Global Power Plant Database Project

In association with Data Trained Academy

I am going to write about a complete end-to-end project for Global Power Plant Database.
I have written down all the steps in the form of sub-topics that I will be explaining one by one. And those sub-topics are as follows:

1. Problem Definition.
2. Data Analysis.
3. EDA Concludi ng Remark.
4. Pre-Processing Pipeline.
5. Building Machine Learning Models.
6. Concluding Remarks.

Introduction:

An affordable, reliable, and environmentally sustainable power sector is central to modern society. Governments, utilities, and companies make decisions that both affect and depend on the power sector. For example, if governments apply a carbon price to electricity generation, it changes how plants run and which plants are built over time.
On the other hand, each new plant affects the electricity generation mix, the reliability of the system, and system emissions. Plants also have significant impact on climate change, through carbon dioxide (CO_2) emissions; on water stress, through water withdrawal and consumption; and on air quality, through sulfur oxides (SO_x), nitrogen oxides (NO_x), and particulate matter (PM) emissions. 2 | Despite the importance of the power sector, there is no global, open-access database of power plants. Existing databases fail to be either truly comprehensive or fully open. Many countries do not report their power sector data at the plant level, and those that do vary wildly in what they report, how they report it, and how frequently they report. The lack of reporting standards makes data gathering time intensive, as the data are in different formats and must be harmonized. This creates a barrier for conducting global and national research and analysis of the power sector.
Thanks to Data Science and Machine Learning, which has been very useful in many industries that have managed to bring accuracy or detect negative incidents. Here in this blog, I have created a Machine Learning model to predict the power plant data on the label of primary fue; and capacity in MW

Detect if the claim is fraudulent or not. Here various features have been used like insured information, insured persons, personal details and the incident information. In total the dataset has 40 features and 1000 entries rows of data. Using all these previously acquired information and analysis done with the data I have achieved a good model that has 95% accuracy. Let's see what are the steps involved to attain this accuracy.

Hardware & Software Requirements & Tools Used:

Hardware used:

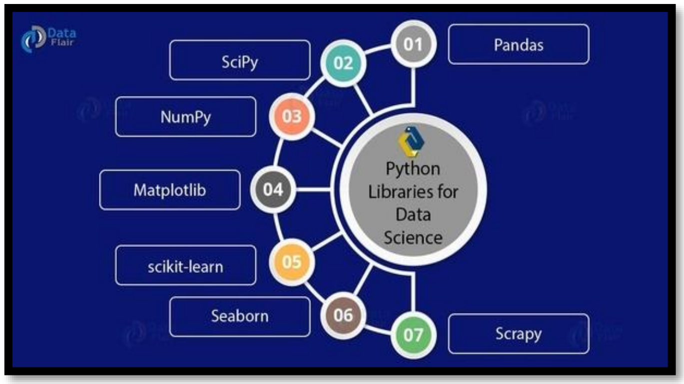
Processor: Core i5 -10300H CPU @ 2.50GHz HP Pavilion Laptop 15-cc1xx
RAM: 8 GB
Operating System: 64-bit
ROM/SSD: 256GB SSD
Graphics: NVIDIA GeForce 940MX
Intel(R) UHD Graphics 620

Software requirement:

Anaconda Navigator - Jupyter Notebook

Libraries Used:

Numpy
Pandas
Matplotlib
Seaborn
Scipy
Date Time
Scikit Learn



1.Problem Definition.

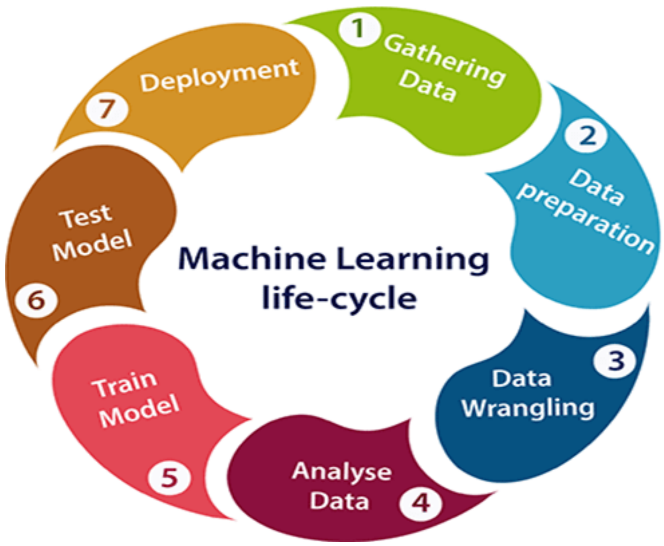
The Global Power Plant Database is a comprehensive, open source database of power plants around the world. It centralizes power plant data to make it easier to navigate, compare and draw insights for one's own analysis. The database covers approximately 35,000 power plants from 167 countries and includes thermal plants (e.g. coal, gas, oil, nuclear, biomass, waste, geothermal) and renewables (e.g. hydro, wind, solar). Each power plant is geolocated and entries contain information on plant capacity, generation, ownership, and fuel type
Make two prediction for labels

1.primary_fuel

2.capacity_mw

2.Data Analysis.

In order to build a Machine Learning Model, we have a Machine Learning Life Cycle that every Machine Learning Project has to touch upon in the life of the model. Let's take a look at the model life cycle and then we will look into the actual machine learning model and understand it better along with the lifecycle as we move forward.



Now that we understand the lifecycle of a Machine Learning Model, let's import the necessary libraries and proceed further.

Importing the necessary Libraries:

To analyze the dataset or even to import the dataset, we have imported all the necessary libraries as shows below.
Pandas has been used to import the dataset and also in creating data frames.
Numpy has been used for numerical tasks.
Seaborn and Matplotlib have been used for Data Visualization.
Date Time has been used to extract day/month/date separately.
Scipy has been used in the Zscore method for removing outliers.
Sklearn has been used in the model building.

```python
# To Read and Process Data
import pandas as pd
import numpy as np


# For data Visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Getting over warning messages
import warnings
warnings.filterwarnings('ignore')

# For Encoding Categorical Data
from sklearn.preprocessing import LabelEncoder

# for scaling
from sklearn.preprocessing import StandardScaler

pd.pandas.set_option('display.max_columns',None)  # To display, all columns
pd.pandas.set_option('display.max_rows',None)  # To display, all columns
# For handling outliers
# importing required libraries
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import zscore
# For machine learning and finding
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

Importing the Dataset

Let's import the dataset first.

```python
df = pd.read_csv('database_IND.csv')
```

Copied the raw data and saved it as a csv file on my local computer after which I imported the entire dataset on this Jupyter Notebook with the help of pandas.
I have imported the dataset which was in "csv" format as "df". Below is how the dataset looks.

	country	country_long	name	gppd_idnr	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	other_fuel2	...	year_of_capacity_data	generation_gwh_2013
0	IND	India	ACME Solar Tower	WRI1020239	2.5	28.1839	73.2407	Solar	NaN	NaN	...	NaN	NaN
1	IND	India	ADITYA CEMENT WORKS	WRI1019881	98.0	24.7663	74.6090	Coal	NaN	NaN	...	NaN	NaN
2	IND	India	AES Saurashtra Windfarms	WRI1026669	39.2	21.9038	69.3732	Wind	NaN	NaN	...	NaN	NaN
3	IND	India	AGARTALA GT	IND0000001	135.0	23.8712	91.3602	Gas	NaN	NaN	...	2019.0	NaN
4	IND	India	AKALTARA TPP	IND0000002	1800.0	21.9603	82.4091	Coal	Oil	NaN	...	2019.0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
902	IND	India	YERMARUS TPP	IND0000513	1600.0	16.2949	77.3568	Coal	Oil	NaN	...	2019.0	NaN
903	IND	India	Yelesandra Solar Power Plant	WRI1026222	3.0	12.8932	78.1654	Solar	NaN	NaN	...	NaN	NaN
904	IND	India	Yelisirur wind power project	WRI1026776	25.5	15.2758	75.5811	Wind	NaN	NaN	...	NaN	NaN
905	IND	India	ZAWAR MINES	WRI1019901	80.0	24.3500	73.7477	Coal	NaN	NaN	...	NaN	NaN
906	IND	India	iEnergy Theni Wind Farm	WRI1026761	16.5	9.9344	77.4768	Wind	NaN	NaN	...	NaN	NaN

907 rows × 27 columns

Checking the shape of the data

```
df.shape
```

```
(907, 13)
```

There are 907 rows and 13 columns in our dataframe

Getting to Overview of Data Types Data

'5]:

```
numerical_data = [feature for feature in df.columns if df[feature].dtype != 'O']
print('Total Numerical Features are = ',len(numerical_data))
```

```
Total Numerical Features are =  10
```

'6]:

```
categorical_data = [feature for feature in df.columns if df[feature].dtype == 'O']
print('Total Categorical Features are = ',len(categorical_data))
```

```
Total Categorical Features are =  3
```

Handling Duplicate Values

```
df.shape
```

```
(907, 13)
```

```
df.drop_duplicates(inplace=True)
```

```
df.shape
```

```
(906, 13)
```

Conclusion

- There are no duplicates in our data set, as there are 891 unique names

Statistical Summary of Dataset

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
capacity_mw	906.0	240.223955	326.730310	0.0000	16.612500	59.600000	386.625000	938.037500
latitude	861.0	21.197918	6.239612	8.1689	16.773900	21.780000	25.512400	34.649000
longitude	861.0	77.464907	4.939316	68.6447	74.256200	76.719500	79.440800	95.408000
commissioning_year	527.0	1997.091082	17.082868	1927.0000	1988.000000	2001.000000	2012.000000	2018.000000
year_of_capacity_data	519.0	2019.000000	0.000000	2019.0000	2019.000000	2019.000000	2019.000000	2019.000000
generation_gwh_2014	398.0	1954.050199	2335.170089	0.0000	223.557672	801.123775	3035.306250	7252.929118
generation_gwh_2015	422.0	1946.278464	2363.744865	0.0000	176.381063	711.181225	3084.121250	7445.731531
generation_gwh_2016	434.0	2018.329679	2461.184325	0.0000	188.285252	737.205450	3282.861313	7924.725403
generation_gwh_2017	440.0	2106.271610	2531.772645	0.0000	177.874930	817.977250	3275.690475	7922.413793
generation_gwh_2018	448.0	2101.351259	2522.735602	0.0000	193.378250	751.644375	3143.535900	7568.772375

Here we can see the statistical analysis of the dataset (numerical only)

We can observe that the count of the columns are same, which means the dataset is balanced. The minimum capacity of the power plant is zero and maximum in 4760 and there is huge difference in mean and standard deviation.From the difference between maximum and 75% percentile we can infer that there are huge outliers present in most of the columns, will remove them using appropriate methods before building our model.

Unique Value of Dataset

```
# Checking number of unique values in each column
df.nunique()
```

```
country                    1
country_long               1
name                     907
gppd_idnr                907
capacity_mw              361
latitude                 836
longitude                827
Fuel_Type                  8
other_fuel1                3
other_fuel2                1
other_fuel3                0
commissioning_year        73
owner                    280
source                   191
url                      304
geolocation_source         3
wepp_id                    0
year_of_capacity_data      1
generation_gwh_2013        0
generation_gwh_2014      371
generation_gwh_2015      396
generation_gwh_2016      403
generation_gwh_2017      408
generation_gwh_2018      410
generation_gwh_2019        0
generation_data_source     1
estimated_generation_gwh   0
dtype: int64
```

the column with one unique value are country, country_long, other_fuel2, year_of_capacity_data and generation_data_source
other_fuel3, wepp_id,generation_gwh_2013, generation_gwh_2019, estimated_generation_gwh have no unique values which means they are filled with only NAN values
Here the columns have only one unique value.. Since these columns have same entries thoughout the dataset so we can drop these columns.

Null Value in Dataset

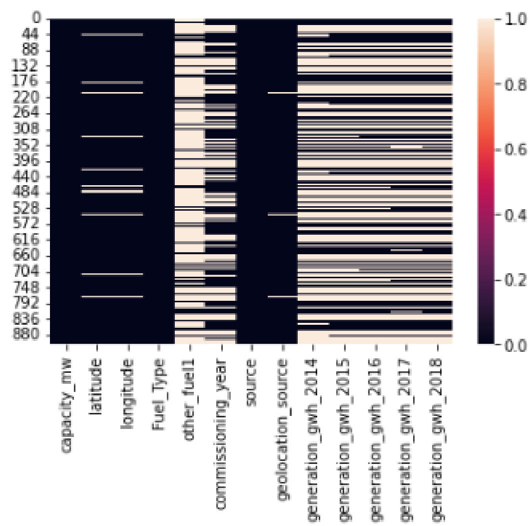
```
# Checking null values again after feature selection
df.isnull().sum()
```

```
capacity_mw              0
latitude                46
longitude               46
Fuel_Type                0
other_fuel1            709
commissioning_year     380
source                   0
geolocation_source      19
generation_gwh_2014    509
generation_gwh_2015    485
generation_gwh_2016    473
generation_gwh_2017    467
generation_gwh_2018    459
dtype: int64
```

Observations:
We can see that there are null values in the dataset.
The dataset contains 3 different types of data namely integer data type, float data type and object data type.

Plotting the graph of null values

```
sns.heatmap(df.isnull())
```



We can clearly observe the white lines in the heat map which indicates the missing values in the dataset.

Imputation:

Dealing with missing value by using imputation technique
For missing value can be filled by using median in numerical variable and
For categorical variable we use mode

In [37]:
```
#checking the mean of latitude
df['latitude'].mean()
```
Out[37]:
```
21.19791811846691
```
In [38]:
```
#checking the mode of other_fuel1
df["other_fuel1"].mode()
```
Out[38]:
```
0    Oil
Name: other_fuel1, dtype: object
```
In [39]:
```
#checking the mode of geolocation_source columns
df["geolocation_source"].mode()
```
Out[39]:
```
0    WRI
Name: geolocation_source, dtype: object
```
Filling the null values

In [40]:
```
df["latitude"] = df["latitude"].fillna(df["latitude"].mean())
df["other_fuel1"] = df["other_fuel1"].fillna(df["other_fuel1"].mode()[0])
df["geolocation_source"] = df["geolocation_source"].fillna(df["geolocation_source"].mode()[0])
df["longitude"] = df["longitude"].fillna(df["longitude"].median())
df["commissioning_year"] = df["commissioning_year"].fillna(df["commissioning_year"].median())
df["generation_gwh_2014"] = df["generation_gwh_2014"].fillna(df["generation_gwh_2014"].median())
df["generation_gwh_2015"] = df["generation_gwh_2015"].fillna(df["generation_gwh_2015"].median())
df["generation_gwh_2016"] = df["generation_gwh_2016"].fillna(df["generation_gwh_2016"].median())
df["generation_gwh_2017"] = df["generation_gwh_2017"].fillna(df["generation_gwh_2017"].median())
df["generation_gwh_2018"] = df["generation_gwh_2018"].fillna(df["generation_gwh_2018"].median())
```

Again checking for missing value

```
# checking for missing values after imputation.
df.isnull().sum()
```
```
capacity_mw            0
latitude               0
longitude              0
Fuel_Type              0
other_fuel1            0
commissioning_year     0
source                 0
geolocation_source     0
generation_gwh_2014    0
generation_gwh_2015    0
generation_gwh_2016    0
generation_gwh_2017    0
generation_gwh_2018    0
dtype: int64
```
Hence we have treated the null values now and the data now shows no null values

Clearly there is no null values

3.Exploratory Data Analysis (EDA)

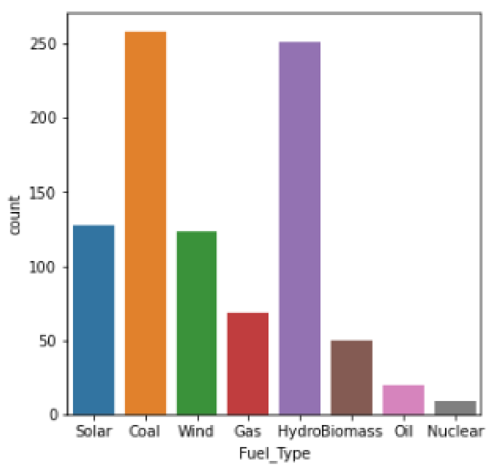
For exploratory data analysis we go through three stage of analysis

3.1 Univariate Analysis

Categorical column visualization

```
print(df['Fuel_Type'].value_counts())   #visualizing the fuel types in Fuel_Type
plt.figure(figsize=(5,5))
sns.countplot(df['Fuel_Type'])
plt.show()
```

```
Coal       258
Hydro      251
Solar      127
Wind       123
Gas         69
Biomass     50
Oil         20
Nuclear      9
Name: Fuel_Type, dtype: int64
```



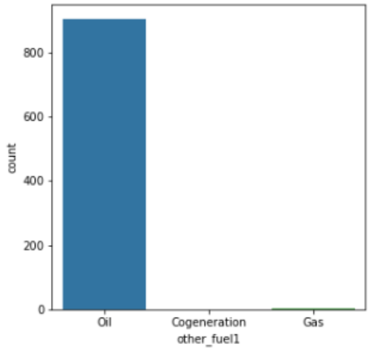
In the above count plot for "primary_fuel" column we can see that the highest number of values have been covered by coal and hydro fuel types then comes solar and wind. Finally we see that gas, biomass, oil and nuclear have very low data counts.

However when we will be considering "primary_fuel" as our target label then this is impose a class imbalance issue while trying to create a classification model and therefore will need to be treated accordingly

In [55]:

```
#checking the count of fuel1
print(df['other_fuel1'].value_counts())
plt.figure(figsize=(5,5))
sns.countplot(df['other_fuel1'])
plt.show()
```

```
Oil             904
Gas               2
Cogeneration      1
Name: other_fuel1, dtype: int64
```

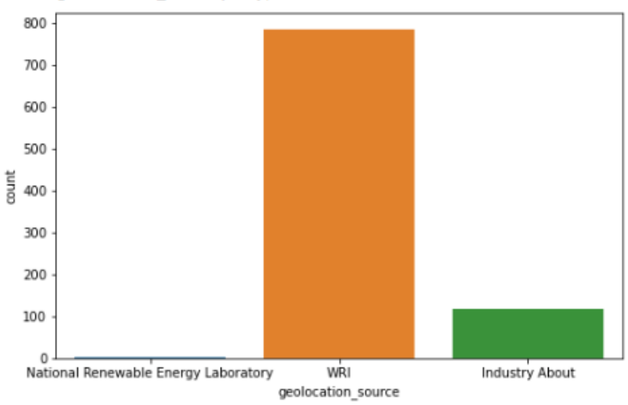


It can be observed that 'other_fuel1' type has 3 unique types namely: 'Oil', 'Cogeneration other fuel', 'Gas'. And it is clearly seen that oil is the max used fuel type.

In [56]:

```
# Visualizing the counts of owner
print(df["geolocation_source"].value_counts())
labels='WRI','Industry About','National Renewable Energy Laboratory'
plt.figure(figsize=(8,5))
sns.countplot(df['geolocation_source'])
plt.show()
```

```
WRI                                     784
Industry About                          119
National Renewable Energy Laboratory      4
Name: geolocation_source, dtype: int64
```

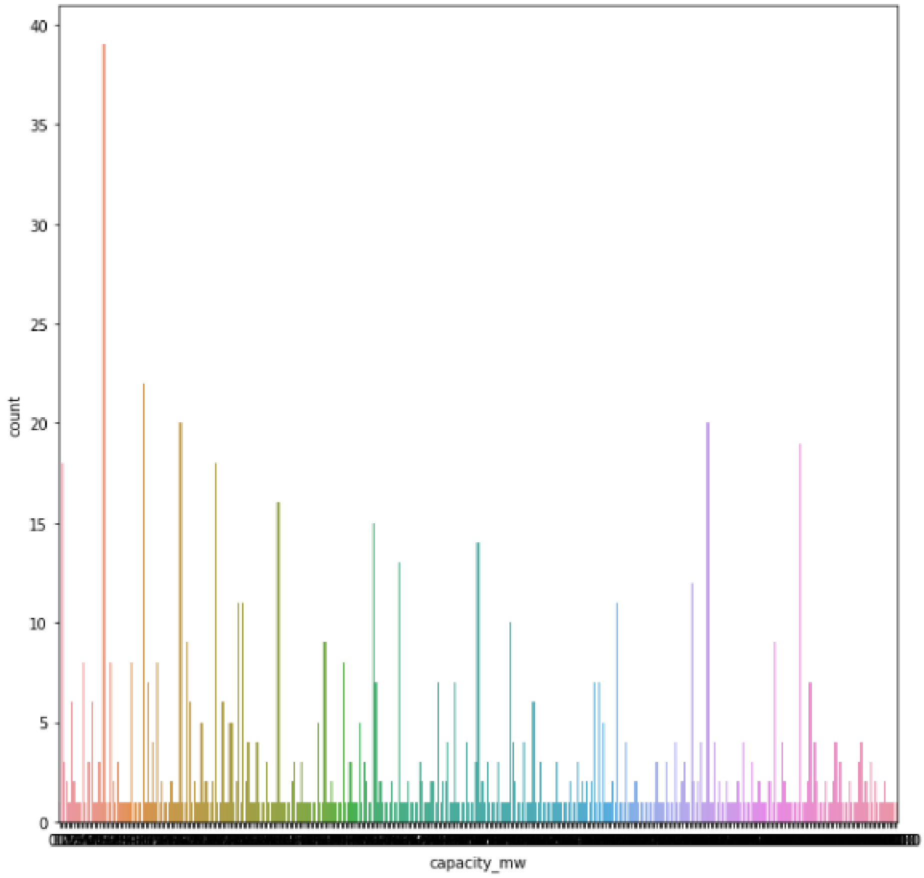


Here it can be seen that the count of WRI is the max, which means that the max information is shared by this source.

In [57]:

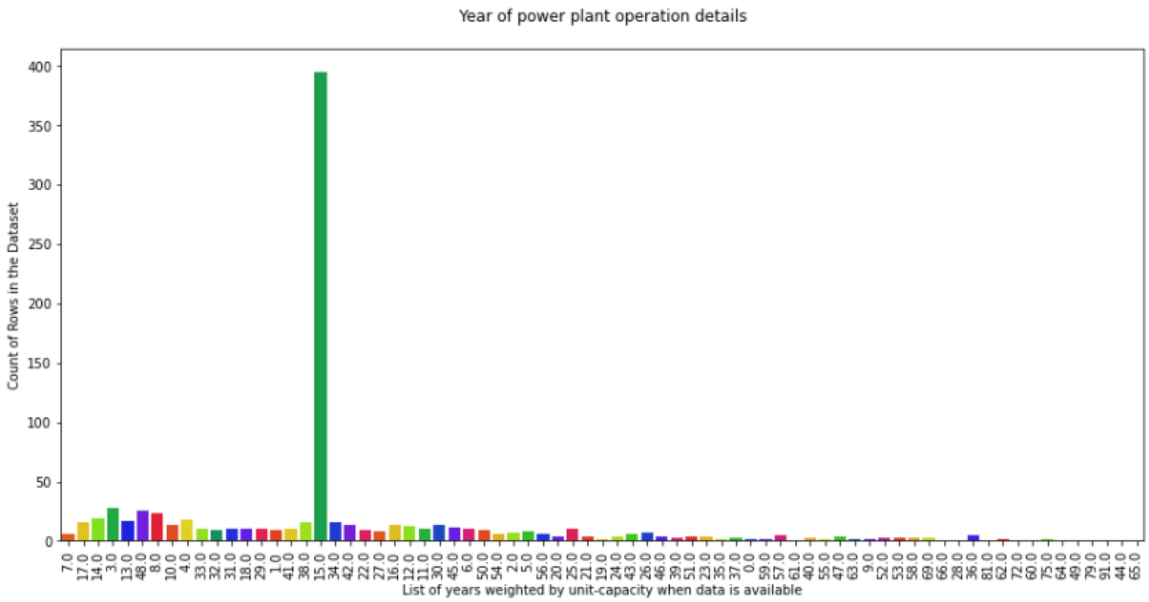
```
print(df['capacity_mw'].value_counts())    #visualizing the capacity_mw
plt.figure(figsize=(10,10))
sns.countplot(df['capacity_mw'])
plt.show()
```

```
Name: capacity_mw, Length: 361, dtype: int64
```



Observation:
Here it can be seen the counts with respect to capacity mw.

```
plt.figure(figsize=(15,7))
values = list(df['Power_plant_age'].unique())
diag = sns.countplot(df["Power_plant_age"], palette="prism")
diag.set_xticklabels(labels=values, rotation=90)
plt.title("Year of power plant operation details\n")
plt.xlabel("List of years weighted by unit-capacity when data is available")
plt.ylabel("Count of Rows in the Dataset")
plt.show()
```



Observation:

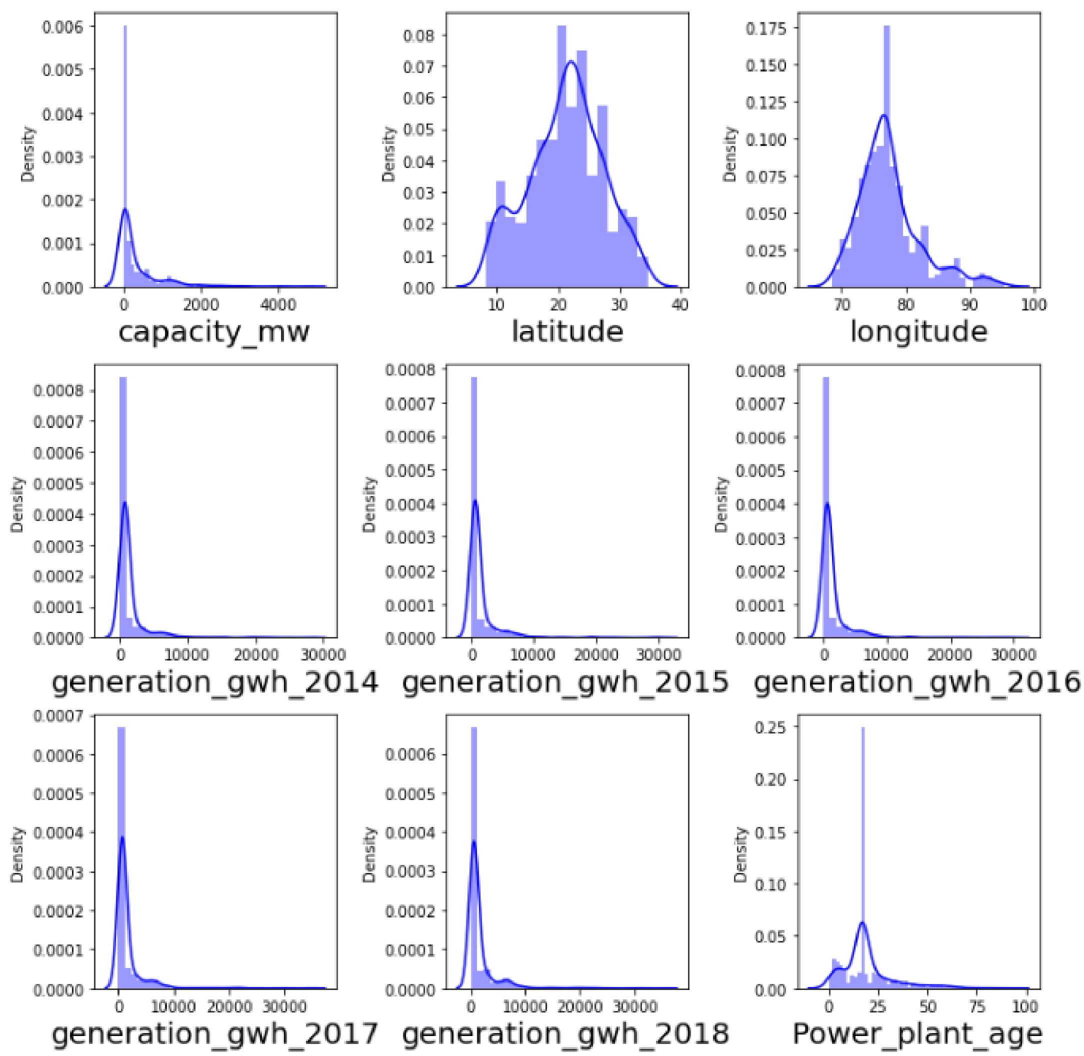
In the above count plot we can see the list of years as to when the power plant data was made available. Since we had missing values in the "commissioning_year" column we replaced it with the median wherein the year "15" covered the most rows in our dataset compared to all the other years.

]

Checking the Distribution of the Dataset, if it is normal

In [60]:

```
# Checking how the data has been distriubted in each column

plt.figure(figsize=(10,10),facecolor='white')
plotnumber=1
for column in num_col:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.distplot(df[column],color="b")
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Observation:

Here in the plots we can see that the data is not normally distributed. Outliers and skewness is present, which needs to be treated.

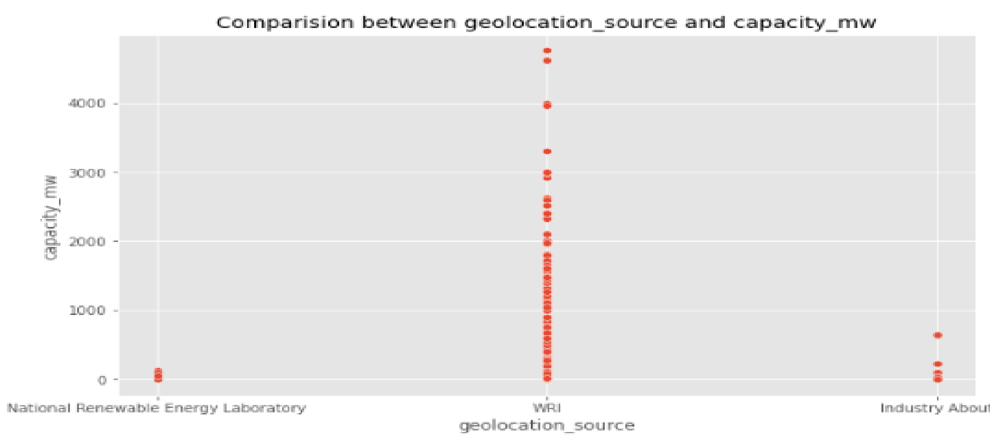
<u>3.2 Bivariate Analysis</u>

<u>Correlation between features and target 'Capacity_mw'</u>

In [61]:

```
#Checking the relation between target capacity_mw and variable geolocation source
plt.figure(figsize=[10,6])
plt.style.use('ggplot')
plt.title('Comparision between geolocation_source and capacity_mw')
sns.scatterplot(df['geolocation_source'],df["capacity_mw"])
```

Out[61]:



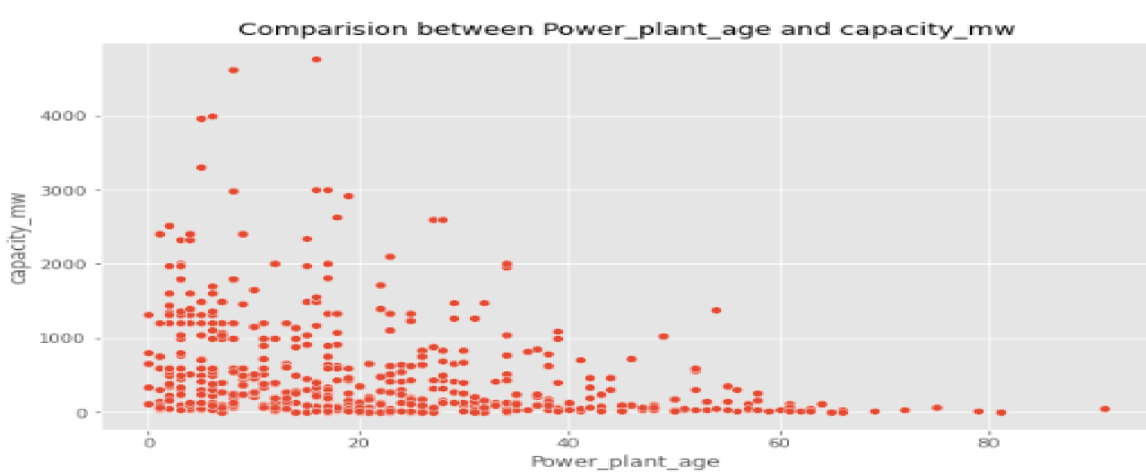
Here also we can see that WRI 'geolocation_source' plays a major role

---

In [62]:

```
#Checking the relation between power plant age and capacity_mw
plt.figure(figsize=[10,6])
plt.style.use('ggplot')
plt.title('Comparision between Power_plant_age and capacity_mw')
sns.scatterplot(df['Power_plant_age'],df["capacity_mw"])
```

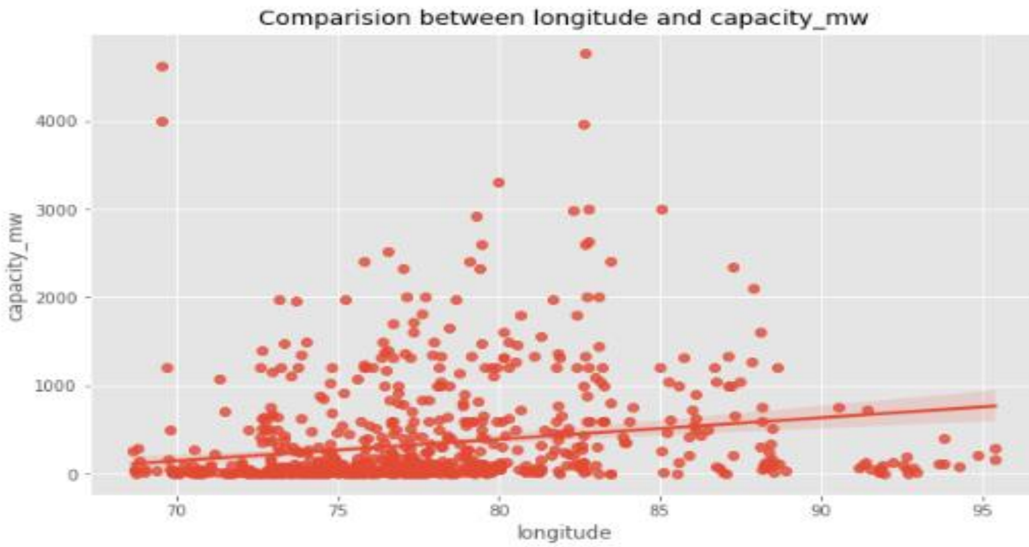
Out[62]:



Here we can see a negative correlation

In [64]:

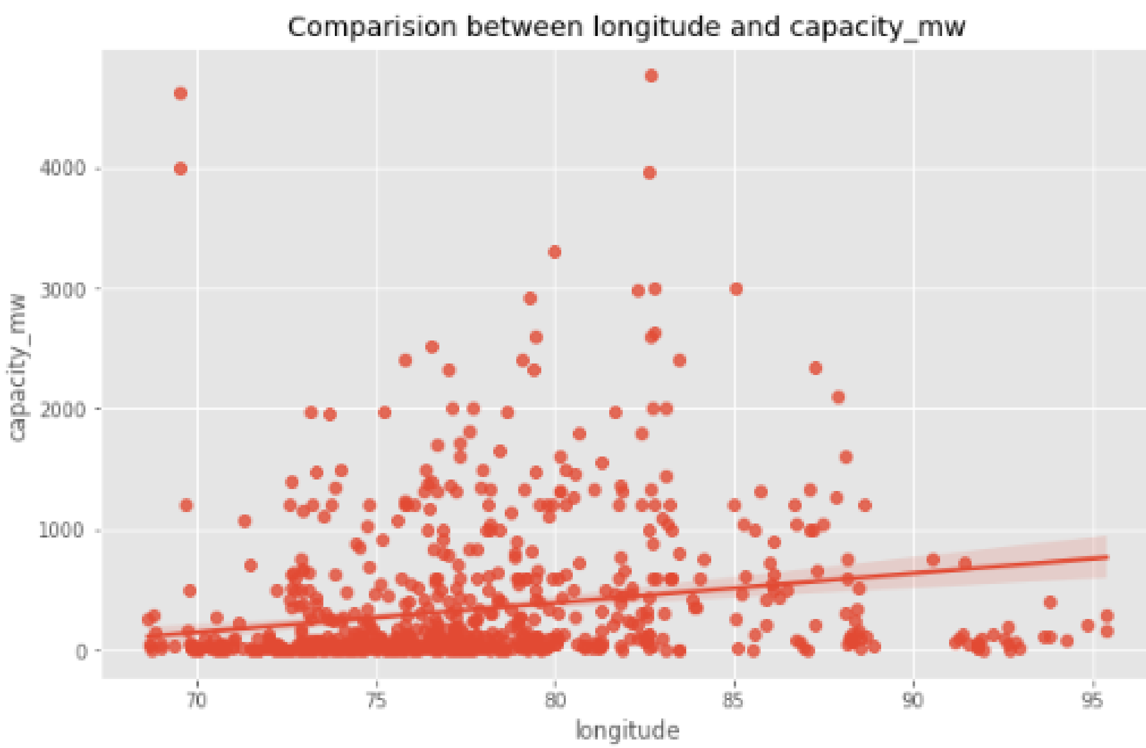
```
# Checking the relationship between target longitude and capacity_mw
plt.figure(figsize=[10,6])
plt.style.use('ggplot')
plt.title('Comparision between longitude and capacity_mw')
sns.regplot(df['longitude'],df["capacity_mw"]);
```



This feature also does not show any linear relationship

In [64]:

```
# Checking the relationship between target longitude and capacity_mw
plt.figure(figsize=[10,6])
plt.style.use('ggplot')
plt.title('Comparision between longitude and capacity_mw')
sns.regplot(df['longitude'],df["capacity_mw"]);
```

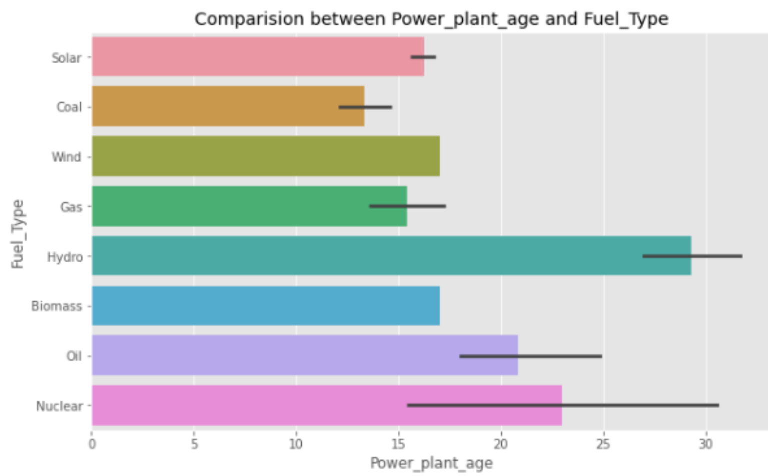


This feature also does not show any linear relationship

In [67]:

```
#Checking the relation between target fuel_type and variable Power_plant_age
plt.figure(figsize=[10,6])
plt.title('Comparision between Power_plant_age and Fuel_Type')
sns.barplot(df['Power_plant_age'],df["Fuel_Type"])
```

Out[67]:



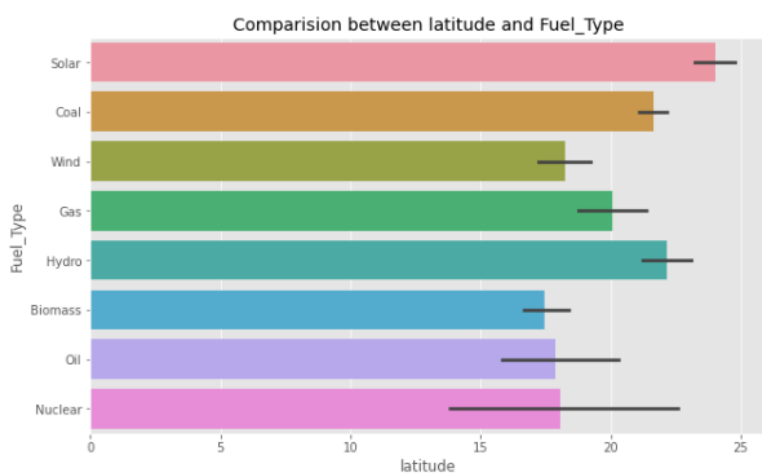
Observation:

Here we can see that older power plants uses Hydro as energy source, followed by oil. The newer power plants are using more of Coal, Solar and Gas

In [68]:

```
# Checking the relation between feature latitude and targer Fuel_Type
plt.figure(figsize=[10,6])
plt.title('Comparision between latitude and Fuel_Type')
sns.barplot(df['latitude'],df["Fuel_Type"])
```

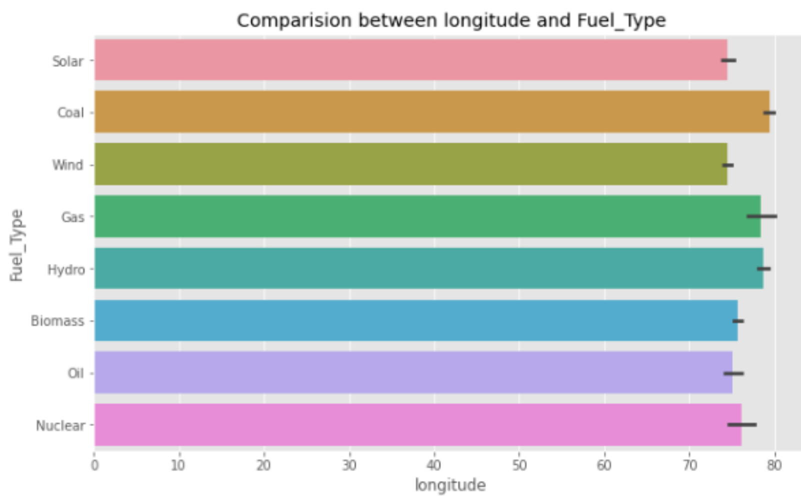
Out[68]:



Observation:

Solar has the highest latitude

In [69]:
```
# Checking the relationship between target longitude and Fuel_Type
plt.figure(figsize=[10,6])
plt.title('Comparision between longitude and Fuel_Type')
sns.barplot(df['longitude'],df["Fuel_Type"]);
```



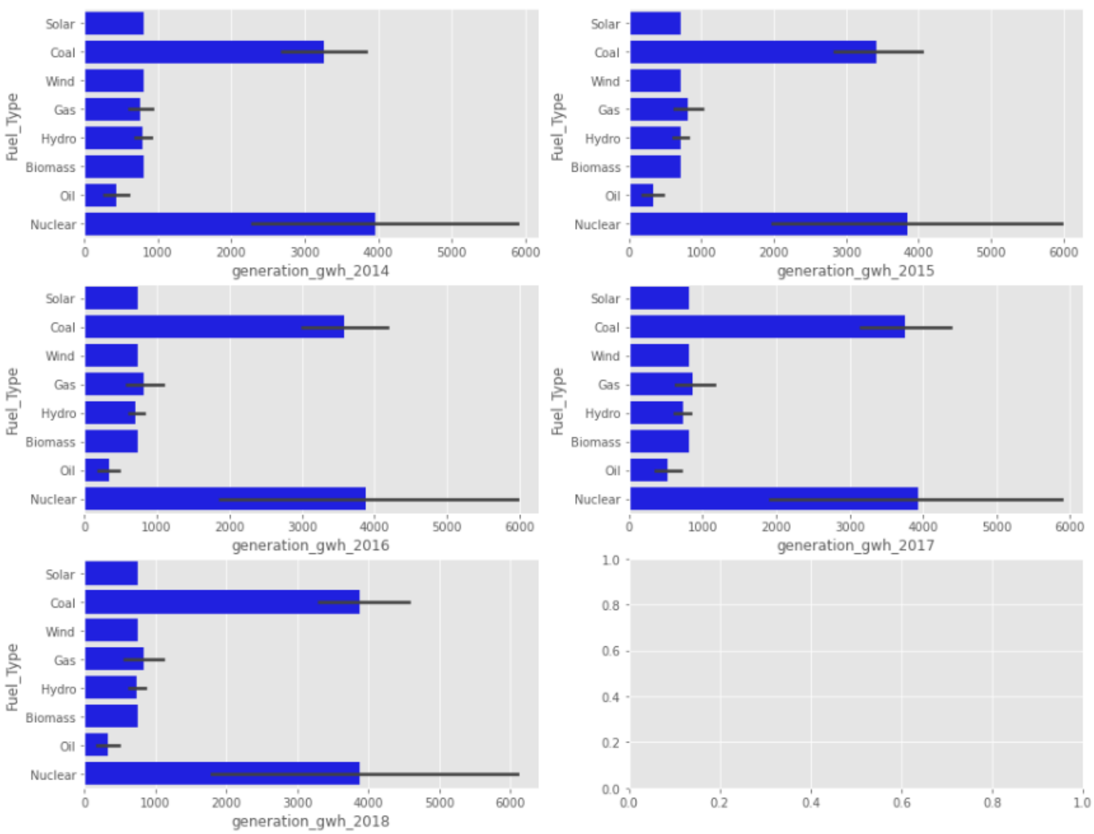
Observation:

Here Gas shows the highest longitude

In [71]:
```
fig,axes=plt.subplots(3,2,figsize=(15,12))

#Checking the relation between feature generation_gwh_2013 and targer Fuel_Type
sns.barplot(x = "generation_gwh_2014", y = "Fuel_Type",ax=axes[0,0],data = df,color="b")

#Checking the relation between feature generation_gwh_2014 and targer Fuel_Type
sns.barplot(x='generation_gwh_2015',y='Fuel_Type',ax=axes[0,1],data=df,color="b")

#Checking the relation between feature generation_gwh_2015 and targer Fuel_Type
sns.barplot(x='generation_gwh_2016',y='Fuel_Type',ax=axes[1,0],data=df,color="b")

#Checking the relation between feature generation_gwh_2016 and targer Fuel_Type
sns.barplot(x='generation_gwh_2017',y='Fuel_Type',ax=axes[1,1],data=df,color="b")

#Checking the relation between feature generation_gwh_2017 and targer Fuel_Type
sns.barplot(x='generation_gwh_2018',y='Fuel_Type',ax=axes[2,0],data=df,color="b")
plt.show()
```

Observation:

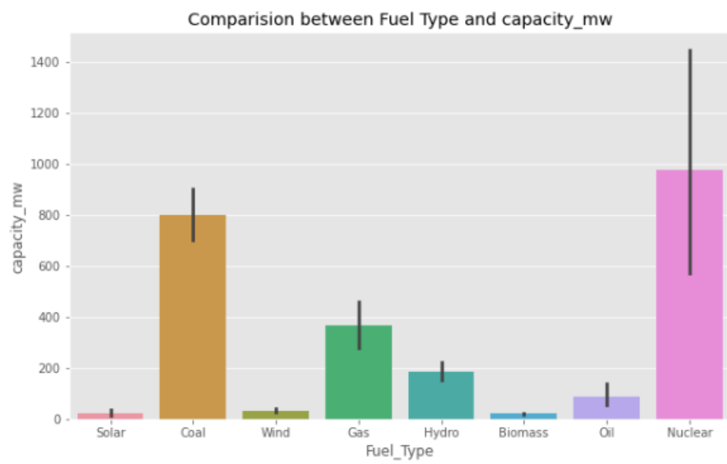
Here we can see that the most used energy source



Observation:
all the years is nuclear followed by coal

Checking the relationship between both the targets

In [74]:
```
plt.figure(figsize = (10,6))
plt.title("Comparision between Fuel Type and capacity_mw")
sns.barplot(x = "Fuel_Type", y = "capacity_mw", data = df)
plt.show()
```



Observation:

Here also it shows that energy source Nuclear has the major contribution

Label Encoding

In [75]:
```
categorical_col = ['Fuel_Type', 'other_fuel1', 'source', 'geolocation_source']
```

In [76]:
```
LE=LabelEncoder()
df[categorical_col]= df[categorical_col].apply(LE.fit_transform)
```

In [77]:
```
df[categorical_col]
```

Out[77]:

	Fuel_Type	other_fuel1	source	geolocation_source
0	6	2	109	1
1	1	2	174	2
2	7	2	21	2
3	2	2	22	2
4	1	2	22	2
...	...	...	...	...
902	1	2	22	2
903	6	2	77	0
904	7	2	21	2
905	1	2	59	2
906	7	2	21	2

907 rows × 4 columns

Now we have encoded the categorical columns

In [78]:

```
df
```

Out[78]:

	capacity_mw	latitude	longitude	Fuel_Type	other_fuel1	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_201
0	2.5	28.1839	73.2407	6	2	109	1	801.123775	711.181225	737.205450	817.97725
1	98.0	24.7663	74.6090	1	2	174	2	801.123775	711.181225	737.205450	817.97725
2	39.2	21.9038	69.3732	7	2	21	2	801.123775	711.181225	737.205450	817.97725
3	135.0	23.8712	91.3602	2	2	22	2	617.789264	843.747000	886.004428	663.77450
4	1800.0	21.9603	82.4091	1	2	22	2	3035.550000	5916.370000	6243.000000	5385.57973
...	...	...	...	...	...	...	...	...	...	...	...
902	1600.0	16.2949	77.3568	1	2	22	2	801.123775	0.994875	233.596650	865.40000
903	3.0	12.8932	78.1654	6	2	77	0	801.123775	711.181225	737.205450	817.97725
904	25.5	15.2758	75.5811	7	2	21	2	801.123775	711.181225	737.205450	817.97725
905	80.0	24.3500	73.7477	1	2	59	2	801.123775	711.181225	737.205450	817.97725
906	16.5	9.9344	77.4768	7	2	21	2	801.123775	711.181225	737.205450	817.97725

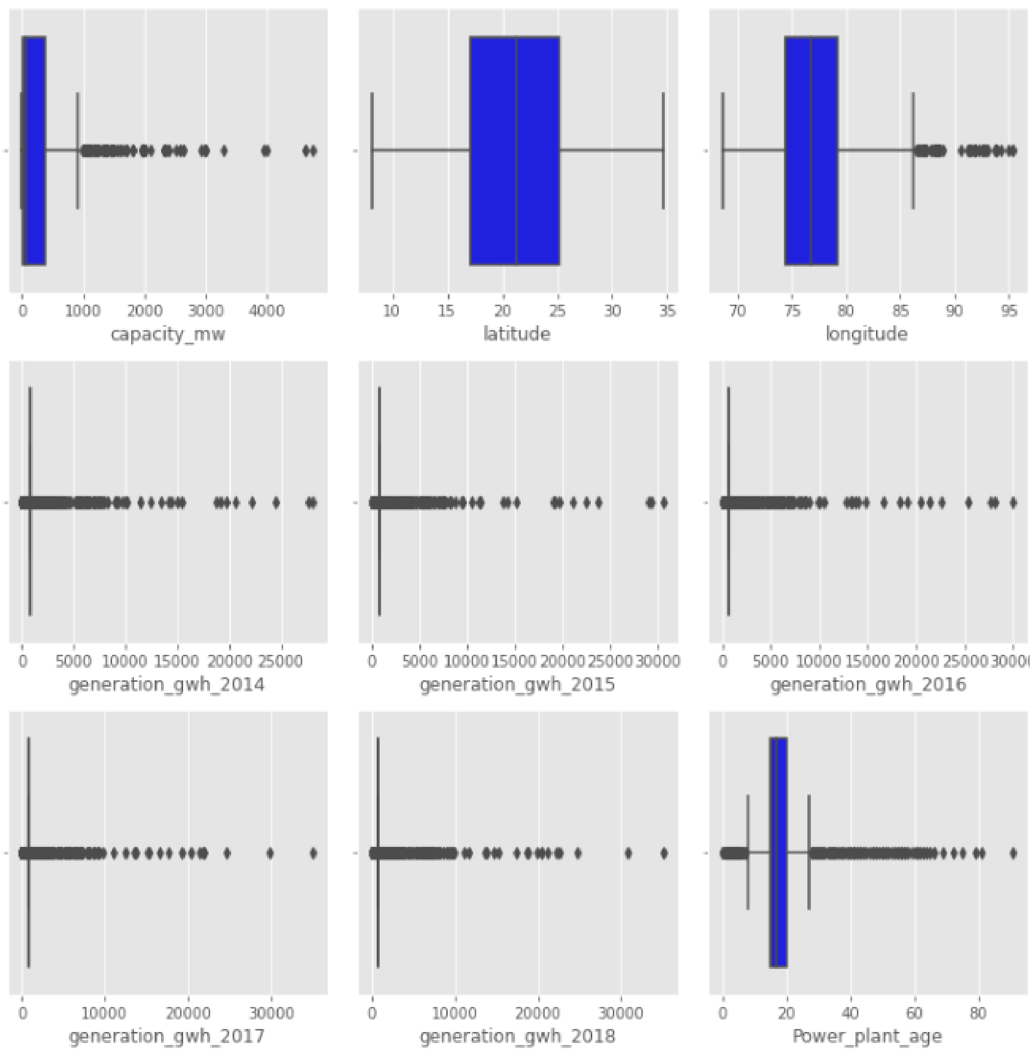
907 rows × 13 columns

4.Pre-Processing Pipeline

Identifying the outliers

In [79]:

```
plt.figure(figsize=(10,10),facecolor='white')
plotnumber=1
for column in num_col:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.boxplot(df[column],color="blue")
        plt.xlabel(column,fontsize=12)
    plotnumber+=1
plt.tight_layout()
```



Observation:

In the boxplot we can notice the outliers present in all the columns except latitude. Even target column has outliers but no need to remove it. Let's remove outliers using Zscore method.

Feauture column with outlier

In [82]:

```
# Features containing outliers
features = df[['longitude', 'generation_gwh_2014', 'generation_gwh_2015', 'generation_gwh_2016', 'generation_gwh_2017', 'generation_gwh_2018','Power_p
```

ZScore

In [83]:

```
z=np.abs(zscore(features))

z
```

Out[83]:

	longitude	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_2018	Power_plant_age
0	0.869917	0.257022	0.267783	0.275737	0.275565	0.288394	0.933076
1	0.585590	0.257022	0.267783	0.275737	0.275565	0.288394	0.172699
2	1.673567	0.257022	0.267783	0.275737	0.275565	0.288394	0.172699
3	2.895239	0.322873	0.223348	0.226194	0.326203	0.327990	0.400812
4	1.035238	0.545554	1.476964	1.557432	1.224379	1.772608	1.237227
...	...	...	...	...	...	...	...
902	0.014609	0.257022	0.505833	0.443415	0.259992	0.308963	1.313265
903	0.153415	0.257022	0.267783	0.275737	0.275565	0.288394	0.172699
904	0.383592	0.257022	0.267783	0.275737	0.275565	0.288394	0.172699
905	0.764564	0.257022	0.267783	0.275737	0.275565	0.288394	0.172699
906	0.010327	0.257022	0.267783	0.275737	0.275565	0.288394	0.172699

907 rows × 7 columns

Creating new dataframe

In [84]:

```
# Creating new dataframe
new_df = df[(z<3).all(axis=1)]
new_df
```

Out[84]:

	capacity_mw	latitude	longitude	Fuel_Type	other_fuel1	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_201
0	2.5	28.1839	73.2407	6	2	109	1	801.123775	711.181225	737.205450	817.97725
1	98.0	24.7663	74.6090	1	2	174	2	801.123775	711.181225	737.205450	817.97725
2	39.2	21.9038	69.3732	7	2	21	2	801.123775	711.181225	737.205450	817.97725
3	135.0	23.8712	91.3602	2	2	22	2	617.789264	843.747000	886.004428	663.77450
4	1800.0	21.9603	82.4091	1	2	22	2	3035.550000	5916.370000	6243.000000	5385.57973
...	...	...	...	...	...	...	...	...	...	...	
902	1600.0	16.2949	77.3568	1	2	22	2	801.123775	0.994875	233.596650	865.40000
903	3.0	12.8932	78.1654	6	2	77	0	801.123775	711.181225	737.205450	817.97725
904	25.5	15.2758	75.5811	7	2	21	2	801.123775	711.181225	737.205450	817.97725
905	80.0	24.3500	73.7477	1	2	59	2	801.123775	711.181225	737.205450	817.97725
906	16.5	9.9344	77.4768	7	2	21	2	801.123775	711.181225	737.205450	817.97725

851 rows × 13 columns

Percentage Data Loss

Percentage data loss:

In [89]:
```
loss_percent=(907-851)/907*100
print(loss_percent,'%')
```

```
6.174200661521499 %
```

Observation:

checking the data loss percentage by comparing the rows in our original data set and the new data set after removal of the outliers. usually less than 10% data loss is acceptable
Correlation between the target variable and features

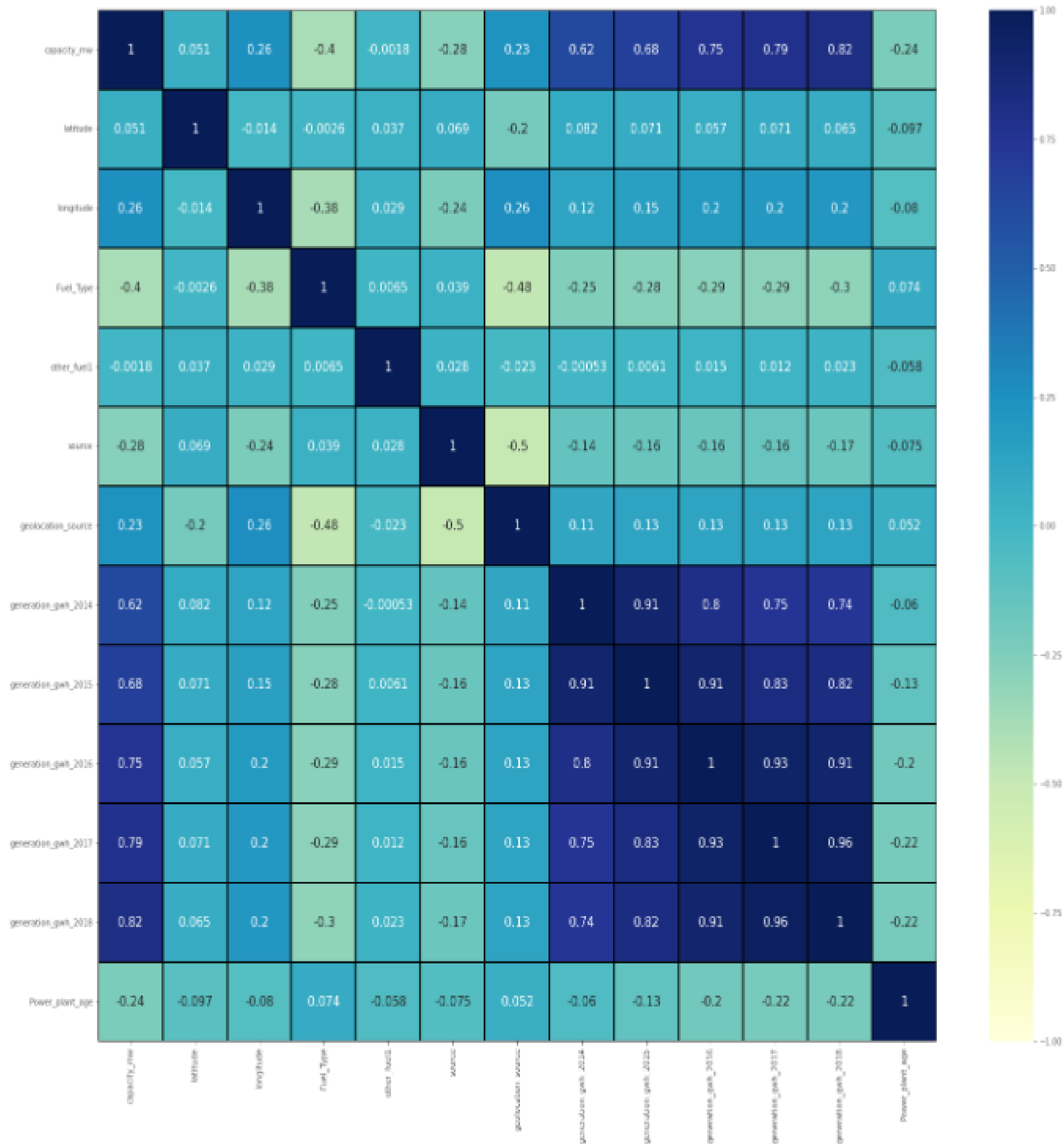
Checking The correlation

In [90]:
```
cor = new_df.corr()
cor
```

Out[90]:

	capacity_mw	latitude	longitude	Fuel_Type	other_fuel1	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016
capacity_mw	1.000000	0.050588	0.257582	-0.398039	-0.001758	-0.275724	0.234543	0.620202	0.680949	0.750088
latitude	0.050588	1.000000	-0.014145	-0.002561	0.037049	0.069430	-0.203340	0.081661	0.071346	0.057301
longitude	0.257582	-0.014145	1.000000	-0.382975	0.029191	-0.235719	0.263739	0.119473	0.154522	0.195299
Fuel_Type	-0.398039	-0.002561	-0.382975	1.000000	0.006463	0.038558	-0.478601	-0.250101	-0.276308	-0.286624
other_fuel1	-0.001758	0.037049	0.029191	0.006463	1.000000	0.028471	-0.022827	-0.000530	0.006092	0.015479
source	-0.275724	0.069430	-0.235719	0.038558	0.028471	1.000000	-0.497893	-0.141496	-0.156317	-0.158561
geolocation_source	0.234543	-0.203340	0.263739	-0.478601	-0.022827	-0.497893	1.000000	0.113446	0.125329	0.127128
generation_gwh_2014	0.620202	0.081661	0.119473	-0.250101	-0.000530	-0.141496	0.113446	1.000000	0.912185	0.801237
generation_gwh_2015	0.680949	0.071346	0.154522	-0.276308	0.006092	-0.156317	0.125329	0.912185	1.000000	0.907984
generation_gwh_2016	0.750088	0.057301	0.195299	-0.286624	0.015479	-0.158561	0.127128	0.801237	0.907984	1.000000
generation_gwh_2017	0.788972	0.071296	0.196529	-0.289232	0.011842	-0.156155	0.125199	0.746130	0.830394	0.932729
generation_gwh_2018	0.815198	0.064558	0.204421	-0.298364	0.022700	-0.165232	0.132477	0.738063	0.821726	0.913720
Power_plant_age	-0.241487	-0.097033	-0.080034	0.074358	-0.058103	-0.075353	0.051608	-0.059768	-0.130252	-0.199087

```
In [91]: plt.figure(figsize=(25,22))
sns.heatmap(new_df.corr(),linewidths=.1,vmin=-1, vmax=1, fmt='.2g', annot = True, linecolor="black",annot_kws={'size':15},cmap="YlGnBu")
plt.yticks(rotation=0)
```



Observation:

From the heat map we can notice most of the features are highly correlated with each other which leads to multicollinearity problem. So will try to solve this problem by Checking VIF value before building our models.

In [93]:
```
new_df.corr()['Fuel_Type'].sort_values()
```

Out[93]:
```
geolocation_source    -0.478601
capacity_mw           -0.398039
longitude             -0.382975
generation_gwh_2018   -0.298364
generation_gwh_2017   -0.289232
generation_gwh_2016   -0.286624
generation_gwh_2015   -0.276308
generation_gwh_2014   -0.250101
latitude              -0.002561
other_fuel1            0.006463
source                 0.038558
Power_plant_age        0.074358
Fuel_Type              1.000000
Name: Fuel_Type, dtype: float64
```

The label Fuel_Type is less correlated with Power_plant_age and source. The label is negatively correlated with geolocation_source, longitude, capacity_mw, and all generation_gwh years.

In [94]:
```
new_df.corr()['capacity_mw'].sort_values()
```

Out[94]:
```
Fuel_Type             -0.398039
source                -0.275724
Power_plant_age       -0.241487
other_fuel1           -0.001758
latitude               0.050588
geolocation_source     0.234543
longitude              0.257582
generation_gwh_2014    0.620202
generation_gwh_2015    0.680949
generation_gwh_2016    0.750088
generation_gwh_2017    0.788972
generation_gwh_2018    0.815198
capacity_mw            1.000000
Name: capacity_mw, dtype: float64
```

Here we can see the co-relation between all the features and the features and targets
The label capacity_mw is highly positively correlated with the features generation_gwh_2017, generation_gwh_2016, generation_gwh_2015, generation_gwh_2014, generation_gwh_2013. And the label is negatively correlated with the features Fuel_Type, source and Power_plant_age. The columns other_fuel1 and latitude have no relation with the label, so we can drop them.

MultiCollinearity with Variance Inflation Factor

In [96]:

```
df1=pd.DataFrame(data=new_df)       # copying the dataframe
df1
```

Out[96]:

	capacity_mw	latitude	longitude	Fuel_Type	other_fuel1	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_201
0	2.5	28.1839	73.2407	6	2	109	1	801.123775	711.181225	737.205450	817.97725
1	98.0	24.7663	74.6090	1	2	174	2	801.123775	711.181225	737.205450	817.97725
2	39.2	21.9038	69.3732	7	2	21	2	801.123775	711.181225	737.205450	817.97725
3	135.0	23.8712	91.3602	2	2	22	2	617.789264	843.747000	886.004428	663.77450
4	1800.0	21.9603	82.4091	1	2	22	2	3035.550000	5916.370000	6243.000000	5385.57973
...	...	...	...	...	...	...	...	...	...	...	...
902	1600.0	16.2949	77.3568	1	2	22	2	801.123775	0.994875	233.596650	865.40000
903	3.0	12.8932	78.1654	6	2	77	0	801.123775	711.181225	737.205450	817.97725
904	25.5	15.2758	75.5811	7	2	21	2	801.123775	711.181225	737.205450	817.97725
905	80.0	24.3500	73.7477	1	2	59	2	801.123775	711.181225	737.205450	817.97725
906	16.5	9.9344	77.4768	7	2	21	2	801.123775	711.181225	737.205450	817.97725

851 rows × 13 columns

In [97]:

```
x1=df1.iloc[:,1:]
y1=df1.iloc[:,0]
```

In [98]:

```
x1
```

Out[98]:

	latitude	longitude	Fuel_Type	other_fuel1	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_
0	28.1839	73.2407	6	2	109	1	801.123775	711.181225	737.205450	817.977250	7
1	24.7663	74.6090	1	2	174	2	801.123775	711.181225	737.205450	817.977250	7
2	21.9038	69.3732	7	2	21	2	801.123775	711.181225	737.205450	817.977250	7
3	23.8712	91.3602	2	2	22	2	617.789264	843.747000	886.004428	663.774500	6
4	21.9603	82.4091	1	2	22	2	3035.550000	5916.370000	6243.000000	5385.579736	72
...	...	...	...	...	...	...	...	...	...	...	
902	16.2949	77.3568	1	2	22	2	801.123775	0.994875	233.596650	865.400000	6
903	12.8932	78.1654	6	2	77	0	801.123775	711.181225	737.205450	817.977250	7
904	15.2758	75.5811	7	2	21	2	801.123775	711.181225	737.205450	817.977250	7
905	24.3500	73.7477	1	2	59	2	801.123775	711.181225	737.205450	817.977250	7
906	9.9344	77.4768	7	2	21	2	801.123775	711.181225	737.205450	817.977250	7

851 rows × 12 columns

In [99]:

```
y1
```

Out[99]:

```
0         2.5
1        98.0
2        39.2
3       135.0
4      1800.0
        ...  
902    1600.0
903       3.0
904      25.5
905      80.0
906      16.5
Name: capacity_mw, Length: 851, dtype: float64
```

Variable Inflation Factor

In [106…

```
x1=df1.drop(['other_fuel1'],axis=1)
```

In [107…

```
calc_vif(x1)
```

Out[107…

	variables	VIF FACTOR
0	capacity_mw	4.925265
1	latitude	13.421012
2	longitude	47.409391
3	Fuel_Type	4.210582
4	source	2.833991
5	geolocation_source	12.954951
6	generation_gwh_2014	10.551644
7	generation_gwh_2015	20.382745
8	generation_gwh_2016	22.935981
9	generation_gwh_2017	30.879133
10	generation_gwh_2018	25.678834
11	Power_plant_age	4.096977

Since latitude has the lowest contribution compared to both the targets lets drop that first and see what happens

In [103…

```
# importing required libraries
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [104…

```
def calc_vif(x1):
    vif=pd.DataFrame()
    vif["variables"]=x1.columns
    vif["VIF FACTOR"]=[variance_inflation_factor(x1.values,i) for i in range(x1.shape[1])]
    return(vif)
```

In [105…

```
calc_vif(x1)
```

Out[105…

	variables	VIF FACTOR
0	latitude	13.872461
1	longitude	244.574821
2	Fuel_Type	4.644438
3	other_fuel1	279.902398
4	source	2.932705
5	geolocation_source	13.596524
6	generation_gwh_2014	10.519302
7	generation_gwh_2015	20.375290
8	generation_gwh_2016	22.939038
9	generation_gwh_2017	30.883761
10	generation_gwh_2018	23.124194
11	Power_plant_age	4.029709

Feature selection by dropping columns

In [110…

```
new_df.drop("other_fuel1",axis=1,inplace=True)
new_df.drop("latitude",axis=1,inplace=True)
```

In [111…

```
new_df.head()
```

Out[111…

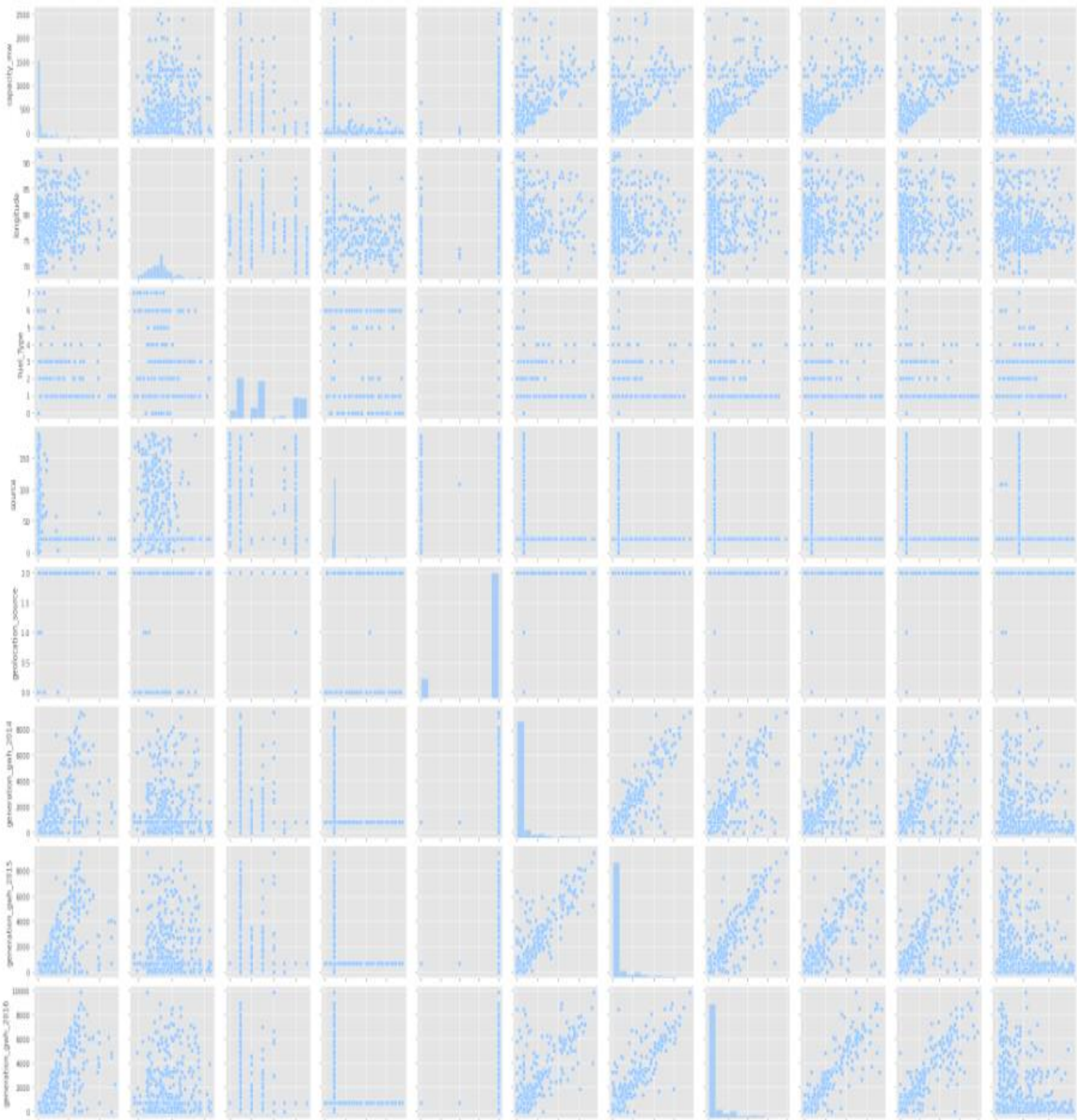
	capacity_mw	longitude	Fuel_Type	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_20
0	2.5	73.2407	6	109	1	801.123775	711.181225	737.205450	817.977250	751.6443
1	98.0	74.6090	1	174	2	801.123775	711.181225	737.205450	817.977250	751.6443
2	39.2	69.3732	7	21	2	801.123775	711.181225	737.205450	817.977250	751.6443
3	135.0	91.3602	2	22	2	617.789264	843.747000	886.004428	663.774500	626.2391
4	1800.0	82.4091	1	22	2	3035.550000	5916.370000	6243.000000	5385.579736	7279.0000

<u>3.3 Multivariate Analysis:</u>

In [112…

```
sns.pairplot(new_df)
```

Out[112…



5. Machine Learning Algorithm

5.1 Predicting "Capacity_mw" Target

Splitting the dataset into Features and Target

In [118…

```
x.skew().sort_values()
```

Out[118…

```
geolocation_source    -2.066536
Fuel_Type              0.413759
longitude              0.945877
Power_plant_age        1.280800
source                 1.734252
generation_gwh_2017    2.546541
generation_gwh_2018    2.597029
generation_gwh_2016    2.645786
generation_gwh_2015    2.714999
generation_gwh_2014    2.943026
dtype: float64
```

The following columns have skewness more than +0.5 and -0.5.

longitude generation_gwh_2013 generation_gwh_2014 generation_gwh_2015 generation_gwh_2016 generation_gwh_2017 Power_plant_age

In [113…
```
x=new_df.drop('capacity_mw', axis=1)
y=new_df["capacity_mw"]
```

In [114…
```
x.shape
```

Out[114…
```
(851, 10)
```

In [115…
```
y.shape
```

Out[115…
```
(851,)
```

In [116…
```
x
```

Out[116…

	longitude	Fuel_Type	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_2018	Power_plan
0	73.2407	6	109	1	801.123775	711.181225	737.205450	817.977250	751.644375	
1	74.6090	1	174	2	801.123775	711.181225	737.205450	817.977250	751.644375	
2	69.3732	7	21	2	801.123775	711.181225	737.205450	817.977250	751.644375	
3	91.3602	2	22	2	617.789264	843.747000	886.004428	663.774500	626.239128	
4	82.4091	1	22	2	3035.550000	5916.370000	6243.000000	5385.579736	7279.000000	
...	...	...	...	...	...	...	...	...	...	
902	77.3568	1	22	2	801.123775	0.994875	233.596650	865.400000	686.500000	
903	78.1654	6	77	0	801.123775	711.181225	737.205450	817.977250	751.644375	
904	75.5811	7	21	2	801.123775	711.181225	737.205450	817.977250	751.644375	
905	73.7477	1	59	2	801.123775	711.181225	737.205450	817.977250	751.644375	
906	77.4768	7	21	2	801.123775	711.181225	737.205450	817.977250	751.644375	

851 rows × 10 columns

In [117…
```
y
```

Out[117…
```
0         2.5
1        98.0
2        39.2
3       135.0
4      1800.0
        ...  
902    1600.0
903       3.0
904      25.5
905      80.0
906      16.5
Name: capacity_mw, Length: 851, dtype: float64
```

Checking for skewness

In [118…
```
x.skew().sort_values()
```

Out[118…
```
geolocation_source    -2.066536
Fuel_Type              0.413759
longitude              0.945877
Power_plant_age        1.280800
source                 1.734252
generation_gwh_2017    2.546541
generation_gwh_2018    2.597029
generation_gwh_2016    2.645786
generation_gwh_2015    2.714999
generation_gwh_2014    2.943026
dtype: float64
```

The following columns have skewness more than +0.5 and -0.5.

longitude generation_gwh_2013 generation_gwh_2014 generation_gwh_2015 generation_gwh_2016 generation_gwh_2017 Power_plant_age

Removing skewness using yeo-johnson method

In [123…

```
from sklearn.preprocessing import PowerTransformer
skew = ['longitude','generation_gwh_2014','generation_gwh_2015','generation_gwh_2016','generation_gwh_2017','generation_gwh_2018','Power_plant_age
transf = PowerTransformer(method='yeo-johnson')
```

In [124…

```
x[skew] = transf.fit_transform(x[skew].values)
x[skew].head()
```

Out[124…

	longitude	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_2018	Power_plant_age
0	-0.922012	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	-1.081421
1	-0.499829	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	0.046187
2	-2.377759	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	0.046187
3	2.430594	-0.268922	0.093773	0.105691	-0.199692	-0.194159	-0.245810
4	1.261979	1.426798	2.286603	2.276671	1.983083	2.347272	-1.758384

In [125…

```
x.skew()
```

Out[125…

```
longitude             -0.000128
Fuel_Type              0.413759
source                 1.734252
geolocation_source    -2.066536
generation_gwh_2014    0.232399
generation_gwh_2015    0.163587
generation_gwh_2016    0.147035
generation_gwh_2017    0.127152
generation_gwh_2018    0.133691
Power_plant_age        0.043734
dtype: float64
```

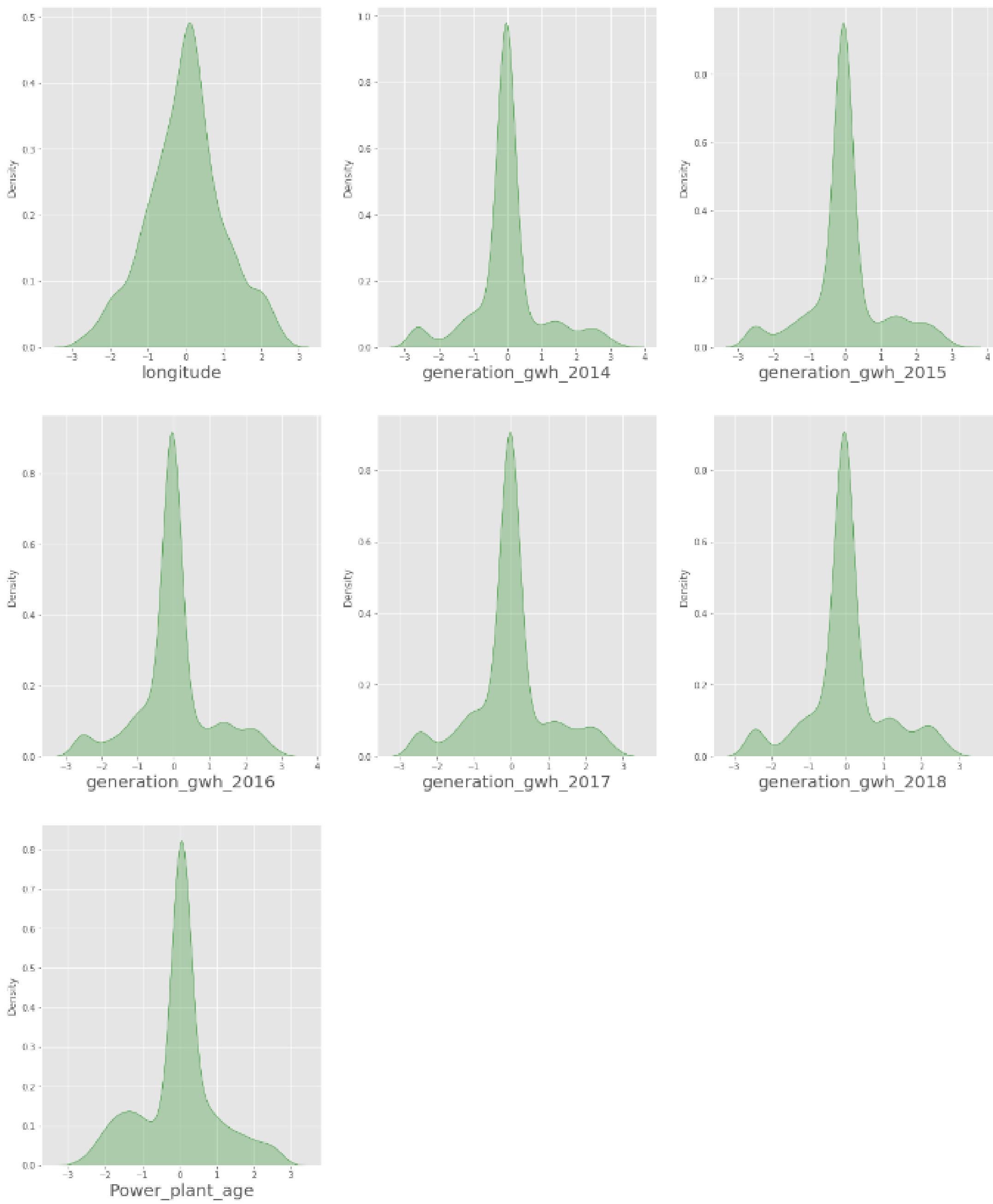
Since Fuel_Type, source and geolocation_source were categorically encoded values we didnt use transformation for skewness removal.

Rest of the numerical data columns the skewness has been removed.

Checking the Distribution of the dataset

In [126…

```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in x[skew]:
    if plotnumber<=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(x[column],color='g',kde_kws={"shade": True},hist=False)
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



Feature Scalling

In [127…

```
#Scalling the data using Standard Scaler
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
x
```

Out[127…

	longitude	Fuel Type	source	geolocation source	generation gwh 2014	generation gwh 2015	generation gwh 2016	generation gwh 2017	generation gwh 2018	Fu
0	-0.922012	1.175506	1.397951	-1.036523	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
1	-0.499829	-0.975797	2.821796	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
2	-2.377759	1.605767	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
3	2.430594	-0.545536	-0.507812	0.407145	-0.268922	0.093773	0.105691	-0.199692	-0.194159	
4	1.261979	-0.975797	-0.507812	0.407145	1.426798	2.286603	2.276671	1.983083	2.347272	
...	...	...	...	...	...	...	...	...	...	
846	0.231932	-0.975797	-0.507812	0.407145	-0.044061	-2.461379	-0.842266	0.010837	-0.126054	
847	0.421592	1.175506	0.696980	-2.480190	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
848	-0.224400	1.605767	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
849	-0.760624	-0.975797	0.302685	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
850	0.260758	1.605767	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	

851 rows × 10 columns

The dataset x has now been scaled.

MultiCollinearity with Variance Inflation Factor

In [128…

```
vif = pd.DataFrame()
vif["VIF values"] = [variance_inflation_factor(x.values,i)
                     for i in range(len(x.columns))]
vif["Features"] = x.columns

# Let's check the values
vif
```

Out[128…

	VIF values	Features
0	1.309948	longitude
1	1.682645	Fuel_Type
2	1.503721	source
3	1.875750	geolocation_source
4	3.603333	generation_gwh_2014
5	6.182235	generation_gwh_2015
6	9.957776	generation_gwh_2016
7	9.750143	generation_gwh_2017
8	8.951489	generation_gwh_2018
9	1.102659	Power_plant_age

VIF values in all the columns are less then 10, hence no multicolinearity problem exists.

Finding best random state

let's find the best random state in which we can build the model.

(Random state ensures that the splits that you generate are reproducible. Scikit-learn use random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.)

For Test size of 0.20

In [133…

```
#getting the best random state for .20 test size
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test, y_train, y_test =train_test_split(x,y, test_size=.20,random_state=i)
    mod=RandomForestRegressor()
    mod.fit(x_train, y_train)
    pred=mod.predict(x_test)
    acc=r2_score(y_test,pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print('R2 Score=', maxAccu, 'Random_State',maxRS)
```

```
R2 Score= 0.8782394329466094 Random_State 125
```

For Test size of 0.20

In [134…

```
#getting the best random state for .30 test size
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test, y_train, y_test =train_test_split(x,y, test_size=.30,random_state=i)
    mod=RandomForestRegressor()
    mod.fit(x_train, y_train)
    pred=mod.predict(x_test)
    acc=r2_score(y_test,pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print('R2 Score=', maxAccu, 'Random_State',maxRS)
```

```
R2 Score= 0.8670313188948057 Random_State 110
```

We got best r2 score of 0.878 at a random state of 125 for test_size=.20

Splitting the dataset into Features and Target

In [135…

```
x_train,x_test, y_train, y_test=train_test_split(x,y,test_size=.20, random_state=125)
```

In [137…

```
# importing all the required libraries

from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

In [138…

```
# creating a function to run all the regressors

def regressor(model, x, y):
    x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.2, random_state=125)

    # Training the model
    model.fit(x_train, y_train)

    # Predicting y_test
    pred = model.predict(x_test)

    # Root Mean Square Error (RMSE)
    rmse = mean_squared_error(y_test, pred, squared=False)
    print("Root Mean Square Error is:", rmse)

    # R2 score
    r2 = r2_score(y_test, pred, multioutput='variance_weighted')*100
    print("R2 Score is:", r2)

    # Cross Validation Score
    cv_score = (cross_val_score(model, x, y, cv=5).mean())*100
    print("Cross Validation Score is:", cv_score)

    # Result of r2 score - cv score
    result = r2 - cv_score
    print("R2 Score - Cross Validation Score is", result)
```

Machine Learning Algorithm

Elastic Net

In [142…

```
model=ElasticNet(alpha=0.001)
regressor(model, x, y)
```

```
Root Mean Square Error is: 245.14355271630316
R2 Score is: 60.52471830409167
Cross Validation Score is: 54.41670015581646
R2 Score - Cross Validation Score is 6.108018148275214
```

Support Vector Regression

In [143…

```
model=SVR(kernel='rbf')
regressor(model, x, y)
```

```
Root Mean Square Error is: 404.7742028213369
R2 Score is: -7.624200572733364
Cross Validation Score is: -11.39657492770902
R2 Score - Cross Validation Score is 3.772374354975656
```

In [144…

```
model=SVR(kernel='poly')
regressor(model, x, y)
```

```
Root Mean Square Error is: 313.2288103301929
R2 Score is: 35.55226062438119
Cross Validation Score is: 26.42896802921332
R2 Score - Cross Validation Score is 9.123292595167872
```

In [145…

```
model=SVR(kernel='linear')
regressor(model, x, y)
```

```
Root Mean Square Error is: 274.9566511710502
R2 Score is: 50.339310438897165
Cross Validation Score is: 43.403823512262655
R2 Score - Cross Validation Score is 6.9354869266345105
```

R2 Score - Cross Validation Score is 9.139924494092483

K Neighbors Regressor

In [150…

```
model=KNeighborsRegressor()
regressor(model, x, y)
```

```
Root Mean Square Error is: 162.60726630197928
R2 Score is: 82.63142518925956
Cross Validation Score is: 72.4607518724892
R2 Score - Cross Validation Score is 10.170673316770362
```

L1 -- Lasso Regression

In [140…

```
model=Lasso(alpha=0.001)
regressor(model, x, y)
```

```
Root Mean Square Error is: 245.21635902893112
R2 Score is: 60.50126693074731
Cross Validation Score is: 54.40459227008451
R2 Score - Cross Validation Score is 6.096674660662806
```

L2 -- Ridge Regression

In [141…

```
model=Ridge(alpha=0.001)
regressor(model, x, y)
```

```
Root Mean Square Error is: 245.2174226821166
R2 Score is: 60.5009242697231
Cross Validation Score is: 54.404378857678324
R2 Score - Cross Validation Score is 6.096545412044776
```

Decision Tree Regressor

In [146…

```
model=DecisionTreeRegressor(random_state=125)
regressor(model, x, y)
```

```
Root Mean Square Error is: 194.07161222939695
R2 Score is: 75.25950903732124
Cross Validation Score is: 58.76758704001117
R2 Score - Cross Validation Score is 16.491921997310065
```

In [147…

```
model=DecisionTreeRegressor()
regressor(model, x, y)
```

```
Root Mean Square Error is: 206.22122476479683
R2 Score is: 72.06484985388744
Cross Validation Score is: 57.62200252871914
R2 Score - Cross Validation Score is 14.442847325168295
```

Random Forest Regressor

In [148…

```
model=RandomForestRegressor()
regressor(model, x, y)
```

```
Root Mean Square Error is: 137.33916402381092
R2 Score is: 87.60994924744097
Cross Validation Score is: 78.21469806444237
R2 Score - Cross Validation Score is 9.395251182998592
```

In [149…

```
model=RandomForestRegressor(random_state=125)
regressor(model, x, y)
```

```
Root Mean Square Error is: 139.04391979729402
R2 Score is: 87.30045119928242
Cross Validation Score is: 78.16052670518994
R2 Score - Cross Validation Score is 9.139924494092483
```

SGD Regressor

In [151…

```
model=SGDRegressor()
regressor(model, x, y)
```

```
Root Mean Square Error is: 241.7983594493085
R2 Score is: 61.59471554324944
Cross Validation Score is: 55.20791681564043
R2 Score - Cross Validation Score is 6.386798727609012
```

Gradient Boosting Regressor

In [152…

```
model=GradientBoostingRegressor()
regressor(model, x, y)
```

```
Root Mean Square Error is: 144.4344823209454
R2 Score is: 86.29667163426653
Cross Validation Score is: 75.23989401383002
R2 Score - Cross Validation Score is 11.056777620436506
```

Ada Boost Regressor

In [153…

```
model=AdaBoostRegressor(random_state=125)
regressor(model, x, y)
```

```
Root Mean Square Error is: 245.82383804952505
R2 Score is: 60.30532263916826
Cross Validation Score is: 58.66999364833373
R2 Score - Cross Validation Score is 1.6353289908345303
```

In [154…

```
model=AdaBoostRegressor()
regressor(model, x, y)
```

```
Root Mean Square Error is: 267.94664398832117
R2 Score is: 52.839225349376186
Cross Validation Score is: 57.69128464530135
R2 Score - Cross Validation Score is -4.852059295925166
```

Extra Trees Regressor

In [155…

```
model=ExtraTreesRegressor(random_state=125)
regressor(model, x, y)
```

```
Root Mean Square Error is: 137.4140603824718
R2 Score is: 87.59643201482244
Cross Validation Score is: 79.43962573802399
R2 Score - Cross Validation Score is 8.156806276798449
```

In [156…

```
model=ExtraTreesRegressor()
regressor(model, x, y)
```

```
Root Mean Square Error is: 137.64693759275738
R2 Score is: 87.55435544862121
Cross Validation Score is: 79.60991781073245
R2 Score - Cross Validation Score is 7.944437637888754
```

Hyper parameter tuning

In [158…
```
#ExtraTreesRegressor?
```

In [157…
```
ExtraTreesRegressor().get_params().keys()
```

Out[157…
```
dict_keys(['bootstrap', 'ccp_alpha', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'max_samples', 'min_impurity_decrease', 'min_samp
eaf', 'min_samples_split', 'min_weight_fraction_leaf', 'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'verbose', 'warm_start'])
```

In [159…
```
# creating parameters list to pass into GridSearchCV

parameters = {'criterion' : ['squared_error', 'absolute_error'],
              'max_features' : ['auto', 'sqrt', 'log2'],
              'n_jobs' : [5, 10, 15]}
```

In [160…
```
GCV = GridSearchCV(ExtraTreesRegressor(), parameters, cv=5)
```

In [161…
```
GCV.fit(x_train,y_train)
```

Out[161…
```
GridSearchCV(cv=5, estimator=ExtraTreesRegressor(),
             param_grid={'criterion': ['squared_error', 'absolute_error'],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_jobs': [5, 10, 15]})
```

In [162…
```
GCV.best_params_      # printing best parameters found by GridSearchCV
```

Out[162…
```
{'criterion': 'absolute_error', 'max_features': 'log2', 'n_jobs': 15}
```

We got the best parameters using Gridsearch CV

In [163…
```
final_model = ExtraTreesRegressor(criterion = 'absolute_error', max_features = 'log2', n_jobs = 15)
```

In [164…
```
final_fit = final_model.fit(x_train,y_train)    # final fit
```

In [165…
```
final_pred = final_model.predict(x_test)    # predicting with best parameters
```

In [166…
```
best_r2=r2_score(y_test,final_pred,multioutput='variance_weighted')*100    # checking final r2_score
print("R2 score for the Best Model is:", best_r2)
```

```
R2 score for the Best Model is: 89.24918510508195
```

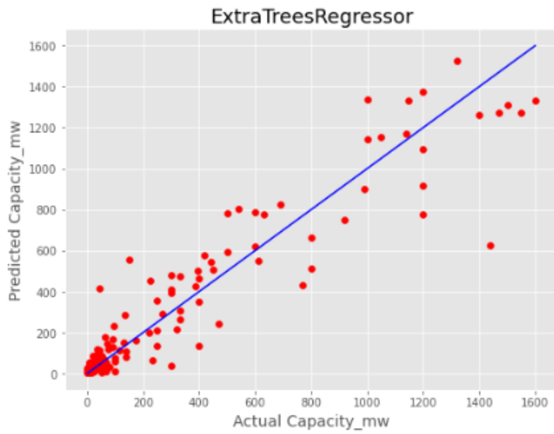
In [167…
```
final_cv_score = (cross_val_score(final_model, x, y, cv=5).mean())*100
print("Cross Validation Score is:", final_cv_score)
```

```
Cross Validation Score is: 79.40239920741098
```

In [168…
```
final_rmse = mean_squared_error(y_test, final_pred, squared=False)
print("Root Mean Square Error is:", final_rmse)
```

In [170…

```
plt.figure(figsize=(8,6))
plt.scatter(x=y_test, y=final_pred, color='r')
plt1 = max(max(final_pred), max(y_test))
plt2 = min(min(final_pred), min(y_test))
plt.plot([plt1, plt2], [plt1, plt2], 'b-')
plt.xlabel('Actual Capacity_mw', fontsize=14)
plt.ylabel('Predicted Capacity_mw', fontsize=14)
plt.title('ExtraTreesRegressor',fontsize=18)
plt.show()
```



Plotting the Final model Actual Capacity_mw vs Predicted Capacity_mw

Hence after Hyper Parameter Tuning on the final model to obtained the best r2_score 89.249% and CV score 79.4% and lowest Root Mean Square Error is: 127.93.

Saving the model in pickle Format

In [171…

```
# pickeling or serialization of a file
import pickle
filename = 'Global_Power_Plant_Capacity_mw_Regression_final_model.pkl'
pickle.dump(final_model, open(filename, 'wb'))
```

Saving the best regression model using pickle

Prediction Conclusion:

In [172…

```
import numpy as np
a=np.array(y_test)
predicted=np.array(final_model.predict(x_test))
df_comparison = pd.DataFrame({"original":a,"predicted":predicted},index= range(len(a)))
df_comparison
```

Out[172…

	original	predicted
0	1.00	6.40400
1	1440.00	626.64900
2	55.00	42.77375
3	10.50	11.32170
4	68.80	12.00370
...	...	...
166	1200.00	1376.02080
167	330.00	306.40010
168	10.00	13.46500
169	56.25	24.23220
170	1147.50	1332.09100

171 rows × 2 columns

Hence predicted the Capacity_mw using the x_test feature columns.

In [173…

```
df_comparison.to_csv('Global_Power_Plant_Capacity_mw_Regression_Prediction.csv')
```

Saving the predicted values in a csv file

5.2. Predicting "Fuel_Type" Target

Seperating the Dataset into Features and Label(Fuel_Type)

In [174…

```
x_df = new_df.drop("Fuel_Type", axis=1)
y_df = new_df["Fuel_Type"]
```

In [175…

```
x_df.shape
```

Out[175…

```
(851, 10)
```

In [176…

```
y_df.shape
```

Out[176…

```
(851,)
```

Checking the skewness

In [177…

```
x_df.skew().sort_values()
```

Out[177…

```
geolocation_source    -2.066536
longitude              0.945877
Power_plant_age        1.280800
source                 1.734252
capacity_mw            2.170245
generation_gwh_2017    2.546541
generation_gwh_2018    2.597029
generation_gwh_2016    2.645786
generation_gwh_2015    2.714999
generation_gwh_2014    2.943026
dtype: float64
```

We can see that there are skewness in most of the columns

Removing the skewness

In [178…

```
skew = ['capacity_mw','longitude','generation_gwh_2014','generation_gwh_2015','generation_gwh_2016','generation_gwh_2017','generation_gwh_2018','Powe

from sklearn.preprocessing import PowerTransformer
transfo = PowerTransformer(method='yeo-johnson')
```

transforming all the numerical columns apart from categorically encoded columns

In [179…

```
x_df[skew] = transfo.fit_transform(x_df[skew].values)
x_df[skew].head()
```

Out[179…

	capacity_mw	longitude	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_2018	Power_plant_age
0	-1.677389	-0.922012	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	-1.081421
1	0.220284	-0.499829	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	0.046187
2	-0.274381	-2.377759	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	0.046187
3	0.391670	2.430594	-0.268922	0.093773	0.105691	-0.199692	-0.194159	-0.245810
4	1.731859	1.261979	1.426798	2.286603	2.276671	1.983083	2.347272	-1.758384

In [180…

```
x_df.skew().sort_values()
```

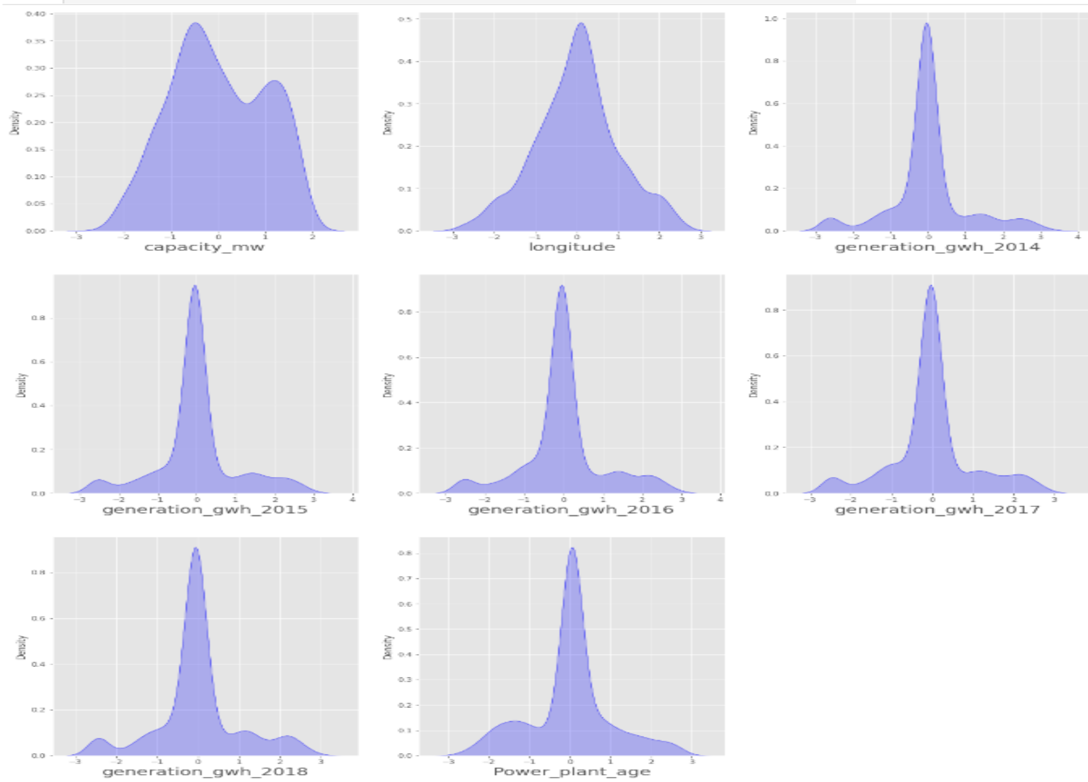
Out[180…

```
geolocation_source    -2.066536
longitude             -0.000128
capacity_mw            0.016303
Power_plant_age        0.043734
generation_gwh_2017    0.127152
generation_gwh_2018    0.133691
generation_gwh_2016    0.147035
generation_gwh_2015    0.163587
generation_gwh_2014    0.232399
source                 1.734252
dtype: float64
```

Distribustion of all the feature

In [181…

```
#Lets visualize the data

plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in x_df[skew]:
    if plotnumber<=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(x_df[column],color='b',kde_kws={"shade": True},hist=False)
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



The dataset looks normally distributed now

Feature Scaling

In [182…

```
scaler=StandardScaler()
x_df=pd.DataFrame(scaler.fit_transform(x_df),columns=x_df.columns)
x_df
```

Out[182…

	capacity_mw	longitude	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_2018	Powe
0	-1.677389	-0.922012	1.397951	-1.036523	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
1	0.220284	-0.499829	2.821796	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
2	-0.274381	-2.377759	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
3	0.391670	2.430594	-0.507812	0.407145	-0.268922	0.093773	0.105691	-0.199692	-0.194159	
4	1.731859	1.261979	-0.507812	0.407145	1.426798	2.286603	2.276671	1.983083	2.347272	
...	...	...	...	...	...	...	...	...	...	
846	1.672840	0.231932	-0.507812	0.407145	-0.044061	-2.461379	-0.842266	0.010837	-0.126054	
847	-1.598186	0.421592	0.696980	-2.480190	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
848	-0.507278	-0.224400	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
849	0.111201	-0.760624	0.302685	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
850	-0.741846	0.260758	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	

851 rows × 10 columns

We have scaled the dataset.

Checking Multicolinearity

In [183…

```
vif = pd.DataFrame()
vif["VIF values"] = [variance_inflation_factor(x_df.values,i)
              for i in range(len(x_df.columns))]
vif["Features"] = x_df.columns

# Let's check the values
vif
```

Out[183…

	VIF values	Features
0	1.811775	capacity_mw
1	1.193670	longitude
2	1.413037	source
3	1.590869	geolocation_source
4	3.621608	generation_gwh_2014
5	6.190754	generation_gwh_2015
6	9.961303	generation_gwh_2016
7	9.767170	generation_gwh_2017
8	8.961146	generation_gwh_2018
9	1.153813	Power_plant_age

All the columns has vif values less then 10, hence there is no multicolinearity that exist.

In [184…

```
y_df.value_counts()
```

Out[184…

```
1    238
3    220
6    126
7    123
2     65
0     50
5     20
4      9
Name: Fuel_Type, dtype: int64
```

We can see that the target Fuel_Type has multiple classes in the mode of energy source, hence we can see that this is a multi classification problem. As the data between the classes are not balanced with 1 having 238 counts and 4 having only 9 counts, we have to do SMOTE oversampling of the data.

SMOTE OverSampling

In [185…

```
from imblearn.over_sampling import SMOTE
sm = SMOTE()
x_df, y_df = sm.fit_resample(x_df,y_df)
```

In [186…

```
y_df.value_counts()
```

Out[186…

```
6    238
1    238
7    238
2    238
3    238
0    238
5    238
4    238
Name: Fuel_Type, dtype: int64
```

Here we can see that the data imbalance has been removed.

In [206…

```
X = x_df   # renaming the features variable
```

In [207…

```
X
```

Out[207…

	capacity_mw	longitude	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_2018	Pow
0	-1.677389	-0.922012	1.397951	-1.036523	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
1	0.220284	-0.499829	2.821796	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
2	-0.274381	-2.377759	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
3	0.391670	2.430594	-0.507812	0.407145	-0.268922	0.093773	0.105691	-0.199692	-0.194159	
4	1.731859	1.261979	-0.507812	0.407145	1.426798	2.286603	2.276671	1.983083	2.347272	
...	...	...	...	...	...	...	...	...	...	
1899	-0.403015	-1.810411	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
1900	-0.289259	-2.425787	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
1901	-1.878258	0.229776	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
1902	-0.638451	-0.734827	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	
1903	-0.424856	-1.807903	-0.529717	0.407145	-0.044061	-0.049141	-0.046103	-0.035226	-0.057181	

1904 rows × 10 columns

In [208…

```
Y = y_df   # renaming the target variable
```

In [209…

```
Y
```

Out[209…

```
0       6
1       1
2       7
3       2
4       1
       ..
1899    7
1900    7
1901    7
1902    7
1903    7
Name: Fuel_Type, Length: 1904, dtype: int32
```

Finding best random state

In [211…

```
maxAccu=0
maxRS=0

for i in range(1,200):
    x_train,x_test, y_train, y_test=train_test_split(X,Y,test_size=.20, random_state=i)
    rfc=RandomForestClassifier()
    rfc.fit(x_train,y_train)
    pred=rfc.predict(x_test)
    acc=accuracy_score(y_test,pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.952755905511811  on Random_state  142
```

Hence we get best accuracy score as 0.952 at Random_state 142 in RandomForestClassifier

train_test_split

In [212…

```
x_train,x_test, y_train, y_test=train_test_split(X,Y,test_size=.20, random_state=142)
```

In [219…

```
# Importing required libraries
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

In [214…

```
# creating a function to run all the classifiers

def classifier(model, X, Y):
    x_train,x_test,y_train,y_test = train_test_split(X, Y, test_size=0.2, random_state=142)

    # Training the model
    model.fit(x_train, y_train)

    # Predicting y_test
    pred = model.predict(x_test)

    # Accuracy Score
    acc_score = (accuracy_score(y_test, pred))*100
    print("Accuracy Score:", acc_score)

    # Classification Report
    class_report = classification_report(y_test, pred)
    print("\nClassification Report:\n", class_report)

    # Cross Validation Score
    cv_score = (cross_val_score(model, X, Y, cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    # Result of accuracy minus cv scores
    result = acc_score - cv_score
    print("\nAccuracy Score - Cross Validation Score is", result)
```

Machine Learning Algorithm

Logistic Regression

In [217…

```
model = LogisticRegression()
classifier(model, X, Y)
```

```
Accuracy Score: 73.75328083989501

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.84      0.83        49
           1       0.46      0.32      0.38        37
           2       0.58      0.43      0.49        42
           3       0.56      0.49      0.52        47
           4       0.68      0.87      0.76        39
           5       0.81      0.86      0.83        50
           6       1.00      0.98      0.99        63
           7       0.71      0.89      0.79        54

    accuracy                           0.74       381
   macro avg       0.70      0.71      0.70       381
weighted avg       0.72      0.74      0.73       381

Cross Validation Score: 72.68821660450338

Accuracy Score - Cross Validation Score is 1.065064235391631
```

Naive Bayes

In [220…

```
model = GaussianNB()
classifier(model, X, Y)
```

```
Accuracy Score: 63.77952755905512

Classification Report:
              precision    recall  f1-score   support

           0       0.59      1.00      0.74        49
           1       0.60      0.08      0.14        37
           2       0.20      0.02      0.04        42
           3       0.33      0.77      0.46        47
           4       0.71      0.56      0.63        39
           5       0.94      0.30      0.45        50
           6       0.82      1.00      0.90        63
           7       1.00      1.00      1.00        54

    accuracy                           0.64       381
   macro avg       0.65      0.59      0.55       381
weighted avg       0.67      0.64      0.59       381

Cross Validation Score: 59.9780356402818

Accuracy Score - Cross Validation Score is 3.8014919187733156
```

SVC Classifier

In [221…

```
model = SVC(kernel='rbf')
classifier(model, X, Y)
```

```
Accuracy Score: 83.2020997375328

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.82      0.80        49
           1       0.77      0.46      0.58        37
           2       0.74      0.76      0.75        42
           3       0.91      0.64      0.75        47
           4       0.80      1.00      0.89        39
           5       0.92      0.88      0.90        50
           6       1.00      1.00      1.00        63
           7       0.72      0.96      0.83        54

    accuracy                           0.83       381
   macro avg       0.83      0.81      0.81       381
weighted avg       0.84      0.83      0.83       381

Cross Validation Score: 80.46152783533638

Accuracy Score - Cross Validation Score is 2.7405719021964217
```

In [222…

```
model = SVC(kernel='linear')
classifier(model, X, Y)
```

```
Accuracy Score: 79.00262467191601

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        49
           1       0.54      0.41      0.46        37
           2       0.56      0.55      0.55        42
           3       0.64      0.60      0.62        47
           4       0.80      1.00      0.89        39
           5       0.98      0.80      0.88        50
           6       1.00      1.00      1.00        63
           7       0.83      0.91      0.87        54

    accuracy                           0.79       381
   macro avg       0.77      0.77      0.76       381
weighted avg       0.79      0.79      0.78       381

Cross Validation Score: 78.25652714463322

Accuracy Score - Cross Validation Score is 0.7460975272827852
```

In [223…

```
model = SVC(kernel='poly')
classifier(model, X, Y)
```

```
Accuracy Score: 69.02887139107612

Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.55      0.63        49
           1       0.94      0.46      0.62        37
           2       0.65      0.48      0.55        42
           3       0.88      0.49      0.63        47
           4       0.94      0.87      0.91        39
           5       0.97      0.58      0.72        50
           6       1.00      0.94      0.97        63
           7       0.38      1.00      0.55        54

    accuracy                           0.69       381
   macro avg       0.81      0.67      0.70       381
weighted avg       0.81      0.69      0.70       381

Cross Validation Score: 65.70272137035502

Accuracy Score - Cross Validation Score is 3.3261500207211014
```

Decision Tree Classifier

In [224…

```
model = DecisionTreeClassifier()
classifier(model, X, Y)
```

```
Accuracy Score: 91.33858267716536

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        49
           1       0.64      0.68      0.66        37
           2       0.82      0.88      0.85        42
           3       0.95      0.83      0.89        47
           4       0.88      0.92      0.90        39
           5       0.96      0.92      0.94        50
           6       1.00      1.00      1.00        63
           7       1.00      1.00      1.00        54

    accuracy                           0.91       381
   macro avg       0.90      0.90      0.90       381
weighted avg       0.92      0.91      0.91       381

Cross Validation Score: 87.13095731454621

Accuracy Score - Cross Validation Score is 4.207625362619154
```

KNeighbors Classifier

In [225…

```
model = KNeighborsClassifier()
classifier(model, X, Y)
```

```
Accuracy Score: 87.4015748031496

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.98      0.90        49
           1       0.73      0.43      0.54        37
           2       0.71      0.95      0.82        42
           3       0.94      0.62      0.74        47
           4       0.86      0.97      0.92        39
           5       0.94      0.90      0.92        50
           6       1.00      1.00      1.00        63
           7       0.92      1.00      0.96        54

    accuracy                           0.87       381
   macro avg       0.87      0.86      0.85       381
weighted avg       0.88      0.87      0.86       381

Cross Validation Score: 85.39908827186075

Accuracy Score - Cross Validation Score is 2.0024865312888522
```

Random Forest Classifier

In [231…

```
model = RandomForestClassifier(random_state=142)
classifier(model, X, Y)
```

```
Accuracy Score: 95.01312335958005

Classification Report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        49
           1       0.90      0.73      0.81        37
           2       0.93      0.98      0.95        42
           3       0.91      0.89      0.90        47
           4       0.93      0.97      0.95        39
           5       0.92      0.96      0.94        50
           6       1.00      1.00      1.00        63
           7       1.00      1.00      1.00        54

    accuracy                           0.95       381
   macro avg       0.94      0.94      0.94       381
weighted avg       0.95      0.95      0.95       381

Cross Validation Score: 91.59649122807018

Accuracy Score - Cross Validation Score is 3.4166321315098713
```

In [232…

```
model = RandomForestClassifier()
classifier(model, X, Y)
```

```
Accuracy Score: 95.01312335958005

Classification Report:
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        49
           1       0.83      0.81      0.82        37
           2       0.91      0.93      0.92        42
           3       0.93      0.85      0.89        47
           4       0.97      1.00      0.99        39
           5       0.94      0.96      0.95        50
           6       1.00      1.00      1.00        63
           7       1.00      1.00      1.00        54

    accuracy                           0.95       381
   macro avg       0.94      0.94      0.94       381
weighted avg       0.95      0.95      0.95       381

Cross Validation Score: 91.5445503522586

Accuracy Score - Cross Validation Score is 3.468573007321453
```

ExtraTrees Classifier

In [228…

```
model = ExtraTreesClassifier()
classifier(model, X, Y)
```

```
Accuracy Score: 94.22572178477691

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95        49
           1       0.87      0.70      0.78        37
           2       0.91      0.95      0.93        42
           3       0.91      0.87      0.89        47
           4       0.95      1.00      0.97        39
           5       0.94      0.96      0.95        50
           6       1.00      1.00      1.00        63
           7       0.98      1.00      0.99        54

    accuracy                           0.94       381
   macro avg       0.94      0.93      0.93       381
weighted avg       0.94      0.94      0.94       381

Cross Validation Score: 92.01657687525902

Accuracy Score - Cross Validation Score is 2.2091449095178888
```

AdaBoost Classifier

In [229…

```
model = AdaBoostClassifier()
classifier(model, X, Y)
```

```
Accuracy Score: 29.133858267716533

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        49
           1       0.17      0.38      0.23        37
           2       0.00      0.00      0.00        42
           3       0.00      0.00      0.00        47
           4       0.16      0.95      0.27        39
           5       0.00      0.00      0.00        50
           6       1.00      0.95      0.98        63
           7       0.00      0.00      0.00        54

    accuracy                           0.29       381
   macro avg       0.17      0.28      0.18       381
weighted avg       0.20      0.29      0.21       381

Cross Validation Score: 27.888796795137445

Accuracy Score - Cross Validation Score is 1.2450614725790885
```

Gradient Boosting Classifier

In [230…

```
model = GradientBoostingClassifier()
classifier(model, X, Y)
```

```
Accuracy Score: 92.91338582677166

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96        49
           1       0.79      0.73      0.76        37
           2       0.88      0.90      0.89        42
           3       0.91      0.83      0.87        47
           4       0.90      0.95      0.92        39
           5       0.92      0.96      0.94        50
           6       1.00      1.00      1.00        63
           7       1.00      1.00      1.00        54

    accuracy                           0.93       381
   macro avg       0.92      0.92      0.92       381
weighted avg       0.93      0.93      0.93       381

Cross Validation Score: 90.23083298798177

Accuracy Score - Cross Validation Score is 2.6825528387898885
```

Comparing all the above the ExtraTreesClassifier gives the best results since the Accuracy Score - Cross Validation Score is the least along with higher Cross Validation Score and the highest Accuracy Score comparing all the models.

Hyper Parameter Tuning

In [233…
```
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=.20,random_state=142)
```

In [ ]:
```
#ExtraTreesClassifier?
```

In [234…
```
# creating parameters list to pass into GridSearchCV

parameters = {'criterion' : ['gini', 'entropy'],
              'max_features' : ['auto', 'sqrt', 'log2'],
              'n_jobs' : [5, 10, 15]}
```

In [235…
```
GCV = GridSearchCV(ExtraTreesClassifier(), parameters, cv=5)
```

In [236…
```
GCV.fit(x_train,y_train)
```

Out[236…
```
GridSearchCV(cv=5, estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_jobs': [5, 10, 15]})
```

In [237…
```
GCV.best_params_      # printing best parameters found by GridSearchCV
```

Out[237…
```
{'criterion': 'gini', 'max_features': 'log2', 'n_jobs': 5}
```

We got the best parameters using Gridsearch CV

In [238…
```
final_modelc = ExtraTreesClassifier(criterion = 'gini', max_features = 'log2', n_jobs = 5)   # final model with best parameters
```

In [239…
```
final_fitc = final_modelc.fit(x_train,y_train)   # final fit
```

In [240…
```
final_predc = final_modelc.predict(x_test)   # predicting with best parameters
```

In [241…
```
best_acc_score = (accuracy_score(y_test, final_predc))*100    # checking accuracy score
print("The Accuracy Score for the Best Model is ", best_acc_score)
```

```
The Accuracy Score for the Best Model is  94.48818897637796
```

We successfully performed the Hyper Parameter Tuning on the Final Model.

In [242…
```
# Final Cross Validation Score
final_cv_score = (cross_val_score(final_modelc, X, Y, cv=5).mean())*100
print("Cross Validation Score:", final_cv_score)
```

```
Cross Validation Score: 92.12211631440807
```

We got final accuracy score of 94.488% and Cross Validation Score of 92.12% which is good

We successfully performed the Hyper Parameter Tuning on the Final Model.

Cross validation Score

In [243…
```
x_test.shape
```
Out[243… (381, 10)

In [244…
```
y_test.shape
```
Out[244… (381,)

In [245…
```
x_train.shape
```
Out[245… (1523, 10)

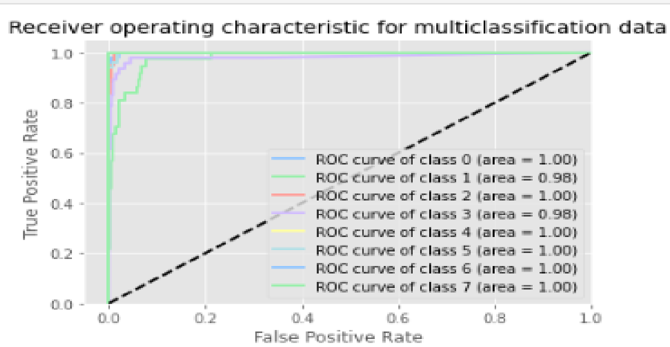
In [246…
```
y_train.shape
```
Out[246… (1523,)

In [247…
```
# Final Classification Report
final_class_report = classification_report(y_test, final_predc)
print("\nClassification Report:\n", final_class_report)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.98      0.97        49
           1       0.86      0.65      0.74        37
           2       0.91      0.98      0.94        42
           3       0.90      0.91      0.91        47
           4       0.95      1.00      0.97        39
           5       0.94      0.96      0.95        50
           6       1.00      1.00      1.00        63
           7       0.98      1.00      0.99        54

    accuracy                           0.94       381
   macro avg       0.94      0.93      0.93       381
weighted avg       0.94      0.94      0.94       381
```

In [249…
```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier
```

In [253…
```
classifier = OneVsRestClassifier(final_modelc)
y_score = classifier.fit(x_train, y_train).predict_proba(x_test)

#Binarize the output
y_test_bin  = label_binarize(y_test, classes=[0,1,2,3,4,5,6,7])
n_classes = 8

# Compute ROC curve and AUC for all the classes
false_positive_rate = dict()
true_positive_rate = dict()
roc_auc = dict()
for i in range(n_classes):
    false_positive_rate[i], true_positive_rate[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(false_positive_rate[i], true_positive_rate[i])


for i in range(n_classes):
    plt.plot(false_positive_rate[i], true_positive_rate[i], lw=2,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic for multiclassification data')
plt.legend(loc="lower right")
plt.show()
```



Saving the model in pickle Format

In [254…

```
# pickeling or serialization of a file
import pickle
filenamec = 'Global_Power_Plant_Fuel_Type_Classification_final_model.pkl'
pickle.dump(final_modelc, open(filenamec, 'wb'))
```

Prediction Conclusion:

In [255…

```
import numpy as np
ac=np.array(y_test)
predictedc=np.array(final_modelc.predict(x_test))
df_comparisonc = pd.DataFrame({"original":ac,"predicted":predictedc},index= range(len(ac)))
df_comparisonc
```

Out[255…

	original	predicted
0	0	0
1	5	5
2	5	5
3	2	2
4	7	7
...	...	...
376	6	6
377	7	7
378	0	0
379	4	4
380	7	7

381 rows × 2 columns

Hence predicted the 'Fuel_Type' using the x_test feature columns.

In [256…

```
df_comparisonc.to_csv('Global_Power_Plant_Fuel_Type_Classification_Prediction.csv')
```

Saving the predicted values in a csv file

----------------------- Thank You----------------------