Finding Two Targets In Global Power Plant Database Using Machine Learning Models

INTRODUCTION

Problem Definition:

Description

The Power Plants produce the electrical energy from another form of energy. As electricity has become necessary part of our life like food, shelter and clothes so the electricity generation has also gain so much importance in the last few years.

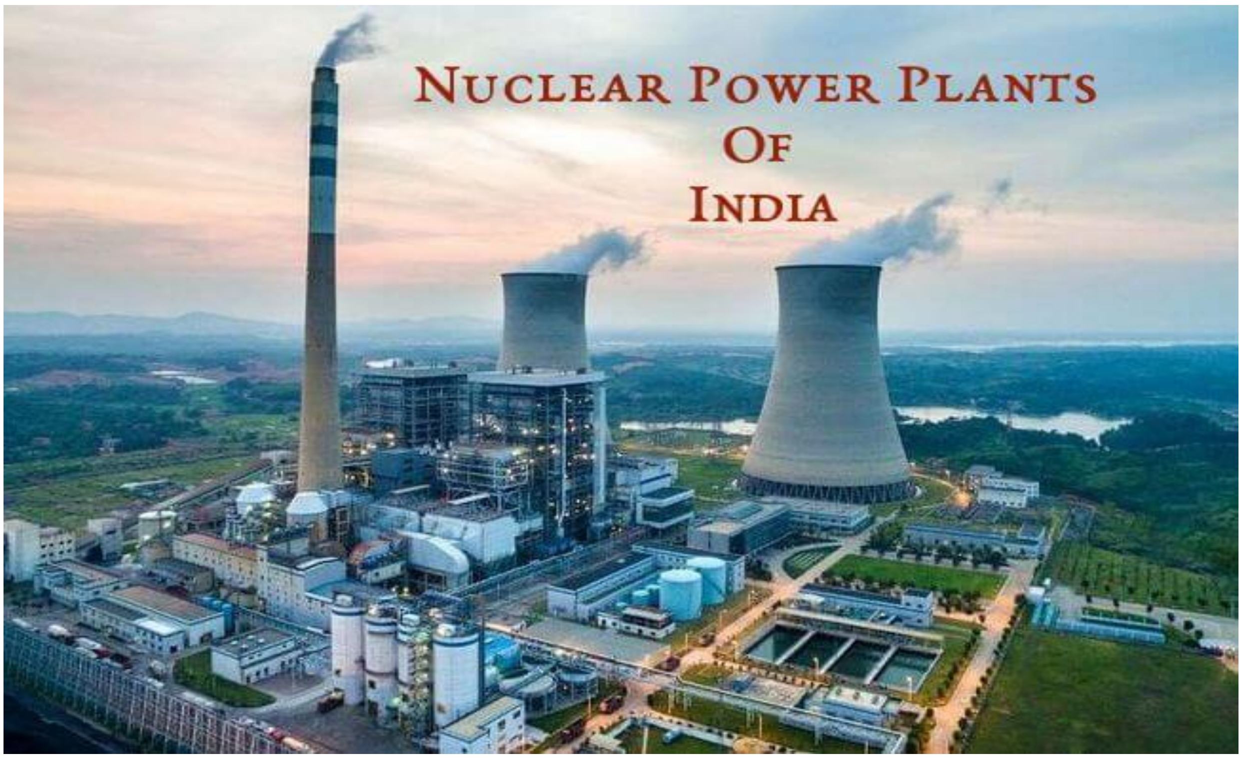


Fig.1(a) Power Plant

There are a lot of ways to produce electrical energy. To generate electricity, the power station requires another source of energy. One source of energy is heat energy, in which we find fossil fuels like coal, natural gas, oil etc, solar thermal energy, geothermal energy. Another source of energy is renewable energy like wind, solar, wave and hydroelectric.

Then there are nuclear power plants and then there is another form which uses potential energy from falling water in a hydroelectric facility/. Also there is a category which uses chemical energy from fuel cells and batteries to produce electrical energy and many more.

Many power stations contain one or more generators, a rotating machine that converts mechanical power into three-phase electric power. The relative motion between a magnetic field and a conductor creates an electric current.

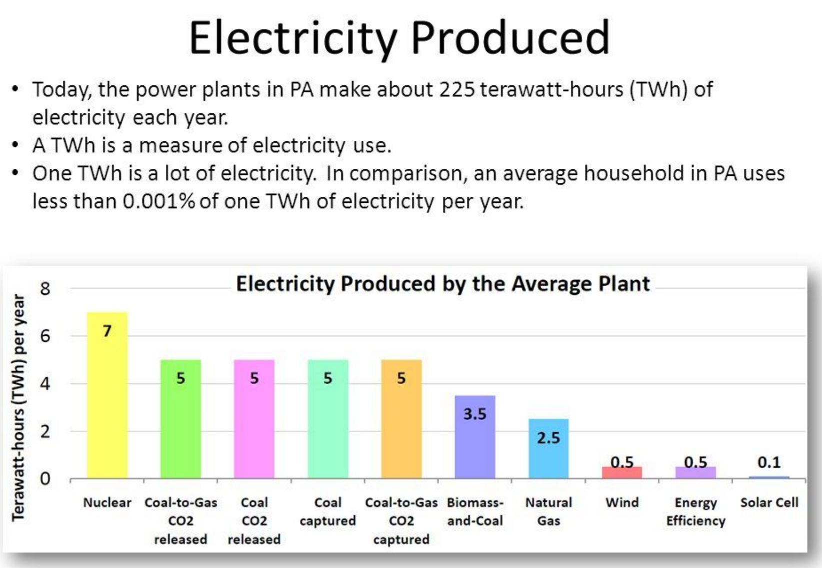


Fig. 1(b). Electricity Production by different types of Power Plant.

ABOUT DATASET

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. It will be continuously updated as data becomes available.

Key attributes of the database

The database includes the following indicators:

- `country`: 3character country code corresponding to the ISO 3166-1 alpha-3 specification
- `country_long`: longer form of the country designation
- `name`: name or title of the power plant, generally in Romanized form
- `gppd_idnr`: 10 or 12 character identifier for the power plant
- `capacity_mw`: electrical generating capacity in megawatts
- `latitude`: geolocation in decimal degrees; WGS84 (EPSG:4326)
- `longitude`: geolocation in decimal degrees; WGS84 (EPSG:4326)
- `primary_fuel`: energy source used in primary electricity generation or export
- `other_fuel1`: energy source used in electricity generation or export
- `other_fuel2`: energy source used in electricity generation or export
- `other_fuel3`: energy source used in electricity generation or export
- `commissioning_year`: year of plant operation, weighted by unit-capacity when data is available
- `owner`: majority shareholder of the power plant, generally in Romanized form
- `source`: entity reporting the data; could be an organization, report, or document, generally in Romanized form
- `url`: web document corresponding to the `source` field
- `geolocation_source` : attribution for geolocation information
- `wepp_id` : a reference to a unique plant identifier in the widely-used PLATTS-WEPP database.
- `year_of_capacity_data`: year the capacity information was reported
- `generation_gwh_2013`: electricity generation in gigawatt-hours reported for the year 2013
- `generation_gwh_2014`: electricity generation in gigawatt-hours reported for the year 2014
- `generation_gwh_2015`: electricity generation in gigawatt-hours reported for the year 2015
- `generation_gwh_2016`: electricity generation in gigawatt-hours reported for the year 2016
- `generation_gwh_2017`: electricity generation in gigawatt-hours reported for the year 2017
- `generation_gwh_2018`: electricity generation in gigawatt-hours reported for the year 2018
- `generation_gwh_2019`: electricity generation in gigawatt-hours reported for the year 2019
- `generation_data_source`: attribution for the reported generation information
- `estimated_generation_gwh_2013`: estimated electricity generation in gigawatt-hours for the year 2013
- `estimated_generation_gwh_2014`: estimated electricity generation in gigawatt-hours for the year 2014
- `estimated_generation_gwh_2015`: estimated electricity generation in gigawatt-hours for the year 2015

This dataset contains 907 rows and 27 columns. This dataset is containing all the values related to INDIAN power plants.

Task:

The task is to predict two targets here:

1). One is **'primary_fuel'** which is a categorical type of target and hence, it becomes a classification problem.

2). Second is **'capacity_mw'** which is a continuous type of target and hence, it becomes a regression problem.

Methodology Used:

As there are two target variables in the dataset, I developed the model using the methodology of choosing one target variable as a feature while predicting the other and then again checking its importance in predicting the other before finally feeding this to the model for training purpose. If it holds some importance after verifying it from the scores, then I continued with that target to be considered as a feature in predicting the other or deleted otherwise.

DATA ANALYSIS

Libraries Used

- Pandas
- Numpy
- Matplotlib
- Seaborn

Data Cleaning

```
In [2]: df=pd.read_csv(r'https://raw.githubusercontent.com/wri/global-power-plant-database/master/source_databases_csv/database_IND.csv')
        pd.options.display.max_columns=None
        df
```

Out[2]:

	country	country_long	name	gppd_idnr	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	other_fuel2	other_fuel3	commissioning_year
0	IND	India	ACME Solar Tower	WRI1020239	2.5	28.1839	73.2407	Solar	NaN	NaN	NaN	2011.0
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	NaN	2004.0
4	IND	India	AKALTARA TPP	IND0000002	1800.0	21.9603	82.4091	Coal	Oil	NaN	NaN	2015.0
...	...	...	...	...	...	...	...	...	...	...	...	...
902	IND	India	YERMARUS TPP	IND0000513	1600.0	16.2949	77.3568	Coal	Oil	NaN	NaN	2016.0
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

Screenshot 1. Dataset

After downloading it, I firstly checked the nan values and got that there were lot of null values in many features as shown in the Screenshot 2:

```
In [4]: df.isna().sum()

Out[4]: country                     0
        country_long                0
        name                        0
        gppd_idnr                   0
        capacity_mw                 0
        latitude                   46
        longitude                  46
        primary_fuel                0
        other_fuel1               709
        other_fuel2               906
        other_fuel3               907
        commissioning_year        380
        owner                     565
        source                      0
        url                         0
        geolocation_source         19
        wepp_id                   907
        year_of_capacity_data     388
        generation_gwh_2013       907
        generation_gwh_2014       509
        generation_gwh_2015       485
        generation_gwh_2016       473
        generation_gwh_2017       467
        generation_gwh_2018       459
        generation_gwh_2019       907
        generation_data_source    458
        estimated_generation_gwh  907
        dtype: int64
```

Screenshot 2. Nan values.

Also. I got that there were many columns which had nan values for more than 900 records. So, I deleted those columns as there was no use of these columns and got 6 columns deleted (Screenshot 3).

In [6]:
```python
for i in df:
    if df[i].isna().sum()>900:
        df=df.drop(i,axis=1)
df
```

Out[6]:

	country	country_long	name	gppd_idnr	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	owner	source
0	IND	India	ACME Solar Tower	WRI1020239	2.5	28.1839	73.2407	Solar	NaN	2011.0	Solar Paces	National Renewable Energy Laboratory
1	IND	India	ADITYA CEMENT WORKS	WRI1019881	98.0	24.7663	74.6090	Coal	NaN	NaN	Ultratech Cement ltd	Ultratech Cement ltd
2	IND	India	AES Saurashtra Windfarms	WRI1026669	39.2	21.9038	69.3732	Wind	NaN	NaN	AES	CDM
3	IND	India	AGARTALA GT	IND0000001	135.0	23.8712	91.3602	Gas	NaN	2004.0	NaN	Central Electricity Authority
4	IND	India	AKALTARA TPP	IND0000002	1800.0	21.9603	82.4091	Coal	Oil	2015.0	NaN	Central Electricity Authority
...	...	...	...	...	...	...	...	...	...	...	...	...
902	IND	India	YERMARUS TPP	IND0000513	1600.0	16.2949	77.3568	Coal	Oil	2016.0	NaN	Central Electricity Authority
903	IND	India	Yelesandra Solar Power Plant	WRI1026222	3.0	12.8932	78.1654	Solar	NaN	NaN	Karnataka Power Corporation Limited	Karnataka Power Corporation Limited
904	IND	India	Yelisirur wind power project	WRI1026776	25.5	15.2758	75.5811	Wind	NaN	NaN	NaN	CDM
905	IND	India	ZAWAR MINES	WRI1019901	80.0	24.3500	73.7477	Coal	NaN	NaN	Hindustan Zinc ltd	Hindustan Zinc ltd
906	IND	India	iEnergy Theni Wind Farm	WRI1026761	16.5	9.9344	77.4768	Wind	NaN	NaN	iEnergy Wind Farms	CDM

907 rows × 21 columns

Screenshot 3.

Then on checking the count_values for each column, I got that features country, country-long, year_of_capacity_data and generation_data_source have one particular or same value for all of the rows. So, I deleted these features.

Also, name, gppd_ idnr and url are of no use, as the former two uses unique values for each row like id which doesn't hold importance in model development and url feature is also not important. So, I deleted these features and left with 14 features. (Screenshot 4 and 5)

```
In [8]: for i in df:
            print(f"for feature {i} \n{df[i].value_counts()} \n")
```

```
for feature generation_gwh_2018
0.000000          39
626.239128         1
505.420200         1
1098.450150        1
17.213500          1
                  ..
220.551700         1
7321.267900        1
6532.350000        1
15305.220000       1
686.500000         1
Name: generation_gwh_2018, Length: 410, dtype: int64

for feature generation_data_source
Central Electricity Authority    449
Name: generation_data_source, dtype: int64
```

Screenshot 4.

```
In [10]: df=df.drop(['country','country_long','year_of_capacity_data','generation_data_source','name','gppd_idnr','url'],axis=1)
         df
```

Out[10]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	owner	source	geolocation_source	generation_gwh_2014	genera
0	2.5	28.1839	73.2407	Solar	NaN	2011.0	Solar Paces	National Renewable Energy Laboratory	National Renewable Energy Laboratory	NaN	
1	98.0	24.7663	74.6090	Coal	NaN	NaN	Ultratech Cement ltd	Ultratech Cement ltd	WRI	NaN	
2	39.2	21.9038	69.3732	Wind	NaN	NaN	AES	CDM	WRI	NaN	
3	135.0	23.8712	91.3602	Gas	NaN	2004.0	NaN	Central Electricity Authority	WRI	617.789264	
4	1800.0	21.9603	82.4091	Coal	Oil	2015.0	NaN	Central Electricity Authority	WRI	3035.550000	
...	...	...	...	...	...	...	...	...	...	...	
902	1600.0	16.2949	77.3568	Coal	Oil	2016.0	NaN	Central Electricity Authority	WRI	NaN	
903	3.0	12.8932	78.1654	Solar	NaN	NaN	Karnataka Power Corporation Limited	Karnataka Power Corporation Limited	Industry About	NaN	
904	25.5	15.2758	75.5811	Wind	NaN	NaN	NaN	CDM	WRI	NaN	
905	80.0	24.3500	73.7477	Coal	NaN	NaN	Hindustan Zinc ltd	Hindustan Zinc ltd	WRI	NaN	
906	16.5	9.9344	77.4768	Wind	NaN	NaN	iEnergy Wind Farms	CDM	WRI	NaN	

907 rows × 14 columns

Screenshot 5.

Then I decided to impute the nan values with Knn Imputer. For that, I firstly categorized columns into categorical columns and continuous columns and then these categorical columns were encoded with LabelEncoder and after that, these were imputed as shown below (Screenshot 6, 7 and 8):

In [17]:
```python
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [18]:
```python
cont_cols=[]
cat_cols=[]
for i in df:
    if df[i].dtype=='object':
        cat_cols.append(i)
    else:
        cont_cols.append(i)
print(cat_cols)
print(cont_cols)
```

```
['primary_fuel', 'other_fuel1', 'owner', 'source', 'geolocation_source']
['capacity_mw', 'latitude', 'longitude', 'commissioning_year', 'generation_gwh_2014', 'generation_gwh_2015', 'generation_gwh_20
16', 'generation_gwh_2017', 'generation_gwh_2018']
```

In [19]:
```python
for i in cat_cols:
    if df[i].isna().sum() > 0:
        ind=[]
        ind = df[i].dropna()
        ind=le.fit_transform(ind)
        df[i].loc[df[i].notnull()]=ind
    else:
        df[i]=le.fit_transform(df[[i]])
df
```

Out[19]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	owner	source	geolocation_source	generation_gwh_2014	generation_gwh
0	2.5	28.1839	73.2407	6	NaN	2011.0	229	109	1	NaN	
1	98.0	24.7663	74.6090	1	NaN	NaN	258	174	2	NaN	
2	39.2	21.9038	69.3732	7	NaN	NaN	2	21	2	NaN	
3	135.0	23.8712	91.3602	2	NaN	2004.0	NaN	22	2	617.789264	843.7
4	1800.0	21.9603	82.4091	1	2	2015.0	NaN	22	2	3035.550000	5916.3
...	...	...	...	...	...	...	...	...	...	...	
902	1600.0	16.2949	77.3568	1	2	2016.0	NaN	22	2	NaN	0.9
903	3.0	12.8932	78.1654	6	NaN	NaN	114	77	0	NaN	

Screenshot 6.

In this way, imputation and encoding both were done for the datasets. I changed the datatype of commissioning_year to be integer as it was having some float values after imputation with Knn Imputer

```
for i in cat_cols:
    if df[i].isna().sum() > 0:
        print(df[i].value_counts())
        df[i]=knn.fit_transform(df[[i]])
        df[i]=df[i].astype(int)
df
```

```
2    195
1      2
0      1
Name: other_fuel1, dtype: int64
5      4
234    4
108    4
205    3
107    3
      ..
239    1
274    1
55     1
111    1
279    1
Name: owner, Length: 280, dtype: int64
2    765
0    119
1      4
Name: geolocation_source, dtype: int64
```

Out[20]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	owner	source	geolocation_source	generation_gwh_2014	generation_gwh
0	2.5	28.1839	73.2407	6	1	2011.000000	229	109	1	2431.823590	2428.2
1	98.0	24.7663	74.6090	1	1	1997.091082	258	174	2	2431.823590	2428.2
2	39.2	21.9038	69.3732	7	1	1997.091082	2	21	2	2431.823590	2428.2
3	135.0	23.8712	91.3602	2	1	2004.000000	140	22	2	617.789264	843.7
4	1800.0	21.9603	82.4091	1	2	2015.000000	140	22	2	3035.550000	5916.3
...	...	...	...	...	...	...	...	...	...	...	
902	1600.0	16.2949	77.3568	1	2	2016.000000	140	22	2	2431.823590	0.9
903	3.0	12.8932	78.1654	6	1	1997.091082	114	77	0	2431.823590	2428.2
904	25.5	15.2758	75.5811	7	1	1997.091082	140	21	2	2431.823590	2428.2
905	80.0	24.3500	73.7477	1	1	1997.091082	91	59	2	2431.823590	2428.2
906	16.5	9.9344	77.4768	7	1	1997.091082	279	21	2	2431.823590	2428.2

907 rows × 14 columns

Screenshot 7.

In [23]:
```
df.isna().sum()
```

Out[23]:
```
capacity_mw            0
latitude               0
longitude              0
primary_fuel           0
other_fuel1            0
commissioning_year     0
owner                  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
```

In [24]:
```
df.describe()
```

Out[24]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	owner	source	geolocation_source	generation_gwh_2014
count	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000
mean	326.223755	21.197918	77.464907	3.206174	1.213892	1997.052922	140.265711	43.847850	1.712238	2431.823590
std	590.085456	6.079148	4.812291	2.280652	0.412959	13.016438	49.807277	44.642818	0.684013	2665.338608
min	0.000000	8.168900	68.644700	0.000000	0.000000	1927.000000	0.000000	0.000000	0.000000	0.000000
25%	16.725000	17.072000	74.388900	1.000000	1.000000	1997.000000	140.000000	22.000000	2.000000	1211.362750
50%	59.200000	21.281800	76.979200	3.000000	1.000000	1997.000000	140.000000	22.000000	2.000000	2431.823590
75%	385.250000	25.176450	79.206100	6.000000	1.000000	2003.000000	140.000000	29.500000	2.000000	2431.823590
max	4760.000000	34.649000	95.408000	7.000000	2.000000	2018.000000	279.000000	190.000000	2.000000	28127.000000

Screenshot 8.

Exploratory Data Analysis (EDA)

Now, moving towards EDA part, let's check univariate analysis and bivariate analysis and based on them, let's take some decisions on outliers, skewness, correlation, feature importance etc.

Univariate Analysis:

1) Firstly, I checked distribution plots of the entire continuous type of columns of the dataset and got the following plot which clearly shows that there is lot of skewness in almost all continuous columns except latitude which shows somewhat less than the others. (Fig 2)

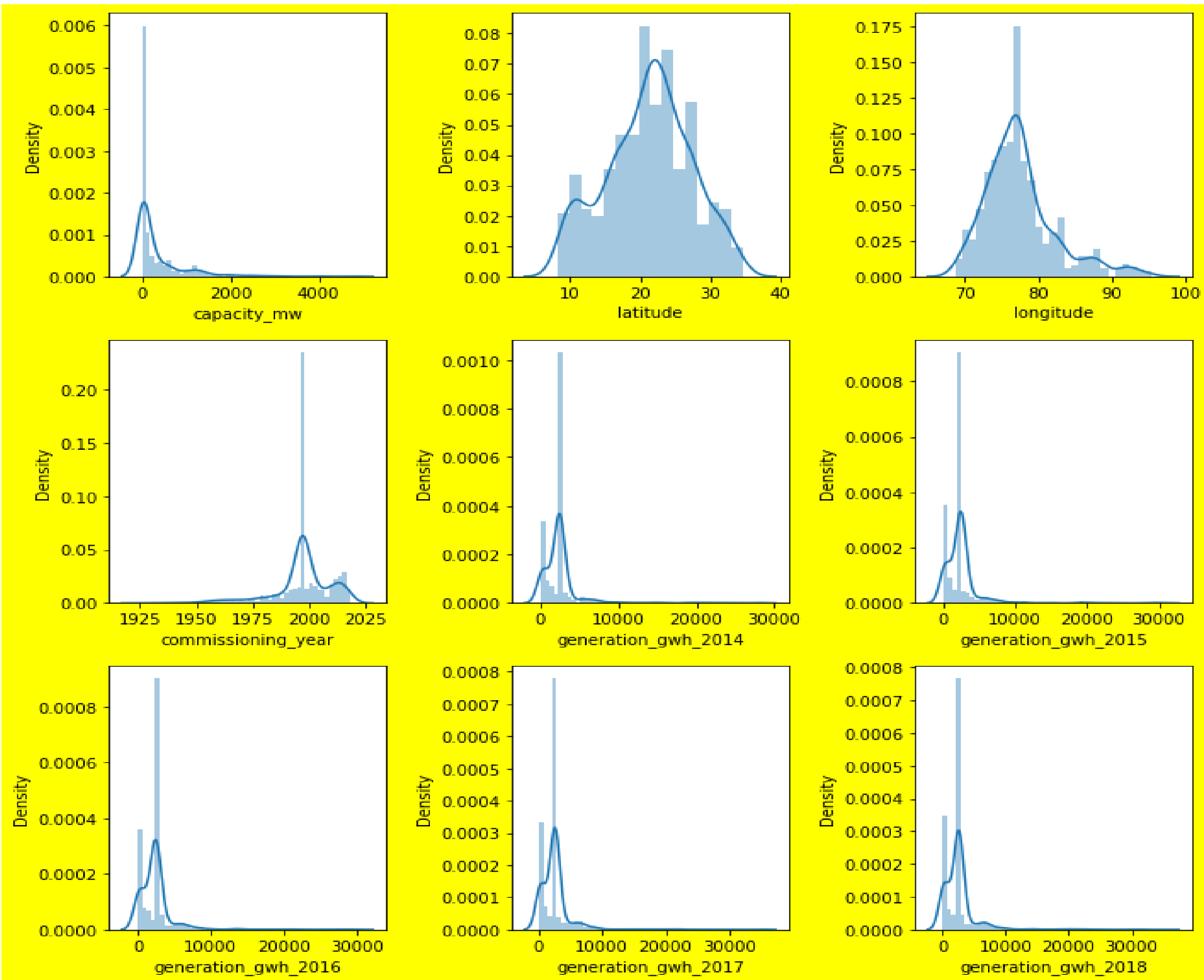


Fig 2. Distribution plot

2) Then, I plotted boxplots for all the continuous columns of the dataset (I am not saying these columns as features or target yet, there is some reason behind that which will be clarified further on dividing these columns into features and target for both of the problems). This plot showed that there were outliers present in almost all of the continuous columns except dataset which I will quantify later in the upcoming discussion.

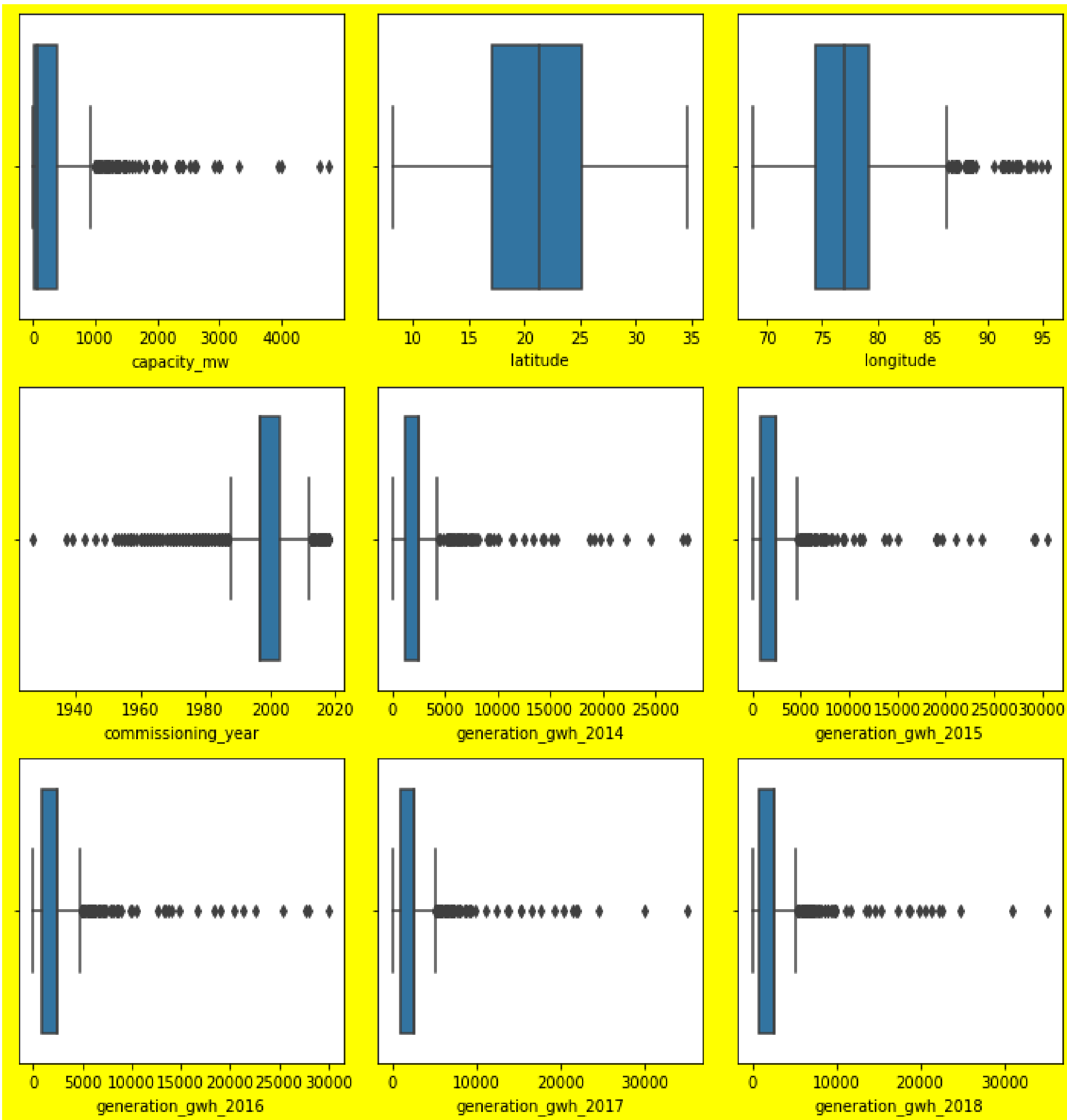


Fig 3. Box Plots

3) Then I plotted heatmap for correlation and found that generation_gwh_2018 and 2017 columns had a lot of skewness associated with them, also both of them had a lot of outliers. And moreover, they were highly correlated with each other. so, I deleted generation_gwh_2017 column from entire dataset (Fig 4).

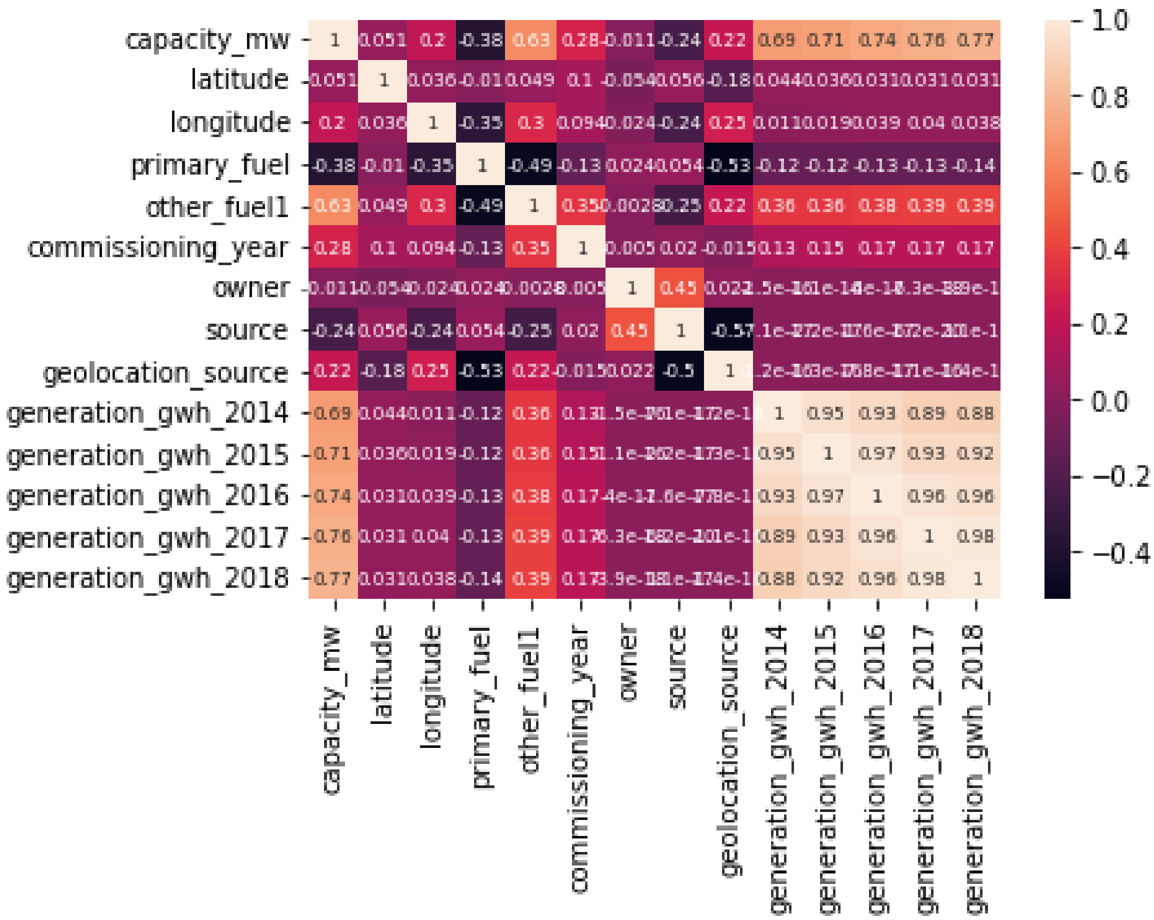


Fig 4. Correlation Heatmap 1.

Bivariate Analysis:

In this section, as there are two targets, so I plotted here two plots showing the relationship of each target with all other columns of the database. These are:

- Regplots of capacity_mw target vs all other columns (Fig. 5).

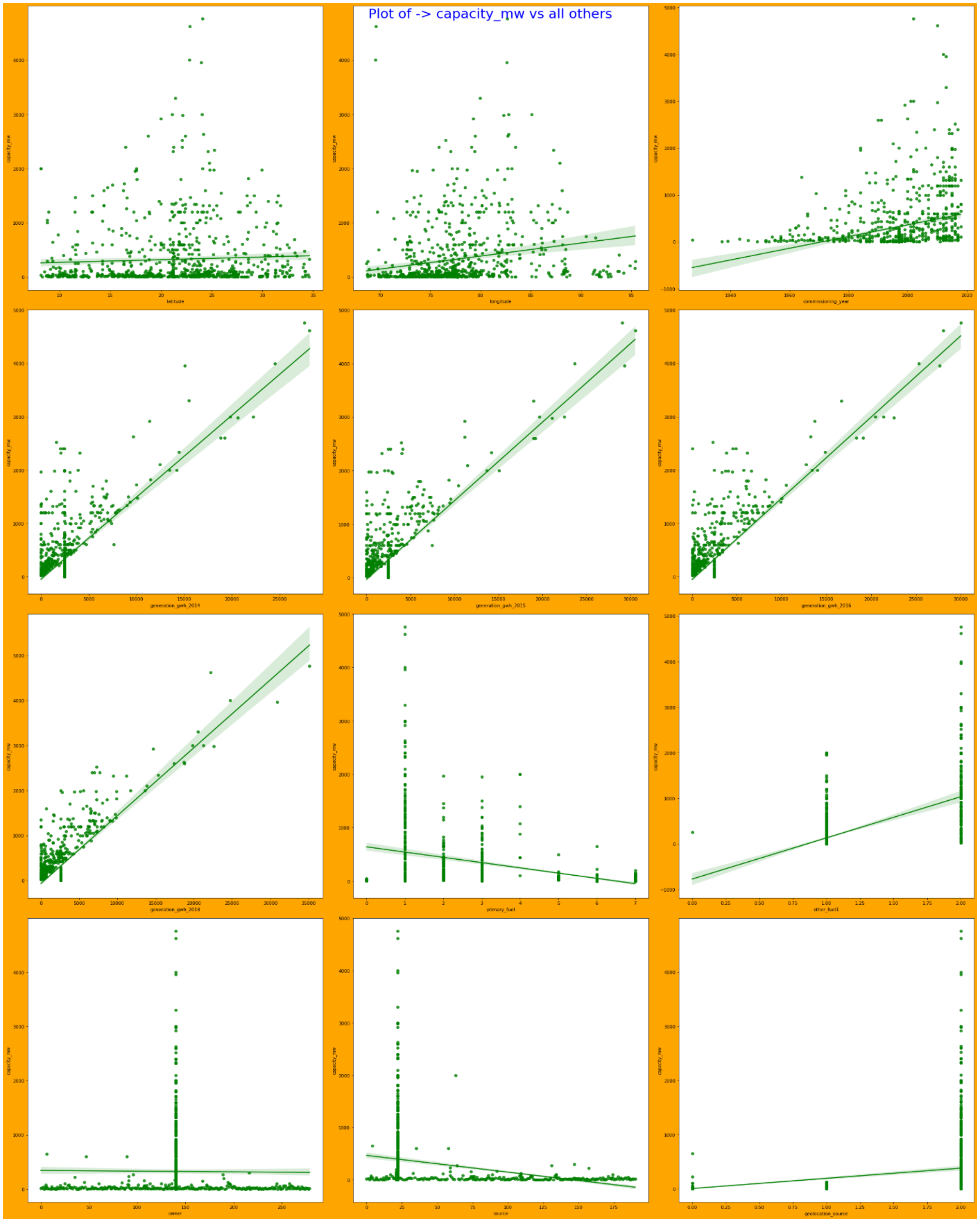


Fig 5. Regplots.

- Stripplots of primary_fuel vs all other columns. (Fig. 6.)

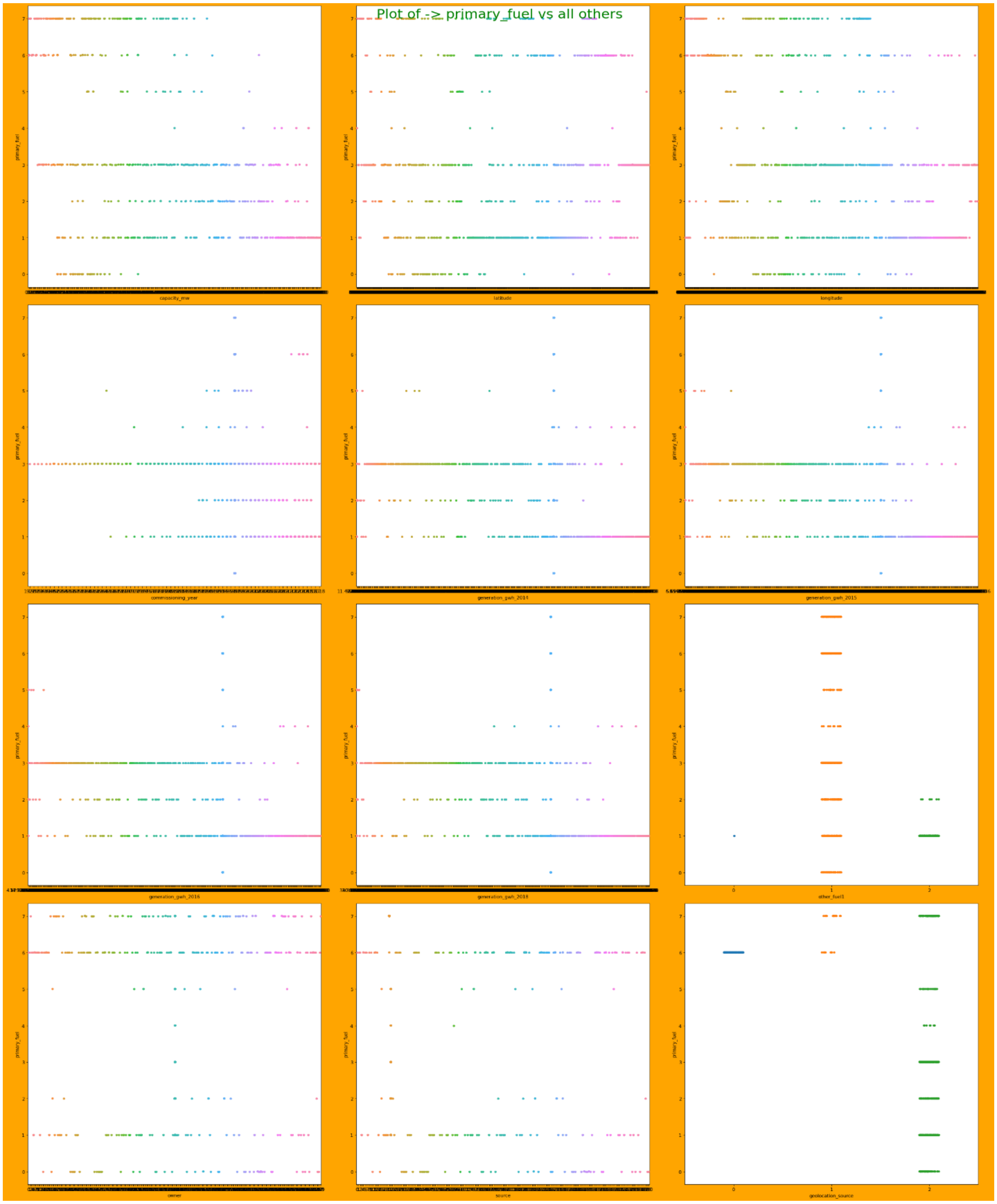


Fig. 6. Stripplots.

EDA Concluding Remark

- The distplots showed that there was lot of skewness associated with almost all continuous columns of database which was needed to be treated.
- The boxplots showed that there were lot of outliers present in each continuous columns which was needed to be removed.
- The correlation heatmap showed that there was high correlation between some of the columns of the database.
- The bivariate analysis of capacity_mw vs all other columns (or its features in this case) showed its relationship with each of the feature. It also showed that it has positive relation with some features and negative relation with some other.
- The bivariate analysis of primary_fuel vs its features showed that some features are very highly correlated with it and some other are very less correlated with it.

Pre-Processing Pipeline

A. Outliers Removal:

Firstly, I made two copies of my dataset for regression and classification purpose and checked outliers using zscores for both of the dataframes. For regression purpose, the capacity_mw was target and it was not included in its continuous features' list. And for classification problem, primary_fuel was the target and capacity_mw was included in its continuous features list. Luckily, I got same indices containing the outliers for both of the datasets and then removed outliers in the parent dataset. As shown in the Screenshot 9 & 10:

In [38]:
```
cont_cols_reg=cont_cols.copy()
cont_cols_reg.remove('capacity_mw')
print(cont_cols_reg)
```

```
['latitude', 'longitude', 'commissioning_year', 'generation_gwh_2014', 'generation_gwh_2015', 'generation_gwh_2016', 'generatio
n_gwh_2018']
```

In [39]:
```
# now, as there are two targets to be find out,, we will consider one target as a feature while finding the other one..
# i will do all the eda side by side for both of the models for finding different targets..
```

In [40]:
```
print(cont_cols)
```

```
['capacity_mw', 'latitude', 'longitude', 'commissioning_year', 'generation_gwh_2014', 'generation_gwh_2015', 'generation_gwh_20
16', 'generation_gwh_2018']
```

In [41]:
```
df_cont_class=pd.DataFrame()
for i in cont_cols:
    df_cont_class[i]=df[i]
print(df_cont_class)
df_cont_reg=pd.DataFrame()
for i in cont_cols_reg:
    df_cont_reg[i]=df[i]
print(df_cont_reg)
```

```
     capacity_mw  latitude  longitude  commissioning_year  \
0            2.5   28.1839    73.2407                2011
1           98.0   24.7663    74.6090                1997
2           39.2   21.9038    69.3732                1997
3          135.0   23.8712    91.3602                2004
4         1800.0   21.9603    82.4091                2015
..           ...       ...        ...                 ...
902       1600.0   16.2949    77.3568                2016
903          3.0   12.8932    78.1654                1997
904         25.5   15.2758    75.5811                1997
905         80.0   24.3500    73.7477                1997
906         16.5    9.9344    77.4768                1997

     generation_gwh_2014  generation_gwh_2015  generation_gwh_2016  \
0            2431.823590          2428.226946          2467.936859
1            2431.823590          2428.226946          2467.936859
2            2431.823590          2428.226946          2467.936859
```

Screenshot 9.

In [42]:
```
ind_class=np.where((np.abs(zscore(df_cont_class[i])))>3)
print(ind_class,len(ind_class[0]))
```

```
(array([ 15, 143, 209, 308, 364, 493, 494, 648, 657, 695, 721, 724, 726,
       786, 808, 880], dtype=int64),) 16
```

In [43]:
```
ind_reg=np.where((np.abs(zscore(df_cont_reg[i])))>3)
print(ind_reg,len(ind_reg[0]))
```

```
(array([ 15, 143, 209, 308, 364, 493, 494, 648, 657, 695, 721, 724, 726,
       786, 808, 880], dtype=int64),) 16
```

In [44]:
```
# as indices which need to be deleted for both of the models are same,, so no need to make new dataframes for
# different target prediction models.. so, we will proceed with one dataframe further..
```

In [45]:
```
df=df.drop(df.index[ind_class])
df
```

Out[45]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	owner	source	geolocation_source	generation_gwh_2014	generation_gwh
0	2.5	28.1839	73.2407	6	1	2011	229	109	1	2431.823590	2428.2
1	98.0	24.7663	74.6090	1	1	1997	258	174	2	2431.823590	2428.2
2	39.2	21.9038	69.3732	7	1	1997	2	21	2	2431.823590	2428.2
3	135.0	23.8712	91.3602	2	1	2004	140	22	2	617.789264	843.7
4	1800.0	21.9603	82.4091	1	2	2015	140	22	2	3035.550000	5916.3
...	...	...	...	...	...	...	...	...	...	...	
902	1600.0	16.2949	77.3568	1	2	2016	140	22	2	2431.823590	0.9
903	3.0	12.8932	78.1654	6	1	1997	114	77	0	2431.823590	2428.2
904	25.5	15.2758	75.5811	7	1	1997	140	21	2	2431.823590	2428.2
905	80.0	24.3500	73.7477	1	1	1997	91	59	2	2431.823590	2428.2
906	16.5	9.9344	77.4768	7	1	1997	279	21	2	2431.823590	2428.2

891 rows × 13 columns

Screenshot 10.

B. Skewness Removal:

Then I again made two copies of the dataset for the classification and regression and removed the skewness using PowerTransformer as shown in Screenshot 11 and 12.

In [51]:
```
df_class=df.copy()
df_reg=df.copy()
print(df_class)
print(df_reg)
```

```
     capacity_mw  latitude  longitude  primary_fuel  other_fuel1  \
0            2.5   28.1839    73.2407             6            1
1           98.0   24.7663    74.6090             1            1
2           39.2   21.9038    69.3732             7            1
3          135.0   23.8712    91.3602             2            1
4         1800.0   21.9603    82.4091             1            2
..           ...       ...        ...           ...          ...
886       1600.0   16.2949    77.3568             1            2
887          3.0   12.8932    78.1654             6            1
888         25.5   15.2758    75.5811             7            1
889         80.0   24.3500    73.7477             1            1
890         16.5    9.9344    77.4768             7            1

     commissioning_year  owner  source  geolocation_source  \
0                  2011    229     109                   1
1                  1997    258     174                   2
2                  1997      2      21                   2
3                  2004    140      22                   2
4                  2015    140      22                   2
```

In [52]:
```
for i in cont_cols:
    df_class[i]=pt.fit_transform(df_class[[i]])
df_class.skew()
```

Out[52]:
```
capacity_mw           0.017980
latitude             -0.076223
longitude            -0.001414
primary_fuel          0.444221
other_fuel1           1.452507
commissioning_year   -0.064209
owner                -0.041015
source                1.798822
geolocation_source   -2.000983
generation_gwh_2014  -0.229759
generation_gwh_2015  -0.299124
```

Screenshot 11.

```
In [53]: for i in cont_cols:
             print(df_class[i].skew())

0.01797988873148643
-0.07622331077864307
-0.0014136579462428456
-0.06420921175248342
-0.22975942096684177
-0.29912445840089
-0.3067152888657833
-0.31654968060426664

In [54]: # hence, skewness removed for classification dataset..

In [55]: for i in cont_cols_reg:
             df_reg[i]=pt.fit_transform(df_reg[[i]])
         df_reg.skew()

Out[55]: capacity_mw            2.216153
         latitude              -0.076223
         longitude             -0.001414
         primary_fuel           0.444221
         other_fuel1            1.452507
         commissioning_year    -0.064209
         owner                 -0.041015
         source                 1.798822
         geolocation_source    -2.000983
         generation_gwh_2014   -0.229759
         generation_gwh_2015   -0.299124
         generation_gwh_2016   -0.306715
         generation_gwh_2018   -0.316550
         dtype: float64

In [56]: for i in cont_cols_reg:
             print(df_reg[i].skew())

-0.07622331077864307
-0.0014136579462428456
-0.06420921175248342
-0.22975942096684177
-0.29912445840089
-0.3067152888657833
-0.31654968060426664

In [57]: # hence, skewness removed for regression dataframe also..
```

Screenshot 12.

C. Features deletion based on correlation matrix:

i. Classification problem dataframe (for predicting primary_fuel):

Here, I plotted heatmaps of the correlation matrix (shown in Fig. 7,8 and 9) of this database and deleted some features which were highly correlated to each other based on these correlation values. The features generation_gwh_2016, generation_gwh_2015 and generation_gwh_2014 were deleted in this dataset and got the final correlation values shown in Fig.10.

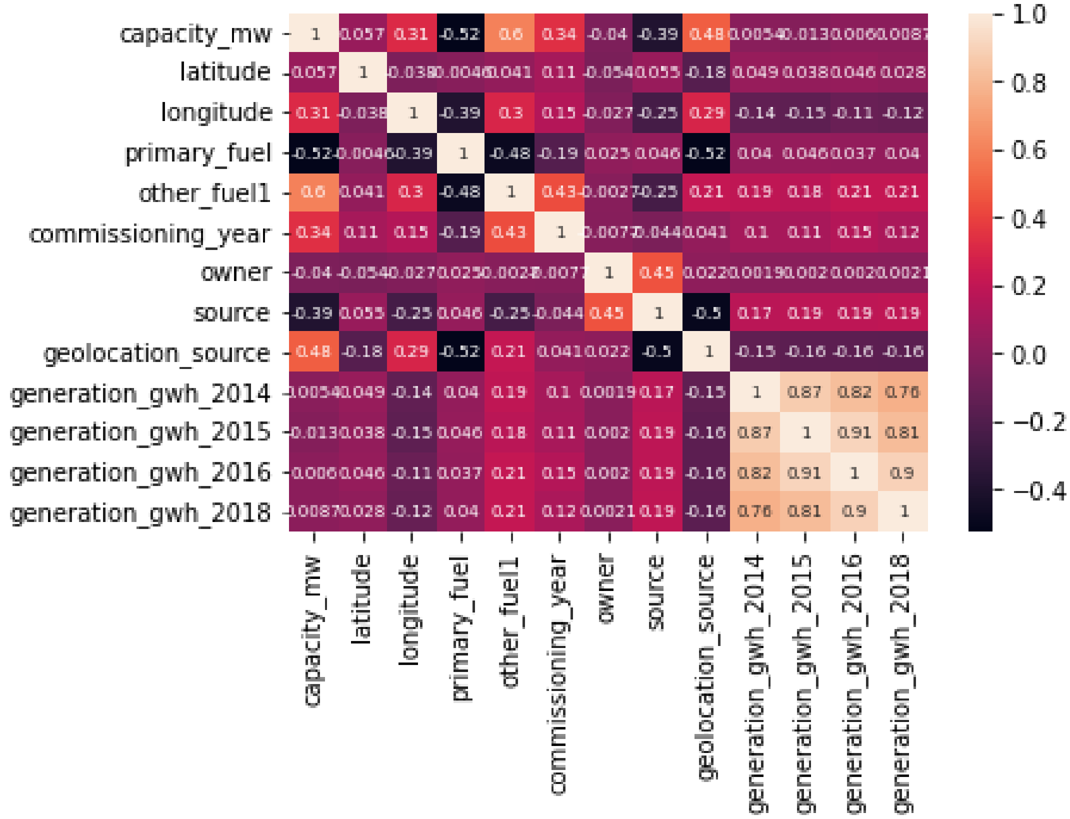


Fig 7. Correlation Heatmap 2.

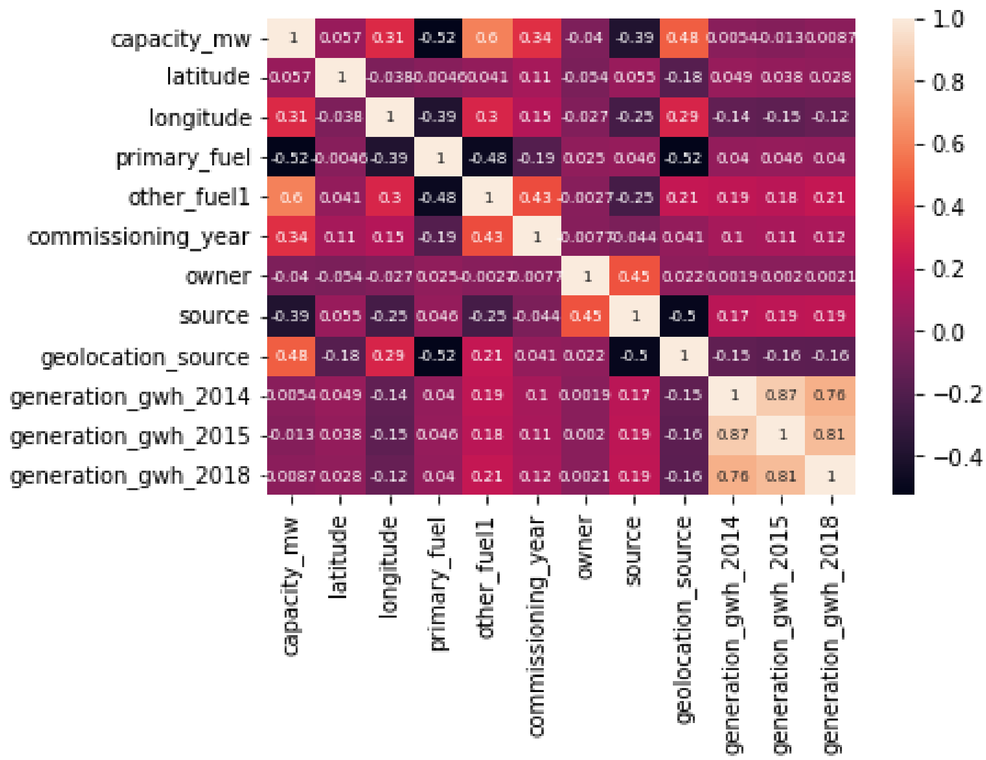


Fig.8 Correlation Heatmap 3.

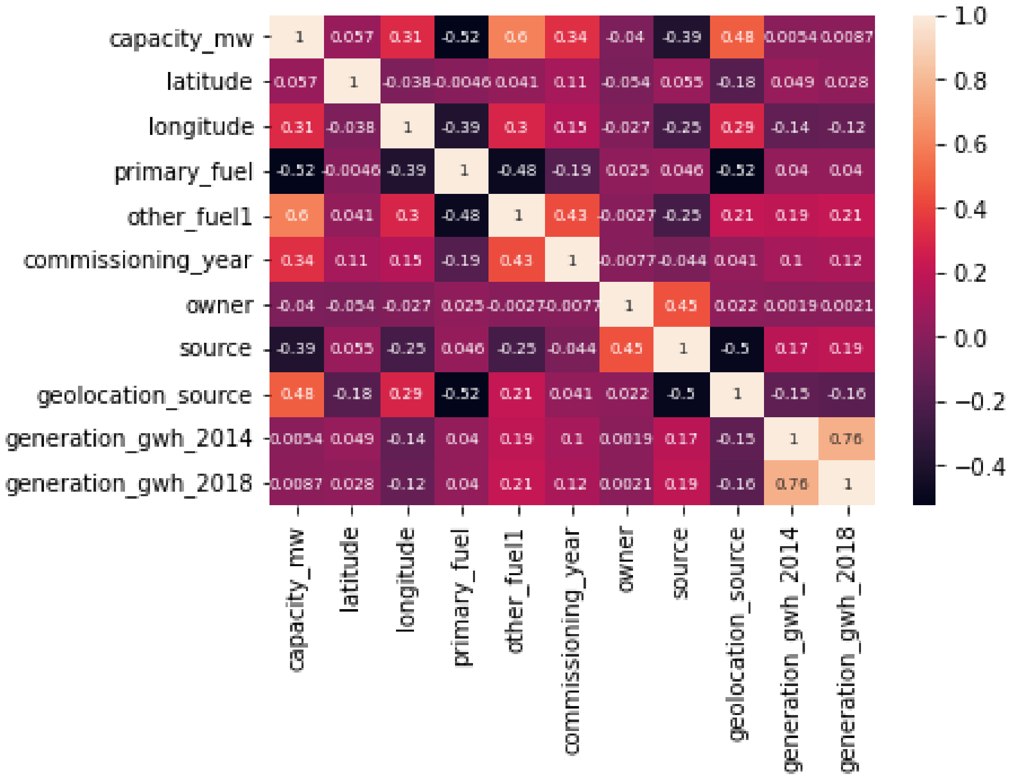


Fig. 9. Correlation Heatmap 4.

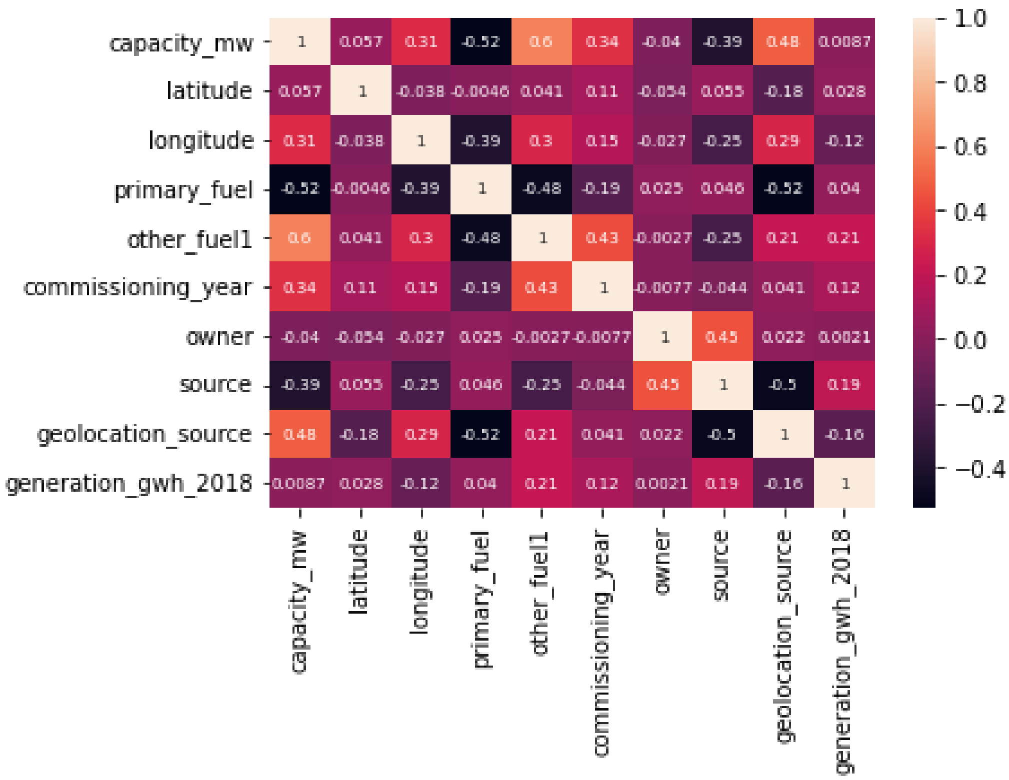


Fig 10. Correlation Heatmap 5.

ii. Regression problem dataframe (for predicting capacity_mw):

Same features were deleted in this dataframe based on the heatmaps shown in Fig. 11,12 and 13. And got the final correlation values as shown in Fig. 14.

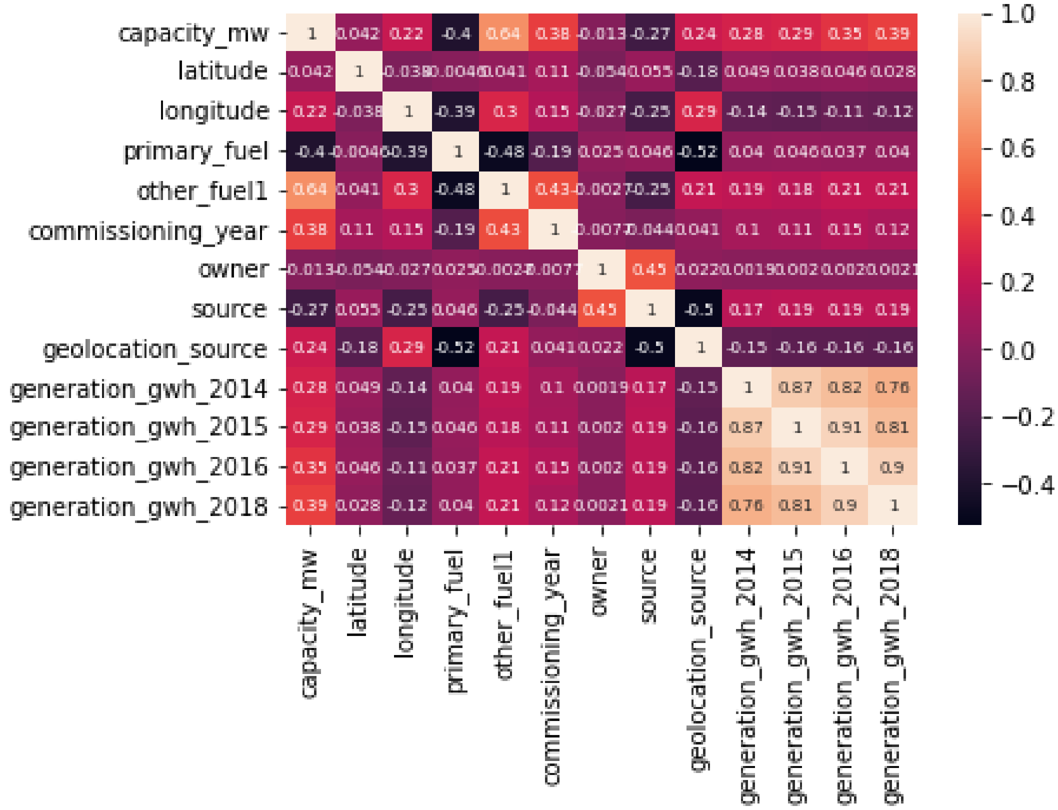


Fig.11 Correlation Heatmap 6.

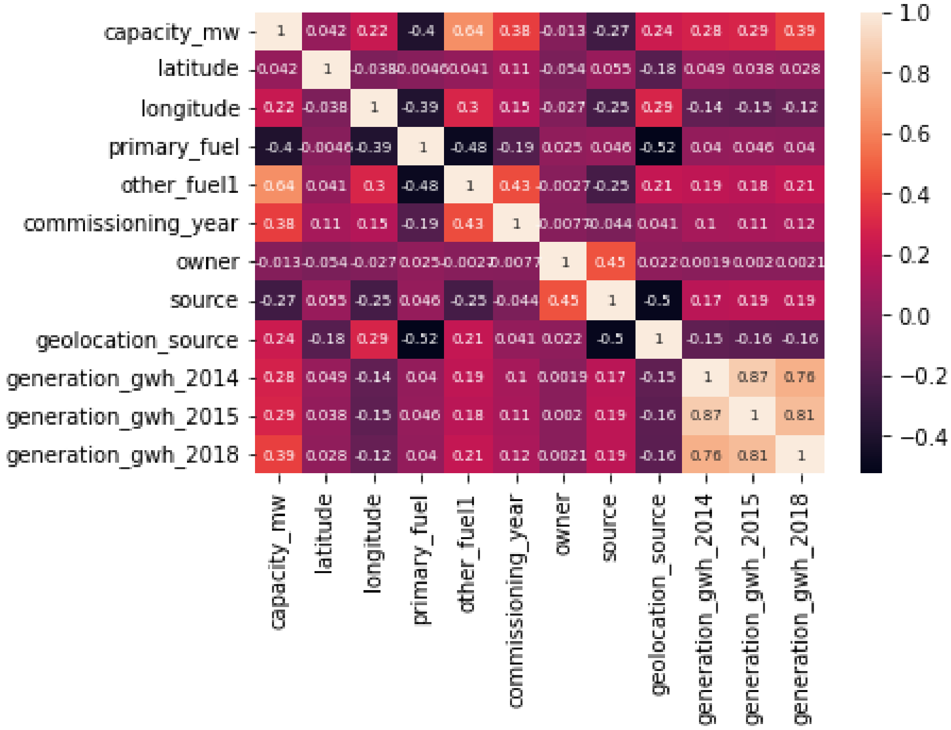


Fig. 12. Correlation Heatmap 7.

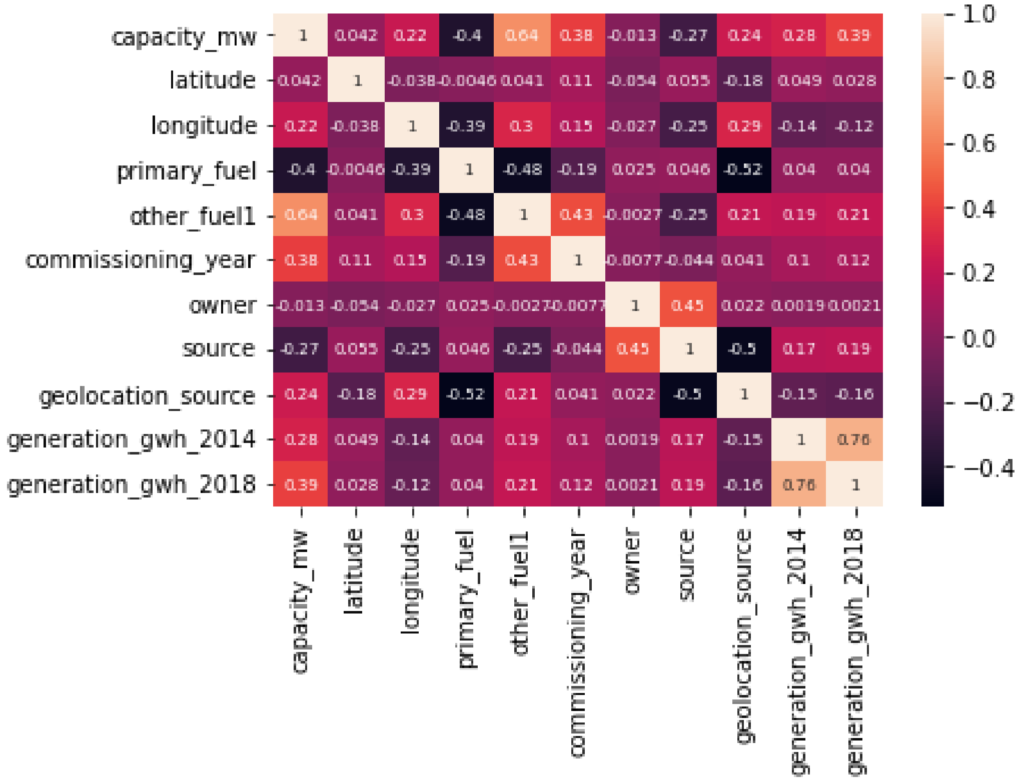


Fig.13. Correlation Heatmap 8.

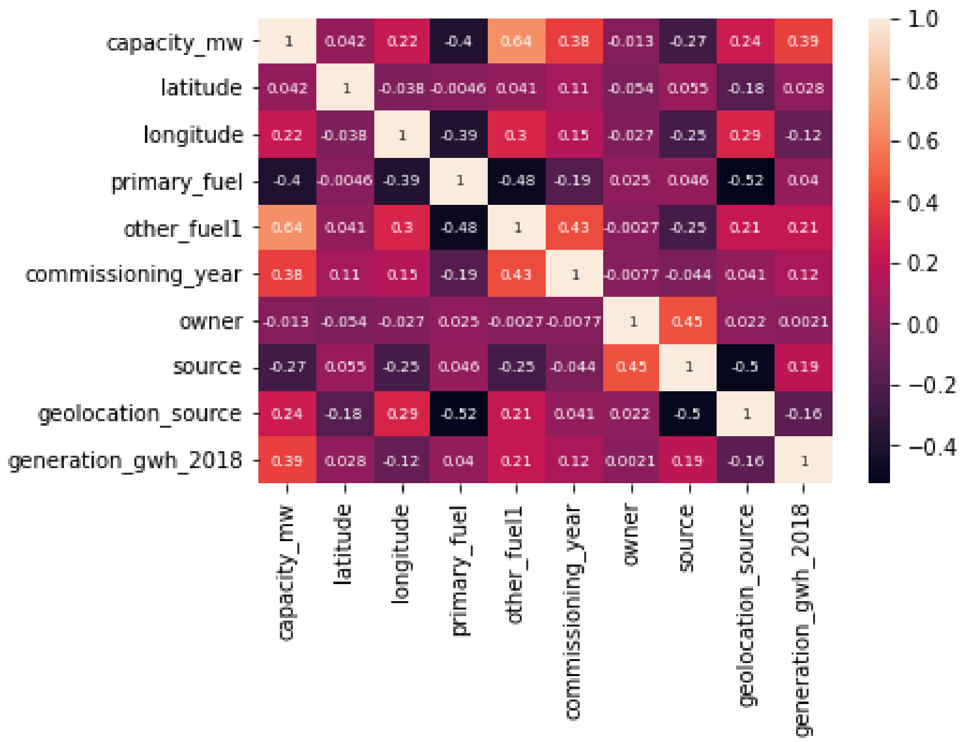


Fig 14. Correlation Heatmap 9

D. Multicollinearity Removal using variance_inflation_factor:

Feature 'owner' was removed from both of the dataframes using vif values and hence, removed multicollinearity.

E. Scaling:

Scaling done on the continuous features of both the datasets using StandardScaler as shown in Screenshot 13:

```
In [88]: from sklearn.preprocessing import StandardScaler,label_binarize
         scaler=StandardScaler()

In [89]: cont_cols_reg.remove('generation_gwh_2015')
         cont_cols_reg.remove('generation_gwh_2016')
         cont_cols_reg.remove('generation_gwh_2014')
         cont_cols.remove('generation_gwh_2015')
         cont_cols.remove('generation_gwh_2016')
         cont_cols.remove('generation_gwh_2014')
         print(cont_cols)
         print(cont_cols_reg)

         ['capacity_mw', 'latitude', 'longitude', 'commissioning_year', 'generation_gwh_2018']
         ['latitude', 'longitude', 'commissioning_year', 'generation_gwh_2018']

In [90]: for i in cont_cols:
             df_class[i]=scaler.fit_transform(df_class[[i]])
         for i in cont_cols_reg:
             df_reg[i]=scaler.fit_transform(df_reg[[i]])
```

Screenshot 13.

F. Feature Selection based on SelectKBest method:

a). In classification problem using f_classif as score_func, I got the scores as below:

```
In [91]: x_class=df_class.drop('primary_fuel',axis=1)
         y_class=df_class['primary_fuel']
         x_reg=df_reg.drop('capacity_mw',axis=1)
         y_reg=df_reg['capacity_mw']
```

```
In [92]: best_features = SelectKBest(score_func=f_classif,k=6)
         fitt_class=best_features.fit(x_class,y_class)
         df_bf_cl=pd.DataFrame({'scores':fitt_class.scores_,'features':x_class.columns})
         df_bf_cl=df_bf_cl.sort_values(by='scores', ascending=False)
         df_bf_cl
```

Out[92]:

	scores	features
6	2452.926209	geolocation_source
3	185.528270	other_fuel1
0	120.130284	capacity_mw
5	108.115102	source
7	51.451866	generation_gwh_2018
2	36.399068	longitude
4	29.135418	commissioning_year
1	14.589664	latitude

Screenshot 14.

These scores were only to check the importance of other target capacity_mw to be as its feature. So, acc. to scores, it can't be ignored during model development but as the number of features were less so I continued with all the features for model development.

b). In regression problem using f_regression as score_func, I got the scores shown in Screenshot 15. So, based on these scores, it was justified that the other target primary_fuel can't be ignored in model development. Similarly, here the number of features were less so I continued with all the features for model development.

```
In [93]: best_features1 = SelectKBest(score_func=f_regression,k=6)
         fitt_reg=best_features1.fit(x_reg,y_reg)
         df_bf_reg=pd.DataFrame({'scores':fitt_reg.scores_,'features':x_reg.columns})
         df_bf_reg=df_bf_reg.sort_values(by='scores', ascending=False)
         df_bf_reg
```

Out[93]:

	scores	features
3	625.892045	other_fuel1
2	166.521082	primary_fuel
7	159.421977	generation_gwh_2018
4	153.211026	commissioning_year
5	67.149909	source
6	54.562412	geolocation_source
1	45.460330	longitude
0	1.544179	latitude

Screenshot 15.

G. Imbalancing removal:

On checking the target variable primary_fuel of classification problem, the problem of imbalancing was found there which was tackled by using SMOTE as shown in Screenshot 16 and the number of rows increased to 2008.

```
In [90]: df['primary_fuel'].value_counts()
Out[90]: 3    251
         1    242
         6    127
         7    123
         2     69
         0     50
         5     20
         4      9
         Name: primary_fuel, dtype: int64

In [91]: class1=[0,1,2,3,4,5,6,7]

In [92]: # so, it's an imbalancing technique in classification problem..hence, going to use smote here..

In [93]: from imblearn.over_sampling import SMOTE
         sm=SMOTE()

In [94]: x_cl,y_cl=sm.fit_resample(x_class,y_class)
```

Screenshot 16.

.

Building Machine Learning Models

a. Classification part:

This is a multi-class problem. So, used label_binarize to encode the target classes and OneVsRestClassifier was used so as to plot roc curves for each class of target. Also, used predict_proba function to get scores which helped to plot these curves and also to get average roc_auc_score.

I coded all this into a single function which was recalled again and again for all the algos I tried. It is shown in Screenshot 17.

The main algos I tried were: LogisticRegression, DecisionTreeClassifier, KNeighborsClassifier, AdaBoostClassifier and RandomForestClassifier.
The output is also shown below one by one:

```python
In [102]: def algo_check (x,y,algo):
              min_diff=1
              max_i=0
              for i in range(100):
                  x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state = i)
                  algo.fit(x_train,y_train)
                  y_pred1 =algo.predict(x_train)
                  acc1= r2_score(y_train,y_pred1)
                  y_pred2 =algo.predict(x_test)
                  acc2= r2_score(y_test,y_pred2)
                  acc=acc1-acc2
                  if acc< min_diff:
                      min_diff=acc
                      max_i = i
                      i+=1
              x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state = max_i)
              algo.fit(x_train,y_train)
              y_pred1 =algo.predict(x_train)
              acc1= r2_score(y_train,y_pred1)
              y_pred2 =algo.predict(x_test)
              acc2= r2_score(y_test,y_pred2)
              cvs=cross_val_score(algo,x_train,y_train,cv=5,scoring='r2')
              ac=cvs.mean()
              mae=mean_absolute_error(y_pred2,y_test)
              mse=mean_squared_error(y_pred2,y_test)
              rmse=np.sqrt(mse)
              print(f'''for algo {algo}, \nthe training accuracy is {acc1}, \ntesing accuracy is {acc2} \nat random_state {max_i}
                  \nand hence, mean square error is {mse} \nand mean_absolute_error is {mae} \nand hence, rmse is {rmse}, \
                  \nalso cross_validation_score is {ac}''')
```

Screenshot 17.

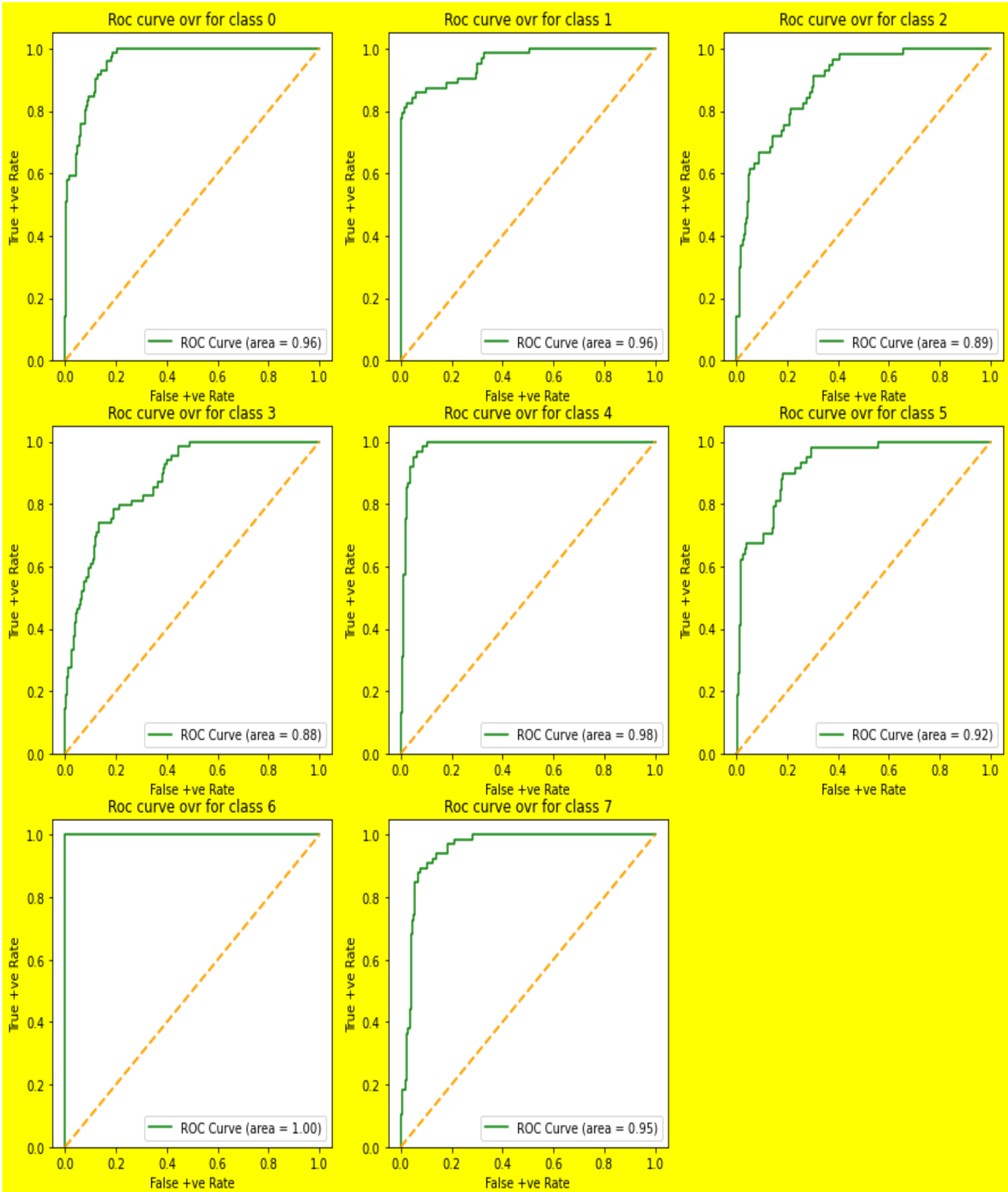
Output for LogisticRegression:

```
for algo LogisticRegression(), the maximum accuracy is 0.7868525896414342 ,
 classification report is :
               precision    recall  f1-score   support

           0       0.88      0.59      0.71        71
           1       0.98      0.78      0.87        63
           2       0.67      0.39      0.49        57
           3       0.76      0.28      0.40        69
           4       0.83      0.79      0.81        61
           5       0.81      0.60      0.69        58
           6       1.00      1.00      1.00        57
           7       0.71      0.67      0.69        66

   micro avg       0.84      0.63      0.72       502
   macro avg       0.83      0.64      0.71       502
weighted avg       0.83      0.63      0.70       502
 samples avg       0.62      0.63      0.62       502
 ,

 at random_state11
 and cross validation score is 0.5730281776793404
one vs rest average roc_auc_score is 0.942730930655201
```


Roc curve ovr for class 0
ROC Curve (area = 0.96)
Roc curve ovr for class 1
ROC Curve (area = 0.96)
Roc curve ovr for class 2
ROC Curve (area = 0.89)
Roc curve ovr for class 3
ROC Curve (area = 0.88)
Roc curve ovr for class 4
ROC Curve (area = 0.98)
Roc curve ovr for class 5
ROC Curve (area = 0.92)
Roc curve ovr for class 6
ROC Curve (area = 1.00)
Roc curve ovr for class 7
ROC Curve (area = 0.95)
True +ve Rate
False +ve Rate
0.0
0.2
0.4
0.6
0.8
1.0

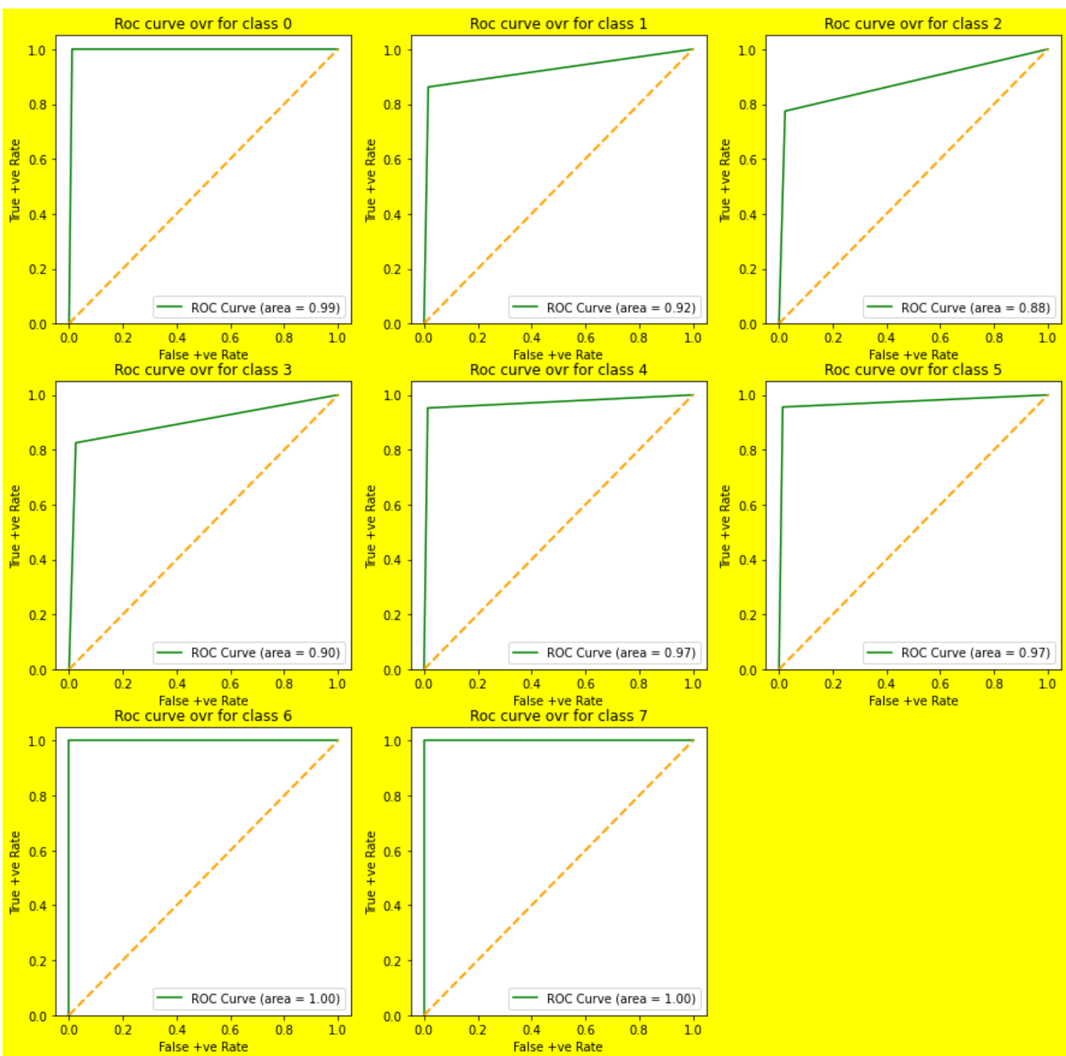
Output for DecisionTreeClassifier:

```
for algo DecisionTreeClassifier(), the maximum accuracy is 0.9402390438247012 ,
 classification report is :
               precision    recall  f1-score   support

           0       0.92      1.00      0.96        59
           1       0.88      0.86      0.87        58
           2       0.83      0.77      0.80        62
           3       0.83      0.83      0.83        63
           4       0.91      0.95      0.93        63
           5       0.92      0.96      0.94        68
           6       1.00      1.00      1.00        72
           7       1.00      1.00      1.00        57

   micro avg       0.91      0.92      0.92       502
   macro avg       0.91      0.92      0.91       502
weighted avg       0.91      0.92      0.92       502
 samples avg       0.90      0.92      0.91       502
 ,

 at random_state37
 and cross validation score is 0.8300233788605881
one vs rest average roc auc score is 0.9542246041278589
```



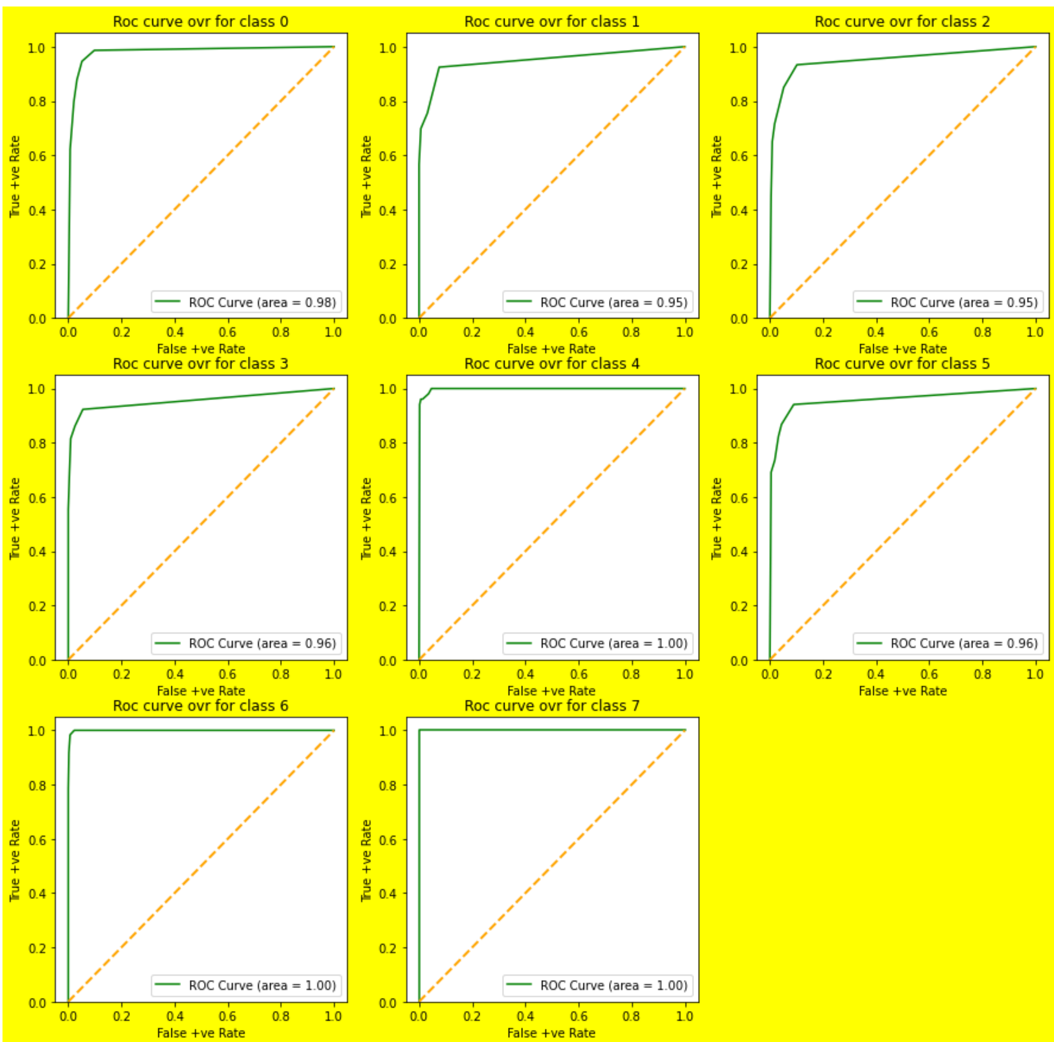
Output for KNeighborsClassifier:

```
for algo KNeighborsClassifier(), the maximum accuracy is 0.8725099601593626 ,
 classification report is :
               precision    recall  f1-score   support

           0       0.82      0.88      0.85        74
           1       0.93      0.70      0.80        53
           2       0.84      0.72      0.77        60
           3       0.93      0.82      0.87        65
           4       0.89      0.96      0.92        51
           5       0.80      0.82      0.81        68
           6       0.98      0.92      0.95        60
           7       0.99      1.00      0.99        71

   micro avg       0.89      0.85      0.87       502
   macro avg       0.90      0.85      0.87       502
weighted avg       0.90      0.85      0.87       502
 samples avg       0.85      0.85      0.85       502
 ,

 at random_state31
 and cross validation score is 0.796167097329888
one vs rest average roc auc score is 0.9733157551989043
```



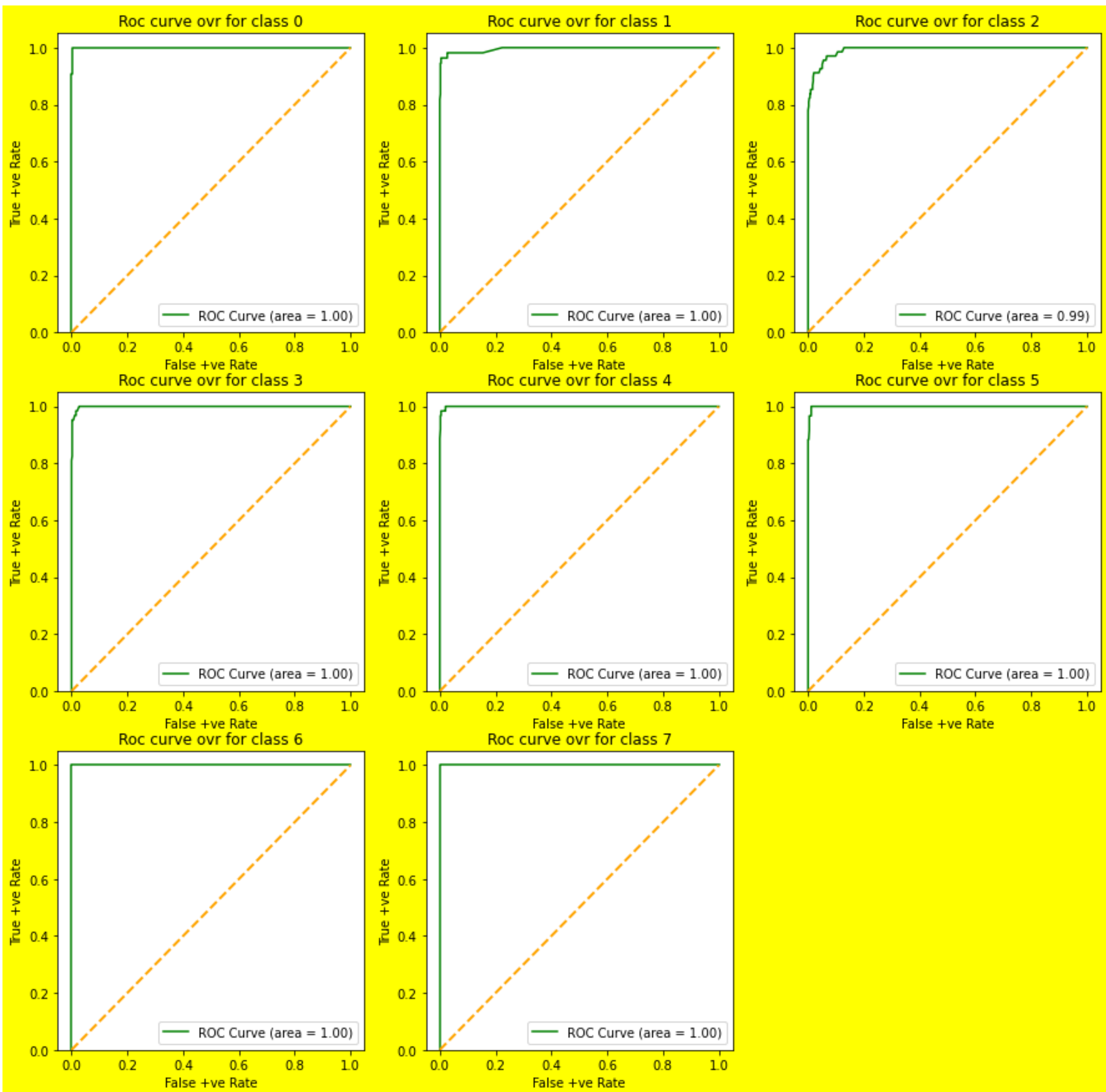
Output for RandomForestClassifier:

```
for algo RandomForestClassifier(), the maximum accuracy is 0.9681274900398407 ,
 classification report is :
               precision    recall  f1-score   support

           0       0.98      0.94      0.96        54
           1       0.98      0.91      0.94        55
           2       0.97      0.82      0.89        68
           3       0.98      0.84      0.91        63
           4       0.97      0.97      0.97        62
           5       0.97      0.97      0.97        60
           6       1.00      1.00      1.00        67
           7       1.00      1.00      1.00        73

   micro avg       0.98      0.93      0.96       502
   macro avg       0.98      0.93      0.95       502
weighted avg       0.98      0.93      0.95       502
 samples avg       0.93      0.93      0.93       502
 ,

 at random_state53
 and cross validation score is 0.8890919158361018
one vs rest average roc_auc_score is 0.9981607262947814
```



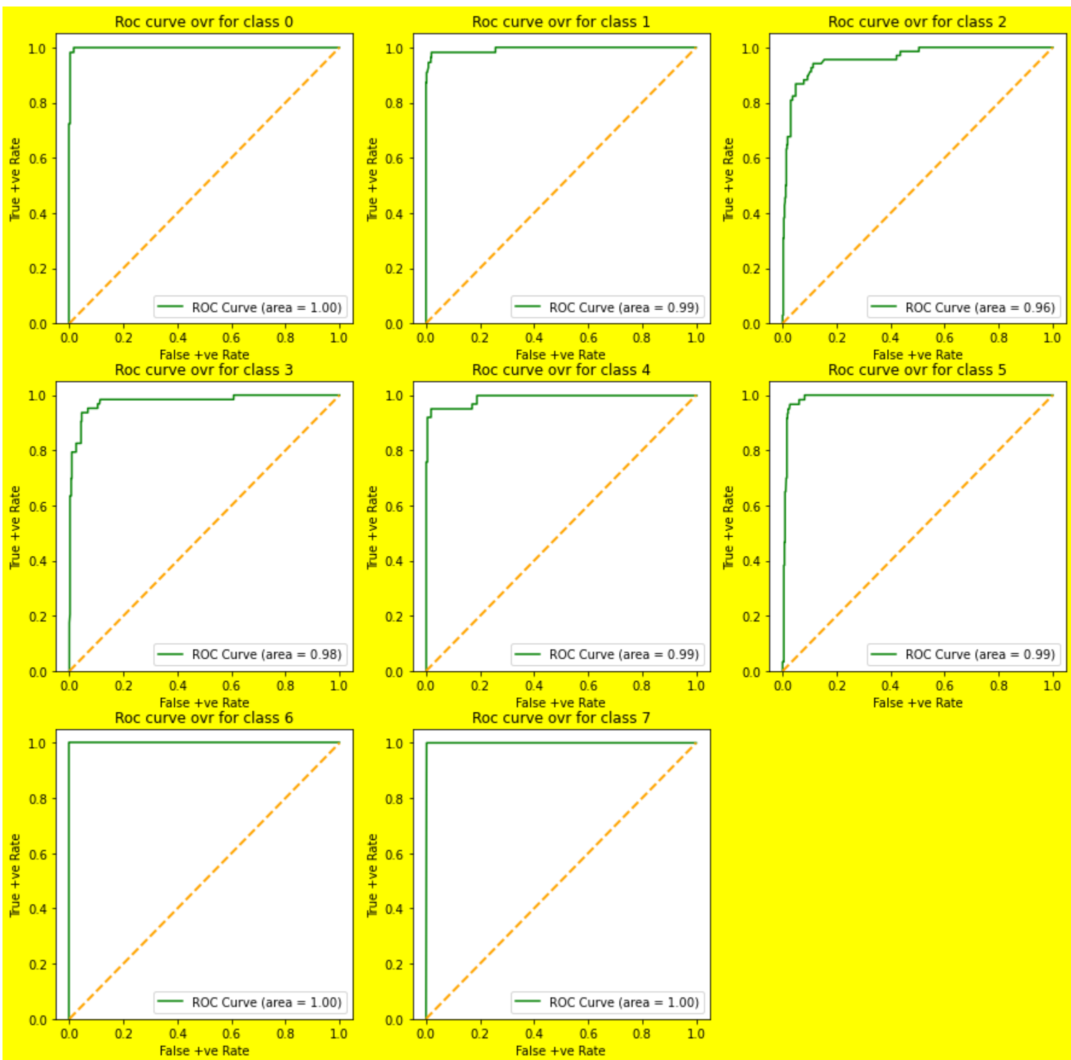
Output for AdaBoostClassifier:

```
for algo AdaBoostClassifier(), the maximum accuracy is 0.36254980079681276 ,
 classification report is :
               precision    recall  f1-score   support

           0       0.96      0.96      0.96        54
           1       0.96      0.91      0.93        55
           2       0.86      0.65      0.74        68
           3       0.80      0.83      0.81        63
           4       0.95      0.92      0.93        62
           5       0.89      0.90      0.89        60
           6       1.00      1.00      1.00        67
           7       0.97      1.00      0.99        73

   micro avg       0.93      0.89      0.91       502
   macro avg       0.92      0.90      0.91       502
weighted avg       0.92      0.89      0.91       502
 samples avg       0.88      0.89      0.88       502
 ,

 at random_state53
 and cross validation score is 0.8432755014150363
one vs rest average roc_auc_score is 0.9883590496849715
```



Hyper-parameter Tuning:

Based on the above results, I found that RFC proved to be the best algo according to my model. I finalized this for the model and trained my model with this algo. Then I hypertuned my model with GridSearchCV and found the following results after instantiating RFC with the parameters given by GridSearchCV. The best parameters I got are also shown in the following Screenshot 18:

```
In [114]: params={'criterion':['gini','entropy','log_loss'],'max_depth':[5,7,8],'min_samples_split':[1,2,3],'min_samples_leaf':[1,2],'max_f
          grid=GridSearchCV(rfc,param_grid=params,cv=5,n_jobs=-1)
          grid.fit(x_cl,y_cl)
          grid.best_params_

Out[114]: {'criterion': 'log_loss',
           'max_depth': 8,
           'max_features': 'log2',
           'min_samples_leaf': 1,
           'min_samples_split': 2}
```

Screenshot 18

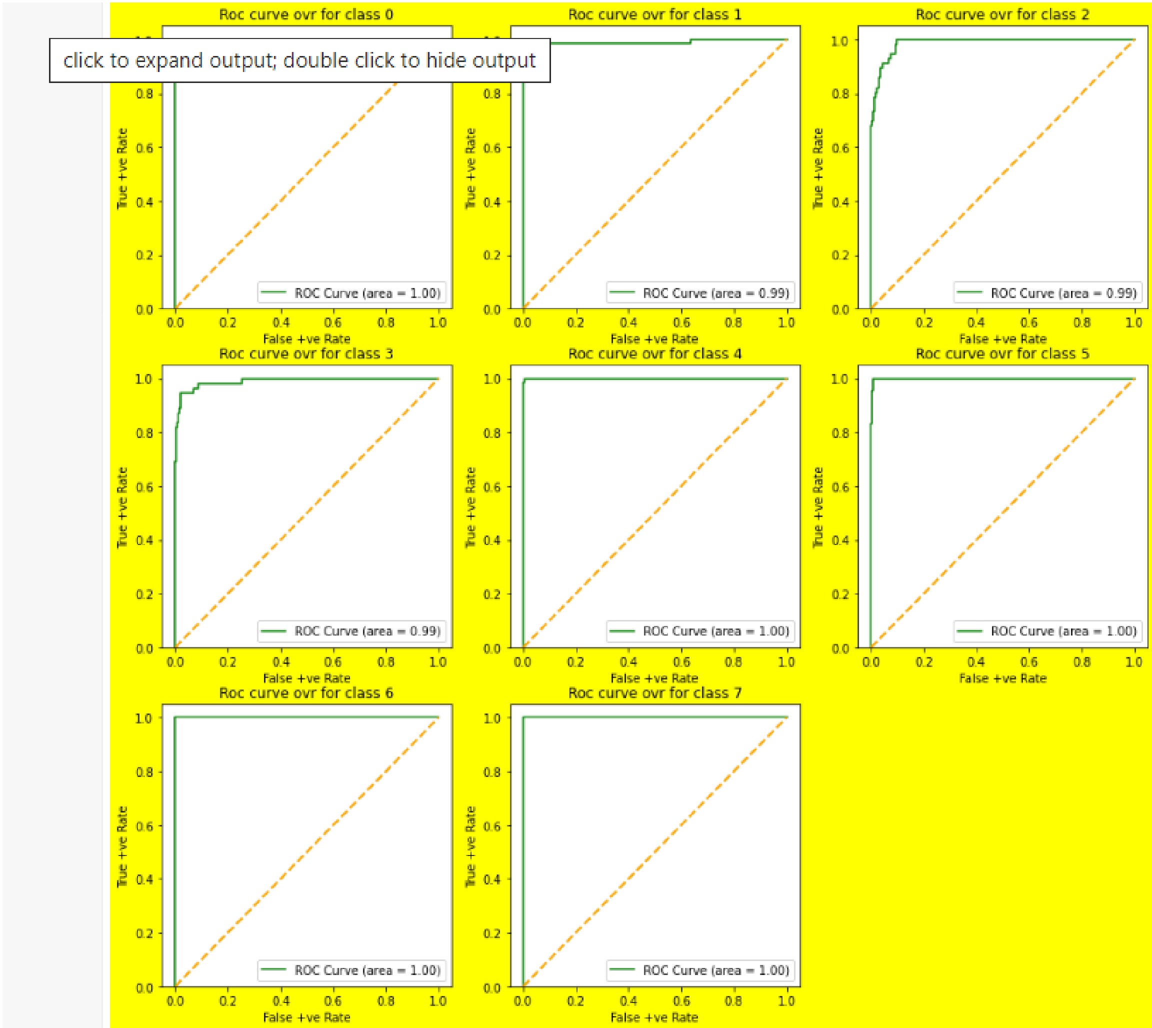
Output got after applying hypertuning parameters to RFC:

```
for Random Forest Classifier with hypertuned parameters, the accuracy is 0.900398406374502 ,
 classification report is :
               precision    recall  f1-score   support

           0       0.98      0.90      0.94        68
           1       0.98      0.91      0.94        64
           2       0.93      0.71      0.81        56
           3       0.95      0.76      0.85        55
           4       0.97      0.98      0.97        59
           5       0.97      0.95      0.96        65
           6       1.00      1.00      1.00        62
           7       1.00      1.00      1.00        73

   micro avg       0.98      0.91      0.94       502
   macro avg       0.97      0.90      0.93       502
weighted avg       0.97      0.91      0.94       502
 samples avg       0.90      0.91      0.91       502
 ,

 and cross validation score is 0.8639165743816906
one vs rest average roc_auc_score is 0.9953648961853152
```



b. Regression Part:

For regression problem, I used the function shown in Screenshot 19 for printing training accuracy and testing accuracy along with the error values for each of the algo I tried. I also printed the cross_validation_score calculated by using cross_val_score for each of the algo I tried

The main algos I tried were: LinearRegression, DecisionTreeRegressor, KNeighborsRegressor, AdaBoostRegressor, RandomForestRegressor, SVR and XGBRegressor.

The outputs I got for each algo is also shown below one by one:

```python
def algo_check (x,y,algo):
    min_diff=1
    max_i=0
    for i in range(100):
        x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state = i)
        algo.fit(x_train,y_train)
        y_pred1 =algo.predict(x_train)
        acc1= r2_score(y_train,y_pred1)
        y_pred2 =algo.predict(x_test)
        acc2= r2_score(y_test,y_pred2)
        acc=acc1-acc2
        if acc< min_diff:
            min_diff=acc
            max_i = i
            i+=1
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state = max_i)
    algo.fit(x_train,y_train)
    y_pred1 =algo.predict(x_train)
    acc1= r2_score(y_train,y_pred1)
    y_pred2 =algo.predict(x_test)
    acc2= r2_score(y_test,y_pred2)
    cvs=cross_val_score(algo,x_train,y_train,cv=5,scoring='r2')
    ac=cvs.mean()
    mae1=mean_absolute_error(y_pred2,y_test)
    mae=mae1/(max(y)-min(y))
    mse1=mean_squared_error(y_pred2,y_test)
    mse=mse1/(max(y)-min(y))
    rmse=np.sqrt(mse)
    print(f'''for algo {algo}, \nthe training accuracy is {acc1}, \ntesing accuracy is {acc2} \nat random_state {max_i}
        \nand hence, mean square error is {mse} \nand mean_absolute_error is {mae} \nand hence, rmse is {rmse},
        \nalso cross_validation_score is {ac}''')
```

Screenshot 19.

Output for LinearRegression:

```
for algo LinearRegression(),
the training accuracy is 0.5140830301477648,
tesing accuracy is 0.6700827602616729
at random_state 97

and hence, mean square error is 29.195587250615265
and mean_absolute_error is 0.08181451681610886
and hence, rmse is 5.403294111059962,
also cross_validation_score is 0.4932120828011256
```

Output for DecisionTreeRegressor:

```
for algo DecisionTreeRegressor(),
the training accuracy is 0.9998890672492123,
tesing accuracy is 0.7433775827726943
at random_state 89

and hence, mean square error is 26.17993942247445
and mean_absolute_error is 0.04963523323130946
and hence, rmse is 5.11663360252368,
also cross_validation_score is 0.5207436599176801
```

Output for KNeighborsRegressor:

```
for algo KNeighborsRegressor(),
the training accuracy is 0.7383622705696262,
tesing accuracy is 0.7655559572272784
at random_state 63

and hence, mean square error is 23.211340620527157
and mean_absolute_error is 0.054742638621965975
and hence, rmse is 4.817814921780117,
also cross_validation_score is 0.5961791276260371
```

Output for AdaBoostRegressor:

```
for algo AdaBoostRegressor(),
the training accuracy is 0.7460123763898853,
tesing accuracy is 0.7644721156538262
at random_state 21

and hence, mean square error is 24.485679587040686
and mean_absolute_error is 0.08060477151986843
and hence, rmse is 4.948300676701113,
also cross_validation_score is 0.5172180540483013
```

Output for RandomForestRegressor:

```
for algo RandomForestRegressor(),
the training accuracy is 0.9613203016761426,
tesing accuracy is 0.8389569406170791
at random_state 89

and hence, mean square error is 16.429186446795775
and mean_absolute_error is 0.04071069740311164
and hence, rmse is 4.053293284083423,
also cross_validation_score is 0.7086413560402668
```

Output for SVR:

```
for algo SVR(),
the training accuracy is -0.183283328403121,
tesing accuracy is -0.07310829392102192
at random_state 29

and hence, mean square error is 65.19958103823392
and mean_absolute_error is 0.07607614286153366
and hence, rmse is 8.074625752208824,
also cross_validation_score is -0.19495628698924167
```

Output for XGBRegressor:

```
for algo XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, ...),
the training accuracy is 0.9995350451915935,
tesing accuracy is 0.8261531235330646
at random_state 89

and hence, mean square error is 17.735397958859576
and mean_absolute_error is 0.042646380290942504
and hence, rmse is 4.211341586580169,
also cross_validation_score is 0.671937273329122
```

Hyper-parameter Tuning:

From the above results, it got clear that RandomForestRegressor is the best algo with cross_val_score 71% with minimum errors. So, I trained my model with this algo and then hypertuned its parameters using GridSearchCV. The parameters I got after tuning it with GridSearchCv are also shown in Screenshot 20.

```
params1={'criterion':['squared_error','absolute_error','poisson'],'max_depth':[5,7,8],'min_samples_split':[1,2,3],'min_samples_le
grid1=GridSearchCV(rfr,param_grid=params1,cv=5,n_jobs=-1)
grid1.fit(x_reg,y_reg)
grid1.best_params_
```

```
{'criterion': 'absolute_error',
 'max_depth': 8,
 'max_features': 'log2',
 'min_samples_leaf': 2,
 'min_samples_split': 2}
```

Screenshot 20.

Output got after applying hypertuning parameters to RFR:

```
for Random Forest Regressor with hyper tuned parameters,
the training accuracy is 0.8767386679864824,
tesing accuracy is 0.8432910595747576
and hence, mean square error is 11.517074924064746

and mean_absolute_error is 0.033682867597337 89
and hence, rmse is 3.393681617957811,
also cross_validation_score is 0.753186131437603
```

CONCLUDING REMARKS:

- IN ORDER TO FIND PRIMARY_FUEL AS TARGET WHICH IS A CLASSIFICATION PROBLEM, I GOT THE BEST RESULTS FROM RANDOMFORESTCLASSIFIER. SO, I FINALIZED THIS FOR MY MODEL.
- THE ACCURACY IN TERMS OF ACCURACY_SCORE I GOT IS IN THE RANGE OF 97% ALONG WITH AVERAGE ROC_AUC_SCORE OF DIFFERENT LABELS TO BE 99% AND 89% CROSS_VAL_SCORE.
- ON CHECKING BY GRIDSEARCHCV, I FOUND THAT RFC IMPROVED MY MODEL WITH ACCURACY TO UPTO 90% WITH ROC_AUC_SCORE 99% AND ALSO CROSS_VAL_SCORE TO UPTO 86%. HENCE, I FINALIZED MY MODEL WITH RANDOMFORESTCLASSIFIER WITH ITS HYPERTUNED PARAMETERS GIVEN BY GRID SEARCH CV.
- I ALSO PRINTED CLASSIFICATION_REPORT, AVERAGE ROC_AUC_SCORE FOR EACH ALGO I TRIED AND ALSO PLOTTED ROC_CURVE FOR ALL ALGOS.
- IN ORDER TO FIND CAPACITY_MW AS TARGET WHICH IS A REGRESSION PROBLEM, I GOT THE BEST RESULTS FROM RANDOMFORESTREGRESSOR. SO, I FINALIZED THIS FOR MY MODEL.
- THE R2_SCORES FOR TRAINING COMES OUT TO BE 0.96 AND FOR TESTING IS 0.83 IN RANDOMFORESTREGRESSOR ALONG WITH THE MINIMUM ERROR TERMS THAN ALL OTHER ALGOS AND ALSO 70.8% CROSS_VAL_SCORE VALUE.
- AFTER HYPERTUNING WITH GRIDSEARCHCV, RANDOMFORESTREGRESSOR GAVE TRAINING_ACCURACY (OR R2_SCORE) TO BE 0.87 AND TESTING_ACCURACY TO BE 0.84 WITH DECREASE IN ERROR TERMS AND INCREASE IN CROSS_VAL_SCORE TO BE 75.3%.
- HENCE, I FINALIZED RANDOMFORESTCLASSIFIER AND RANDOMFORESTREGRESSOR WITH THEIR HYPERTUNED PARAMETERS AS THE FINAL ALGOS FOR MY MODEL.
- I ALSO SAVED MY MODEL IN PICKLE USING NAME 'Global_power_plant_evaluation_project'.