National Level Online Hack-A-Thon

On

Sustainable Energy

DETAILED REPORT

SUBMITTED BY:

VIT/OW/55

TEAM MEMBERS:

Sarthak Kiran Sarode

(18bee1074)

Ashutosh Sanjay Lembhe

(18bee1025)

DATE: 08/01/2022

PROBLEM STATEMENT:

1. OLAP operation of the data in front end (dice, slice, roll up/ down, filter)

2. Ability to notify significant changes in the time series dataset imported in the tool

3. Ability to select from and to time stamp in the time series visualization and give a label or annotation

4. Annotation tool - Data labelling where the customer can import the data and the multiple columns render it in the chart where they can select from all and to frame and labelled the part and save in the database.

5. Exploratory data analysis- Where they can explore the data and find its relationship with the different parameters.

6. Prediction Analysis/ modelling - Where they can pass the data and application has to automatically select which model is best and show it's all the model accuracy results.

7. Have a dashboard to display aggregated values.

8. Results should be in pictorial representation.

9. Data cleaning/Data sanitisation must be done (Should not have null values).

10. Working video of the application is expected.

SOFTWARES USED:

LIBRARIES USED:

EXPLORATORY ANALYSIS ON SOLAR DATASET

ENERGY EFFICIENCY PREDICTION BY AUTOML USING PYCART WITH A REGRESSION

(Sub problem statement 5, 9 covered)

Introduction: We have taken the solar dataset and we are going to develop the regression machine learning model for the **temperature parameter**. As the solar panel's output mainly depends upon the temperature.

1) Importing the essential libraries and importing data set in python jupyter notebook.

In [1]:
```
from pycaret.classification import*
```

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

Exploratory Data Analysis

In [3]:
```
data=pd.read_csv('Final_SolarData.csv')
```

In [4]:
```
data.head()
```

Out[4]:

	location	date_time	solarvoltage	solarcurrent	solarenergy	solarpower	batteryvoltage	batterycurrent	batterypower	load_energy1	load_power1	load_curren
0	Peru	2015-12-02 00:00:27	0.0	0.0	0.0	0.0	98.78	0.0	0.0	0.00	182.47	0.!
1	Peru	2015-12-02 00:01:40	0.0	0.0	0.0	0.0	98.80	0.0	0.0	0.01	192.18	1.(
2	Peru	2015-12-02 00:02:52	0.0	0.0	0.0	0.0	98.55	0.0	0.0	0.00	185.28	0.!
3	Peru	2015-12-02 00:04:05	0.0	0.0	0.0	0.0	98.64	0.0	0.0	0.00	175.68	0.!
4	Peru	2015-12-02 00:05:18	0.0	0.0	0.0	0.0	98.59	0.0	0.0	0.01	188.92	1.(

Basic Analysis of checking the datatypes and info of the dataframe

```
In [6]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119296 entries, 0 to 119295
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   location               119296 non-null  object 
 1   date_time              119296 non-null  object 
 2   solarvoltage           119296 non-null  float64
 3   solarcurrent           119296 non-null  float64
 4   solarenergy            119296 non-null  float64
 5   solarpower             119296 non-null  float64
 6   batteryvoltage         119296 non-null  float64
 7   batterycurrent         119296 non-null  float64
 8   batterypower           119296 non-null  float64
 9   load_energy1           119296 non-null  float64
 10  load_power1            119296 non-null  float64
 11  load_current1          119296 non-null  float64
 12  load_voltage1          119296 non-null  float64
 13  load_energy2           119296 non-null  float64
 14  load_power2            119296 non-null  float64
 15  load_current2          119296 non-null  float64
 16  load_voltage2          119296 non-null  float64
 17  inverter_input_power   119296 non-null  float64
 18  inverter_output_power  119296 non-null  float64
 19  inverter_input_energy  119296 non-null  float64
 20  inverter_output_energy 119296 non-null  float64
 21  irradiance             119296 non-null  int64  
 22  temperature            119120 non-null  float64
dtypes: float64(20), int64(1), object(2)
memory usage: 20.9+ MB
```

Here the data type of date_time column is object we are supposed to convert it into standard format which is done ahead.

- -

In [5]:
```
data.dtypes
```

Out[5]:
```
location                  object
date_time                 object
solarvoltage             float64
solarcurrent             float64
solarenergy              float64
solarpower               float64
batteryvoltage           float64
batterycurrent           float64
batterypower             float64
load_energy1             float64
load_power1              float64
load_current1            float64
load_voltage1            float64
load_energy2             float64
load_power2              float64
load_current2            float64
load_voltage2            float64
inverter_input_power     float64
inverter_output_power    float64
inverter_input_energy    float64
inverter_output_energy   float64
irradiance                 int64
temperature              float64
dtype: object
```

Shape of the dataframe before data cleaning i.e missing values cleaning

In [6]:
```
data.shape
```

Out[6]:
```
(119296, 23)
```

2) Missing values cleaning and Data Sanitization (9 th sub problem statement)

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
location                  0
date_time                 0
solarvoltage              0
solarcurrent              0
solarenergy               0
solarpower                0
batteryvoltage            0
batterycurrent            0
batterypower              0
load_energy1              0
load_power1               0
load_current1             0
load_voltage1             0
load_energy2              0
load_power2               0
load_current2             0
load_voltage2             0
inverter_input_power      0
inverter_output_power     0
inverter_input_energy     0
inverter_output_energy    0
irradiance                0
temperature             176
dtype: int64
```

```
In [8]: data.isnull().sum().sort_values(ascending=False)

Out[8]: temperature              176
        load_voltage1              0
        irradiance                 0
        inverter_output_energy     0
        inverter_input_energy      0
        inverter_output_power      0
        inverter_input_power       0
        load_voltage2              0
        load_current2              0
        load_power2                0
        load_energy2               0
        location                   0
        date_time                  0
        load_power1                0
        load_energy1               0
        batterypower               0
        batterycurrent             0
        batteryvoltage             0
        solarpower                 0
        solarenergy                0
        solarcurrent               0
        solarvoltage               0
        load_current1              0
        dtype: int64
```

Temperature has some missing values. Checking which row of temperature has missing value.

```
In [9]: missing_values= pd.isnull(data["temperature"])
        missing_values

Out[9]: 0         False
        1         False
        2         False
        3         False
        4         False
                  ...
        119291    False
        119292    False
        119293    False
        119294    False
        119295    False
        Name: temperature, Length: 119296, dtype: bool
```

“False” means no missing values. Dropping rows having missing values.

```
In [7]: data.dropna()
```

Out[7]:

	location	date_time	solarvoltage	solarcurrent	solarenergy	solarpower	batteryvoltage	batterycurrent	batterypower	load_energy1	load_power1	load_c
0	Peru	2015-12-02 00:00:27	0.0	0.00	0.0	0.0	98.78	0.00	0.00	0.00	182.47	
1	Peru	2015-12-02 00:01:40	0.0	0.00	0.0	0.0	98.80	0.00	0.00	0.01	192.18	
2	Peru	2015-12-02 00:02:52	0.0	0.00	0.0	0.0	98.55	0.00	0.00	0.00	185.28	
3	Peru	2015-12-02 00:04:05	0.0	0.00	0.0	0.0	98.64	0.00	0.00	0.00	175.68	
4	Peru	2015-12-02 00:05:18	0.0	0.00	0.0	0.0	98.59	0.00	0.00	0.01	188.92	
...	...	...	...	...	...	...	...	...	...	...	...	
119291	Peru	2016-03-14 22:22:02	0.0	1.11	0.0	0.0	94.88	1.50	142.87	0.00	124.63	
119292	Peru	2016-03-14 22:23:14	0.0	1.11	0.0	0.0	94.58	0.47	44.50	0.01	114.46	
119293	Peru	2016-03-14 22:24:27	0.0	1.23	0.0	0.0	93.97	0.28	26.53	0.00	131.01	
119294	Peru	2016-03-14 22:25:40	0.0	1.17	0.0	0.0	94.41	1.19	112.54	0.00	121.10	
119295	Peru	2016-03-14	0.0	1.17	0.0	0.0	94.33	1.28	121.32	0.00	124.50	

Deleting the columns which has missing values.

```
In [10]: columns_with_na_dropped = data.dropna(axis=1)
         columns_with_na_dropped.head()
```

Out[10]:

	location	date_time	solarvoltage	solarcurrent	solarenergy	solarpow
0	Peru	2015-12-02 00:00:27	0.0	0.0	0.0	(
1	Peru	2015-12-02 00:01:40	0.0	0.0	0.0	(
2	Peru	2015-12-02 00:02:52	0.0	0.0	0.0	(
3	Peru	2015-12-02 00:04:05	0.0	0.0	0.0	(
4	Peru	2015-12-02 00:05:18	0.0	0.0	0.0	(

Final conclusion on cleaning the missing value.

In [9]:
```
print("Columns in original dataset: %d \n" % data.shape[1])
print("Columns with na's dropped: %d" % columns_with_na_dropped.shape[1])
```

```
Columns in original dataset: 23

Columns with na's dropped: 22
```

3) Date datatype converting to standard format.

In [12]:
```
data['date_time'] = pd.to_datetime(data['date_time'])
```

In [11]:
```
data.dtypes
```

Out[11]:
```
location                       object
date_time              datetime64[ns]
solarvoltage                  float64
solarcurrent                  float64
solarenergy                   float64
solarpower                    float64
batteryvoltage                float64
batterycurrent                float64
batterypower                  float64
load_energy1                  float64
load_power1                   float64
load_current1                 float64
load_voltage1                 float64
load_energy2                  float64
load_power2                   float64
load_current2                 float64
load_voltage2                 float64
inverter_input_power          float64
inverter_output_power         float64
inverter_input_energy         float64
inverter_output_energy        float64
irradiance                      int64
temperature                   float64
dtype: object
```

4) Statistical Analysis of the dataset.Checking the skewness and the kurtosis of the dataset.

In [13]:
```
data.skew()
```

Out[13]:
```
solarvoltage               0.169811
solarcurrent               1.085848
solarenergy               58.399654
solarpower                 0.879099
batteryvoltage            -0.542736
batterycurrent             1.119845
batterypower               1.151977
load_energy1             151.550088
load_power1                1.113178
load_current1              1.148885
load_voltage1             -9.679776
load_energy2               5.504227
load_power2                0.050937
load_current2              0.073336
load_voltage2             -9.679776
inverter_input_power       2.435407
inverter_output_power    235.642088
inverter_input_energy     48.835040
inverter_output_energy    21.545588
irradiance                 1.900642
temperature                2.636329
dtype: float64
```

```
In [14]: data.kurt()

Out[14]: solarvoltage                  -1.867285
         solarcurrent                   0.343499
         solarenergy                 8778.367807
         solarpower                    -0.640969
         batteryvoltage                11.710920
         batterycurrent                 0.087627
         batterypower                   0.196991
         load_energy1               38758.832564
         load_power1                    1.795199
         load_current1                  1.867863
         load_voltage1                 99.725862
         load_energy2                 229.848240
         load_power2                    0.011164
         load_current2                  0.127850
         load_voltage2                 99.725862
         inverter_input_power           8.011426
         inverter_output_power      56855.173231
         inverter_input_energy       6060.402081
         inverter_output_energy      1439.582866
         irradiance                     3.105551
         temperature                   12.743482
         dtype: float64
```

If skewness is less than -1 or greater than 1, the distribution is highly skewed. If skewness is between -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed. If skewness is between -0.5 and 0.5, the distribution is approximately symmetric.

Low kurtosis in a data set is an indicator that data has light tails or lack of outliers. ... The peak is lower and broader than Mesokurtic, which means that data are light-tailed or lack of outliers. The reason for this is because the extreme values are less than that of the normal distribution.

Anyways, we have omitted the skewness and outliers while developing the setup model for regression which will be explained later.

Checking the statistical data like mean, standard deviation of the entire dataset

```
In [14]: data.describe()
```

Out[14]:

	solarvoltage	solarcurrent	solarenergy	solarpower	batteryvoltage	batterycurrent
count	119296.000000	119296.000000	119296.000000	119296.000000	119296.000000	119296.000000
mean	81.551298	3.123387	0.008706	508.336857	98.691006	4.170616
std	85.631798	3.768051	0.018784	651.506325	2.883481	5.474155
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	97.020000	0.000000
50%	0.000000	1.170000	0.000000	0.000000	98.770000	0.970000
75%	182.390000	6.300000	0.020000	1130.600000	100.530000	8.840000
max	198.750000	18.350000	3.230000	2980.960000	112.070000	23.990000

Printing all the columns name.

```
In [12]: data.columns

Out[12]: Index(['location', 'date_time', 'solarvoltage', 'solarcurrent', 'solarenergy',
                'solarpower', 'batteryvoltage', 'batterycurrent', 'batterypower',
                'load_energy1', 'load_power1', 'load_current1', 'load_voltage1',
                'load_energy2', 'load_power2', 'load_current2', 'load_voltage2',
                'inverter_input_power', 'inverter_output_power',
                'inverter_input_energy', 'inverter_output_energy', 'irradiance',
                'temperature'],
               dtype='object')
```

Storing all the columns inside the cat_F variable which will be used later for the analysis.

```
In [75]: cat_f=['location', 'date_time', 'solarvoltage', 'solarcurrent', 'solarenergy',
                'solarpower', 'batteryvoltage', 'batterycurrent', 'batterypower',
                'load_energy1', 'load_power1', 'load_current1', 'load_voltage1',
                'load_energy2', 'load_power2', 'load_current2', 'load_voltage2',
                'inverter_input_power', 'inverter_output_power',
                'inverter_input_energy', 'inverter_output_energy', 'irradiance',
                'temperature']
```

As the energy efficiency depends upon the temperature model and we are going to establish the regression relationship between temperature and other parameters**. Let's train the data and before that split in the test and train data set.**

```
In [15]: data['temperature'].describe()

Out[15]: count    119120.000000
         mean         38.506775
         std           0.436293
         min          36.540000
         25%          38.440000
         50%          38.440000
         75%          38.440000
         max          45.540000
         Name: temperature, dtype: float64
```

```
In [18]: traindata = data[0:100000]
         testdata = data[100000:]
         print('Data for Modeling: ' + str(traindata.shape))
         print('Unseen Test Data For Predictions: ' + str(testdata.shape))

         Data for Modeling: (100000, 23)
         Unseen Test Data For Predictions: (19296, 23)
```

```
In [19]: target = 'temperature'
         data = traindata
```

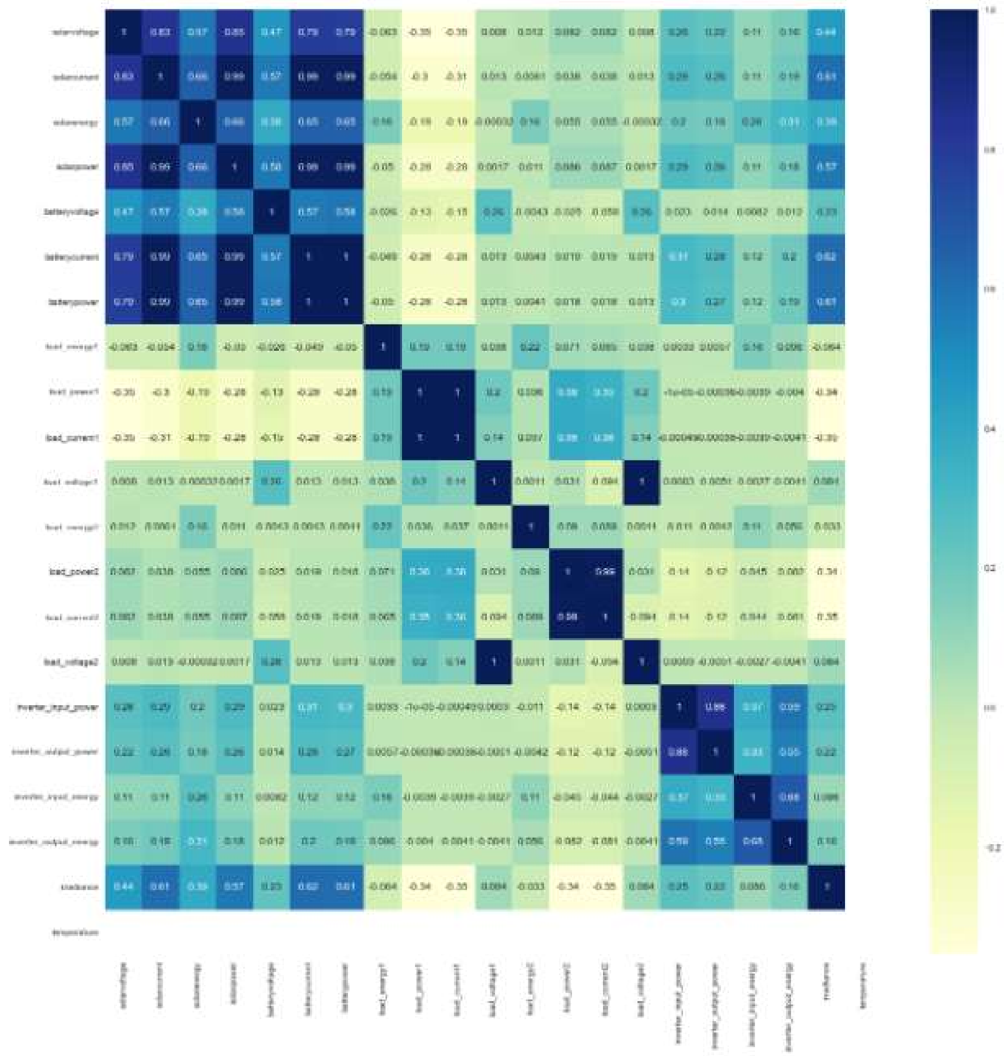
5) Checking the correlation of the temperature on the other parameters. The parameters which have negative correlation in heatmap will be ignore in the regression model development.

In [27]:
```
target = 'temperature'
data = traindata
```

In [25]:
```
corr_matrix=data.corr()
corr_matrix["temperature"].sort_values(ascending=False)
```

Out[25]:
```
irradiance               1.000000
batterycurrent           0.616224
batterypower             0.613680
solarcurrent             0.608385
solarpower               0.573467
solarvoltage             0.444946
solarenergy              0.385098
inverter_input_power     0.253651
batteryvoltage           0.234874
inverter_output_power    0.218651
inverter_output_energy   0.155758
inverter_input_energy    0.086225
load_voltage2            0.084396
load_voltage1            0.084396
load_energy2            -0.032779
load_energy1            -0.064380
load_power1             -0.342297
load_power2             -0.344840
load_current1           -0.350428
load_current2           -0.350453
temperature                   NaN
Name: irradiance, dtype: float64
```

In [30]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(20,20))         # Sample figsize in inches
dataplot = sns.heatmap(data.corr(), cmap="YlGnBu", annot=True)
```

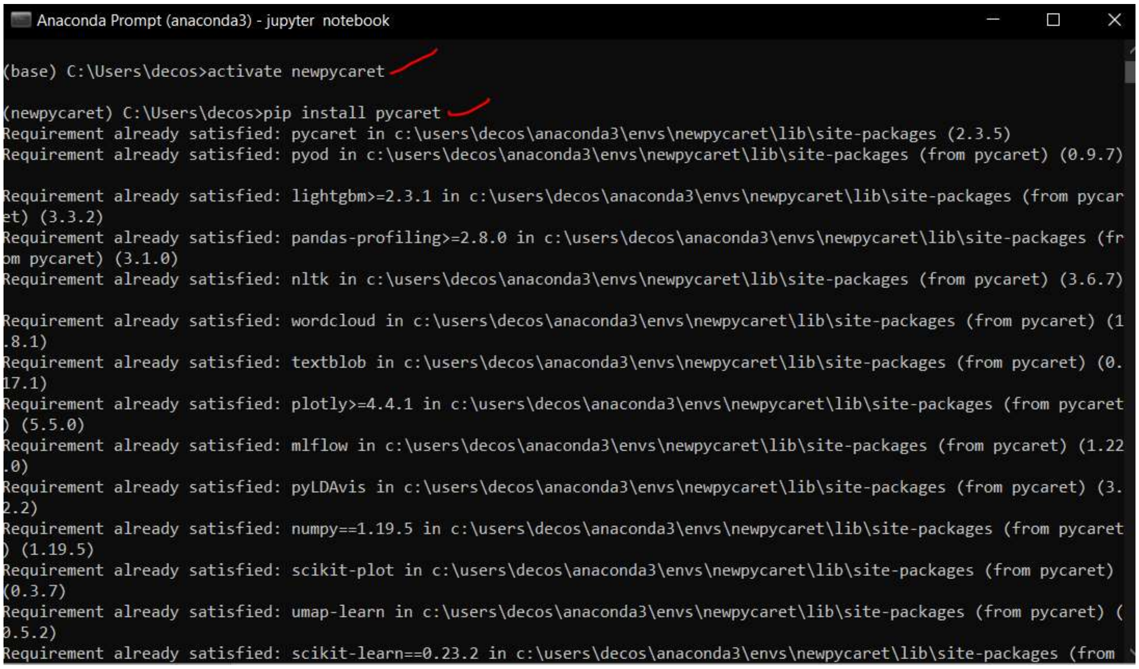


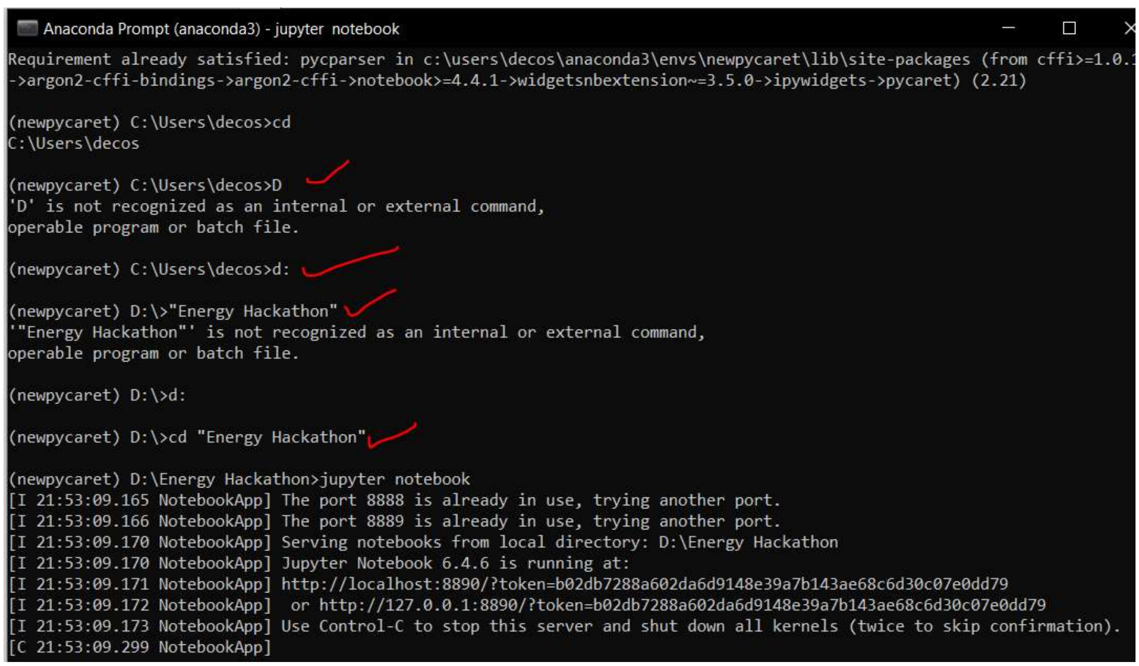
PREDICTIVE ANALYSIS ON SOLAR DATASET

ENERGY EFFICIENCY PREDICTION BY AUTOML USING PYCART WITH A REGRESSION

(Sub problem statement 6,8 covered)

6) **Setting up the environment:** Setting up an environment in Caret to run regression models hassle-free. Installing the picrate library in anaconda prompt in D drive folder and then Jupiter notebook is launched from there.





7) Setup of the Regression model

```
In [84]: from pycaret.regression import*
reg = setup(data=data,target='irradiance',session_id=123,train_size=0.80,
          ignore_features=['load_energy2','load_energy1','load_power1','load_power2','load_current1','load_current2','temperatur
          normalize=True,
          transform_target = True,remove_outliers=True,
          remove_multicollinearity= True,
          combine_rare_levels=True,
           high_cardinality_features=['location'],
           log_experiment=True,experiment_name='life'
          )
```

	Description	Value
0	session_id	123
1	Target	irradiance
2	Original Data	(100000, 23)
3	Missing Values	False
4	Numeric Features	13
5	Categorical Features	1
6	Ordinal Features	False
7	High Cardinality Features	True

We have removed the **outliers and skewness** here. The columns which have less correlation are also ignored.

```
In [31]: exp_clf = setup(data=data, target = target, feature_selection=True, session_id=100)
```

	Description	Value
0	session_id	100
1	Target	irradiance
2	Target Type	Multiclass
3	Label Encoded	None
4	Original Data	(100000, 23)
5	Missing Values	False
6	Numeric Features	20
7	Categorical Features	1
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(69999, 51)
12	Transformed Test Set	(30001, 51)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	08da
22	Imputation Type	simple
23	Iterative Imputation Iteration	None
24	Numeric Imputer	mean
25	Iterative Imputation Numeric Model	None
26	Categorical Imputer	constant
27	Iterative Imputation Categorical Model	None
28	Unknown Categoricals Handling	least_frequent
29	Normalize	False
30	Normalize Method	None
31	Transformation	False
32	Transformation Method	None
33	PCA	False
34	PCA Method	None
35	PCA Components	None
36	Ignore Low Variance	False
37	Combine Rare Levels	False
38	Rare Level Threshold	None
39	Numeric Binning	False
40	Remove Outliers	False
41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None

41	Outliers Threshold	None
42	Remove Multicollinearity	False
43	Multicollinearity Threshold	None
44	Remove Perfect Collinearity	True
45	Clustering	False
46	Clustering Iteration	None
47	Polynomial Features	False
48	Polynomial Degree	None
49	Trignometry Features	False
50	Polynomial Threshold	None
51	Group Features	False
52	Feature Selection	True
53	Feature Selection Method	classic
54	Features Selection Threshold	0.800000
55	Feature Interaction	False
56	Feature Ratio	False
57	Interaction Threshold	None
58	Fix Imbalance	False
59	Fix Imbalance Method	SMOTE

8) Comparing all models and picking up the best one which has maximum **R2 score.**

Checking all the machine learning models available for the regression.

```
In [33]: models()
```

Out[33]:

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model._logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors._classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree._classes.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model._stochastic_gradient.SGDC...	True
rbfsvm	SVM - Radial Kernel	sklearn.svm._classes.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process._gpc.GaussianProcessC...	False
mlp	MLP Classifier	sklearn.neural_network._multilayer_perceptron....	False
ridge	Ridge Classifier	sklearn.linear_model._ridge.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble._forest.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QuadraticDiscrim...	True
ada	Ada Boost Classifier	sklearn.ensemble._weight_boosting.AdaBoostClas...	True
gbc	Gradient Boosting Classifier	sklearn.ensemble._gb.GradientBoostingClassifier	True
lda	Linear Discriminant Analysis	sklearn.discriminant_analysis.LinearDiscrimina...	True
et	Extra Trees Classifier	sklearn.ensemble._forest.ExtraTreesClassifier	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMClassifier	True
dummy	Dummy Classifier	sklearn.dummy.DummyClassifier	True

Comparing the models and selecting the best

```
In [30]: best=compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
et	Extra Trees Regressor	26.6548	2948.6166	53.9859	0.9866	1.3599	0.2936	33.1970
rf	Random Forest Regressor	28.2483	3931.7792	62.4063	0.9822	1.3607	0.2958	36.3570
lightgbm	Light Gradient Boosting Machine	32.8953	4688.6018	68.2727	0.9788	1.3793	0.3401	0.3600
dt	Decision Tree Regressor	36.9204	8092.8985	89.5469	0.9633	1.5765	0.4221	1.9350
gbr	Gradient Boosting Regressor	57.5751	16019.1460	126.4891	0.9274	1.5019	0.6441	8.2970
ada	AdaBoost Regressor	104.8898	41529.3795	203.7687	0.8115	1.8563	1.0281	5.4410
knn	K Neighbors Regressor	85.3995	44372.0735	210.5934	0.7988	1.4614	0.6989	11.6910
br	Bayesian Ridge	184.7286	84977.6860	291.4726	0.6147	2.7075	3.0617	0.8220
ridge	Ridge Regression	184.4190	85277.7863	291.9856	0.6133	2.6991	3.0434	0.3160
lr	Linear Regression	183.9752	85278.4413	291.9864	0.6133	2.7023	3.0314	10.9560
lasso	Lasso Regression	186.4486	87178.6743	295.2201	0.6047	2.7020	3.0721	3.4580
en	Elastic Net	189.9523	94361.9050	307.1307	0.5722	2.6763	2.9125	3.9800
omp	Orthogonal Matching Pursuit	194.5018	95592.2663	309.1336	0.5666	2.6968	3.0077	0.2030
huber	Huber Regressor	137.3688	138578.6230	372.1007	0.3722	1.6677	0.5965	8.4960
llar	Lasso Least Angle Regression	331.4704	200936.0550	448.1752	0.0892	3.4293	5.8360	0.2530
dummy	Dummy Regressor	360.2472	220623.9754	469.6346	-0.0002	3.5015	6.4006	0.0420
par	Passive Aggressive Regressor	563.8945	786019.2617	813.4953	-2.6170	2.8621	5.3574	1.1960
lar	Least Angle Regression	49272.7756	41822460423.8974	74520.0509	-185411.0018	5.5062	1015.5241	0.1940

We select par model for its significant RMSE value.

9) Create model:

We selected par model of regression

In [31]:
```
par = create_model('par')
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	514.6130	524456.3100	724.1936	-1.3248	2.7444	4.4240
1	345.0681	206719.1784	454.6638	0.0680	3.0776	4.6494
2	250.3099	130066.2587	360.6470	0.3956	2.9542	4.5294
3	350.8714	230263.9748	479.8583	0.0341	2.7159	4.2097
4	686.5742	1115570.8366	1056.2059	-3.9879	2.8738	5.1397
5	830.0463	1482158.2902	1217.4392	-5.8601	2.8747	6.6104
6	843.0010	1522174.7916	1233.7645	-6.0245	2.8514	6.7425
7	792.0349	1271424.8527	1127.5748	-5.0559	2.9084	6.8975
8	786.9194	1243917.0838	1115.3103	-4.8212	2.8985	6.8820
9	239.5067	133441.0406	365.2958	0.4062	2.7222	3.4896
Mean	563.8945	786019.2617	813.4953	-2.6170	2.8621	5.3574
SD	237.6617	561197.6599	352.4835	2.6256	0.1066	1.2287

10) Tune Model: Automatically tuning the hyperparameters of a regression model.

```
In [32]: tuned_par = tune_model(par)
```

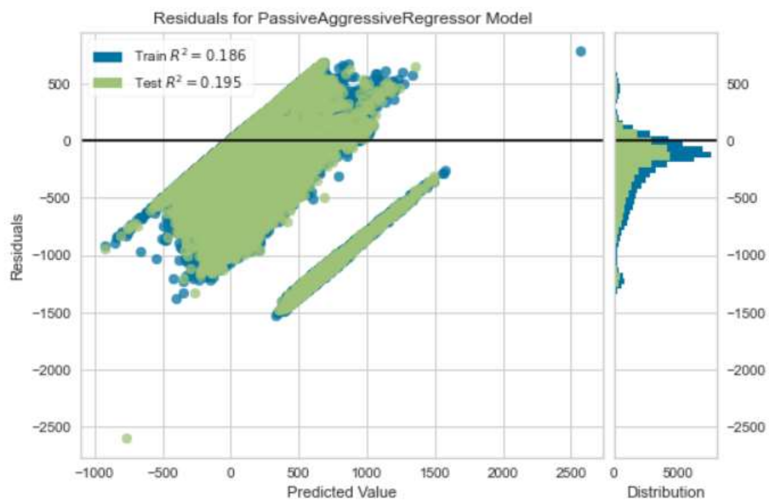
	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	304.6009	187464.1668	432.9713	0.1690	2.9479	3.5383
1	290.5549	170707.5036	413.1676	0.2303	2.9012	3.5134
2	296.2495	176744.3004	420.4097	0.1787	2.9169	3.5773
3	300.3711	184247.1895	429.2402	0.2271	2.9190	3.4642
4	298.7297	178657.7549	422.6793	0.2012	2.9274	3.5977
5	288.1370	168807.8217	410.8623	0.2187	2.9586	3.4228
6	294.6856	173826.1599	416.9246	0.1978	2.9264	3.6928
7	290.5770	168183.1439	410.1014	0.1989	2.9270	3.6297
8	300.8704	178597.1336	422.6075	0.1642	2.9526	3.5828
9	606.9557	451449.2030	671.8997	-1.0088	3.9986	13.4043
Mean	327.1732	203868.4377	445.0864	0.0777	3.0376	4.5423
SD	93.3931	82743.6632	75.9379	0.3628	0.3208	2.9549

11) Plot Model & Results : plotting the performance of various models.

```
In [33]: print(tuned_par)

PassiveAggressiveRegressor(C=0.911, average=False, early_stopping=False,
                           epsilon=0.3, fit_intercept=False,
                           loss='squared_epsilon_insensitive', max_iter=1000,
                           n_iter_no_change=5, random_state=100, shuffle=False,
                           tol=0.001, validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

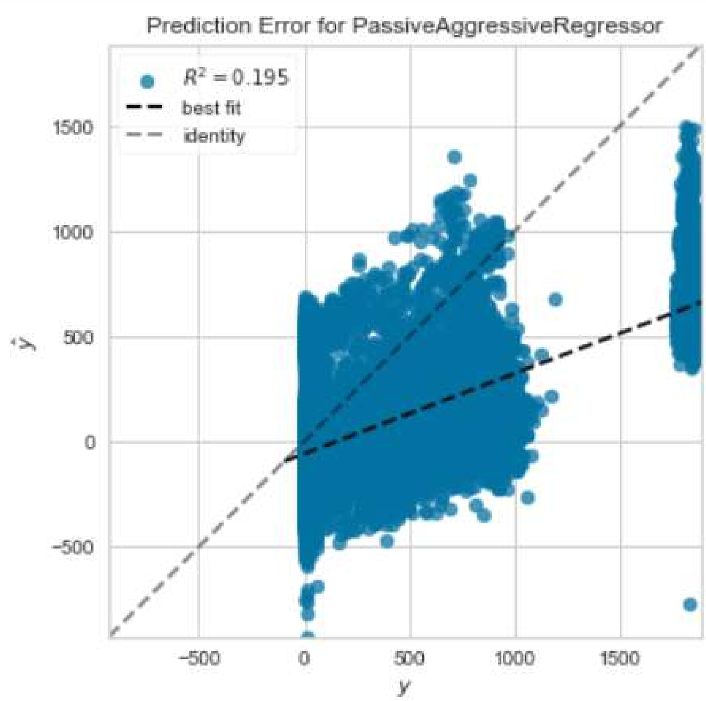
```
warm_start=False)
```

```
In [34]: plot_model(tuned_par)
```



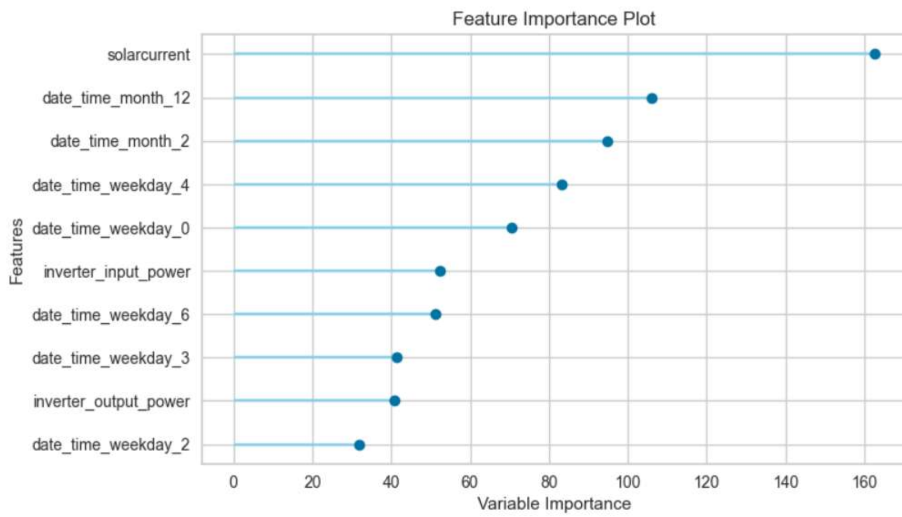
Prediction error plot:

```
In [35]: plot_model(tuned_par, plot = 'error')
```

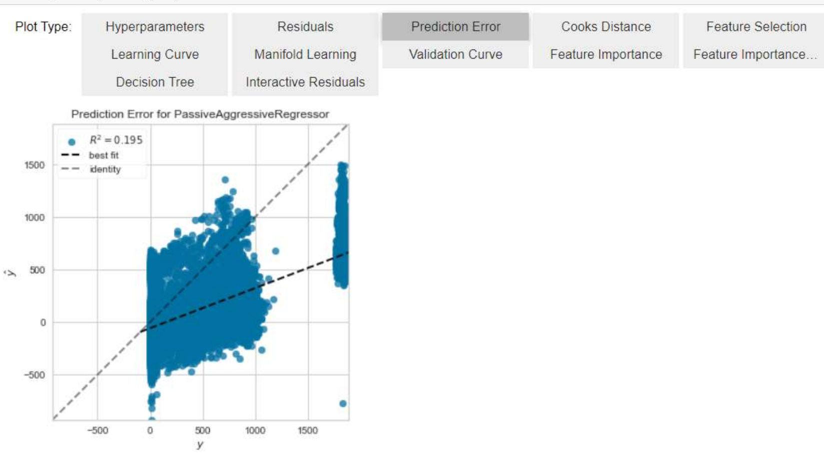


Feature importance plot:

```
In [36]: plot_model(tuned_par, plot = 'feature')
```




```
In [37]: evaluate_model(tuned_par)
```



Hyperparameters:

```
In [37]: evaluate_model(tuned_par)
```

Plot Type: Hyperparameters | Residuals | Prediction Error | Cooks Distance | Feature Selection | Learning Curve | Manifold Learning | Validation Curve | Feature Importance | Feature Importance... | Decision Tree | Interactive Residuals

	Parameters
C	0.911
average	False
early_stopping	False
epsilon	0.3
fit_intercept	False
loss	squared_epsilon_insensitive
max_iter	1000
n_iter_no_change	5
random_state	100
shuffle	False
tol	0.001
validation_fraction	0.1
verbose	0
warm_start	False

Residuals:

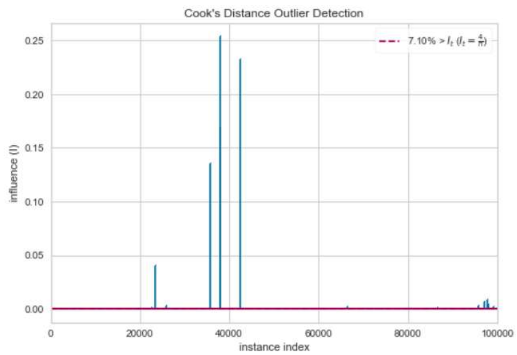
```
In [37]: evaluate_model(tuned_par)
```

Plot Type: Hyperparameters | Residuals | Prediction Error | Cooks Distance | Feature Selection | Learning Curve | Manifold Learning | Validation Curve | Feature Importance | Feature Importance... | Decision Tree | Interactive Residuals

Cooks distance:

```
In [37]: evaluate_model(tuned_par)
```

Plot Type: Hyperparameters | Residuals | Prediction Error | Cooks Distance | Feature Selection | Learning Curve | Manifold Learning | Validation Curve | Feature Importance | Feature Importance... | Decision Tree | Interactive Residuals



11) Finalize model: How to select and finalize the best model at the end of the experiment.

```
In [38]: predict_model(tuned_par)
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Passive Aggressive Regressor	296.5226	176686.1359	420.3405	0.1954	2.9547	3.5948

Out[38]:

date_time_hour_14	date_time_hour_15	batteryvoltage	inverter_input_power	date_time_weekday_6	date_time_hour_0	date_time_hour_18	irradiance	Label
0.0	0.0	105.849998	0.20	0.0	0.0	0.0	930	-57.849471
0.0	0.0	100.070000	0.29	0.0	0.0	1.0	45	-161.798587
0.0	0.0	99.379997	0.19	1.0	0.0	0.0	15	-232.231633
0.0	0.0	93.419998	0.18	1.0	0.0	0.0	0	-144.850875
0.0	0.0	100.059998	0.20	0.0	0.0	0.0	815	247.637599
...	...	...	...	...	...	...	...	...
0.0	0.0	99.349998	0.19	0.0	0.0	0.0	125	-77.081246
0.0	0.0	96.339996	0.19	0.0	0.0	0.0	15	-89.908268
0.0	0.0	92.720001	0.41	0.0	0.0	0.0	15	-62.723250
0.0	0.0	94.169998	0.18	0.0	1.0	0.0	1832	587.353067
0.0	0.0	99.300003	0.44	0.0	0.0	0.0	15	-131.383088

```
In [39]: final_par = finalize_model(tuned_par)
         print(final_par)

PassiveAggressiveRegressor(C=0.911, average=False, early_stopping=False,
                           epsilon=0.3, fit_intercept=False,
                           loss='squared_epsilon_insensitive', max_iter=1000,
                           n_iter_no_change=5, random_state=100, shuffle=False,
                           tol=0.001, validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

12) **Predict Model**: making predictions on new data.

```
In [40]: unseen_predictions = predict_model(final_par, data=testdata)
         unseen_predictions.head()
```

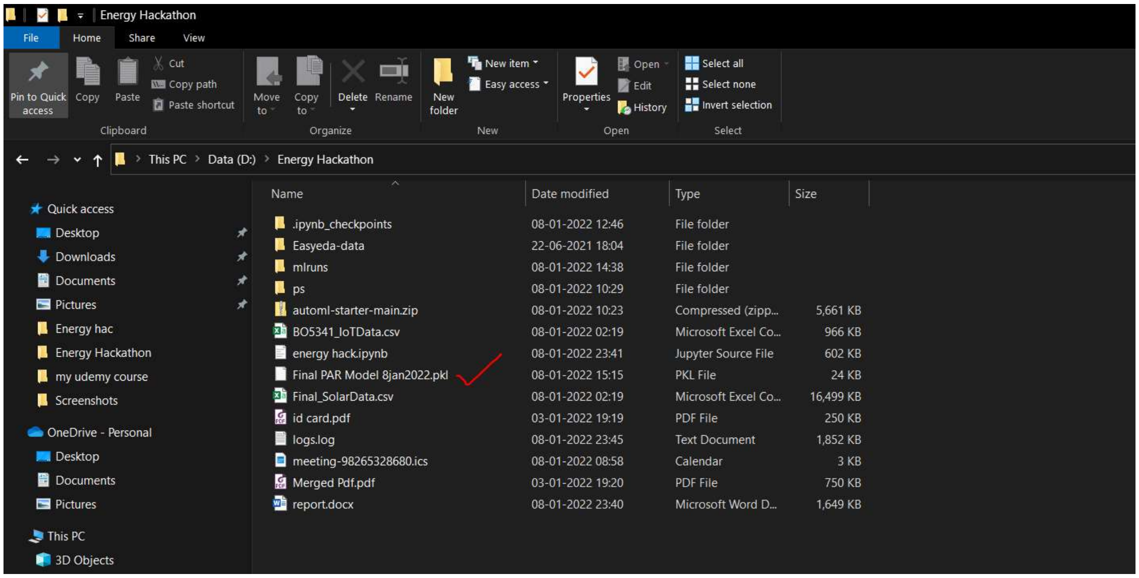
Out[40]:

	location	date_time	solarvoltage	solarcurrent	solarenergy	solarpower	batteryvoltage	batterycurrent	batterypower	load_energy1	...	load_power2	lo:
100000	Peru	2016-02-27 06:50:35	0.0	1.42	0.0	0.0	93.42	1.06	99.63	0.0	...	48.65	
100001	Peru	2016-02-27 06:51:48	0.0	1.35	0.0	0.0	93.57	0.56	52.83	0.0	...	64.76	
100002	Peru	2016-02-27 06:53:01	0.0	1.48	0.0	0.0	93.48	1.63	152.49	0.0	...	64.80	
100003	Peru	2016-02-27 06:54:14	0.0	1.35	0.0	0.0	93.51	1.69	158.41	0.0	...	58.40	
100004	Peru	2016-02-27 06:55:26	0.0	1.42	0.0	0.0	93.45	1.69	158.30	0.0	...	61.55	

5 rows × 24 columns

13) **Save Model**: Saving the model for future use.

```
In [122]: save_model(final_par,'Final PAR Model 8jan2022')

          Transformation Pipeline and Model Successfully Saved

Out[122]: (Pipeline(memory=None,
                    steps=[('dtypes',
                            DataTypes_Auto_infer(categorical_features=[],
                                                 display_types=True, features_todrop=[],
                                                 id_columns=[], ml_usecase='regression',
                                                 numerical_features=[],
                                                 target='temperature', time_features=[])),
                           ('imputer',
                            Simple_Imputer(categorical_strategy='not_available',
                                           fill_value_categorical=None,
                                           fill_value_numerical=None,
                                           numeric_stra...
                           ('fix_multi', 'passthrough'), ('dfs', 'passthrough'),
                           ('pca', 'passthrough'),
                           ['trained_model',
                            PassiveAggressiveRegressor(C=5.654, average=False,
                                                       early_stopping=False, epsilon=0.3,
                                                       fit_intercept=True,
                                                       loss='squared_epsilon_insensitive',
                                                       max_iter=1000, n_iter_no_change=5,
                                                       random_state=100, shuffle=True,
                                                       tol=0.001, validation_fraction=0.1,
                                                       verbose=0, warm_start=False)]],
                    verbose=False),
           'Final PAR Model 8jan2022.pkl')
```



TIME SERIES DATA VISUALIZATION AND DATE ANNOTATIONS

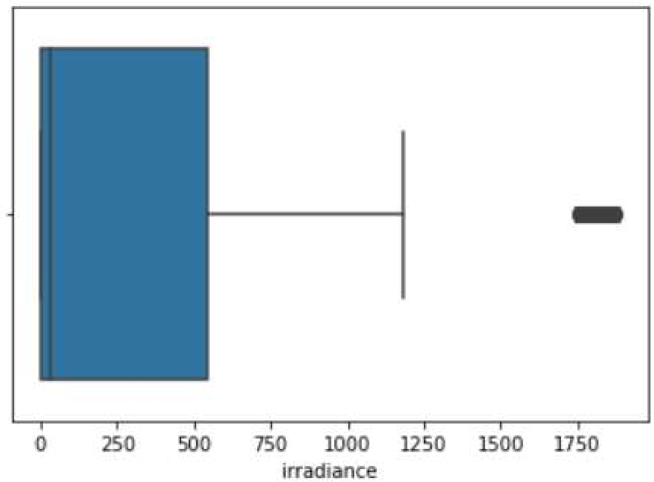
(sub problem statement 2,3 covered)

Outlier in the irradiance and clearing it.

In [16]:
```
sns.boxplot(data["irradiance"])
```

```
C:\Users\decos\anaconda3\lib\site-packages\seaborn\_decorator
rg: x. From version 0.12, the only valid positional argument
yword will result in an error or misinterpretation.
  warnings.warn(
```

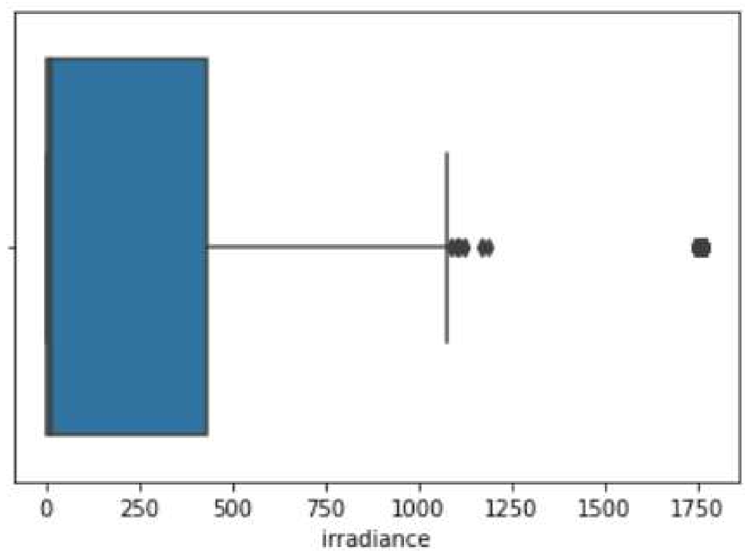
Out[16]:
```
<AxesSubplot:xlabel='irradiance'>
```



In [18]:
```
from scipy import stats
from scipy.stats import zscore

z_scores = stats.zscore(data["irradiance"])

abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3)
data = data[filtered_entries]

sns.boxplot(data["irradiance"]) #outliers removed
```

```
C:\Users\decos\anaconda3\lib\site-packages\seaborn\_dec
rg: x. From version 0.12, the only valid positional arg
yword will result in an error or misinterpretation.
  warnings.warn(
```

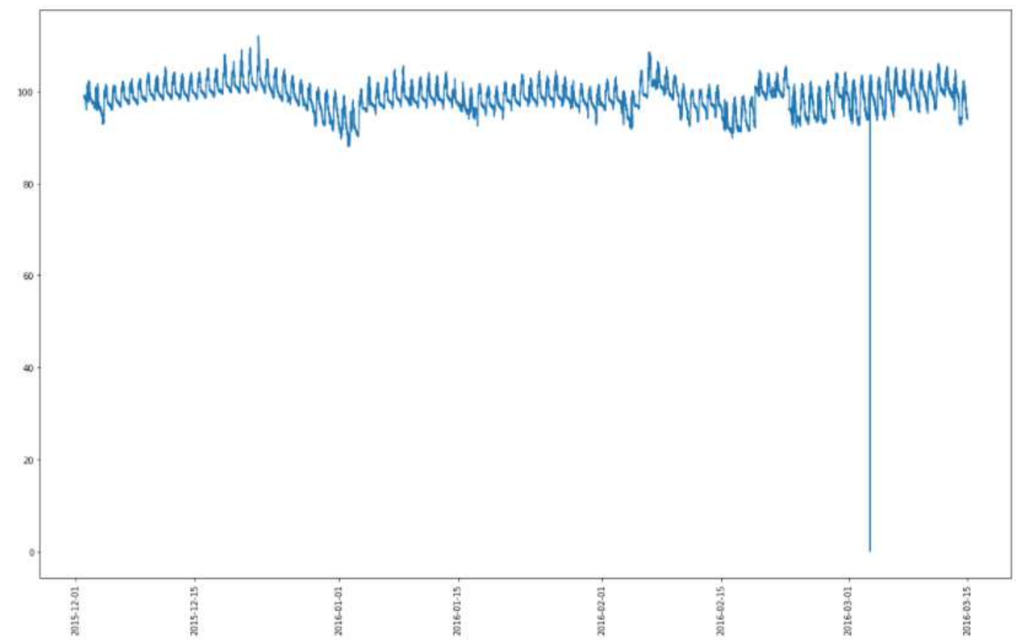
Out[18]: <AxesSubplot:xlabel='irradiance'>



Time series of batteryvoltage

In [16]:
```
plt.figure(figsize=(20,12))
plt.plot(data['date_time'],data['batteryvoltage'])
plt.xticks(rotation='vertical')
```

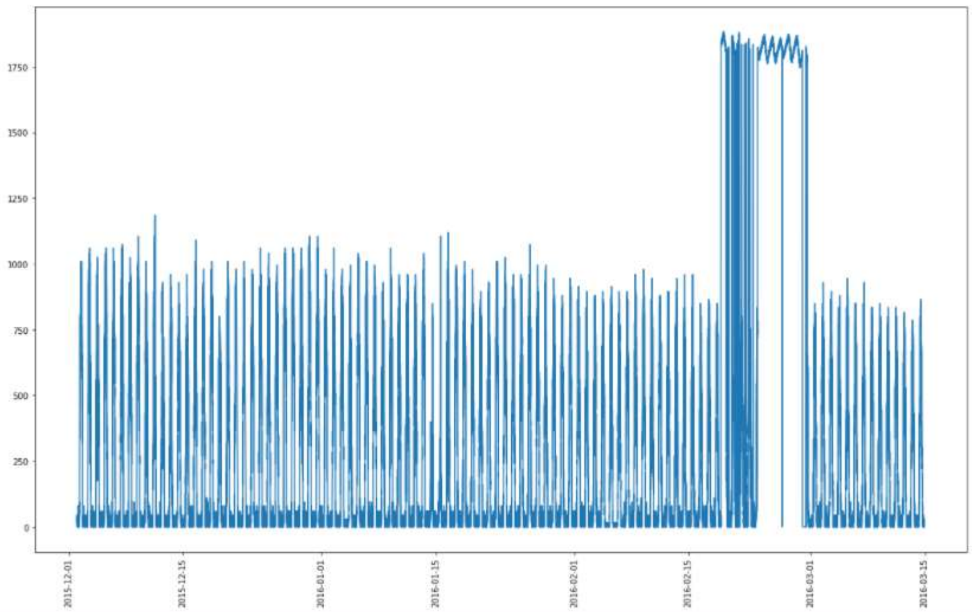
Out[16]:
```
(array([16770., 16784., 16801., 16815., 16832., 16846., 16861., 16875.]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



Time series of irradiance

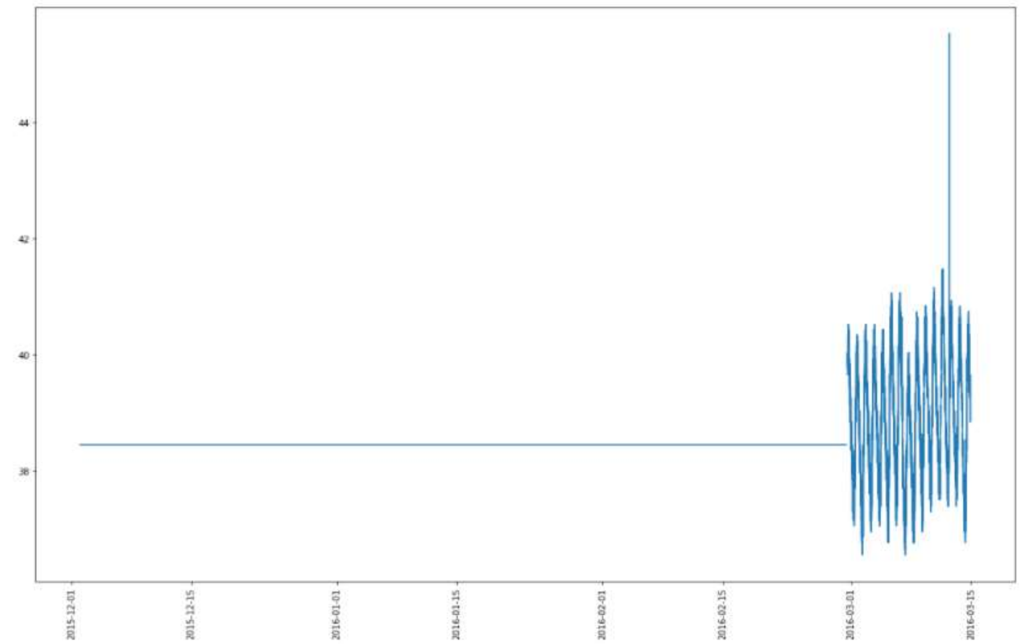
```
In [12]: plt.figure(figsize=(20,12))
         plt.plot(data['date_time'],data['irradiance'])
         plt.xticks(rotation='vertical')
```

```
Out[12]: (array([16770., 16784., 16801., 16815., 16832., 16846., 16861., 16875.]),
          [Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, '')])
```



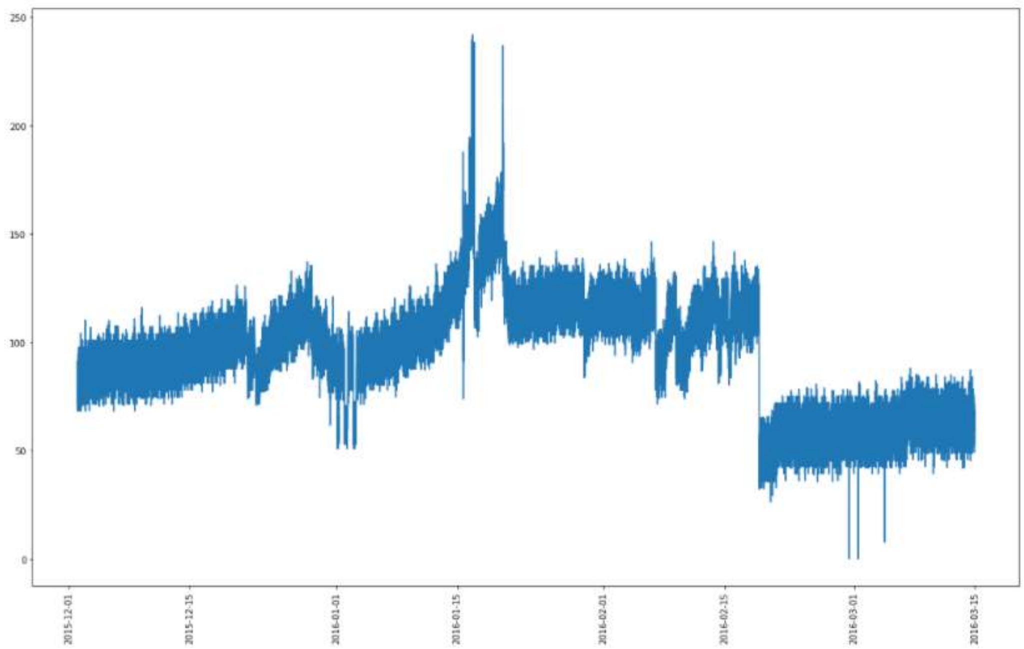
Time series of Temperature

```
In [15]: plt.figure(figsize=(20,12))
         plt.plot(data['date_time'],data['temperature'])
         plt.xticks(rotation='vertical')

Out[15]: (array([16770., 16784., 16801., 16815., 16832., 16846., 16861., 16875.]),
          [Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, '')])
```



Time series of load_power 2

```
In [17]: plt.figure(figsize=(20,12))
         plt.plot(data['date_time'],data['load_power2'])
         plt.xticks(rotation='vertical')

Out[17]: (array([16770., 16784., 16801., 16815., 16832., 16846., 16861., 16875.]),
          [Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, ''),
           Text(0, 0, '')])
```



DASHBORAD CREATION AND RESULT VISUALIZATION

(Sub problem statements 7 & 10 covered in the video submitted)

http://localhost:8892/voila/render/OneDrive/Desktop/Energy%20hac/Descriptive_Analysis.ipynb

The Dashboard is created by using **Voila library** in python which converts python jupyter notebook in to html page and then the results can be visualized on the web browser.

This thing is demonstrated in the submitted video.

DATA ANNOTATION AND DATA SAVING IN DATABSE

(Sub problem statement 4 covered)

Tortus library is used in python to make the data labelling in python.

Libraries installation and dataset importing.

```
In [1]: import pandas as pd
        from datetime import datetime
        import sysconfig
        from ipywidgets import Image, HTML, Button, IntProgress, \
            Box, HBox, VBox, GridBox, Layout, ButtonStyle, Output
        from IPython.display import display, clear_output
```

```
In [4]: data=pd.read_csv("Final_SolarData.csv")
        data.head()
```

Out[4]:

	location	date_time	solarvoltage	solarcurrent	solarenergy	solarpower	batteryvoltage	batterycurrent	batterypower	load_energy1	...	load_energy2	load_pc
0	Peru	2015-12-02 00:00:27	0.0	0.0	0.0	0.0	98.78	0.0	0.0	0.00	...	0.00	
1	Peru	2015-12-02 00:01:40	0.0	0.0	0.0	0.0	98.80	0.0	0.0	0.01	...	0.01	
2	Peru	2015-12-02 00:02:52	0.0	0.0	0.0	0.0	98.55	0.0	0.0	0.00	...	0.00	
3	Peru	2015-12-02 00:04:05	0.0	0.0	0.0	0.0	98.64	0.0	0.0	0.00	...	0.00	
4	Peru	2015-12-02 00:05:18	0.0	0.0	0.0	0.0	98.59	0.0	0.0	0.01	...	0.00	

Output:

```
In [5]: from tortus import Tortus
        tortus = Tortus(data,
            'location',
            num_records=3,
            id_column='temperature'
        )
```

```
In [6]: tortus.annotate()
```

tortus 1/3

Click on the label corresponding with the text below. Each selection requires confirmation before proceeding to the next item.

Peru

Positve | Negative | Neutral | Skip

Annotated and labelled data saved:

In [11]:
```
tortus.annotations
```
Out[11]: See Full Dataframe in Mito

	temperature	location	label	annotated_at
0	38.44	Peru	positve	2022-01-08 17:54:32
1	38.44	Peru	neutral	2022-01-08 17:54:42
2	39.65	Peru	negative	2022-01-08 17:54:44