Global Power Plant Database

Problem Statement:

Description

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.

Features

- country (text): 3 character country code corresponding to the ISO 3166-1 alpha-3 specification [5]
- country_long (text): longer form of the country designation
- name (text): name or title of the power plant, generally in Romanized form
- gppd_idnr (text): 10 or 12 character identifier for the power plant
- capacity_mw (number): electrical generating capacity in megawatts
- latitude (number): geolocation in decimal degrees; WGS84 (EPSG:4326)
- longitude (number): geolocation in decimal degrees; WGS84 (EPSG:4326)
- primary_fuel (text): energy source used in primary electricity generation or export

- other_fuel1 (text): energy source used in electricity generation or export
- other_fuel2 (text): energy source used in electricity generation or export
- other_fuel3 (text): energy source used in electricity generation or export
- commissioning_year (number): year of plant operation, weighted by unit-capacity when data is available
- owner (text): majority shareholder of the power plant, generally in Romanized form
- source (text): entity reporting the data; could be an organization, report, or document, generally in Romanized form
- url (text): web document corresponding to the source field
- geolocation_source (text): attribution for geolocation information
- wepp_id (text): a reference to a unique plant identifier in the widely-used PLATTS-WEPP database.
- year_of_capacity_data (number): year the capacity information was reported
- generation_gwh_2013 (number): electricity generation in gigawatt-hours reported for the year 2013
- generation_gwh_2014 (number): electricity generation in gigawatt-hours reported for the year 2014
- generation_gwh_2015 (number): electricity generation in gigawatt-hours reported for the year 2015
- generation_gwh_2016 (number): electricity generation in gigawatt-hours reported for the year 2016
- generation_gwh_2017 (number): electricity generation in gigawatt-hours reported for the year 2017
- generation_gwh_2018 (number): electricity generation in gigawatt-hours reported for the year 2018
- generation_gwh_2019 (number): electricity generation in gigawatt-hours reported for the year 2019

- generation_data_source (text): attribution for the reported generation information
- estimated_generation_gwh (number): estimated electricity generation in gigawatt-hours

Prediction :

Make two prediction 1) Primary Fuel 2) capacity_mw

Loading the data

#reading csv and storing it in df

df=pd.read_csv("https://raw.githubusercontent.com/wri/global-power-plant-database/master/source_databases_csv/database_IND.csv")

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

907 rows × 27 columns

The dataset has 907 rows and 27 columns

deleting columns url , owner, source and geolocation_source as it has nothing to do with power generation

```
df.drop(['url','owner','source', 'geolocation_source'],inplace=True,axis=1)
df
```

EDA

- checking for nulls
  it was found that few features had more than 80% of its data as nulls, as imputing them would lead to undesirable results dropping those features would be better

```
#filtering columns where more than 80% nulls are present
pos=np.where(df.isnull().sum()>680)
df.columns[pos]
```

```
Index(['other_fuel1', 'other_fuel2', 'other_fuel3', 'wepp_id',
       'generation_gwh_2013', 'generation_gwh_2019',
       'estimated_generation_gwh'],
      dtype='object')
```

The columns which needed imputed were

1. latitude and longitude
2. commissioning_year
3. year_of_capacity_data
4. generation_gwh_2014, generation_gwh_2015, generation_gwh_2016, generation_gwh_2017, generation_gwh_2018
5. generation_data_source

- **check for unique values present**

1. country and country_long can be deleted as it has only one unique data and its considering data in one country
2. year_of_capacity_data can be deleted as it has only one unique data that is 2019
3. generation_data_source can be deleted as it has only one unique data that is Central Electricity Authority
4. gppd_idnr can be deleted as its for identiifcation purpose
5. owner can be deleted as it has nothing to do with power generation

- **dividing the dataset into numerical and categorical data**

```
discretecols=[]
continuecols=[]

for column in numericalCol:
     if df[column].nunique()>10:
        continuecols.append(column)

     else:
        discretecols.append(column)

print("The discrete columns are:",discretecols)
print('\n')
print("The continuous columns are:",continuecols)
```

- **check for duplicates**
  there was one duplicate hence dropped

- **Imputation**
  a. The latitude and longitude is grouped using primary fuel

```
lat_long_mean=df.pivot_table(values=['latitude','longitude'], index='primary_fuel')
lat_long_mean
```

primary_fuel	latitude	longitude
Biomass	17.460458	75.679052
Coal	21.657714	79.431460
Gas	20.050144	78.408238
Hydro	22.258483	78.846256
Nuclear	18.081478	76.124056
Oil	17.311847	74.833806
Solar	24.095380	74.352328
Wind	17.857224	74.181553

The index value where nulls are present are stored

```
#storing index values where nulls are present
pos = df['latitude'].isnull()
a=df['longitude'].isnull()
```

Imputing the values for nulls

```
# impute values
df.loc[pos, 'latitude'] = df.loc[pos, 'primary_fuel'].apply(lambda x: lat_long_mean.loc[x])
df.loc[pos, 'longitude'] = df.loc[a, 'primary_fuel'].apply(lambda x: lat_long_mean.loc[x])
```

b. commissioning year

```
year=df.groupby(['primary_fuel'])[['commissioning_year']].mean()
year
```

	commissioning_year
primary_fuel	
Biomass	NaN
Coal	2006.021164
Gas	2002.830508
Hydro	1988.709163
Nuclear	1994.250000
Oil	1994.583333
Solar	2013.375000
Wind	NaN

- Biomass and wind power plants has no data for commissioning year

Imputing the value for commissioning year as 2012 for biomass and wind

```
# as we dont know when Biomass and Wind power plant were established assuming that all plants were commisioned in 2012
# the first large scale Biomass was commsioned on 2012, source interent


pos=np.where(df['primary_fuel']=='Biomass')
for i in pos:
    df.loc[i,'commissioning_year']=2012

# wind plant was commisioned from late 1986,but i am not sure if these plants were used for commerical power generation in 1986
# assume that all wind plant commisioned from 2012

pos=np.where(df['primary_fuel']=='Wind')
for i in pos:
    df.loc[i,'commissioning_year']=2012
```

c. generation_gwh_2014, generation_gwh_2015, generation_gwh_2016, generation_gwh_2017, generation_gwh_2018

Assumption:

- assigning the value of power generation as 0, when the commissioning year is greater than or equal to generation year
- eg if commissioning year is 2015 generation year is 2014, the generation is zero as the plant is not ready for generating power for commercial use
- when commissioning year value is 2015, it implies that power plant was available for generating power only at the beginning of the next year, here its 2016

```
pos=np.where(df['commissioning_year']>=2015)
for i in pos:
    df.loc[i,'generation_gwh_2014']=0

pos=np.where(df['commissioning_year']>=2016)
for i in pos:
    df.loc[i,'generation_gwh_2015']=0

pos=np.where(df['commissioning_year']>=2017)
for i in pos:
    df.loc[i,'generation_gwh_2016']=0

pos=np.where(df['commissioning_year']>=2018)
for i in pos:
    df.loc[i,'generation_gwh_2017']=0

pos=np.where(df['commissioning_year']>=2019)
for i in pos:
    df.loc[i,'generation_gwh_2018']=0
```

Imputing where nulls are present

- $\text{Generation} = \frac{\text{capacity of plant} \times 24(\text{hours}) \times 365\ (\text{days}) \times \text{efficency}}{1000}$
- efficeny considered for solar is 50%, wind is 40% and for biomass is 35% and for the rest of the sources is 80%

```
for j in range(2014,2019):
    pos=0
    pos=np.where(df['generation_gwh_{}'.format(j)].isna()==True)
    for k in pos:
        if [df['primary_fuel'].iloc[k]]=='Solar':
            df['generation_gwh_{}'.format(j)].iloc[k] =((df['capacity_mw'].iloc[k]*24*365*0.5)/1000)
        elif [df['primary_fuel'].iloc[k]]=='Wind':
            df['generation_gwh_{}'.format(j)].iloc[k]=((df['capacity_mw'].iloc[k]*24*365*0.4)/1000)
        elif [df['primary_fuel'].iloc[k]]=='Biomass':
            df['generation_gwh_{}'.format(j)].iloc[k]=((df['capacity_mw'].iloc[k]*24*365*0.35)/1000)
        else:
            df['generation_gwh_{}'.format(j)].iloc[k]=((df['capacity_mw'].iloc[k]*24*365*0.8)/1000)
```

d. capacity

checking which all data had zero as capacity, which is not possible

```
pos=np.where(df['capacity_mw']==0)
df.loc[pos]
```

	capacity_mw	latitude	longitude	primary_fuel	commissioning_year	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_201
828	0.0	12.8491	77.6752	Solar	2013.375	0.0	0.0	0.0	0

Only one such case exists imputing it with mean value according to its primary_fuel

```
capacity_mean=df.pivot_table(values=['capacity_mw'], index='primary_fuel')
capacity_mean
```

	capacity_mw
primary_fuel	
Biomass	20.065200
Coal	797.826434
Gas	364.818928
Hydro	185.026972
Nuclear	975.555556
Oil	88.942000
Solar	21.712598
Wind	33.519262

```
# imputing the avg value

df.loc[828,'capacity_mw']=21.712598
```

- **Checking if nulls are present after imputing**

```
df.isna().sum().sum()
```

```
0
```

checking how the data is defined statistically for numerical continuous data and visualising

```
df[continuecols].describe().T
```

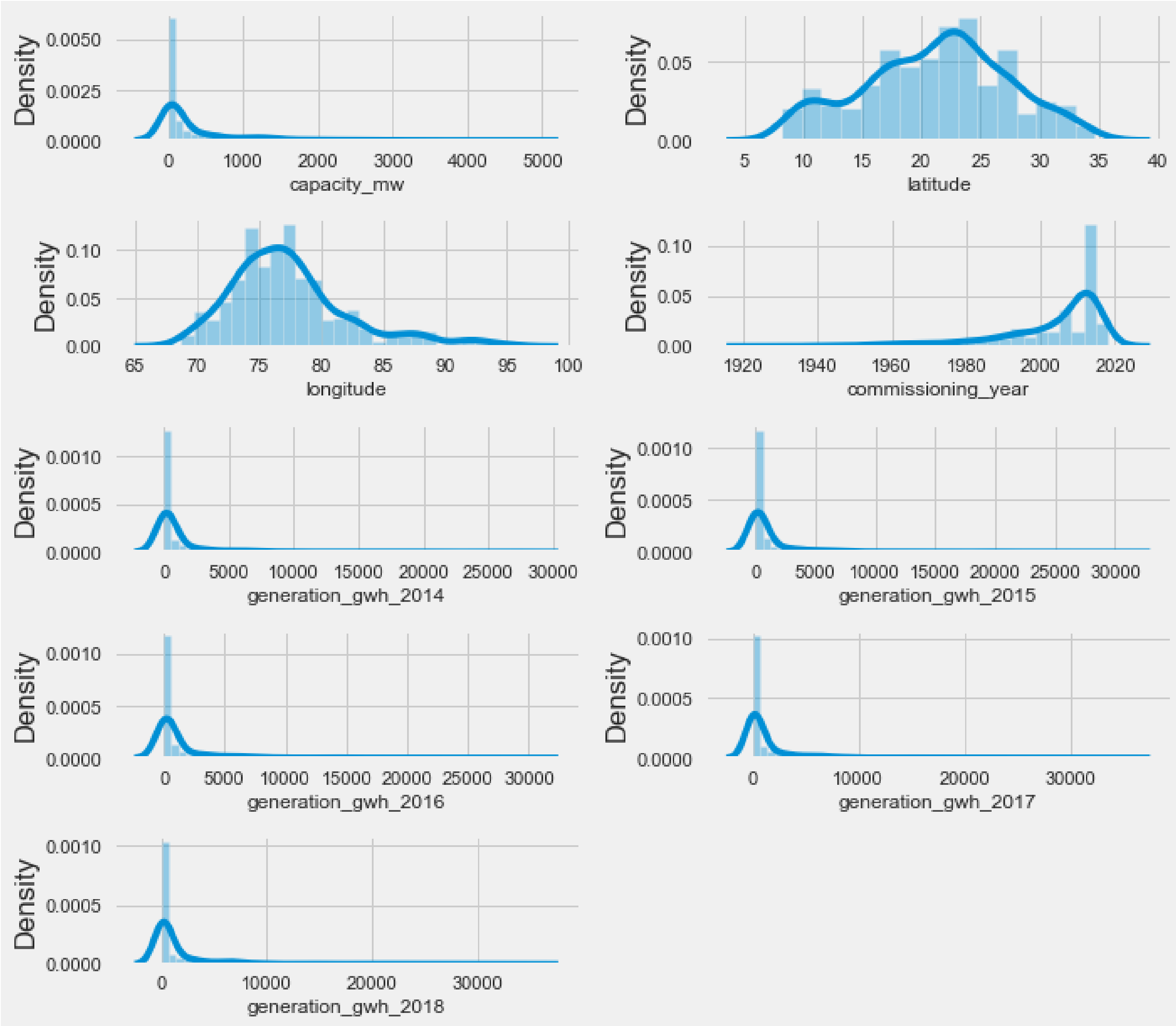
	count	mean	std	min	25%	50%	75%	max
capacity_mw	906.0	326.582957	590.312125	1.0000	16.962500	59.600000	386.625000	4760.000
latitude	906.0	21.168562	6.105802	8.1689	17.065350	21.778300	25.178075	34.649
longitude	906.0	77.431055	4.845145	68.6447	74.247525	76.729350	79.326675	95.408
commissioning_year	906.0	2002.778485	14.861056	1927.0000	1997.000000	2010.000000	2013.000000	2018.000
generation_gwh_2014	906.0	1183.234008	2932.911942	0.0000	57.561998	178.704000	700.800000	28127.000
generation_gwh_2015	906.0	1254.770839	3110.361979	0.0000	68.139488	182.604000	738.487538	30539.000
generation_gwh_2016	906.0	1309.564381	3134.863586	0.0000	70.080000	196.224000	837.655125	30015.000
generation_gwh_2017	906.0	1373.576986	3187.333180	0.0000	73.584000	206.153650	919.216750	35116.000
generation_gwh_2018	906.0	1416.043900	3297.231619	0.0000	73.584000	213.560875	971.382512	35136.000

- capacity_mw
  - mean to std is 0.55
  - min is 1 for solar power plant, max is 4760 produced by coal power plant
  - the avg capacity of power plant is 590.3
  - the differnce between each quantile is not uniform

- commissioning_year
  - the oldest power plant was commissioned in 1927 (hydro) and the latest being commissioned at 2018 (coal and hydro)

- generation_gwh_2014, generation_gwh_2015, generation_gwh_2016, generation_gwh_2017, generation_gwh_2018
  - it has min as zero, this is because the plants were commissioned not in that year

- the generation increased as the years increased, this could be due to increase in no. of power plants or due to high demand for energy causing the plants to be run at peak capacity



- capacity_mw
  - data is not uniformly distributed
  - presence of outliers

- latitude
  - values range from 7.5 to 35
  - negatively skewed

- longitude
  - value in the range 70-88
  - positively skewed
  - presence of outliers

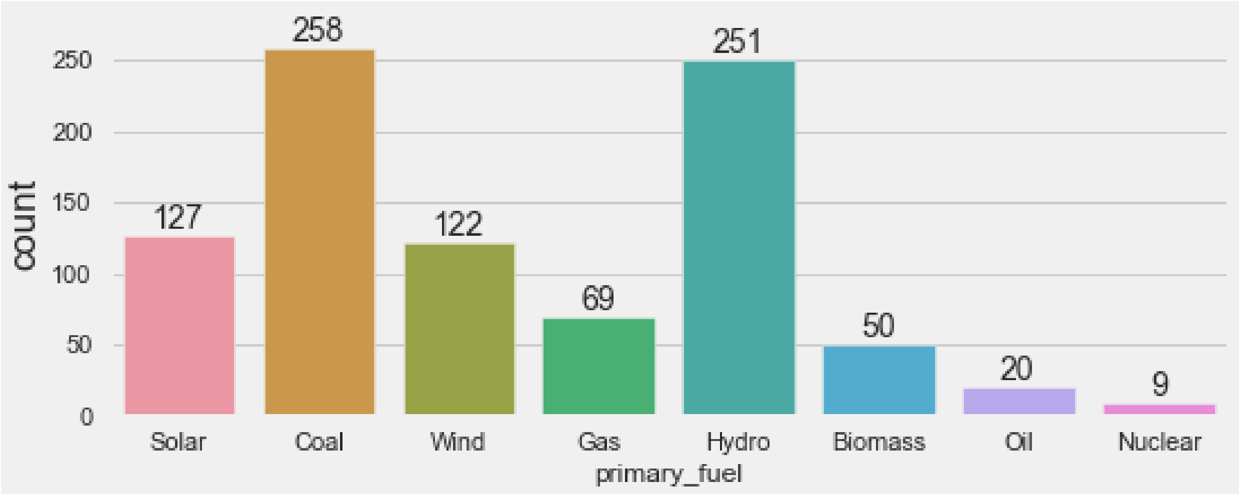
- commissioning year
  - negatively skewed
  - presence of outliers
  - large no.of power plants were commissioned in 2000-2020

- generation_gwh_2014, generation_gwh_2015, generation_gwh_2016, generation_gwh_2017, generation_gwh_2018
  - presence of outliers
  - there is high concentration at value zero this is because many power plants taken into consideration were not commissioned that year also oil power plant stopped generating from 2000

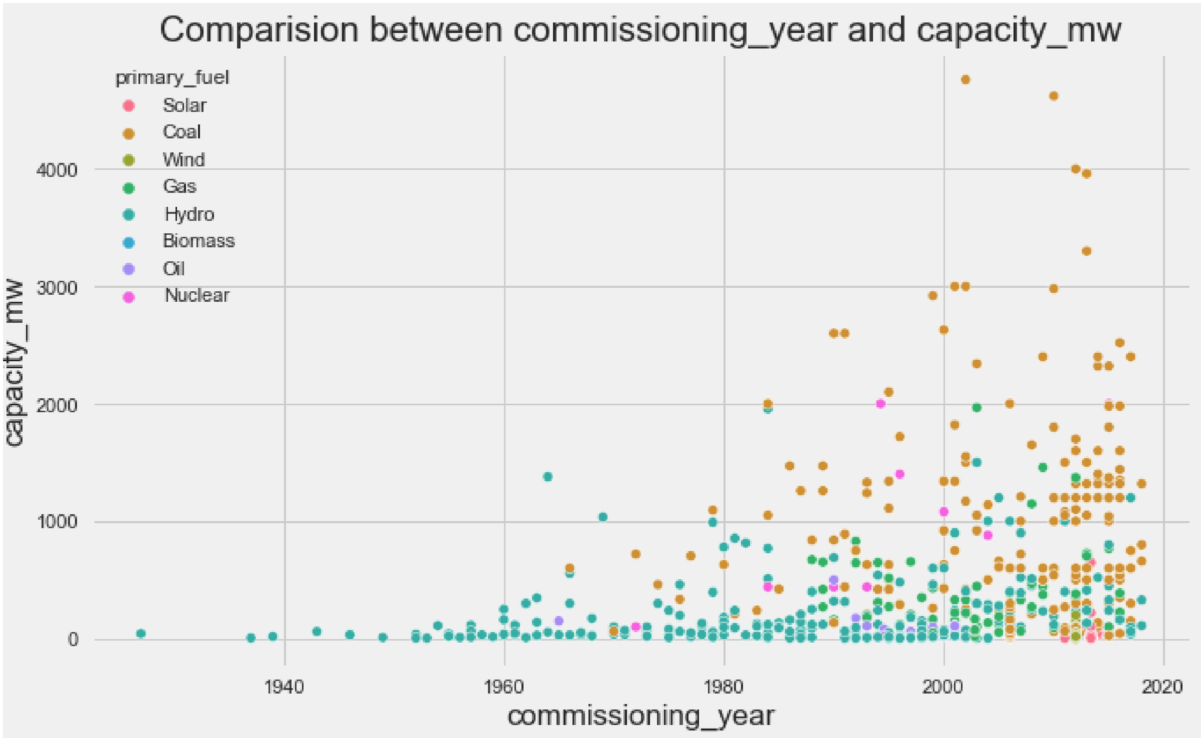
- **checking how the data is defined statistically for categorical data's visualising**

```
df[objectColumns].describe(include=['O']).T
```

	count	unique	top	freq
primary_fuel	906	8	Coal	258



- Coal is the most widely used power plant, followed by hydro power plant
- Oil and nuclear has the least no. of power plants installed
- Solar, Wind and Biomass power plants are renewable power plants account for 32.9% of total power plant installed



- as the years increased, capacity of the plants also increased, the types of power plants installed increased
- from 1940-1980 hydro power plants were only used
- 1981-2000 hydro and coal power plants were used

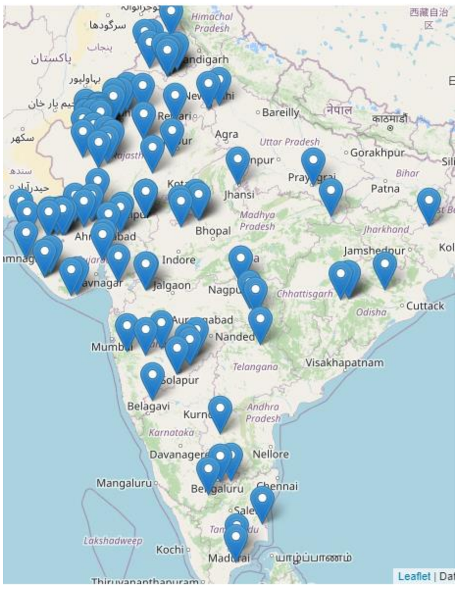
- 2000-2020 other types of power plants were installed, but their capacity was low compared to coal and hydro

Geographical mapping to see the location of power plants

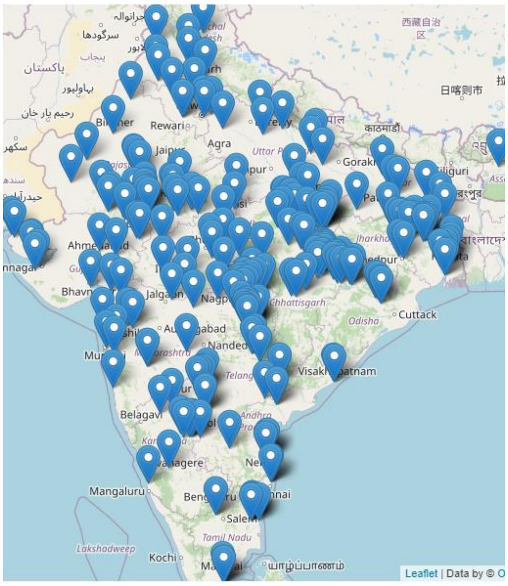
```
import folium
import numpy as np
import pandas as pd
import os


def plotPointsOnMap(dataframe,beginIndex,endIndex,latitudeColumn,latitudeValue,longitudeColumn,longitudeValue,zoom):
    df = dataframe[beginIndex:endIndex]
    location = [latitudeValue,longitudeValue]
    plot = folium.Map(location=location,zoom_start=zoom)
    for i in range(0,len(df)):
        popup = folium.Popup(str(df.primary_fuel[i:i+1]))
        folium.Marker([df[latitudeColumn].iloc[i],df[longitudeColumn].iloc[i]],popup=popup).add_to(plot)
    return(plot)
```

```
pos=np.where(df['primary_fuel']=='Solar')
everything=df.loc[pos]
india_latitudeLower = everything['latitude'] > 5
india_latitudeUpper = everything['latitude'] < 35
india_longitudeLower = everything['longitude'] > 68
india_longitudeUpper = everything['longitude'] < 95
india_only = everything[india_latitudeLower & india_latitudeUpper & india_longitudeLower & india_longitudeUpper]
plotPointsOnMap(india_only,0,906,'latitude',23,'longitude',73,5)
```

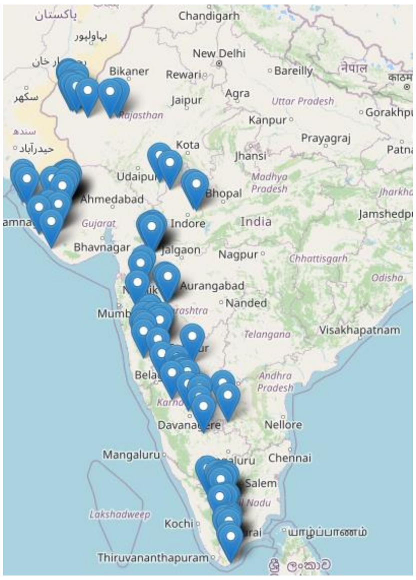
Solar



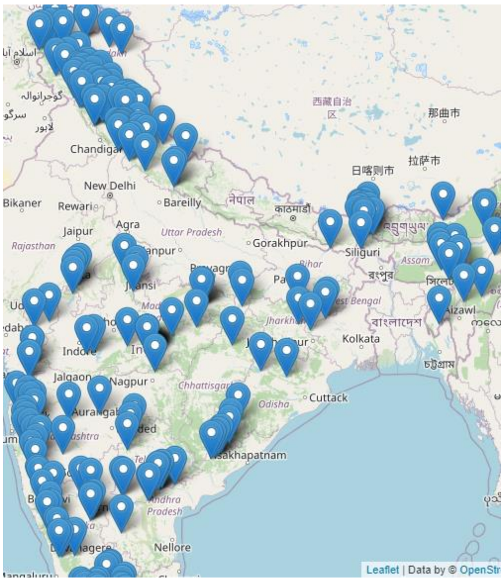
Coal



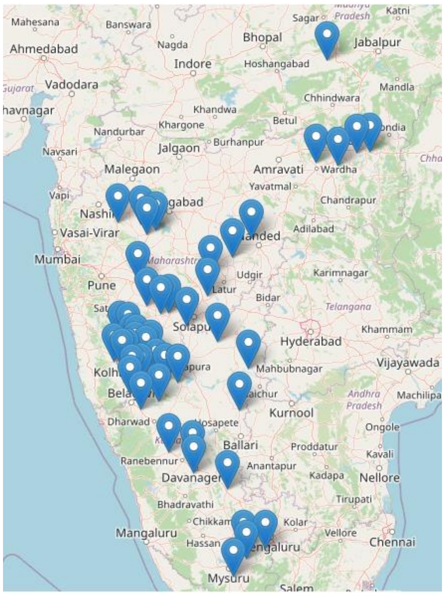
Wind



Hydro

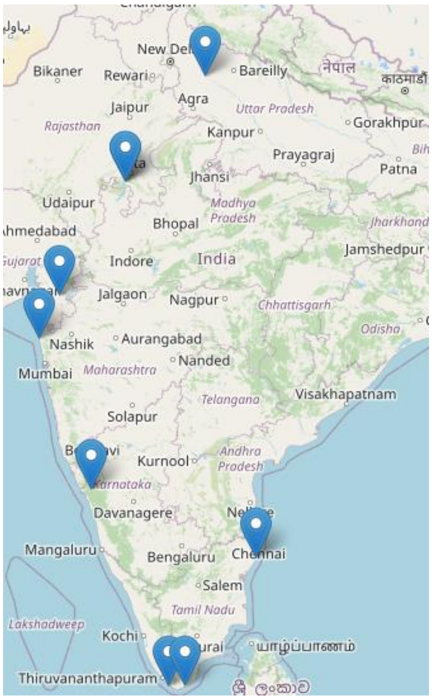


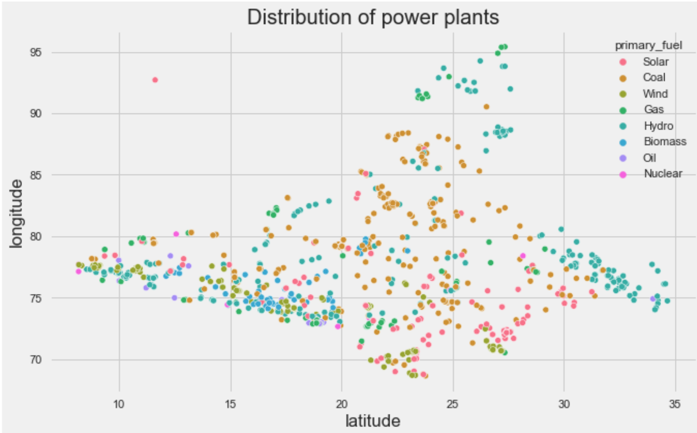
Biomass



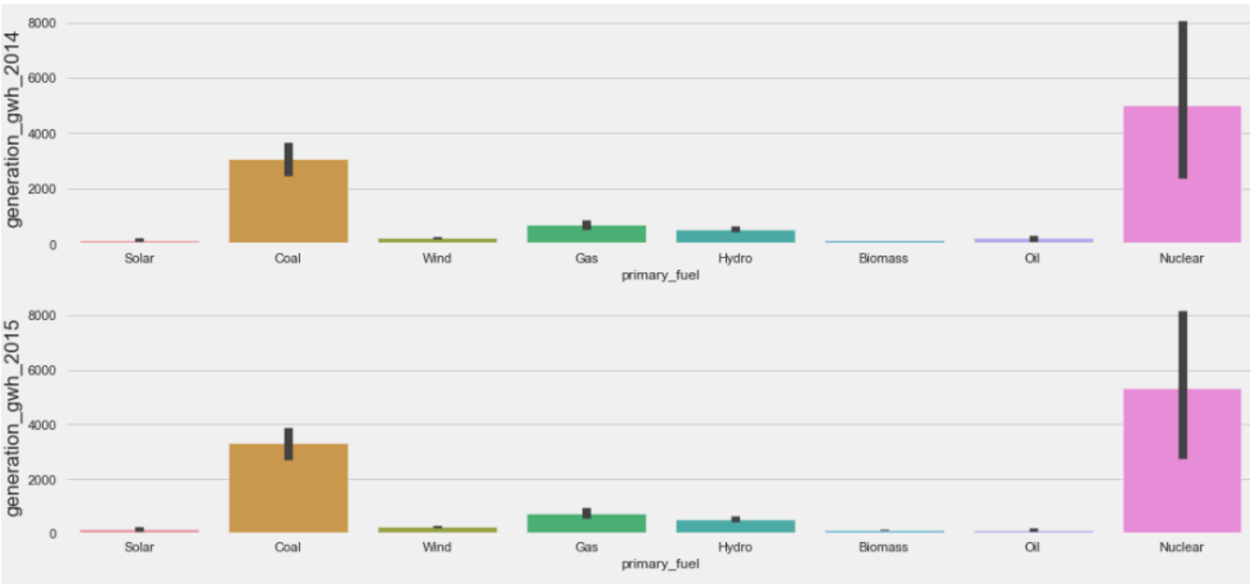
Oil

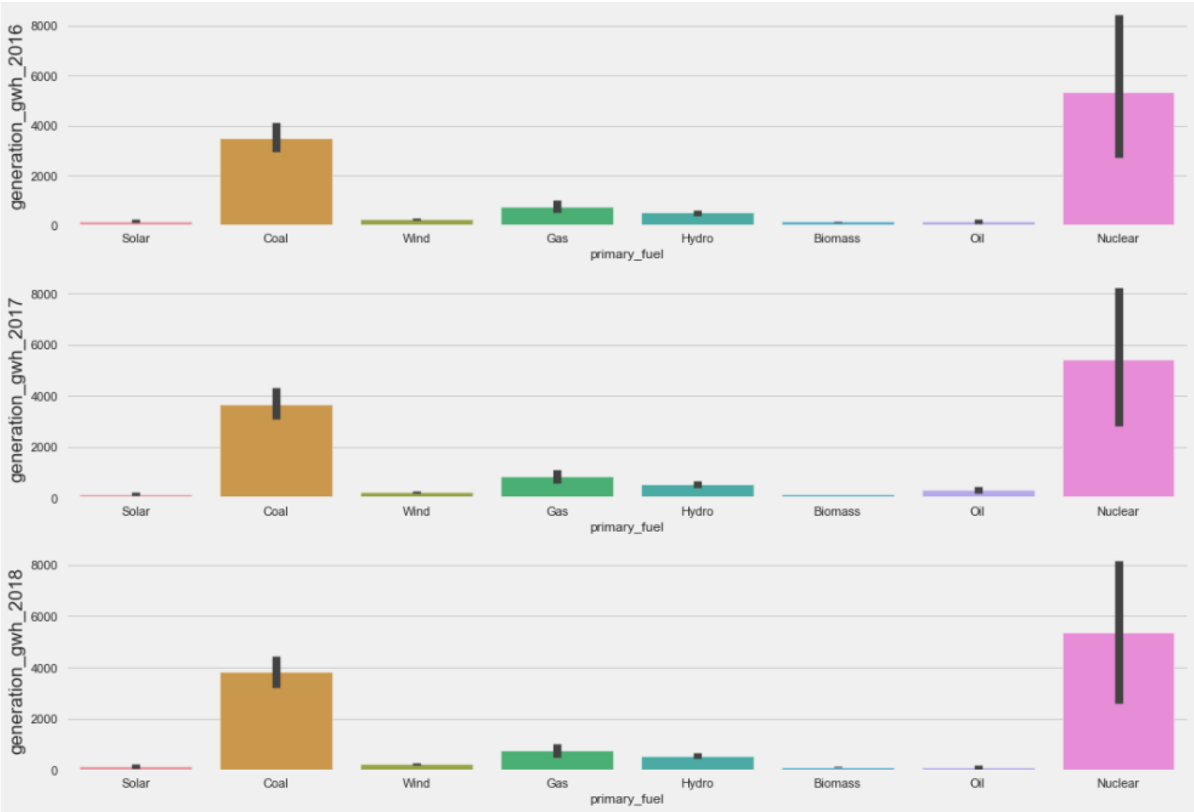
Nuclear





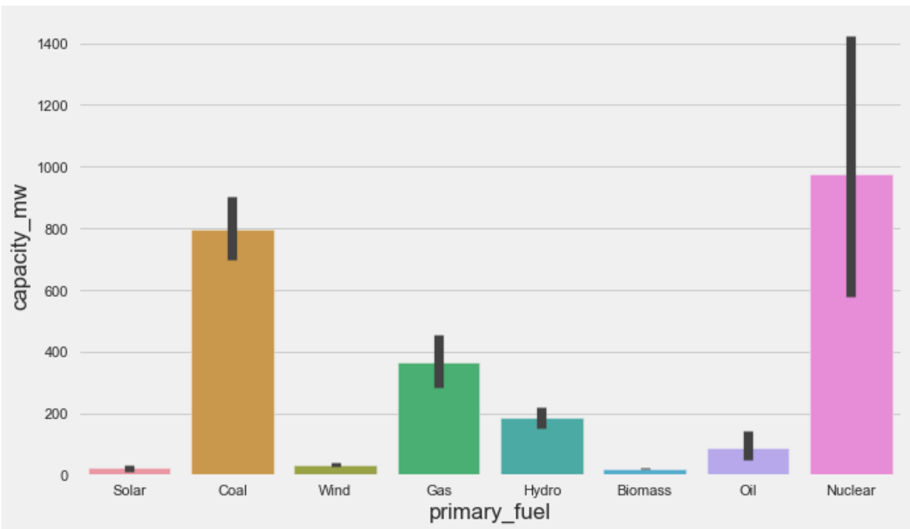
Power generated by power plants throughout the years





- the trend is same for all the years
- Nuclear power plant has the highest power generation even though they account for the least no. of power plant to be installed
- Coal power plant is the second highest power generation

Capacity and primary fuel



Nuclear power plant has the highest capacity followed by coal

- **Encoding**

```
p_f={'Solar':0, 'Coal':1, 'Wind':2, 'Gas':3, 'Hydro':4, 'Biomass':5, 'Oil':6, 'Nuclear':7}
df['primary_fuel'] = df['primary_fuel'].map(p_f)
```

Primary fuel was the only object data type present and encoded as above

- **check skewness**

the skewness for the numerical columns were checked and treated using power transform for those features which didnt confine with the limits

df.skew()[continuecol].sort_values()

```
latitude             -0.136277
longitude             1.135365
generation_gwh_2018   4.812275
generation_gwh_2016   4.937614
generation_gwh_2017   4.939805
generation_gwh_2014   4.944424
generation_gwh_2015   5.211376
```

using power transform to transform the dataset

```
#using power transform to transform and normalize the dataset and storing it in A and copying it to df
from sklearn.preprocessing import power_transform
B=df[continuecol].copy()
A=power_transform(B)
A=pd.DataFrame(A,columns=B.columns)
df[B.columns]=A.copy()
df
```

	capacity_mw	latitude	longitude	primary_fuel	commissioning_year	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2
0	2.5	1.157279	-0.935061	0	2011.000000	-0.949040	-1.043435	-1.111020	-1.201
1	98.0	0.582782	-0.528447	1	2006.021164	0.562891	0.528595	0.500524	0.465
2	39.2	0.107239	-2.365284	2	2012.000000	0.137878	0.092130	0.057339	0.012
3	135.0	0.433499	2.154880	3	2004.000000	0.512121	0.630711	0.628505	0.448
4	1800.0	0.116571	1.119213	1	2015.000000	-1.885575	1.675390	1.682375	1.570
...	...	...	...	...	...	...	...	...	
901	1600.0	-0.807181	0.166291	1	2016.000000	-1.885575	-2.048938	-0.018314	0.583
902	3.0	-1.348304	0.344092	0	2013.375000	-0.886190	-0.976929	-1.041926	-1.128
903	25.5	-0.970484	-0.265395	2	2012.000000	-0.050606	-0.102709	-0.141497	-0.191
904	80.0	0.513289	-0.779153	1	2006.021164	0.465961	0.429392	0.400057	0.363
905	16.5	-1.808635	0.193378	2	2012.000000	-0.234453	-0.293553	-0.336880	-0.393

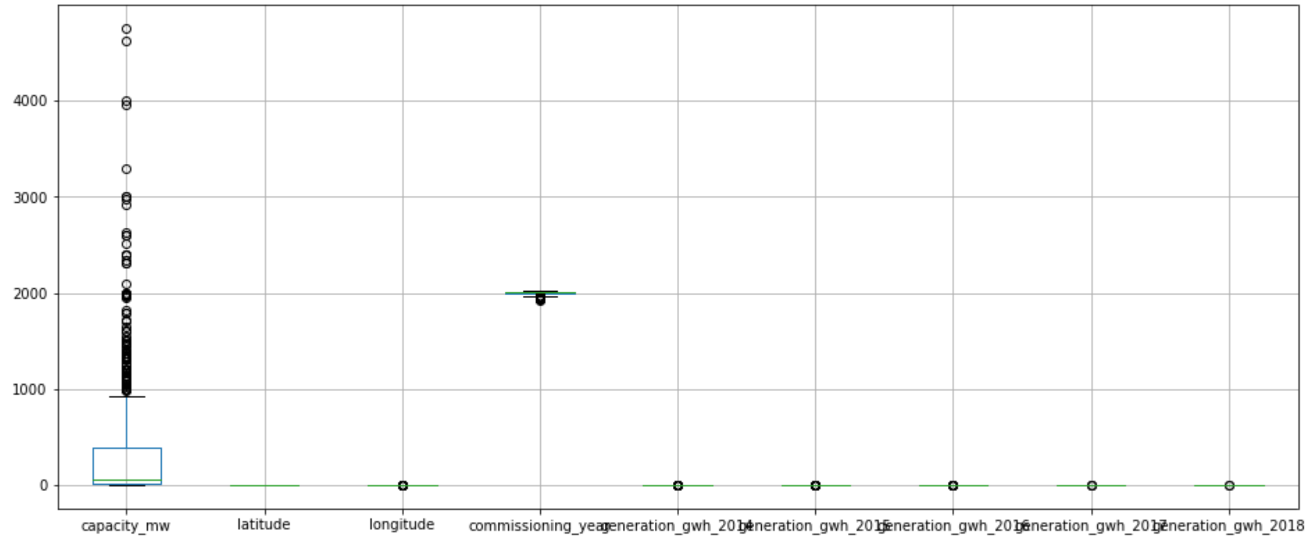
906 rows × 10 columns

After transforming the data the skewness was within limits (-0.65 to 0.65)

- **Outliers check**

Outliers were checked using box plot

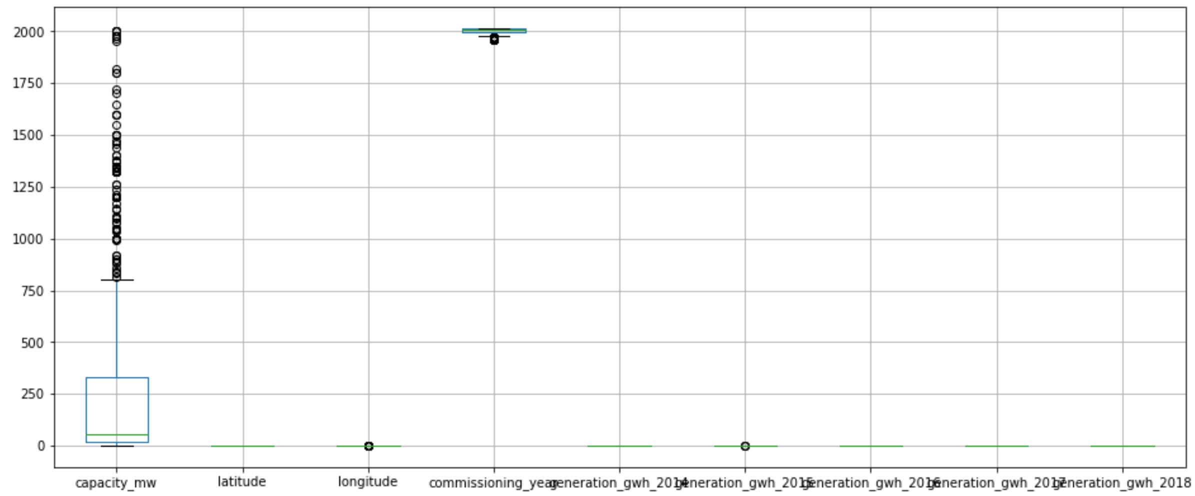
```
# visualizing
df[numericalCol].iloc[:,:].boxplot(figsize = (16,8))
plt.subplots_adjust(bottom=0.25)
plt.show()
```



It can be seen that there are outliers present. Outliers were treated using zscore method

```
from scipy.stats import zscore
z=np.abs(zscore(df[numericalCol]))
df_x=df[(z<3).all(axis=1)]
data_loss=((df.shape[0]-df_x.shape[0])/df.shape[0])*100
print("data loss ", data_loss, " %")
```

```
data loss  4.304635761589404  %
```



It can be seen that outliers have been reduced to a large extend, further reduction of outliers are not done as it seems not ideal to remove the outliers present in target variable capacity_mw

As a regression problem target is capacity_mw

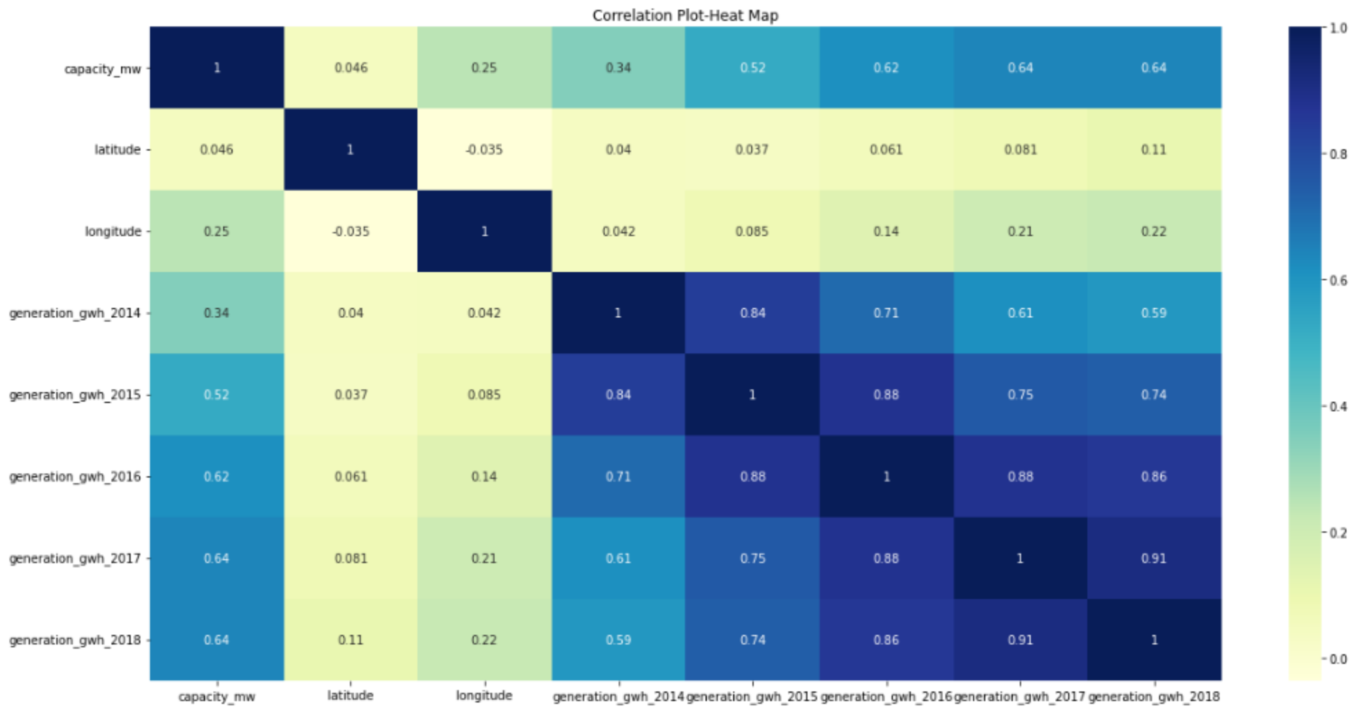
- **Check for correlation**

```
latitude               0.045716
longitude              0.245901
generation_gwh_2014    0.344925
generation_gwh_2015    0.522813
generation_gwh_2016    0.615157
generation_gwh_2017    0.641001
generation_gwh_2018    0.643428
capacity_mw            1.000000
```

It can be seen that latitude and longitude are least corelated to target

plotting heatmap to see the correlation feature to feature

```
plt.figure(figsize=(20,10))
sns.heatmap(df1[numer].corr(),annot=True,cmap="YlGnBu")
plt.title("Correlation Plot-Heat Map")
plt.show()
```



- generation_gwh_2014, generation_gwh_2015, generation_gwh_2016, generation_gwh_2017, generation_gwh_2018 has high correlation this is expected as the values imputed is the same
- need to check for multicollinearity

- **using vif method to remove multicollinearity**

	vif	features
4	9.211605	generation_gwh_2016
3	7.817567	generation_gwh_2015
5	7.674750	generation_gwh_2017
6	6.822085	generation_gwh_2018
2	3.501804	generation_gwh_2014
1	1.078486	longitude
0	1.021897	latitude

It can be seen that vif is within limits less than 10

Splitting the dataset into X and Y with Y having target variable and X having features except target

```
X=df_x.drop(['capacity_mw'],axis=1)
Y=df_x['capacity_mw']
```

running the algorithm, In each case the algorithm is fitted such that it picks the best random state having the highest r2score and cv_score having the least difference between test and train accuracy is selected

```
#importing necessary libriaries
#A=[] // stores test accuracy
#B=[] // stores cv_mean
#C=[] // stores mean_squared_error
#D=[] // min diff between test accuracy and cv_score
#mae=[] // stores mean_absolute_error

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
A=[]
B=[]
C=[]
D=[]
mae=[]
```

```python
#importing necessary librairies
#A=[] // stores test accuracy
#B=[] // stores cv_mean
#C=[] // stores mean_squared_error
#D=[] // min diff between test accuracy and cv_score
#mae=[] // stores mean_absolute_error

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
A=[]
B=[]
C=[]
D=[]
mae=[]
#loop used to find the best random state
def maxr2_score(regr,X,Y):
    max_r_score=0
    for r_state in range(0,100):
        x_train, x_test, y_train, y_test = train_test_split(X, Y,random_state = r_state,test_size=0.20)
        regr.fit(x_train,y_train)
        y_pred = regr.predict(x_test)
        r2_scr=(r2_score(y_test,y_pred))*100
        print("r2 score corresponding to ",r_state," is ",r2_scr)
        if r2_scr>max_r_score:
            max_r_score=r2_scr
            final_r_state=r_state
    print("max r2 score corresponding to ",final_r_state," is ",max_r_score)
    return final_r_state
```

```python
# used to get test accuracy, train accuracy, mse, mae
def te_t(regr,x_train,x_test,y_train,y_test,R):
    regr.fit(x_train,y_train)
    y_tr=regr.predict(x_train)
    y_te=regr.predict(x_test)
    print(f"test accuracy is {round(r2_score(y_test,y_te)*100,1)}")
    A.append(round(r2_score(y_test,y_te)*100,1))
    print(f"train accuracy is {round(r2_score(y_train,y_tr)*100,1)}")
    C.append(mean_squared_error(y_test,y_te))
    mae.append(mean_absolute_error(y_test,y_te))
```

```python
# used to find the best cv_score
def score(regr,x_train,x_test,y_train,y_test,R):
    max_cv_mean=0
    min_dif=100
    r=0
    k=0
    y_tr=regr.predict(x_train)
    y_te=regr.predict(x_test)
    t_ac=round(r2_score(y_train,y_tr)*100,1)
    te_ac=round(r2_score(y_test,y_te)*100,1)
    for j in range(2,20):
        cv_score=cross_val_score(regr,X,Y,cv=j)
        cv_mean=cv_score.mean()*100
        d=np.abs(cv_mean-te_ac)
        print(f"At cv is {j} cv score is {round(cv_mean,2)}  \n test accuracy is {te_ac} \n")
        if cv_mean>max_cv_mean:
            max_cv_mean=cv_mean
            k=j
        if d<min_dif:
            min_dif=d
            r=j
    B.append(max_cv_mean)
    print("min diff between test accuracy and cv score ",min_dif," at ", r," max cv ",max_cv_mean," at ",k)
    D.append(min_dif)
```

```python
from sklearn.tree import DecisionTreeRegressor
reg= DecisionTreeRegressor()
R=maxr2_score(reg,X,Y)
```

Similarly running the different algorithm

On running the algorithm, the following results are obtained

	test accuracy	cv_score	diff	mse	mae
ADA	72.9	67.716074	5.183926	62705.579905	213.522287
DT	80.9	73.316960	7.583040	31377.184270	75.857149
RF	91.6	83.768818	7.831182	15502.740713	62.005804
GRAD	91.7	82.750296	8.949704	15433.290355	69.232813
KNN	76.0	63.963485	12.036515	39346.769486	106.208039
LR	56.9	43.524272	13.375728	76524.821289	199.776474

inference

- random forest is the best model
    - second highest test accuracy score
    - highest cv_score
    - 3rd highest differnce between cv_score and test accuracy
    - least error compared to other models

Hyper parameter tuning

The following parameters were used for tuning

- criterion
- max_depth
- max_features
- min_samples_split
- n_estimators

the parameters are tuned using gridsearch cv

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

```
par={'n_estimators': [300,350,400,450],
     'max_features': ['log2', 'sqrt','auto'],
     'criterion': ["squared_error", "friedman_mse", "absolute_error","poisson"],
     'max_depth': [350,375,400,425],
     'min_samples_split':[2, 3, 4]
}
```

```
grid=GridSearchCV( rf_reg,par,cv=2)
grid.fit(x2_train,y2_train)
grid.best_params_
```

```
{'criterion': 'squared_error',
 'max_depth': 350,
 'max_features': 'sqrt',
 'min_samples_split': 4,
 'n_estimators': 300}
```

```
rf_reg=RandomForestRegressor( criterion= 'squared_error', max_depth= 350, max_features= 'sqrt',min_samples_split=4,n_estimators=
rf_reg.fit(x2_train,y2_train)
y_te=rf_reg.predict(x2_test)
r2=round(r2_score(y2_test,y_te)*100,2)
r2
```

```
92.22
```

```
cv_score=cross_val_score(rf_reg,X,Y,cv=8)
cv_mean=round(cv_score.mean()*100,2)
print(cv_mean)
```

```
84.39
```

Saving the model in pickle format

```
import pickle
filename='global_power_plant_reg.pkl'
pickle.dump(rf_reg,open(filename,'wb'))
```

```
l_m=pickle.load(open('global_power_plant_reg.pkl','rb'))
re=l_m.score(x2_test,y2_test)
print(re*100)
```

```
92.21603998739843
```

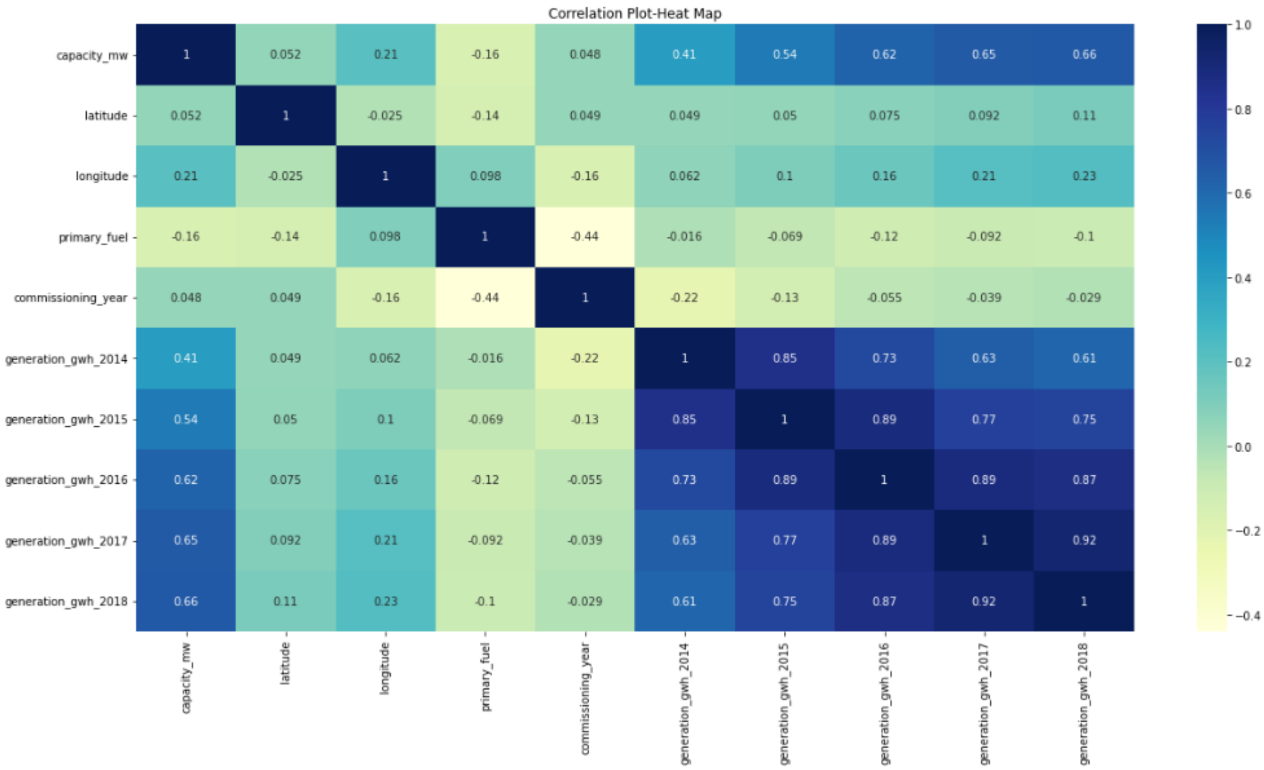
as classification problem target is Primary Fuel

- **Check for correlation**

```
commissioning_year    -0.438358
latitude              -0.143913
capacity_mw           -0.104138
generation_gwh_2016   -0.080943
generation_gwh_2018   -0.060084
generation_gwh_2017   -0.050399
generation_gwh_2015   -0.036193
generation_gwh_2014    0.016347
longitude              0.103212
primary_fuel           1.000000
```

generation_gwh_2017, generation_gwh_2015, generation_gwh_2014 and longitude has the least correlation with the target variable

- **plotting heatmap to see the correlation feature to feature**



- **using vif method to remove multicollinearity**

	vif	features
7	9.816182	generation_gwh_2017
8	8.596263	generation_gwh_2018
6	8.382506	generation_gwh_2016
5	3.920342	generation_gwh_2015
0	1.898963	capacity_mw
4	1.376403	generation_gwh_2014
3	1.310259	commissioning_year
2	1.130495	longitude
1	1.039463	latitude

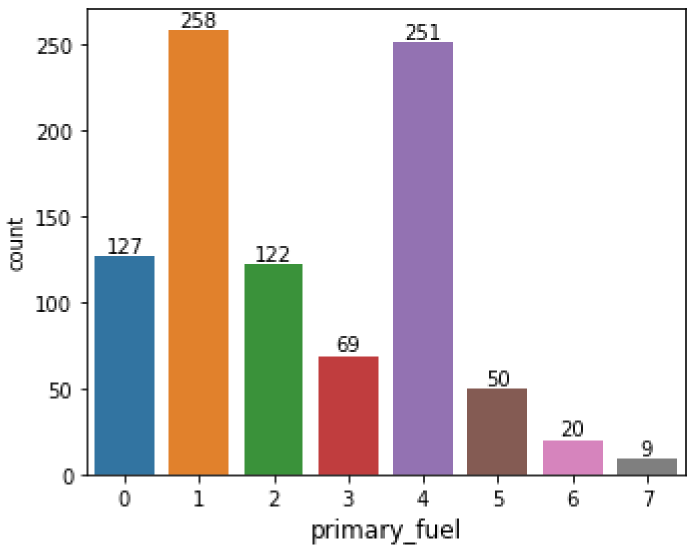
It can be seen that vif is within limits less than 10

Splitting the dataset into X and Y with Y having target variable and X having features except target

```
X=df.drop('primary_fuel',axis=1)
Y=df['primary_fuel']
```

as it is classification problem need to check if the classes in the target variable are balanced

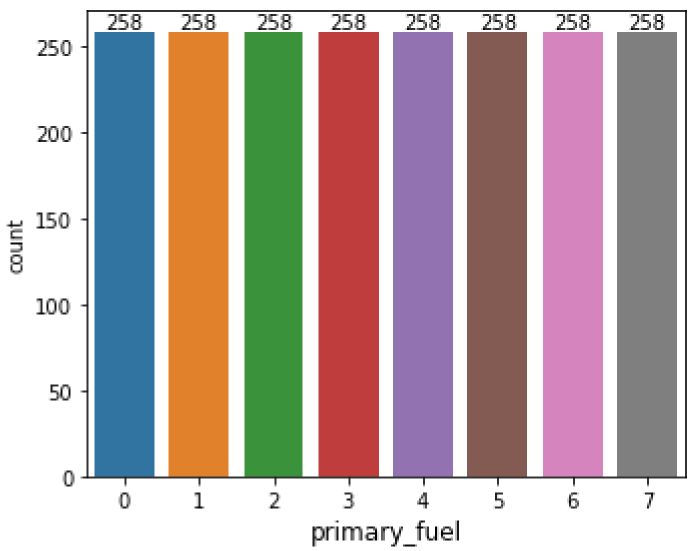
```
# plotting countplot graph
plt.figure(figsize=(5,4))
ax =sns.countplot(df['primary_fuel'])
ax.bar_label(ax.containers[0]);
plt.xlabel('primary_fuel',fontsize=12)
plt.tight_layout()
```



It can be seen that it is not balanced

Using smote to balance the classes

```
# plotting countplot graph
plt.figure(figsize=(5,4))
ax =sns.countplot(Y)
ax.bar_label(ax.containers[0]);
plt.xlabel('primary_fuel',fontsize=12)
plt.tight_layout()
```



It is now balanced

running the algorithm, In each case the algorithm is fitted such that it picks the best random state having the highest accuracy score and cv_score having the least difference between test and train accuracy is selected

```
#importing necessary libririries
#A=[] // stores test accuracy
#B=[] // stores cv_mean
#C=[] // stores mean_squared_error
#D=[] // min diff between test accuracy and cv_score
#mae=[] // stores mean_absolute_error


from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
A=[]
B=[]
C=[]
D=[]
E=[]
mae=[]
```

```
#loop used to find the best random state
def max_aucroc_score(regr,X,Y):
    max_aucroc_score=0
    for r_state in range(0,100):
        x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state = r_state, test_size=0.20,stratify=Y)
        regr.fit(x_train,y_train)
        y_pred = regr.predict(x_test)
        aucroc_scr=(accuracy_score(y_test,y_pred))*100
        print("accuracy score corresponding to ",r_state," is ",aucroc_scr)
        if aucroc_scr>max_aucroc_score:
            max_aucroc_score=aucroc_scr
            final_r_state=r_state
    print("max accuracy score corresponding to ",final_r_state," is ",max_aucroc_score)
    return final_r_state
```

```
# used to get test accuracy, train accuracy, mse, mae,F1score,confusion matrix,classification report and auc score
def te_t(regr,x_train,x_test,y_train,y_test):
    regr.fit(x_train,y_train)
    y_tr=regr.predict(x_train)
    y_te=regr.predict(x_test)
    print(f"test accuracy is {round(accuracy_score(y_test,y_te)*100,1)}")
    A.append(round(accuracy_score(y_test,y_te)*100,1))
    print(f"train accuracy is {round(accuracy_score(y_train,y_tr)*100,1)}")
    C.append(mean_squared_error(y_test,y_te))
    mae.append(mean_absolute_error(y_test,y_te))
    print("Confusion matrix \n",confusion_matrix(y_test,y_te))
    print('\n')
    print("classification report \n",classification_report(y_test,y_te))
```

```
# used to find the best cv_score
def score(regr,x_train,x_test,y_train,y_test):
    max_cv_mean=0
    min_dif=100
    r=0
    k=0
    y_tr=regr.predict(x_train)
    y_te=regr.predict(x_test)
    t_ac=round(accuracy_score(y_train,y_tr)*100,1)
    te_ac=round(accuracy_score(y_test,y_te)*100,1)
    for j in range(2,20):
        cv_score=cross_val_score(regr,X,Y,cv=j)
        cv_mean=cv_score.mean()*100
        d=np.abs(cv_mean-te_ac)
        print(f"At cv is {j} cv score is {round(cv_mean,2)}  \n test accuracy is {te_ac} \n")
        if cv_mean>max_cv_mean:
            max_cv_mean=cv_mean
            k=j
        if d<min_dif:
            min_dif=d
            r=j
    B.append(max_cv_mean)
    print("min diff between test accuracy and cv score ",min_dif," at ", r," max cv ",max_cv_mean," at ",k)
    D.append(min_dif)
```

```
from sklearn.tree import DecisionTreeClassifier
reg=DecisionTreeClassifier()
R=max_aucroc_score(reg,X,Y)
```

Similarly the other algorithm are run

The following results were obtained after running the algorithm

	test accuracy	max_cv_score	diff	mse	mae
DT	87.9	88.032946	0.011358	0.796610	0.283293
KNN	76.8	77.906977	0.043871	2.648910	0.702179
RF	90.8	92.832949	0.624419	0.731235	0.242131
ADA	24.2	29.312016	1.042248	6.000000	1.893462

inference

- random forest is the best model
    - highest test accuracy and cv_score
    - very low error compared to other models

Hyper parameter tuning

The following parameters were used

- criterion
- max_depth
- max_features
- min_samples_split
- n_estimators

tuning improved the cv_score such that difference between cv_Score and test accuracy became 0.01

the model is stored in pickle format and can be loaded for later use

Storing model and loading it

```
import pickle
filename='global_power_plant_c.pkl'
pickle.dump(rf,open(filename,'wb'))
```

```
l_m=pickle.load(open('global_power_plant_c.pkl','rb'))
re=l_m.score(x2_test,y2_test)
print(re*100)
```

```
92.00968523002422
```