDQnt	0.0	0.0	4157.000000	5.207580e+05	23.202871	22.535908	0.0
67694	2020-06-17 23:45:00	4136001	rrq4fwE8jgrTyWY	0.0	0.0	3931.000000	1.211314e+08	23.202871	22.535908	0.0
67695	2020-06-17 23:45:00	4136001	vOuJvMaM2sgwLmb	0.0	0.0	4322.000000	2.427691e+06	23.202871	22.535908	0.0
67696	2020-06-17 23:45:00	4136001	xMblugepa2P7lBB	0.0	0.0	4218.000000	1.068964e+08	23.202871	22.535908	0.0
67697	2020-06-17 23:45:00	4136001	xoJJ8DcxJEcupym	0.0	0.0	4316.000000	2.093357e+08	23.202871	22.535908	0.0

67698 rows × 10 columns

Table 2. Merged Data for Future Study

3 Exploratory Data Analysis

3.1 Inspection of Data Validity

Initial investigation dataset led to an interesting discovery. A solar inverter or PV inverter, is a type of electrical converter which converts the variable direct current (DC) output of a photovoltaic (PV) solar panel into a utility frequency alternating current (AC)

that can be fed into a commercial electrical grid network. Typical grid-tied inverter efficiencies exceed 95% under most operating conditions. In our dataset, in plant 1 data, DC to AC power conversion rate is extremely low (less than 10% vs. 90%) compared with that from plant 2 (Figure 1). This raises concerns about the usability of plant 1 data as this level of inefficiency should not appear in a commercial plant if the data itself is collected correctly. For code please refer to Appendix A.2.

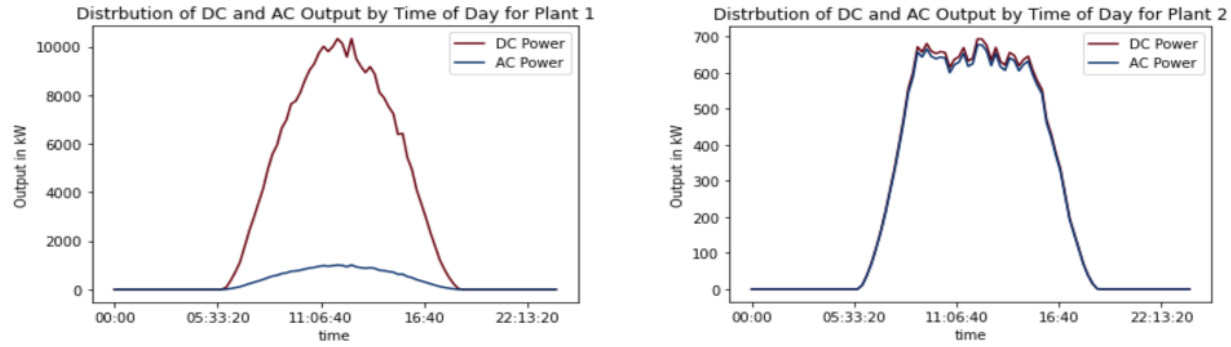


Figure 1. Comparison of DC AC Conversion Rate between Plant 1 & 2

3.2 Discovery of Linear Relationships and Outliers

Firstly, we investigated the relationship between power outputs and weather data. From Figure 2, we could observe strong linear correlation between some features. The strongest linear relationship exists between DC and AC outputs with correlation coefficient at 0.999. Another noticeable linear relationship is between irradiation and module temperature at 0.947. One interesting finding is that the correlation coefficient between irradiation and DC output is only 0.604. For code please refer to Appendix A.3.

	DC_POWER	AC_POWER	DAILY_YIELD	TOTAL_YIELD	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION	month	hour
DC_POWER	1.000000	0.999997	0.005593	0.004528	0.563232	0.749676	0.780978	-0.080431	0.027817
AC_POWER	0.999997	1.000000	0.005395	0.004533	0.563324	0.749604	0.780851	-0.080248	0.027842
DAILY_YIELD	0.005593	0.005395	1.000000	-0.068472	0.321785	0.046787	-0.107987	-0.040094	0.596705
TOTAL_YIELD	[illegible]	[illegible]	-0.068472	1.000000	0.002774	-0.004646	-0.006720	-0.032167	-0.003695
AMBIENT_TEMPERATURE	0.563232	0.563324	0.321785	0.002774	1.000000	0.848976	0.671998	-0.355423	0.360336
MODULE_TEMPERATURE	0.749676	0.749604	0.046787	-0.004646	0.848976	1.000000	0.947057	-0.183838	0.150493
IRRADIATION	0.780978	0.780851	-0.107987	-0.006720	0.671998	0.947057	1.000000	-0.092924	0.021706
month	[illegible]	[illegible]	-0.040094	-0.032167	-0.355423	-0.183838	-0.092924	1.000000	-0.005384
hour	0.027817	0.027842	0.596705	-0.003695	0.360336	0.150493	0.021706	-0.005384	1.000000

Figure 2. Correlation Matrix between Variables from the Merged Dataset

Upon visualization (Figure 3), we could see the relationship between irradiation and DC output is mostly linear but for some cases, when irradiation is high the DC output remains 0 or deviates from the linear pattern. We suspect that this could be caused by

malfunctioning of some equipment or some performing sub-optimally compared to others. For code please refer to Appendix A.4.

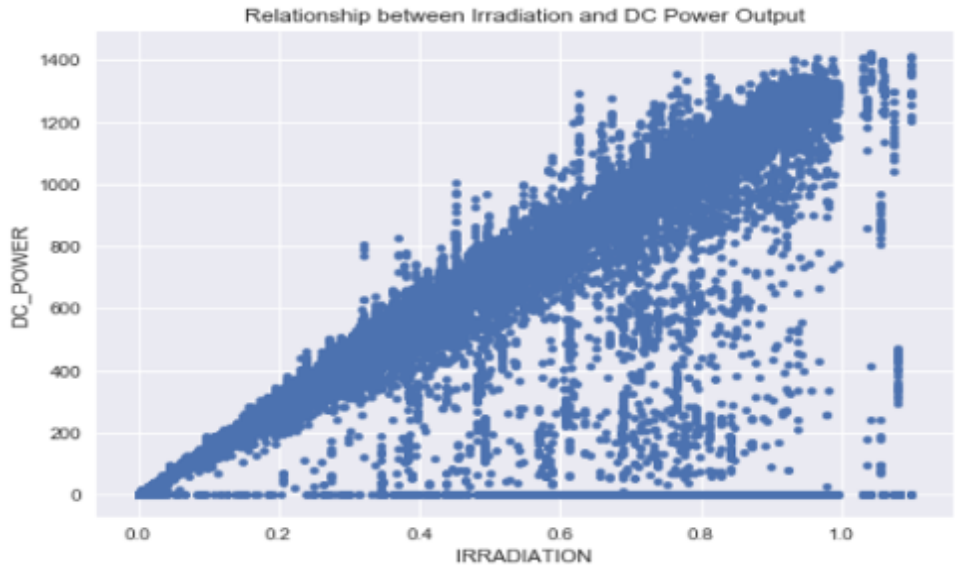


Figure 3. Outliers between Irradiation and DC Power Output

3.3 Performance for Individual Inverters

The performance of each inverter would be under deeper scrutiny to explore possible explanations for outliers discovered in above parts. The performance of inverters are represented with DC power output divided by irradiation (DC conversion rate). By plotting the distribution of DC conversion rate, we could identify high and low performing inverters. For code please refer to Appendix A.5.

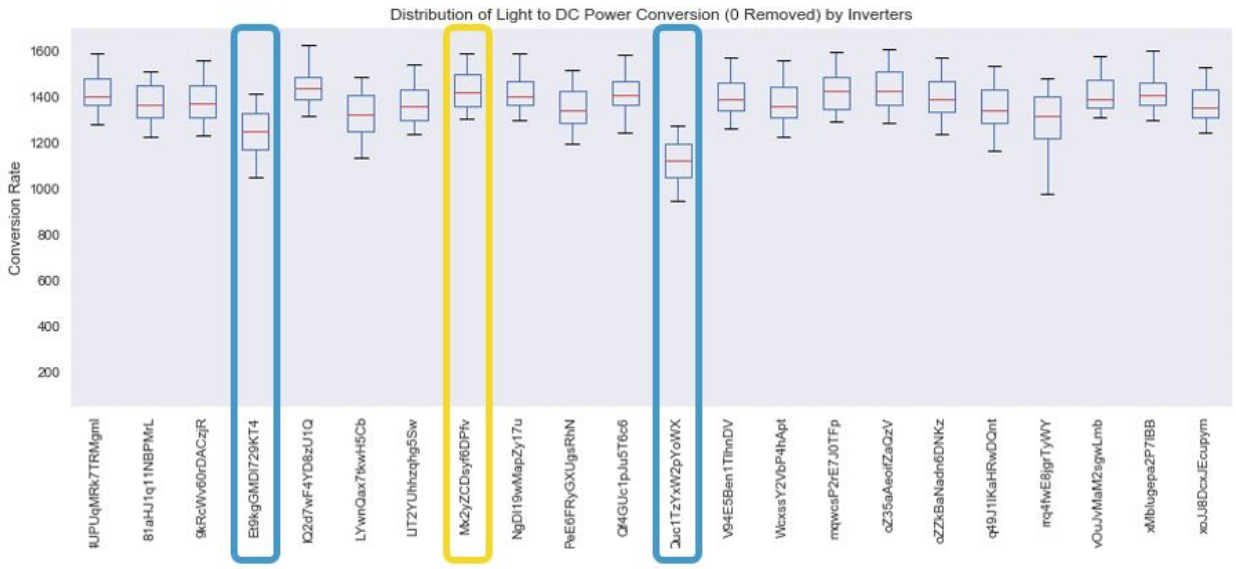


Figure 4. Identifying the High and Low Performing Inverters

In the above figure, data points, which DC conversion rate is 0 are removed. It represents situations when all inverters have some level of output. The yellow circled inverter is performing relatively well as its mean and upper and lower quantile of DC conversion rate takes lead across all 22 inverters. While the 2 blue circled inverters are performing sub-optimally.

Further study of the 3 inverters selected shows that the mean DC conversion rates differ drastically between them given across time of day. And the gap is more significant around noon. For code please refer to Appendix A.6.

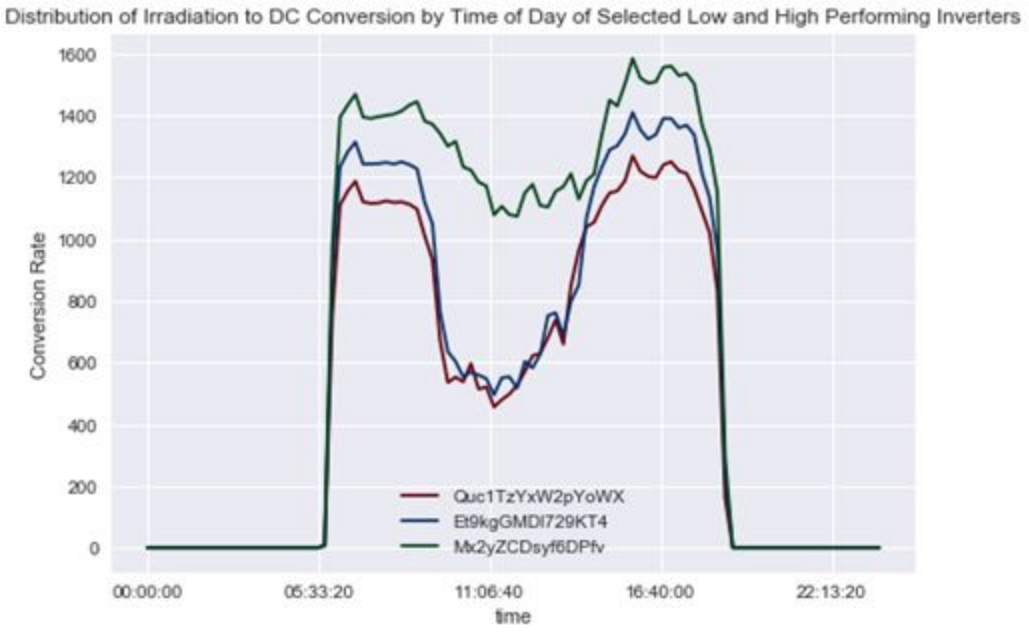


Figure 5. Comparing DC Conversion Rates across Time of Day

These findings further points out that it is very likely that low performing inverters require more maintenance and have higher rates of malfunction compared to high performing ones.

4 Research Questions

We addressed the following three research questions in our project:

1. Can we identify the faulty equipment in the plant?
2. Can we identify panels that may need cleaning and maintenance?
3. Can we predict the output of power for future days that would help better grid management?

5 Analysis Techniques

5.1 Methodologies towards Research Questions

5.1.1 Solving Research Question 1 & 2

To answer research question 1 & 2, identification of faulty or need maintenance inverters a 3-step approach is proposed.

- Step 1: Apply an outlier detection technique to filter out the anomalies deviating from the linear pattern between irradiation and power output.
- Step 2: Construct and select Machine Learning models based on the filtered data obtained through step 1 to serve as a ‘golden standard’. This model would take relevant input such as irradiation, temperature and time and output the expected AC power output, given the solar systems running at optimal performance.
- Step 3: Comparing difference between actual AC output and the predicted one by the ‘golden standard’ model. Identify those outlier data points with large gaps using IQR rules and develop a labeling technique to label those outlier data points as ‘faulty’ or ‘need maintenance’.

5.1.2 Solving Research Question 3

To answer research question 3, forecasting power output in future days, typical Machine Learning models that developed based on the independent and identical distribution assumption can no longer be useful. Time series related models are required. 2 approaches are proposed to solve the forecasting problem.

- Dedicated time series models.
- Data manipulation on time series data, making past observations a feature for the current data points. This allows I.I.D Machine Learning models to be implemented on time series data.

5.2 Machine Learning Techniques

A wide range of Machine Learning related techniques are applied in this study. For preprocessing, Label Encoder and MinMax Scaler. For hyperparameter tuning, GridSearchcv method is applied.

Label Encoder is to transfer text from categorical variables to integers so that they can be utilized in constructing models.

MinMax Scaler is to standardize every numerical feature into a scale of [0,1] by the below function. This step would eliminate the effect of different scales of different features on the application of distance based algorithms.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

GridSearchcv is an exhaustive search method to arrive at the optimal model. A grid, which contains all possible combinations of hyperparameters that are input by users. And each combination is used to construct a model. Cross validation is used on each construction to reduce variance. Then the average performance metrics for each construction obtained through cross validation is used to represent the performance of each combination. Combinations are compared against each other to identify the best one.

Below example is based on tuning of a SVR model. Penalty term C and Kernel type are 2 hyperparameters to tune. Each one is provided with 3 possible values. A total 9 combinations are obtained and tested and one best combination is identified.

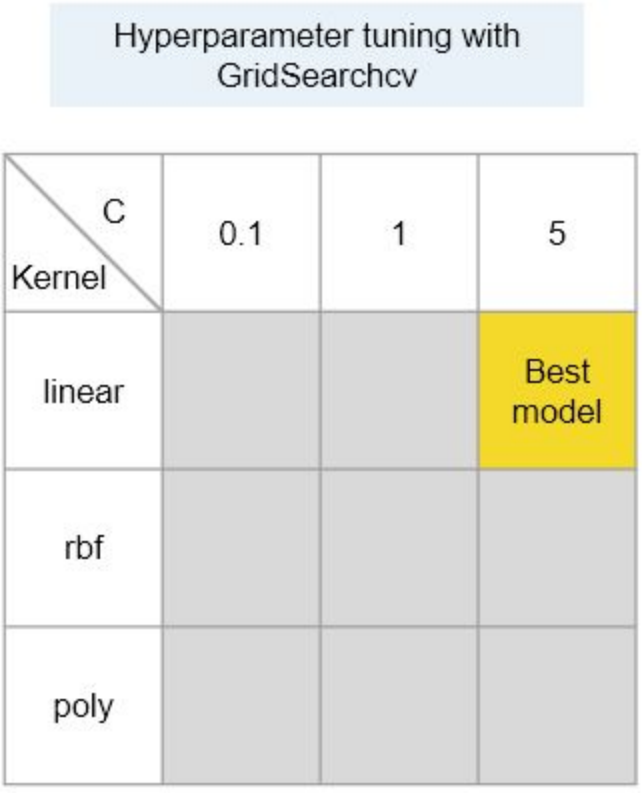


Figure 6. Example of GridSearchcv with SVR

6 Machine Learning in Action

6.1 Filtering Out Performance Outliers

As described in 5.1, outliers that distort the linear relationship between irradiation and AC power output need to be identified and then removed to create the filtered dataset. Random Sample Consensus (RANSAC) algorithm is applied to this process. RANSAC is a robust estimation that is able to arrive at a model that extracts the linear relationship between irradiation and AC output ignoring the effect of outlier. Detailed explanation of the RANSAC algorithm is in Appendix B.

Below, data for a selected inverter is used to demonstrate the application of RANSAC.

Figure 7. Simple Linear Regression Applied to Irradiation AC Relationship

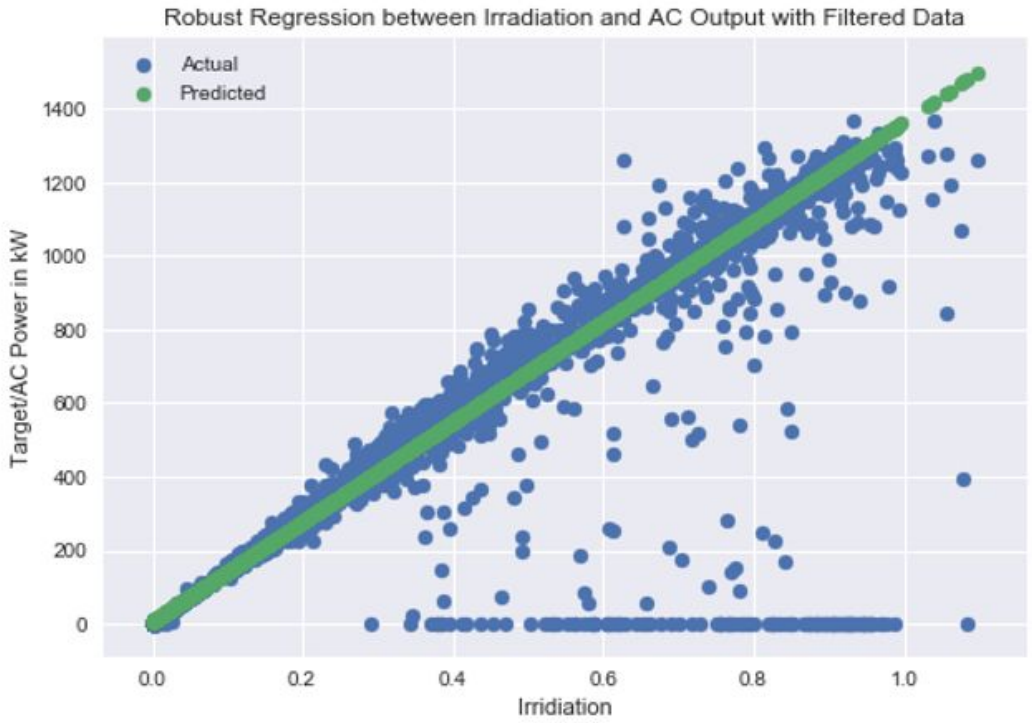


Figure 8. RANSAC Applied to Irradiation AC Relationship

Comparing Figure 7 and 8, RANSAC is capable of fitting a model that ignores the effect of outlier and captures the true linear relationship between irradiation and AC power output. While a simple Linear Regression would suffer from the effect of the outliers and fail to represent the inherent linear relationship. For code please refer to Appendix A.7.

This technique would be applied on the entire dataset across inverters to produce the filtered dataset that is used in the search for ‘golden standard’. A most strict filter, which is able to filter out most number of outliers, is desired in this case. To do so, hyperparameter tuning is conducted on the RANSAC model. The hyperparameter tuned is Residual Threshold t. This value is used in the algorithm to set the criteria of which data points shall be included as inliers. For code please refer to Appendix A.8.

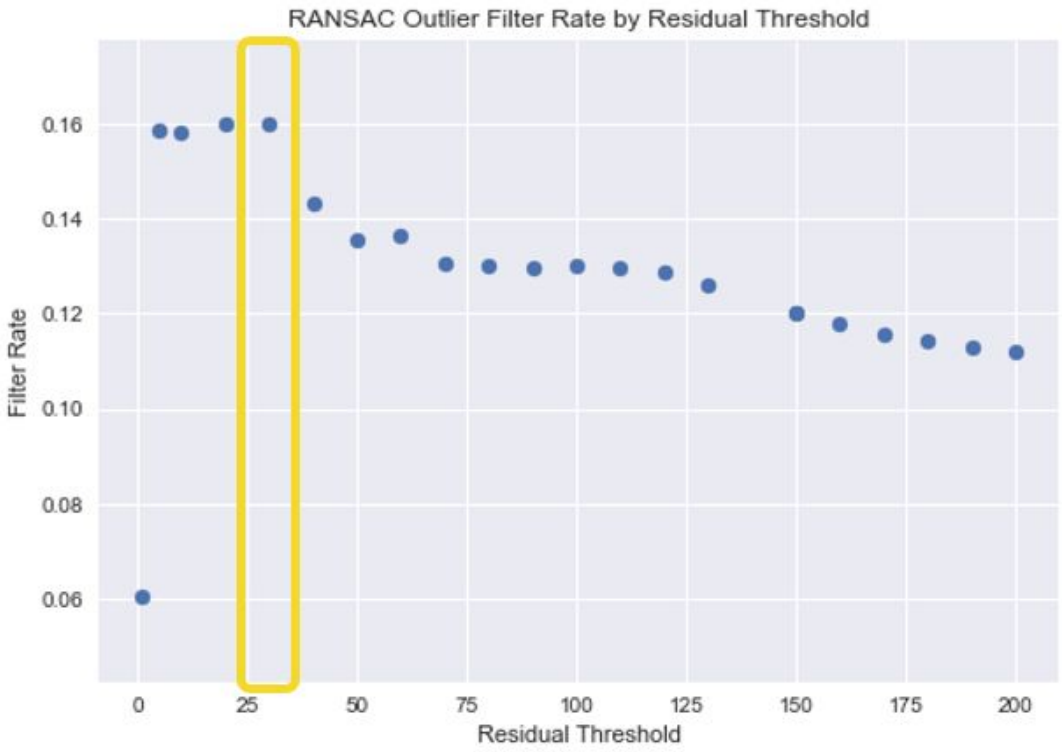


Figure 9. Selection of the Best Residual Threshold

In Figure 9 it could be observed that when Residual Threshold equals to 30, the filtered out rate is the highest at 16%. The filtering result is shown below. The linear relationship between irradiation and AC power output is preserved while the outliers are removed. For code please refer to Appendix A.9.

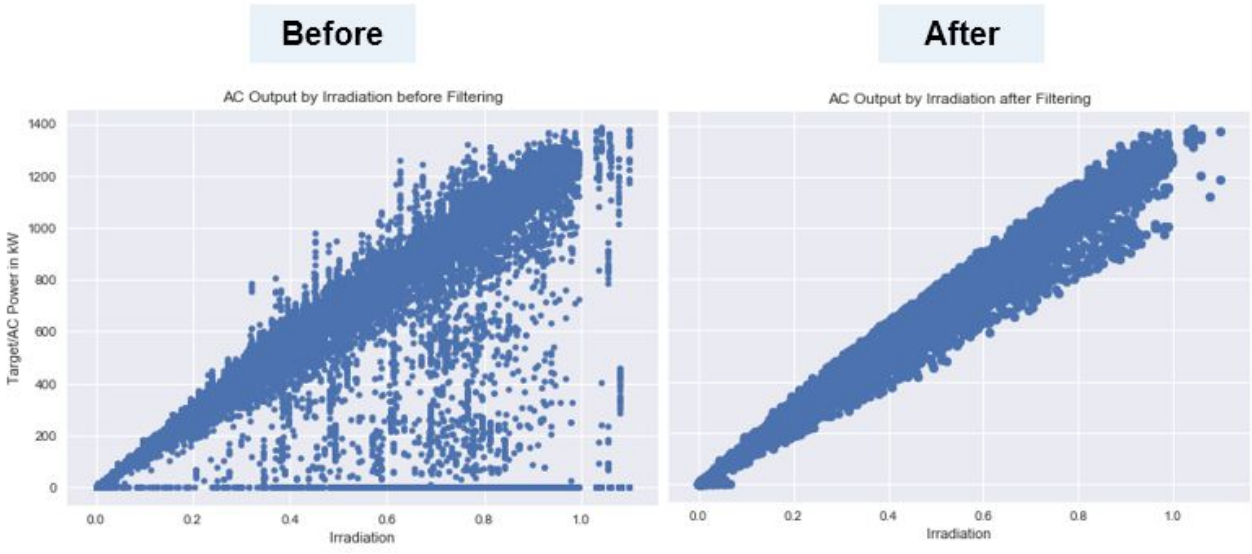


Figure 10. Comparison: Before and After Filtering Applied

6.2 Machine Learning Models for the ‘Golden Standard’

Several Machine Learning models are constructed based on the new dataset created through the filtering method illustrated in 6.1. These models are the candidates for selecting the ‘golden standard’, which is the optimal performance (AC power output) expected, without malfunctions or need for maintenance, for any inverters given a set of inputs.

The input data is shown in the table below.

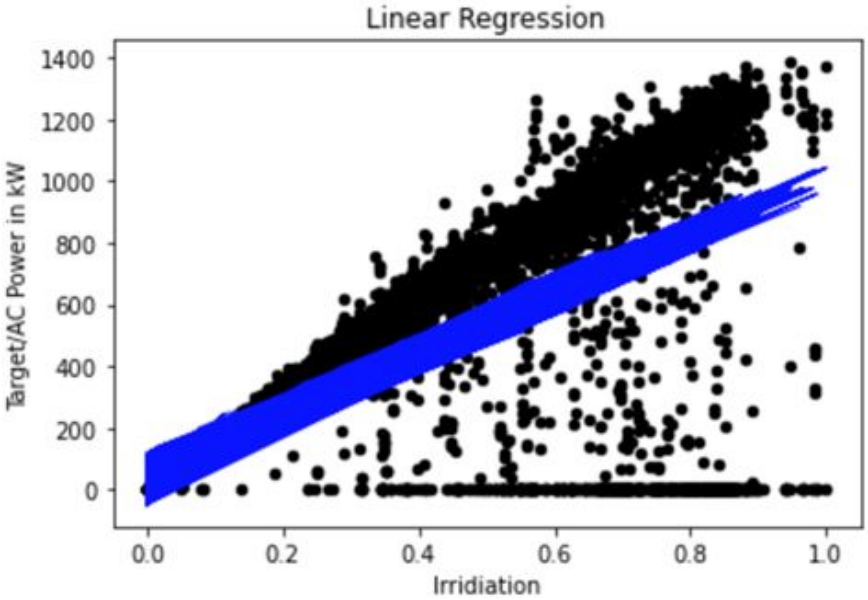
	AMBIENT_TEMPERATURE	MODULE_TEMPERATURE	IRRADIATION	month	day_of_month	hour	minute	day_of_week
17	27.004764	25.060789	0.000000	5	15	0	0	4
39	26.880811	24.421869	0.000000	5	15	0	15	4
61	26.682055	24.427290	0.000000	5	15	0	30	4
83	26.500589	24.420678	0.000000	5	15	0	45	4
105	26.596148	25.088210	0.000000	5	15	1	0	4
127	26.512740	25.317970	0.000000	5	15	1	15	4
149	26.494339	25.217193	0.000000	5	15	1	30	4

Table 3. Input Variables for Machine Learning Models

First 3 columns from the left are environmental variables. They are driving factors for power output of a solar system. The rest of the columns are dissected from date time timestamp. By including them we are assuming that time itself would also provide some explanation power to the target variable, AC power output. The time variable may implicitly factor in effects such as weekday / weekends, staff shifts schedule etc..These factors presumably have some noticeable level of impact on the performance of the solar plant and are highly correlated with time yet we could not obtain them directly.

From conducting EDA, it is shown that the environmental variables have high linear correlation with the AC power output. It is evident that the prediction of power output is a linear problem. The problem is first approached with Linear Regression.

First, the Linear Regression model was trained and run-on unfiltered data. Below figure shows the effect of outliers on the model, resulting in high RMSE value. For code, please refer to Appendix A.10.

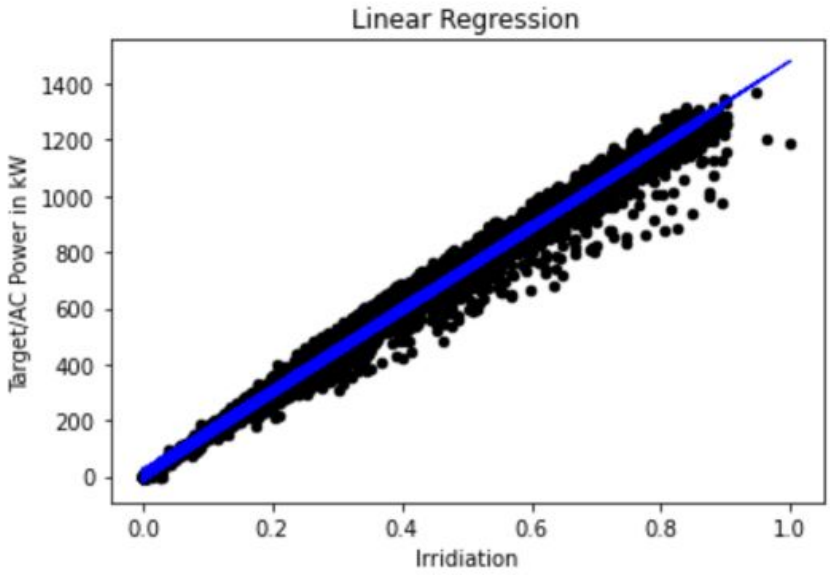


Model: Linear Regression **RMSE:** 223.83 **R2:** 0.62

Figure 11. Linear Regression on Unfiltered Data

Then, Linear regression model was tried with the filtered data. Below figure shows the impressive improvement in the model efficiency. The RMSE score dropped from 223 previously to 29.5. For code, please refer to Appendix A.11.

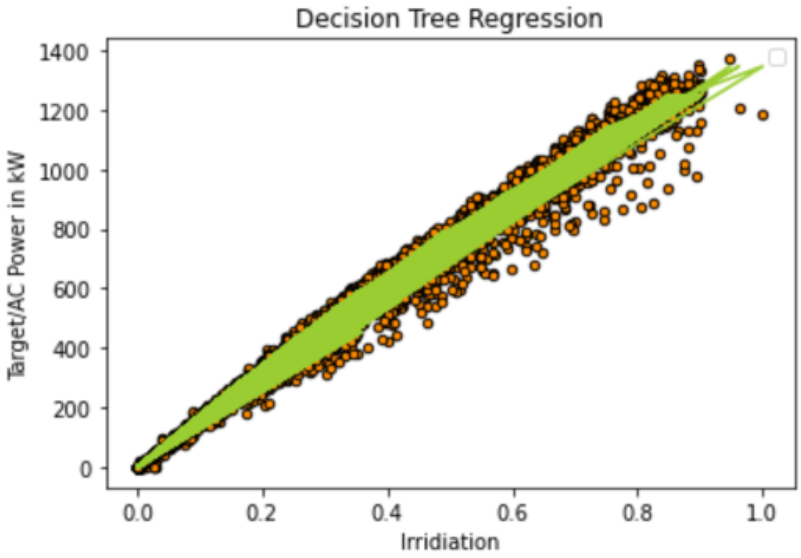
This demonstrates again the importance of outlier filtering.



Model: Linear Regression **RMSE:** 29.54 **R2:** 0.993

Figure 12. Linear Regression on Filtered Data

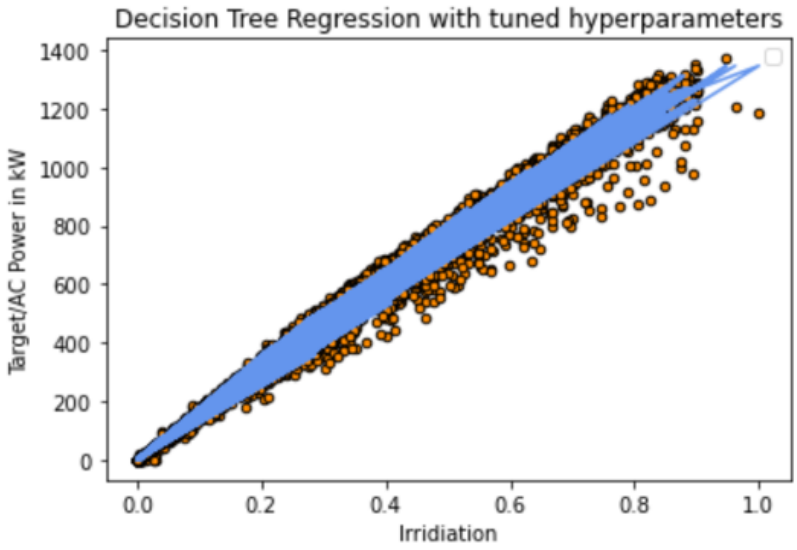
To see if we can get better results, another Machine Learning model, Decision Tree Regression was tried with the filtered data and the results improved slightly (as shown below). For code, please refer to Appendix A.12.



Model: Regression Tree **RMSE:** 26.36 **R2:** 0.9944

Figure 13. Decision Tree without Tuning on Filtered Data

Hyper parameters tuning of Decision Tree Regression model improved the model efficiency and gave even better results. For code, please refer to Appendix A.13.

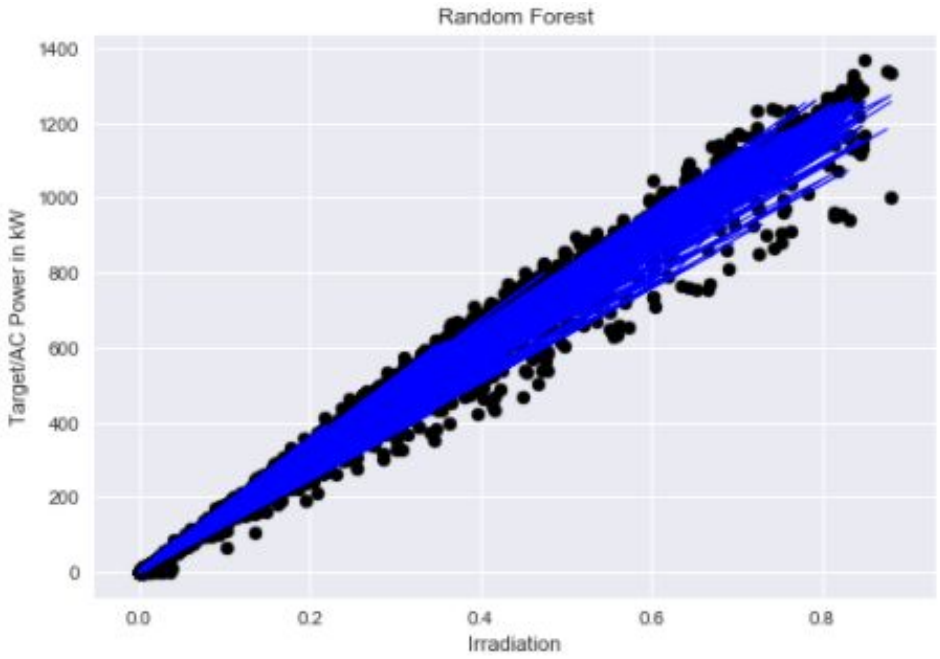


Model: Regression Tree Best **RMSE:** 24.14 **R2:** 0.9954

Figure 14. Decision Tree with Tuning on Filtered Data

Apart from Linear Regression and Decision Tree, more advanced models are utilized to try to achieve better results for the ‘golden standard’. Random Forest is chosen because it is a more powerful improvement based on the Decision Tree.

With grid search cv hyperparameter tuning and running on filtered data Random Forest yields results as shown in the below figure. For code please refer to Appendix A.14.

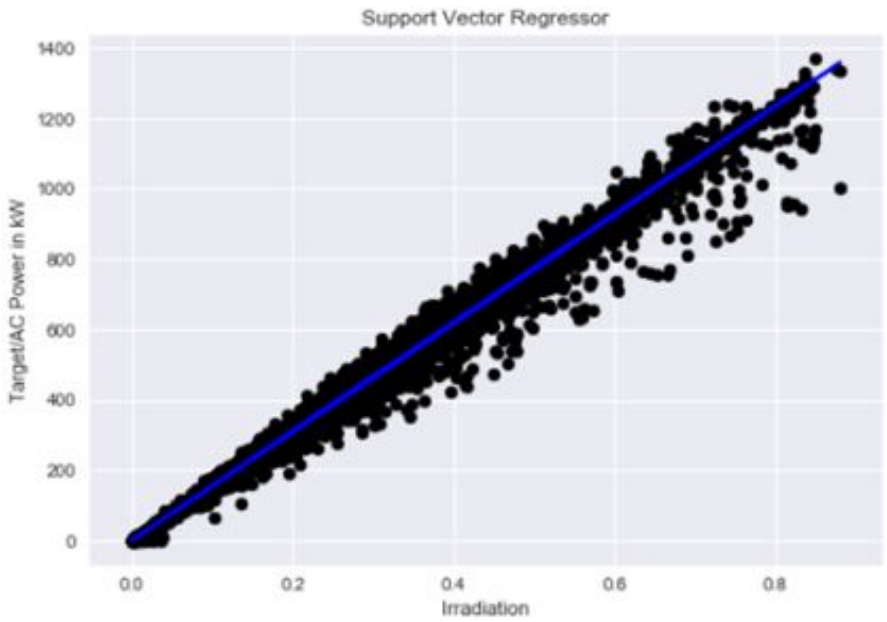


Model: Random Forest **RMSE:** 21.46 **R2:** 0.9948

Figure 15. Random Forest with Tuning on Filtered Data

Another model implemented is a Support Vector Regressor (SVR). This model is a stronger and more advanced approach towards linear problems. This algorithm applies Support Vector Machine into regression settings.

With grid search cv hyperparameter tuning and running on filtered data SVR yields results as shown in the below figure. For code please refer to Appendix A.15.



Model: SVR **RMSE:** 26.87 **R2:** 0.9918

Figure 16. SVR with Tuning on Filtered Data

All 4 categories of models mentioned above are compared in the below table.

Model	RMSE	R2
Linear Regression	29.54	0.993
Regression Tree	24.14	0.9954
Random Forest	21.46	0.9948
SVR	26.87	0.9918

Table 4. Comparison of Metrics for Models Built

In this case, in terms of RMSE, Random Forest is the best model we have ever constructed with the lowest value to 21.46. Random Forest would then be selected in later study as the ‘golden standard’ model.

6.3 Solution to Research Questions 1 & 2 - Labelling

Once the ‘golden standard’ model is selected, it is applied to identify and label inverters according to their status. Solving our research questions 1 and 2 required spotting and identifying date times and equipments with DC/AC power outputs were not optimal. The following 3 step procedure was followed to identify such records with underperformance. For code, please refer to Appendix A.16.

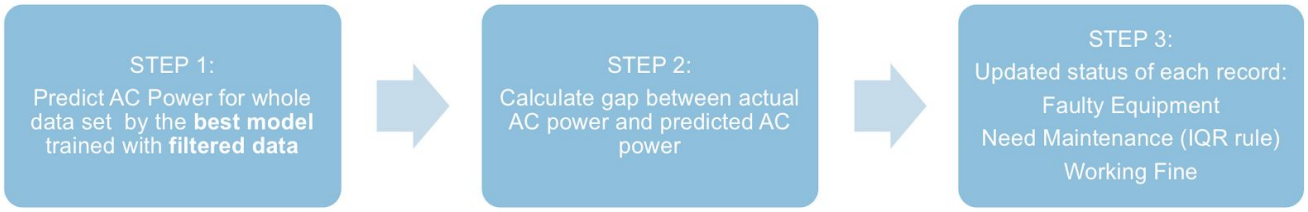


Figure 17. The 3-Step Approach to Label Inverter Performance

6.3.1 Identifying Faulty Equipments

The whole data set was checked for the following conditions:

Irradiation is greater than zero, meaning there is sunlight falling on the solar cells, and predicted AC power is greater than zero. Then,

- DC output is equal zero, meaning no power is being generated by the cells – All records meeting this condition were labelled as 'Faulty Equipment'.
- DC output is greater than zero, but AC output if zero - All records meeting this condition were labelled as 'Faulty Inverters'.

For code, please refer to Appendix A.17.

6.3.2 Identifying Equipments needing Maintenance/Cleaning

For Identifying equipments that were performing and giving some output but not optimally, we followed the 1.5 IQR rule. Interquartile range was calculated for the 'Gap' column (containing the difference between predicted AC power and actual AC power). A cut-off was set to 1.5*IQR to identify the outlier values. For code, please refer to Appendix A.18, Appendix A.17.

Q25: -0.53

Q75: 0.62

IQR: 1.14

Cut off: -2.24 (Q25 – 1.5 * IQR)

Next, all records with Irradiation greater than zero, and actual AC power less than 2.24kW than predicted AC power, were labelled as 'Need Maintenance'.

All other remaining records were labelled as 'Working Fine'.

6.3.3 Subsequent Findings

Out of total number of 67698 records

Number of records with ‘Working Fine’ condition: 52264

Number of records with ‘Need Maintenance’ condition: 11464

Number of records with ‘Faulty Equipment’ condition: 3970

- The most faults were recorded on 2020-06-06 and 2020-06-07
- The most maintenance needs were recorded on 2020-06-07 and 2020-06-02
- The top 3 inverters with most faults or underperformance are Quc1TzYxW2pYoWX, Et9kgGMDl729KT4 and rrq4fwE8jgrTyWY

These findings lay in sync with the findings of exploratory data analysis.

Below figure shows the distribution of AC Power of a Faulty module and a normally functioning module on 7th June. The faulty module stopped producing any power around 9AM till 2PM. For code, please refer to Appendix A.19.

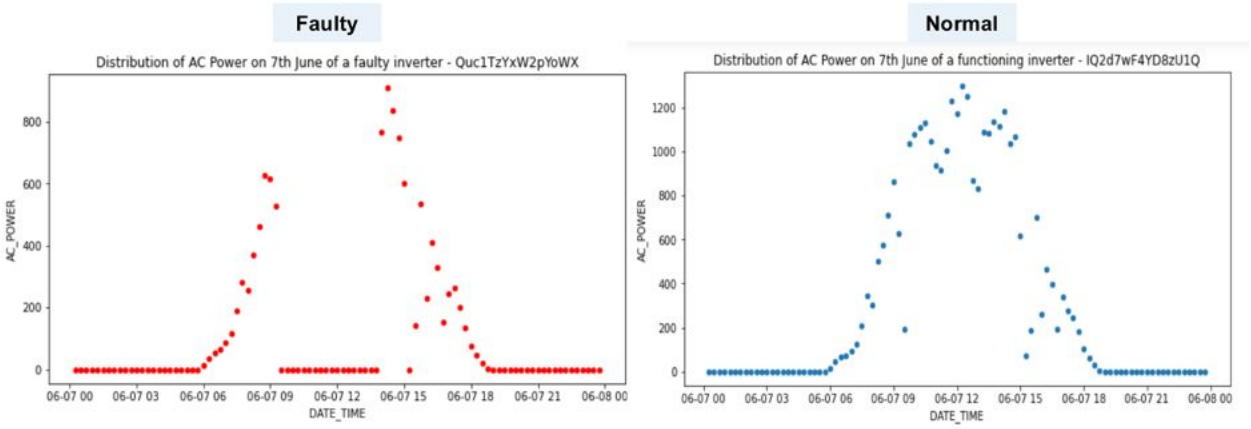


Figure 18. Faulty and Normal Inverter Performance on Same Day

On 7th June a lot of solar modules showed abnormal behavior, generating less than usual power, while other modules kept performing usually. Below figure shows the distribution of AC power of all modules on 7th June and 25th May. The modules with lower than expected outputs were identified as needing maintenance. For code, please refer to Appendix A.20.

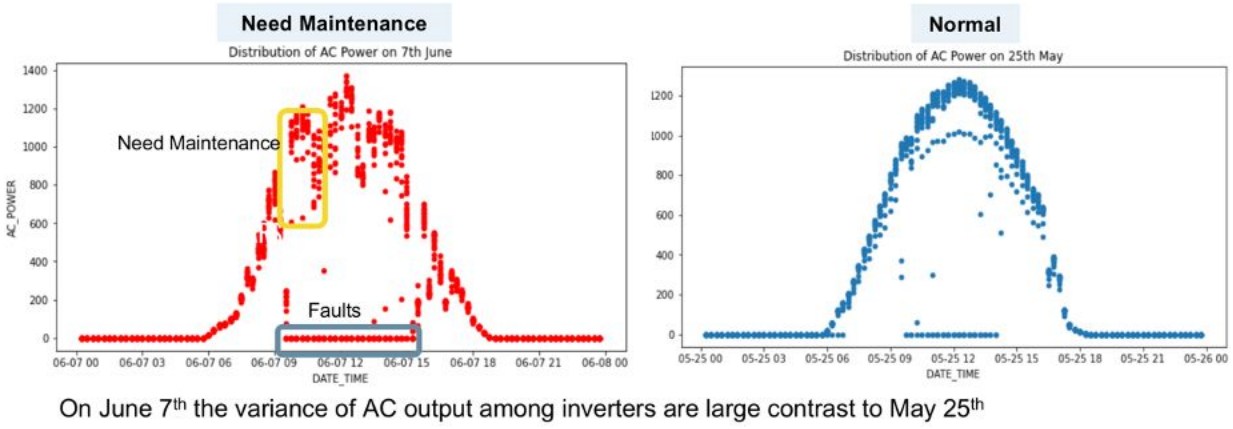


Figure 19. Variance of Power Output Differs between Dates

Below figure shows the distribution of AC Power of an underperforming module and a normally functioning module on 25thMay. A drop in out can be observed for modules on the left around 9AM. This module was identified as needing maintenance. For code, please refer to Appendix A.21.

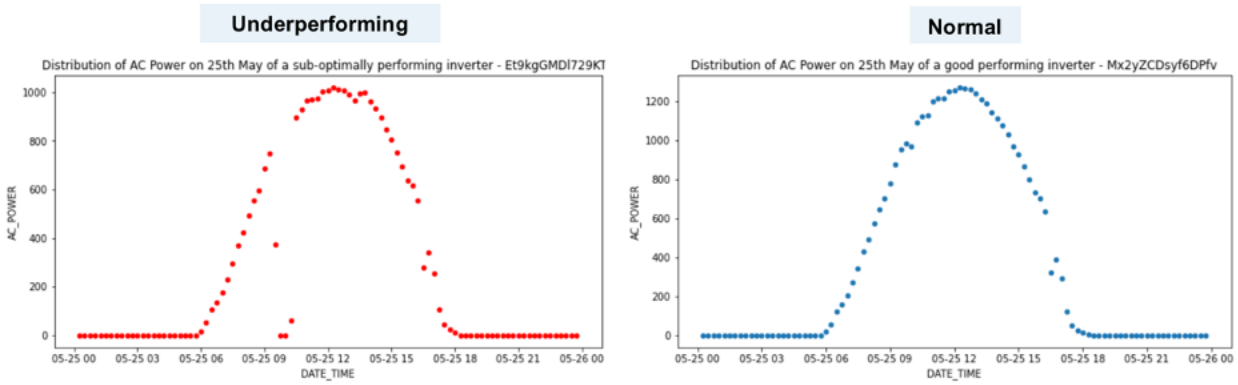


Figure 20. Comparison between Under and Normal Performing Inverters on Same Day

Below figure shows a comparison AC power distribution of an underperforming module and a normally performing module over 34 days period. For code, please refer to Appendix A.22.

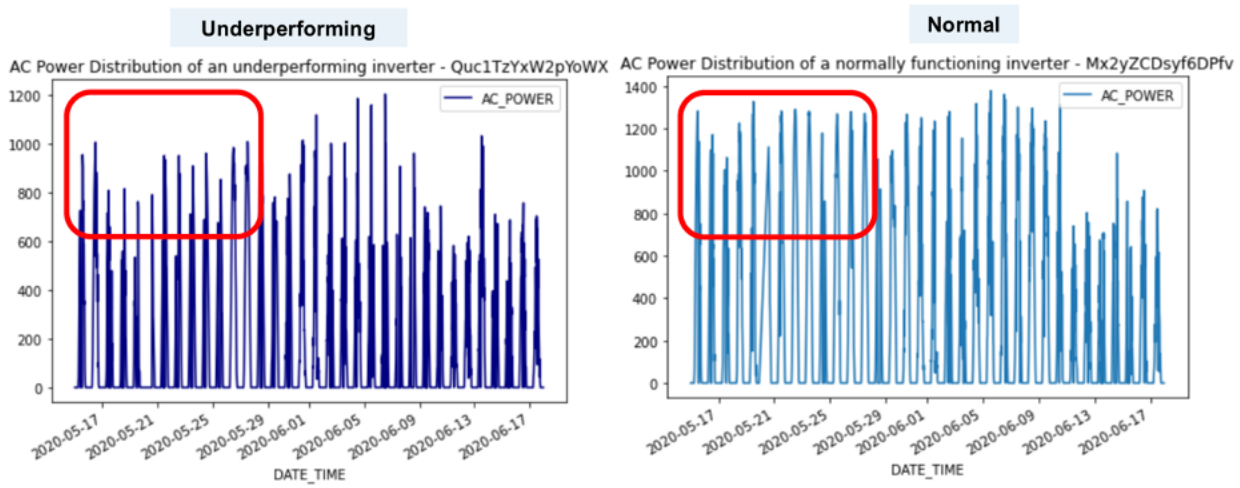


Figure 21.Comparison between Under and Normal Performing Inverters across Dates

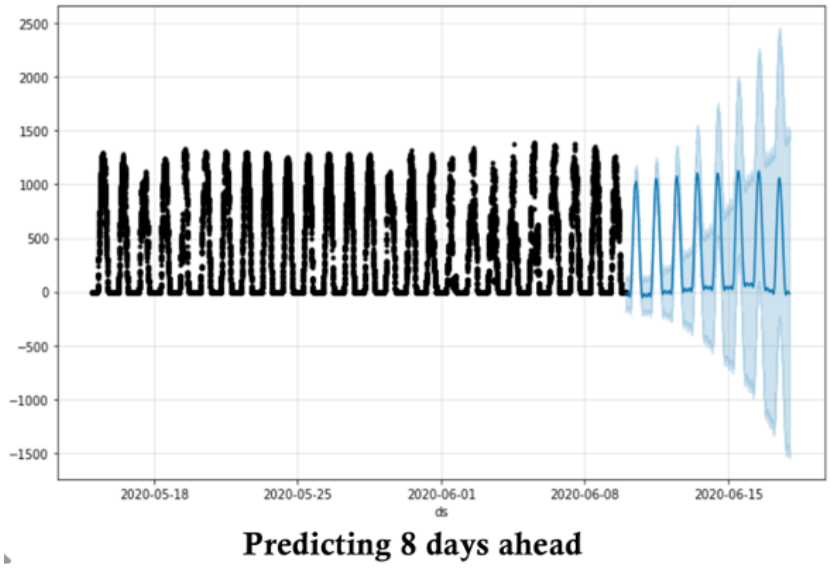
6.4 Machine Learning Models for Power Output Forecast (Research Question 3)

In the above sections Machine Learning models are constructed to predict power output given the variables observed at the same timestamp. These models are capable of monitoring system performance in real-time. Yet for better integration of solar power output with the existing power grid, forecasting output for a given period ahead of time is necessary. These models have no place in that process. This is because these models are constructed under an assumption that each observation is independently and identically distributed (I.I.D.). They fail to model the effect of time that connects observations from one another in forecasting problems.

Time series methods are introduced into this study to conduct forecasting.

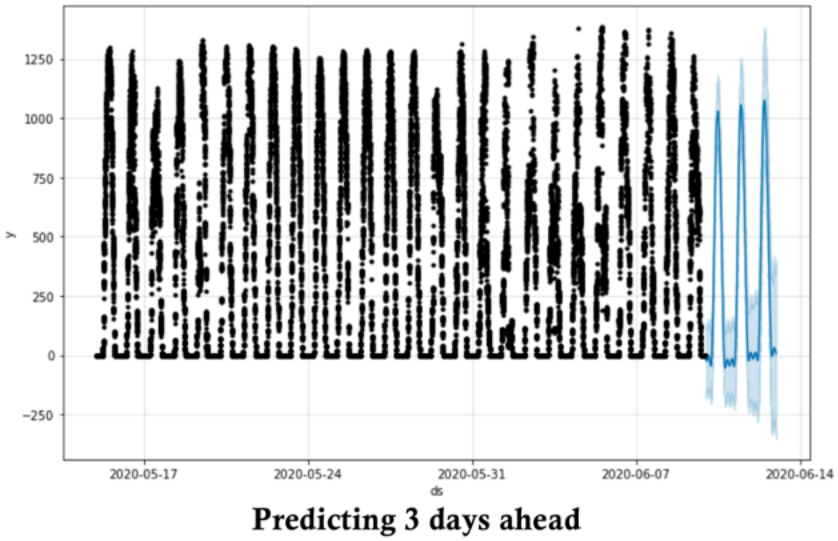
6.4.1 Forecasting with Facebook Prophet

```
df_train = df_input[df_input['DATE_TIME'] < '2020-06-10']
df_predict = df_input[df_input['DATE_TIME'] >= '2020-06-10']
```



- **MAE:** 328.502
- Same performance for filtered and unfiltered data

For code, please refer to Appendix A.23.



- **MAE:** 321.192
- Same performance for filtered and unfiltered data

Figure 22.Forecasting Results for FB Prophet

For code, please refer to Appendix A.24.

6.4.2 Forecasting with VARMA

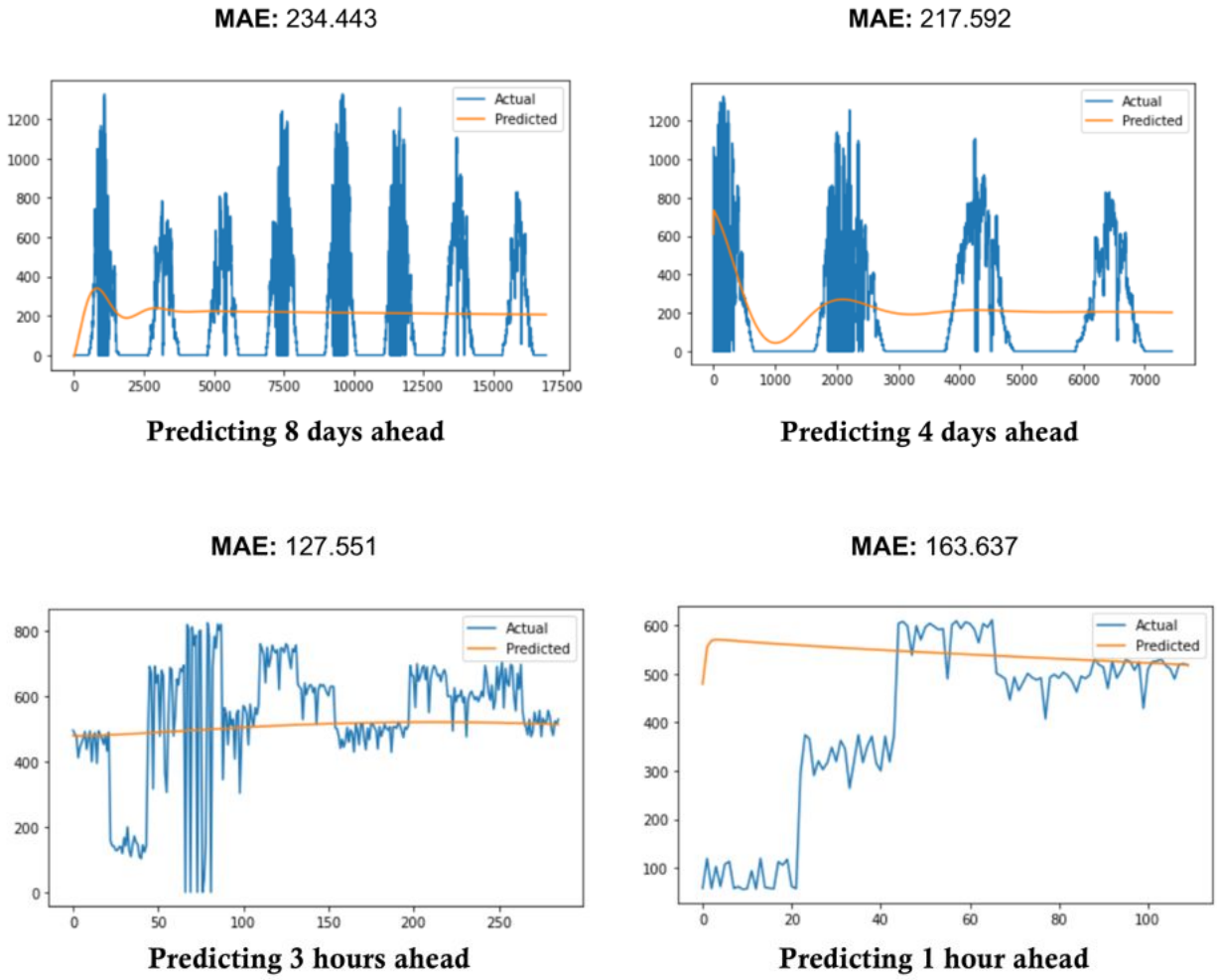


Figure 23.Forecasting Results for VARMA

For code, please refer to Appendix A.25.

While VARMA performed better than Facebook Prophet in predicting future power outputs of the plant, the errors were still high.

6.4.3 Forecasting with I.I.D approach

The I.I.D approach towards time series forecasting is centered around data manipulation. For example, there is a forecasting problem requiring observation on T day is forecasted using data from T-1 day and older. Then original dataset would look like below:

Time	Irradiation	AC output
T day 15:00	1.0	1000
T day 15:15	0.9	920
T day 15:30	1.1	1115
T day 15:45	0.8	788
T day 16:00	0.7	713

Table 5. Original Time Series Data before Manipulation

The manipulation would make observations from T-1, T-2, etc. as features and create a new dataset as below.

Time	AC output	T-1 Irradiation	T-1 AC output	T- 2 Irradiation	T-2 AC output
T day 15:00	1000	1.1	1100	0.9	923
T day 15:15	920	1.0	996	0.9	931
T day 15:30	1115	1.0	1002	0.8	774
T day 15:45	788	0.8	832	0.6	610
T day 16:00	713	0.9	903	0.7	721

Table 6. New Dataset after Manipulation

By doing so, each observation could still be independently and identically distributed, the connection of time is represented in the connection between T-1 and T-2 features. After the manipulation, the new dataset could be input into Machine Learning models that were built based on the I.I.D assumption.

In this study the AC power output at T day is forecasted by 1 day and 3 days ahead. For code please refer to Appendix A.26.Random Forest is used as the I.I.D model for forecasting and the results are shown below. For code please refer to Appendix A.27.

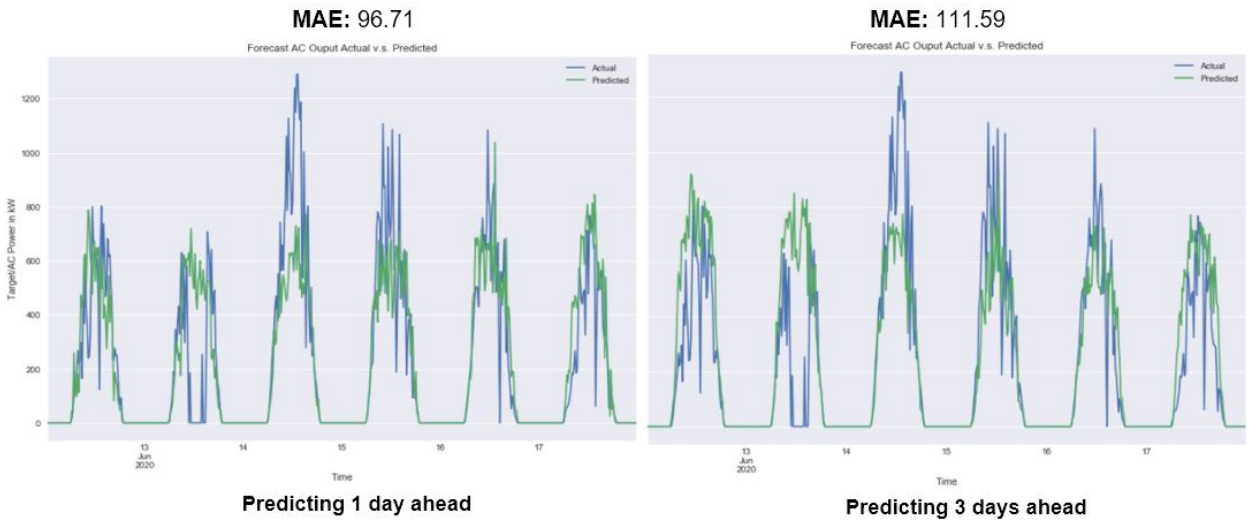


Figure 24. Forecast v.s. Actual with I.I.D Approach

The I.I.D model is trained on data from before June 10th while tested on data after to prevent leakage between train and test sets.

The I.I.D approach performs better forecasting a shorter time horizon 1 day v.s. 3 days ahead.

7 Discussion: Comparison between Machine Learning Models

In part 6.2, 4 models (Linear Regression, Regression Tree, Random Forest and SVR) are used to predict the optimal AC output given environmental and time related inputs. The result is, in terms of RMSE, Random Forest < Regression Tree < SVR < Linear Regression.

In this part, conceptual theories behind some of those models are explained and comparisons are drawn to discuss why some models perform better than others.

7.1 Linear Regression v.s. Decision Regression Tree

Linear Regression is used to predict continuous outputs where there is a linear relationship between the features of the dataset and the output variable. Decision trees can also be used for regression problems and are useful for complex datasets. They work by splitting the dataset, in a tree-like structure, into smaller and smaller subsets and then predict the output value by taking the average of all of the examples that fall into a certain leaf on the decision tree. They learn by splitting the training examples in a way such that the sum of squared residuals is minimized. Decision trees are useful

when there are complex relationships between the features and the output variables. They also work well compared to other algorithms when there are missing features. For our dataset, the Decision Tree Regression model performed better than Linear Regression model due to the dataset having more variations than simple linear relationships between features.

7.2 Random Forest v.s. Regression Tree

Random Forest algorithm is a more complex and flexible improvement of Decision Tree. In Random Forest. a large number of decision trees are built. Each would be constructed on a randomly selected set of features. Then the final output of the Random Forest model is based on the voting of (taking mean in regression problems) outputs from all trees in the forest. This ensemble approach replies on the combination of prediction power from the large number of weak-learning Decision Trees. Compared to a single Decision Tree, Random Forest can better handle the bias-variance trade-off. The different trees built upon random subsets of features can ensure coverage of a large feature space yet without a single Decision Tree's concern of overfitting by needing to grow very deep. Bias and variance are taken care of at the same time.

8 Self Assessment

Both the team members contributed equally to the project. Below is the topic wise distribution.

Topic	**Sub-topic**	**Responsible**
Introduction		Mansi
Dataset Description		Mansi
EDA	Inspection of Data Validity - Discovery of Plant 1 issue	Mansi
	Discovery of Linear Relationships and Outliers	Miao
	Performance for Individual Inverters	Miao
Methodologies towards Research Questions	Filtering Out Performance Outliers	Miao
Machine Learning Models for the 'Golden Standard'	Linear Regression	Mansi

	Decision Tree Regression	Mansi
	SVR	Miao
	Random Forest	Miao
Machine Learning Techniques	Solution to Research Questions 1 & 2 - Labelling	Mansi
Machine Learning Models for Power Output Forecast (Research Question 3)	Facebook Prophet	Mansi
	VARMA	Mansi
	IID	Miao
Comparison between Machine Learning Models		Miao

Table 7. Contribution Table

9 Appendices

Appendix A: Python Code

A.1 Merge Weather and Power Generation Data

```
df_power_2 = pd.read_csv('Plant_2_Generation_Data.csv', header = 0, engine = 'c')
df_weather_2 = pd.read_csv('Plant_2_Weather_Sensor_Data.csv', header = 0, engine = 'c')
# unify the datetime
df_power_2['DATE_TIME'] = pd.to_datetime(df_power_2['DATE_TIME'], errors='coerce')
df_weather_2['DATE_TIME'] = pd.to_datetime(df_weather_2['DATE_TIME'], errors='coerce')
# join power data and weather data, based on DATE_TIME
df_merged_2 = pd.merge(left = df_power_2, right = df_weather_2, on = 'DATE_TIME')
df_merged_2 = df_merged_2.rename(columns = {"PLANT_ID_x":"PLANT_ID", "SOURCE_KEY_x":"Inverter_ID"})
df_merged_2 = df_merged_2.drop(columns = ['PLANT_ID_y','SOURCE_KEY_y'])
# break down DATE_TIME
df_merged_2['month'] = df_merged_2.DATE_TIME.dt.month
df_merged_2['day_of_month'] = df_merged_2.DATE_TIME.dt.day
df_merged_2['hour'] = df_merged_2.DATE_TIME.dt.hour
df_merged_2['minute'] = df_merged_2.DATE_TIME.dt.minute
df_merged_2['day_of_week'] = df_merged_2.DATE_TIME.dt.dayofweek
df_merged_2.head()
```

A.2 DC to AC Power Conversion of Plants

```
df_merged_2.groupby('time')['DC_POWER'].agg('mean').plot(legend=True, colormap='Reds_r', label = 'DC Power')
df_merged_2.groupby('time')['AC_POWER'].agg('mean').plot(legend=True, colormap='Blues_r', label = 'AC Power')
plt.title('Distrbution of DC and AC Output by Time of Day for Plant 1')
plt.ylabel('Output in kW')
plt.show()
```

```
df_merged_1.groupby('time')['DC_POWER'].agg('mean').plot(legend=True, colormap='Reds_r', label = 'DC Power')
df_merged_1.groupby('time')['AC_POWER'].agg('mean').plot(legend=True, colormap='Blues_r', label = 'AC Power')
plt.title('Distrbution of DC and AC Output by Time of Day for Plant 2')
plt.ylabel('Output in kW')
plt.show()
```

A.3 Correlation Matrix

```
# plot correlation matrix
df2 = df_merged_2.drop('PLANT_ID', axis=1)
corr = df2.corr()
corr.style.background_gradient(cmap='coolwarm')
```

A.4 Relationship Between Irradiation and DC Output

```
df_plant_2.plot(x = 'IRRADIATION', y = 'DC_POWER', kind = 'scatter')
plt.title('Relationship between Irradiation and DC Power Output')
plt.show()
```

A.5 Identifying the High and Low Performing Inverters

```
df_plant_1 = df_merged_2.copy()
df_plant_1['Light_conversion'] = df_plant_1['DC_POWER'] / df_plant_1['IRRADIATION']
df_plant_1['Light_conversion'] = df_plant_1['Light_conversion'].fillna(0)
df_plant_1['DCAC_conversion'] = df_plant_1['AC_POWER'] / df_plant_1['DC_POWER']
df_plant_1['DCAC_conversion'] = df_plant_1['DCAC_conversion'].fillna(0)
Light_conv = df_plant_1.loc[df_plant_1['DC_POWER']>0,:].pivot_table(
    values='Light_conversion', index='time', columns='Inverter_ID')
Light_conv.boxplot(figsize=(15,5), grid=False, rot=90)
plt.title('Distribution of Light to DC Power Conversion (0 Removed) by Inverters')
plt.ylabel('Conversion Rate')
plt.show()
```

A.6 Comparing DC Conversion Rates across Time of Day

```
df_plant_1.loc[df_plant_1['Inverter_ID'] == 'Quc1TzYxW2pYoWX',:].groupby('time')['Light_conversion'].agg(
    'mean').plot(legend=True, colormap='Reds_r', label = 'Quc1TzYxW2pYoWX')
df_plant_1.loc[df_plant_1['Inverter_ID'] == 'Et9kgGMD1729KT4',:].groupby('time')['Light_conversion'].agg(
    'mean').plot(legend=True, colormap='Blues_r', label = 'Et9kgGMD1729KT4')
df_plant_1.loc[df_plant_1['Inverter_ID'] == 'Mx2yZCDsyf6DPfv',:].groupby('time')['Light_conversion'].agg(
    'mean').plot(legend=True, colormap='Greens_r', label = 'Mx2yZCDsyf6DPfv')
plt.title(
    'Distribution of Irradiation to DC Conversion by Time of Day of Selected Low and High Performing Inverters')
plt.ylabel('Conversion Rate')
plt.show()
```

A.7 Before and After RANSAC Application

```
df_temp = list_df_inverter[0][['IRRADIATION','AC_POWER']]
X_temp = df_temp['IRRADIATION'].values.reshape(-1,1)
Y_temp = df_temp['AC_POWER'].values.reshape(-1,1)
lr_temp = LinearRegression().fit(X = X_temp, y = Y_temp)
lr_pred_temp = lr_temp.predict(X_temp)
```

```
a1 = plt.scatter(x = X_temp, y = Y_temp)
#plt.scatter(x = X_temp, y = high_temp)
a2 = plt.scatter(x = X_temp, y = lr_pred_temp, c = 'red')
plt.xlabel("Irridiation")
plt.ylabel("Target/AC Power in kW")
plt.title("Linear Regression between Irradiation and AC Output with Unfiltered Data")
plt.legend([a1,a2],['Actual','Predicted'])
plt.show()
```

```
df_temp = list_df_inverter[0][['IRRADIATION','AC_POWER']]
X_temp = df_temp['IRRADIATION'].values.reshape(-1,1)
Y_temp = df_temp['AC_POWER'].values.reshape(-1,1)
lr_temp = RANSACRegressor( residual_threshold = 300, random_state = 0).fit(X_temp, Y_temp)
lr_pred_temp = lr_temp.predict(X_temp)
```

```
a1 = plt.scatter(x = X_temp, y = Y_temp)
#plt.scatter(x = X_temp, y = high_temp)
a2 = plt.scatter(x = X_temp, y = lr_pred_temp)
plt.xlabel("Irridiation")
plt.ylabel("Target/AC Power in kW")
plt.title("Robust Regression between Irradiation and AC Output with Filtered Data")
plt.legend([a1,a2],['Actual','Predicted'])
plt.show()
```

A.8 Selection of the Best Residual Threshold

```
threshold_list = [1,5,10,20,30,40,50,60,70,80,90,100,110,120,130,150,150,160,170,180,190,200]
df_len_list = []
for x in threshold_list:
    list_df_filtered = []
    for i in range(len(inverter_list)):
        df_temp = list_df_inverter[i]
        X_temp = df_temp['IRRADIATION'].values.reshape(-1,1)
        Y_temp = df_temp['AC_POWER'].values.reshape(-1,1)
        lr_temp = RANSACRegressor( residual_threshold = x, random_state = 0).fit(X_temp, Y_temp)
        lr_pred_temp = lr_temp.predict(X_temp)
        a = lr_temp.estimator_.coef_[0,0]
        b = -1
        c = 0
        Distance = abs( a*X_temp + b*Y_temp + c ) / (np.sqrt( a*a + b*b))
        Dis = Distance.reshape(-1)
        dis_iqr = stats.iqr(Dis)
        TF_dis = Distance < ( pd.Series(Dis).quantile(0.75) + 1.5*dis_iqr)
        TF_dis = TF_dis.reshape(-1)
        df_temp = df_temp.loc[TF_dis, :]
        list_df_filtered.append(df_temp)
    df_filter = pd.concat(list_df_filtered)
    df_len_list.append(len(df_filter))
thres_choose = pd.DataFrame({'Threshold': threshold_list, 'df_len': df_len_list})
plt.scatter(x = thres_choose['Threshold'], y = 1 - thres_choose['df_len']/len(df_merged_2))
plt.xlabel("Residual Threshold")
plt.ylabel("Filter Rate")
plt.title("RANSAC Outlier Filter Rate by Residual Threshold")
plt.show()
```

A.9 Comparison: Before and After Filtering Applied

```
list_df_filtered = []
for i in range(len(inverter_list)):
    df_temp = list_df_inverter[i]
    X_temp = df_temp['IRRADIATION'].values.reshape(-1,1)
    Y_temp = df_temp['AC_POWER'].values.reshape(-1,1)
    lr_temp = RANSACRegressor( residual_threshold = 30, random_state = 0).fit(X_temp, Y_temp)
    lr_pred_temp = lr_temp.predict(X_temp)
    a = lr_temp.estimator_.coef_[0,0]
    b = -1
    c = 0
    Distance = abs( a*X_temp + b*Y_temp + c ) / (np.sqrt( a*a + b*b))
    Dis = Distance.reshape(-1)
    dis_iqr = stats.iqr(Dis)
    TF_dis = Distance < ( pd.Series(Dis).quantile(0.75) + 1.5*dis_iqr)
    TF_dis = TF_dis.reshape(-1)
    df_temp = df_temp.loc[TF_dis, :]
    list_df_filtered.append(df_temp)
df_filter = pd.concat(list_df_filtered)
plt.scatter(x = df_filter.loc[:,'IRRADIATION'], y = df_filter.loc[:,'AC_POWER'])
plt.xlabel("Irradiation")
plt.ylabel("Target/AC Power in kW")
plt.title("AC Output by Irradiation after Filtering")
plt.show()
```

A.10 Linear Regression on Unfiltered Data

```
df_input = df_merged_2.copy()
df_input = df_input[['AC_POWER','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION','month','day_of_month',
                     'hour','minute','day_of_week']]
Input = df_input.iloc[:,1:]
Target = df_input.iloc[:,0]
scaler = MinMaxScaler()
Input_scaled = scaler.fit_transform(Input)
Input_scaled  = pd.DataFrame(Input_scaled, columns = Input.columns)
Target = pd.DataFrame(Target)
X = Input_scaled
Y = Target
features = Input.columns
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regr = LinearRegression()
regr.fit(X_train, Y_train)
y_pred = regr.predict(X_test)

## Print outputs
test_set_rmse_LR = (np.sqrt(mean_squared_error(Y_test, y_pred)))
test_set_r2_LR = r2_score(Y_test, y_pred)
output = {'Model':['Linear Regression'], 'RMSE': [test_set_rmse_LR], 'R2' : [test_set_r2_LR]}
print(output)

## Plot outputs
plt.scatter(X_test['IRRADIATION'], Y_test, s=20, color='black', label = True)
plt.plot(X_test['IRRADIATION'], y_pred, color='blue', linewidth=1)
plt.xlabel("Irridiation")
plt.ylabel("Target/AC Power in kW")
plt.title("Linear Regression")
```

A.11 Linear Regression on Filtered Data

```
df_input = df_filter.copy()
df_input = df_input[['AC_POWER','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION','month','day_of_month',
                     'hour','minute','day_of_week']]
Input = df_input.iloc[:,1:]
Target = df_input.iloc[:,0]
scaler = MinMaxScaler()
Input_scaled = scaler.fit_transform(Input)
Input_scaled  = pd.DataFrame(Input_scaled, columns = Input.columns)
Target = pd.DataFrame(Target)
X = Input_scaled
Y = Target
features = Input.columns
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

regr = LinearRegression()
regr.fit(X_train, Y_train)
y_pred = regr.predict(X_test)

## Print outputs
test_set_rmse_LR = (np.sqrt(mean_squared_error(Y_test, y_pred)))
test_set_r2_LR = r2_score(Y_test, y_pred)
output = {'Model':['Linear Regression'], 'RMSE': [test_set_rmse_LR], 'R2' : [test_set_r2_LR]}
print(output)

## Plot outputs
plt.scatter(X_test['IRRADIATION'], Y_test, s=20, color='black', label = True)
plt.plot(X_test['IRRADIATION'], y_pred, color='blue', linewidth=1)
plt.xlabel("Irridiation")
plt.ylabel("Target/AC Power in kW")
plt.title("Linear Regression")
```

A.12 Decision Tree Regression on Filtered Data

```
df_input = df_filter.copy()
df_input = df_input[['AC_POWER','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION','month',
                     'day_of_month','hour','minute','day_of_week']]
Input = df_input.iloc[:,1:]
Target = df_input.iloc[:,0]
scaler = MinMaxScaler() ##standard scale, data cleaning, tune hyper parameters
Input_scaled = scaler.fit_transform(Input)
Input_scaled  = pd.DataFrame(Input_scaled, columns = Input.columns)
Target = pd.DataFrame(Target)
X = Input_scaled
Y = Target
features = Input.columns
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Fit regression model

regr = DecisionTreeRegressor(max_depth=6)
regr.fit(X_train, Y_train)

# Predict
y = regr_2.predict(X_test)

# Plot the results
plt.figure()
plt.scatter(X_test['IRRADIATION'], Y_test, s=20, edgecolor="black", c="darkorange")
plt.plot(X_test['IRRADIATION'], y, color="yellowgreen")
plt.xlabel("Irridiation")
plt.ylabel("Target/AC Power in kW")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

A.13 Decision Tree Regression with Tuning on Filtered Data

```
start_time = timeit.default_timer()
parameters = {'splitter' : ['best', 'random'], 'max_depth' : [4,5,6,15] ,
              'min_samples_split' : [50,60,75,100,1000]}

# Fit regression model
regrT = DecisionTreeRegressor()
clf = GridSearchCV(regrT, parameters, n_jobs = -1, scoring = 'neg_mean_squared_error', cv = 5)
best_regrT = clf.fit(X_train, Y_train)
elapsed = timeit.default_timer() - start_time

# Predict
RegrT_model = best_regrT.best_estimator_
regrT_result = RegrT_model.predict(X_test)

# Plot the results
plt.scatter(X_test['IRRADIATION'], Y_test, s=20, edgecolor="black", c="darkorange")
plt.plot(X_test['IRRADIATION'], regrT_result, color="cornflowerblue")
plt.xlabel("Irridiation")
plt.ylabel("Target/AC Power in kW")
plt.title("Decision Tree Regression with tuned hyperparameters")
plt.legend()
test_set_rmse_RT_Best = (np.sqrt(mean_squared_error(Y_test, regrT_result)))
test_set_r2_RT_Best = r2_score(Y_test, regrT_result)

##Print outputs
output = {'Model':['Regression Tree Best'], 'RMSE': [test_set_rmse_RT_Best], 'R2' : [test_set_r2_RT_Best]}
print(output)
```

A.14 Random Forest with Tuning on Filtered Data

```
start_time = timeit.default_timer()
parameters = {'n_estimators': (101,501,1001,2001), 'max_depth': (
    None,10), 'max_features': (None, 'sqrt')}
RF = RandomForestRegressor(random_state = 0)
clf = GridSearchCV(RF, parameters, n_jobs = -1,
                   scoring = 'neg_mean_squared_error', cv = 3)
best_RF = clf.fit(X_train, Y_train)
elapsed = timeit.default_timer() - start_time
```

```
RF_model = best_RF.best_estimator_
RF_result = RF_model.predict(X_test)
```

```
RF_mse = metrics.mean_squared_error(Y_test, RF_result)
RF_r2 = metrics.r2_score(Y_test, RF_result)
RF_mae = metrics.mean_absolute_error(Y_test, RF_result)
```

A.15 SVR with Tuning on Filtered Data

```
start_time = timeit.default_timer()
parameters_svr = {'kernel': ( 'linear','rbf', 'poly'),
                  'C': [0.1, 1, 5], 'cache_size': [1000]}
svr = SVR()
clf_svr = GridSearchCV(svr, parameters_svr, n_jobs = -1,
                       scoring = 'neg_mean_squared_error', cv = 3)
best_svr = clf_svr.fit(X_train, Y_train)
elapsed = timeit.default_timer() - start_time
```

```
svr_model = best_svr.best_estimator_
svr_result = svr_model.predict(X_test)
```

```
SVR_mse = metrics.mean_squared_error(Y_test, svr_result)
SVR_r2 = metrics.r2_score(Y_test, svr_result)
SVR_mae = metrics.mean_absolute_error(Y_test, svr_result)
```

A.16 Solution to Research Questions 1 & 2 - Steps 1 & 2

```
df_input = df_merged_2.copy()
df_input = df_input[['AC_POWER','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION','month',
                     'day_of_month','hour','minute','day_of_week']]

# Define Inputs, Outputs
Input = df_input.iloc[:,1:]
Target = df_input.iloc[:,0]
scaler = MinMaxScaler() ##standard scale, data cleaning, tune hyper parameters
Input_scaled = scaler.fit_transform(Input)
Input_scaled  = pd.DataFrame(Input_scaled, columns = Input.columns)
Target = pd.DataFrame(Target)
#Input_scaled.head()
X = Input_scaled
Y = Target

# Predict AC Output with Linear Regression model
predicted_values = regr.predict(X)

AC_Power_Predicted = pd.DataFrame(predicted_values)
AC_Power_Predicted = AC_Power_Predicted.rename(columns={0: "AC_Power_Predicted"})
df_predict = df_merged_2.copy()

# Merging predicted AC power with Merged DataFrame of plant 2
df_predict_merged = pd.concat([df_predict,AC_Power_Predicted], axis=1, sort=False)

df_predict_updated = df_predict_merged.copy()

# Updating predicted AC power with 0 if predicted power is <0, otherwise keeping the predicted value
df_predict_updated['AC_Power_Predicted'] = np.where(df_predict_updated['AC_Power_Predicted']<0, 0,
                                                    df_predict_updated['AC_Power_Predicted'])

# Calculating Gap between Actual AC Power and Predicted AC Power and storing in a new Column
df_predict_updated['Gap'] = df_predict_updated['AC_POWER'] - df_predict_updated['AC_Power_Predicted']
df_predict_updated['Date'] = df_predict_updated['DATE_TIME'].dt.date
```

A.17 Solution to Research Questions 1 & 2 - Labelling (Step 3)

```
# Function to update status of each record as faulty, working fine or need maintenance
def update_status(lower):
        for i in range(len(df_predict_updated)):
            gap = df_predict_updated.iloc[i]["Gap"]
            AC_Power = df_predict_updated.iloc[i]["AC_POWER"]
            AC_Power_Predicted = df_predict_updated.iloc[i]["AC_Power_Predicted"]
            Irridiation = df_predict_updated.iloc[i]["IRRADIATION"]
            DC_Power = df_predict_updated.iloc[i]["DC_POWER"]
            if(Irridiation > 0):
                if((AC_Power == 0.0) & (AC_Power_Predicted > 0)):
                    if(DC_Power > 0.0):
                        df_predict_updated.at[i, 'Status'] = "Faulty Inverter"
                    else:
                        df_predict_updated.at[i, 'Status'] = "Faulty Equipment"
                elif(gap < lower):
                    df_predict_updated.at[i, 'Status'] = "Need Maintenance"
                else:
                    df_predict_updated.at[i, 'Status'] = "Working Fine"
            elif(Irridiation == 0.0):
                    df_predict_updated.at[i, 'Status'] = "Working Fine"
```

A.18 IQR and Cut-off Calculation for Gap

```
q75, q25 = np.percentile(df_predict_updated.Gap, [75,25])
iqr = q75 - q25
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
print("q25:", q25 ,"\nq75:",q75,"\nIQR:",iqr, "\nLower Cut Off:",lower, "\nUpper Cut Off:",upper )
```

A.19 Faulty and Normal Inverter Performance on Same Day

```
df_merged_2[((df_merged_2['Inverter_ID'] == 'Quc1TzYxW2pYoWX') & (df_merged_2['DATE_TIME'] > '2020-06-07') &
             (df_merged_2['DATE_TIME'] < '2020-06-08'))].plot(x = 'DATE_TIME', y = 'AC_POWER', kind = 'scatter',
                                                              color = 'red')
plt.title("Distribution of AC Power on 7th June of a faulty inverter - Quc1TzYxW2pYoWX")
df_merged_2[((df_merged_2['Inverter_ID'] == 'IQ2d7wF4YD8zU1Q') & (df_merged_2['DATE_TIME'] > '2020-06-07') &
             (df_merged_2['DATE_TIME'] < '2020-06-08'))].plot(x = 'DATE_TIME', y = 'AC_POWER', kind = 'scatter')
plt.title("Distribution of AC Power on 7th June of a functioning inverter - IQ2d7wF4YD8zU1Q")
plt.show()
```

A.20 Variance of Power Outputs of all Inverters on Two Different Dates

```
df_merged_2[((df_merged_2['DATE_TIME'] > '2020-06-07') & (df_merged_2['DATE_TIME'] < '2020-06-08'))].plot
            (x = 'DATE_TIME', y = 'AC_POWER', kind = 'scatter', color = 'red',figsize=(10,5))
plt.title("Distribution of AC Power on 7th June")
df_merged_2[((df_merged_2['DATE_TIME'] > '2020-05-25') & (df_merged_2['DATE_TIME'] < '2020-05-26'))].plot
            (x = 'DATE_TIME', y = 'AC_POWER', kind = 'scatter',figsize=(10,5))
plt.title("Distribution of AC Power on 25th May")
plt.show()
```

A.21 Comparison Same Day Under and Normal Performing Inverters

```
df_merged_2[((df_merged_2['Inverter_ID'] == 'Et9kgGMDl729KT4') & (df_merged_2['DATE_TIME'] > '2020-05-25') &
             (df_merged_2['DATE_TIME'] < '2020-05-26'))].plot(x = 'DATE_TIME', y = 'AC_POWER', kind = 'scatter',
                                                              color = 'red',figsize=(10,5))
plt.title("Distribution of AC Power on 25th May of a sub-optimally performing inverter - Et9kgGMDl729KT4")
df_merged_2[((df_merged_2['Inverter_ID'] == 'Mx2yZCDsyf6DPfv') & (df_merged_2['DATE_TIME'] > '2020-05-25') &
             (df_merged_2['DATE_TIME'] < '2020-05-26'))].plot(x = 'DATE_TIME', y = 'AC_POWER', kind = 'scatter',
                                                              figsize=(10,5))
plt.title("Distribution of AC Power on 25th May of a good performing inverter - Mx2yZCDsyf6DPfv")
plt.show()
```

A.22 Comparison across Dates Under and Normal Performing Inverters

```
df_merged_2.loc[df_merged_2['Inverter_ID'] == 'Quc1TzYxW2pYoWX',:].groupby('DATE_TIME')['AC_POWER']
               .agg('sum').plot(x = 'DATE_TIME', y = 'AC_POWER', color = 'navy',figsize=(7,5))
plt.title("AC Power Distribution of an underperforming inverter - Quc1TzYxW2pYoWX")
plt.legend()
```

```
df_merged_2.loc[df_merged_2['Inverter_ID'] == 'Mx2yZCDsyf6DPfv',:].groupby('DATE_TIME')['AC_POWER']
               .agg('sum').plot(x = 'DATE_TIME', y = 'AC_POWER',figsize=(7,5))
plt.title("AC Power Distribution of a normally functioning inverter - Mx2yZCDsyf6DPfv")
plt.legend()
```

A.23 Forecasting with Facebook Prophet - Predicting 8 Days Ahead

```
# Define Input and Output
df_input = df_merged_2.copy()
df_input = df_input[['DATE_TIME','AC_POWER']]
df_train = df_input[df_input['DATE_TIME'] < '2020-06-10']
df_predict = df_input[df_input['DATE_TIME'] >= '2020-06-10']
df_train.columns = ['ds', 'y']
df_predict = df_predict.drop('AC_POWER', axis = 1)
df_predict.columns = ['ds']

model = Prophet()
# fit the model
model.fit(df_train)

# use the model to make a forecast
forecast = model.predict(df_predict)

# summarize the forecast
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())

# plot forecast
model.plot(forecast)
```

A.24 Forecasting with Facebook Prophet - Predicting 3 Days Ahead

```
df_train = df_input[df_input['DATE_TIME'] < '2020-06-10']
df_predict = df_input[(df_input['DATE_TIME'] >= '2020-06-10') & (df_input['DATE_TIME'] < '2020-06-13')]
df_train.columns = ['ds', 'y']
df_predict = df_predict.drop('AC_POWER', axis = 1)
df_predict.columns = ['ds']
```

```
model = Prophet()
# fit the model
model.fit(df_train)
# use the model to make a forecast
forecast = model.predict(df_predict)
# summarize the forecast
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())
# plot forecast
model.plot(forecast)
```

A.25 Forecasting with VARMA

```
# VARMA example
from statsmodels.tsa.statespace.varmax import VARMAX
from random import random

df_input = df_merged_2.copy()
#df_input = df_input[['AC_POWER','IRRADIATION','month']]
df_input = df_input[['AC_POWER','DATE_TIME','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION','month',
                     'day_of_month','hour','minute','day_of_week']]
df_input = df_input.drop('DATE_TIME', axis=1)
df_input.index = df_merged_2.DATE_TIME

#creating the train and validation set
train = df_input.loc[:'2020-06-17 14:45:00']
valid = df_input.loc['2020-06-17 13:45:00': '2020-06-17 14:45:00']
# train = int(len(df_input[df_input['DATE_TIME'] < '2020-06-10']))
# valid = int(len(df_input[df_input['DATE_TIME'] >= '2020-06-10']))

#fit the model
from statsmodels.tsa.vector_ar.var_model import VAR

model = VAR(endog=train)
model_fit = model.fit()

# make prediction on validation
prediction = model_fit.forecast(model_fit.y, steps=len(valid))
```

```
cols = df_input.columns
#converting predictions to dataframe

pred = pd.DataFrame(index=range(0,len(prediction)),columns=[cols])
for j in range(0,2):
    for i in range(0, len(prediction)):
        pred.iloc[i][j] = prediction[i][j]

#pred.info()
pred.columns = pred.columns.get_level_values(0)
pred['AC_POWER'] = pred['AC_POWER'].astype(float)
y_trueV = valid['AC_POWER'].values
y_predV = pred['AC_POWER'].values
mae = mean_absolute_error(y_trueV, y_predV)
print('MAE: %.3f' % mae)

# plot expected vs actual
pyplot.figure(figsize=(7,4))
pyplot.plot(y_trueV, label='Actual')
pyplot.plot(y_predV, label='Predicted')

pyplot.legend()
#pyplot.show()
```

A.26 Data Manipulation for I.I.D Approach

```
df_ts = df_merged_2.copy()
t_time = df_ts['DATE_TIME']
t_1_time = t_time - dt.timedelta(days = 1)
t_2_time = t_time - dt.timedelta(days = 2)
t_3_time = t_time - dt.timedelta(days = 3)
df_t_1 = df_ts.loc[df_ts['DATE_TIME'].isin(t_1_time),:]
df_t_1['DATE_TIME_original'] = df_t_1['DATE_TIME'] + dt.timedelta(days = 1)
df_t_1 = df_t_1[['DATE_TIME_original','Inverter_ID','AC_POWER','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION']]
df_t_1.columns = ['DATE_TIME','Inverter_ID','AC_POWER_1','AMBIENT_TEMPERATURE_1','MODULE_TEMPERATURE_1','IRRADIATION_1']
df_t_1.reset_index()
df_t_2 = df_ts.loc[df_ts['DATE_TIME'].isin(t_2_time),:]
df_t_2['DATE_TIME_original'] = df_t_2['DATE_TIME'] + dt.timedelta(days = 2)
df_t_2 = df_t_2[['DATE_TIME_original','Inverter_ID','AC_POWER','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION']]
df_t_2.columns = ['DATE_TIME','Inverter_ID','AC_POWER_2','AMBIENT_TEMPERATURE_2','MODULE_TEMPERATURE_2','IRRADIATION_2']
df_t_2.reset_index()
df_t_3 = df_ts.loc[df_ts['DATE_TIME'].isin(t_3_time),:]
df_t_3['DATE_TIME_original'] = df_t_3['DATE_TIME'] + dt.timedelta(days = 3)
df_t_3 = df_t_3[['DATE_TIME_original','Inverter_ID','AC_POWER','AMBIENT_TEMPERATURE','MODULE_TEMPERATURE','IRRADIATION']]
df_t_3.columns = ['DATE_TIME','Inverter_ID','AC_POWER_3','AMBIENT_TEMPERATURE_3','MODULE_TEMPERATURE_3','IRRADIATION_3']
df_t_3.reset_index()
```

...

```
df_ts = df_ts[['DATE_TIME','Inverter_ID','AC_POWER']]
```

```
df_ts = pd.merge(df_ts, df_t_1, how = 'right', on = ['DATE_TIME','Inverter_ID'])
df_ts = pd.merge(df_ts, df_t_2, how = 'right', on = ['DATE_TIME','Inverter_ID'])
df_ts = pd.merge(df_ts, df_t_3, how = 'right', on = ['DATE_TIME','Inverter_ID'])
```

```
df_input = df_ts.copy()
df_input = df_input.dropna()
df_input = df_input[['AC_POWER','Inverter_ID','AC_POWER_1','AMBIENT_TEMPERATURE_1','MODULE_TEMPERATURE_1','IRRADIATION_1',
                     'AC_POWER_2','AMBIENT_TEMPERATURE_2','MODULE_TEMPERATURE_2','IRRADIATION_2','AC_POWER_3',
                     'AMBIENT_TEMPERATURE_3','MODULE_TEMPERATURE_3','IRRADIATION_3']]
le = LabelEncoder()
col_Inverter_ID = le.fit_transform(df_input['Inverter_ID'])
df_input['Inverter_ID'] = col_Inverter_ID
scaler = MinMaxScaler()
Input = df_input.iloc[:,1:]
Target = df_input.iloc[:,0]
Input_scaled = scaler.fit_transform(Input)
Input_scaled  = pd.DataFrame(Input_scaled, columns = Input.columns)
X = Input_scaled
Y = Target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

A.27 Forecast v.s. Actual with I.I.D Approach

```
start_time = timeit.default_timer()
parameters = {'n_estimators':[100,500,800,1000]}
RF = RandomForestRegressor(random_state = 0)
clf = GridSearchCV(RF, parameters, n_jobs = -1, scoring = 'neg_mean_squared_error', cv = 3)
best_RF = clf.fit(X_train, Y_train)
elapsed = timeit.default_timer() - start_time
```

```
RF_model = best_RF.best_estimator_
RF_result = RF_model.predict(X_test)
```

```
RF_mse = metrics.mean_squared_error(Y_test, RF_result)
RF_r2 = metrics.r2_score(Y_test, RF_result)
RF_mae = metrics.mean_absolute_error(Y_test, RF_result)
```

```
Rf_forecast = pd.DataFrame({'Actual': Y_test, 'Predicted': RF_result})
Date_time = df_ts[['DATE_TIME','Inverter_ID']]
Rf_forecast = Rf_forecast.merge(Date_time, left_index = True, right_index = True)
```

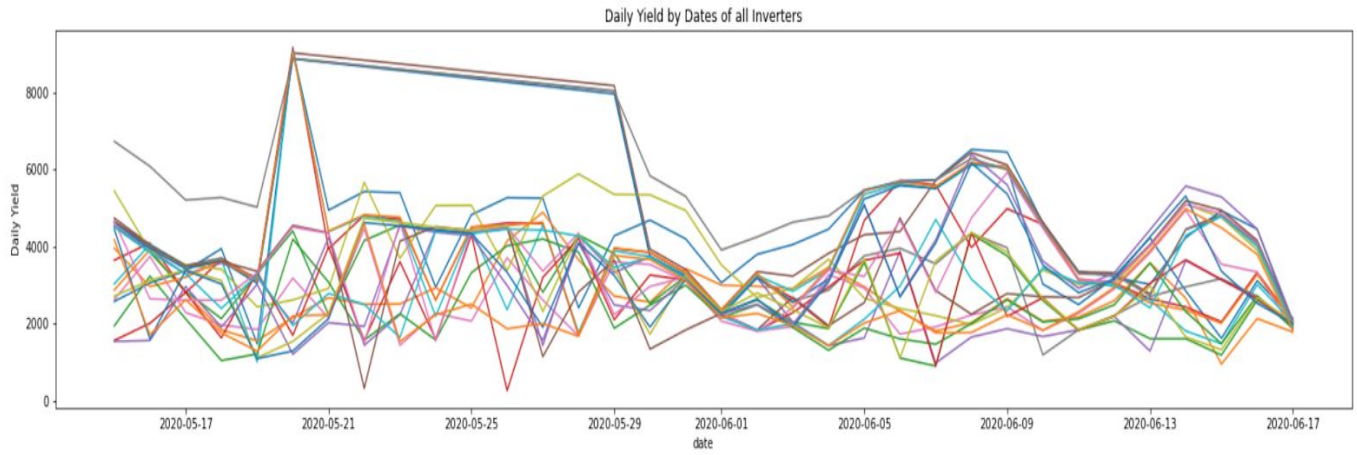
```
Rf_forecast['DATE_TIME']= pd.to_datetime(Rf_forecast.DATE_TIME)
Rf_forecast = Rf_forecast.loc[(Rf_forecast['DATE_TIME'] > '2020-06-12' )&
                              (Rf_forecast['Inverter_ID'] == 'xMbIugepa2P71BB'),:]
```

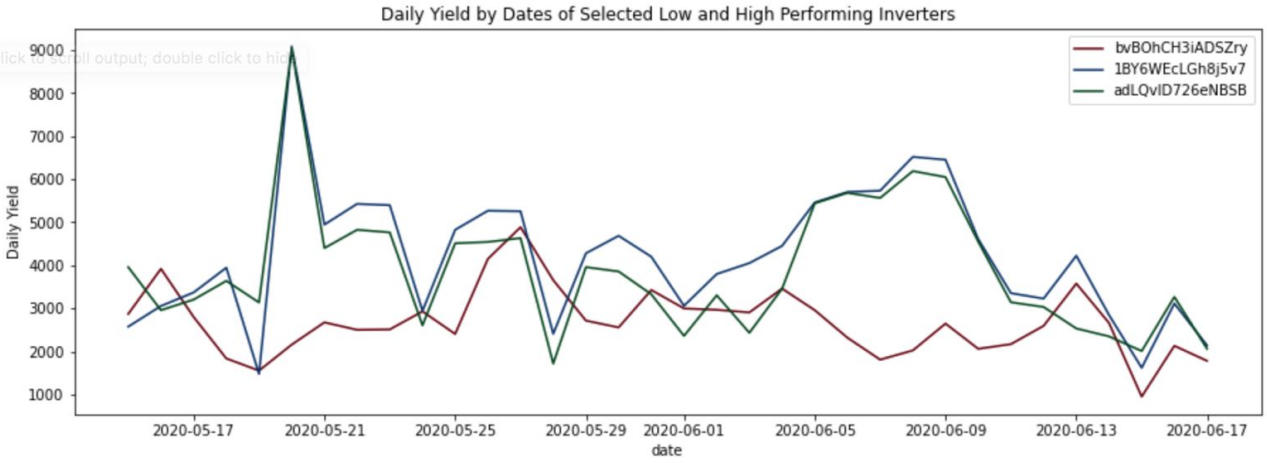
```
Rf_forecast.plot(x = 'DATE_TIME',figsize=(12,8))
plt.xlabel("Time")
plt.ylabel("Target/AC Power in kW")
plt.title("Forecast AC Ouput Actual v.s. Predicted")
plt.show()
```

Appendix B: Pseudo Code for RANSAC

```
Given:
    data – A set of observations.
    model – A model to explain observed data points.
    n – Minimum number of data points required to estimate model parameters.
    k – Maximum number of iterations allowed in the algorithm.
    t – Threshold value to determine data points that are fit well by model.
    d – Number of close data points required to assert that a model fits well to data.

Return:
    bestFit – model parameters which best fit the data (or null if no good model is found)

iterations = 0
bestFit = null
bestErr = something really large

while iterations < k do
    maybeInliers  := n randomly selected values from data
    maybeModel  := model parameters fitted to maybeInliers
    alsoInliers  := empty set
    for every point in data not in maybeInliers do
        if point fits maybeModel with an error smaller than t
             add point to alsoInliers
    end for
    if the number of elements in alsoInliers is > d then
        // This implies that we may have found a good model
        // now test how good it is.
        betterModel  := model parameters fitted to all points in maybeInliers and alsoInliers
        thisErr  := a measure of how well betterModel fits these points
        if thisErr < bestErr then
            bestFit  := betterModel
            bestErr  := thisErr
        end if
    end if
    increment iterations
end while

return bestFit
```

Appendix C: Additional EDA


Daily Yield by Dates of all Inverters
Daily Yield
0
2000
4000
6000
8000
2020-05-17
2020-05-21
2020-05-25
2020-05-29
2020-06-01
2020-06-05
2020-06-09
2020-06-13
2020-06-17
date


Daily Yield by Dates of Selected Low and High Performing Inverters
bvBOhCH3iADSZry
1BY6WEcLGh8j5v7
adLQvlD726eNBSB
Daily Yield
1000
2000
3000
4000
5000
6000
7000
8000
9000
2020-05-17
2020-05-21
2020-05-25
2020-05-29
2020-06-01
2020-06-05
2020-06-09
2020-06-13
2020-06-17
date