Problem Statement:

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.

Importing Required Library

In [125]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from scipy.stats import zscore
import scikitplot as skplt
pd.set_option('display.max_columns', None) # # For display maximum column
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sco
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, plot
import xgboost as xgb
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Reading Data

In [2]:
```
df = pd.read_csv(r"C:\Users\Kushal Arya\Desktop\csv file\database_IND.csv")
df.head()
```

Out[2]:

	country	country_long	name	gppd_idnr	capacity_mw	latitude	longitude	primary_fuel
0	IND	India	ACME Solar Tower	WRI1020239	2.5	28.1839	73.2407	Solar
1	IND	India	ADITYA CEMENT WORKS	WRI1019881	98.0	24.7663	74.6090	Coal
2	IND	India	AES Saurashtra Windfarms	WRI1026669	39.2	21.9038	69.3732	Wind
3	IND	India	AGARTALA GT	IND0000001	135.0	23.8712	91.3602	Gas
4	IND	India	AKALTARA TPP	IND0000002	1800.0	21.9603	82.4091	Coal

Check no of row and column

In [3]:
```
print('No of Rows and Columns ----->', df.shape )
```

```
No of Rows and Columns -----> (908, 25)
```

Checking for Null values

In [4]:
```python
print('----------------------\n')
print(df.isnull().sum())
print('\n----------------------')
```

```
----------------------

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               907
other_fuel3               908
commissioning_year        380
owner                     566
source                      0
url                         0
geolocation_source         19
wepp_id                   908
year_of_capacity_data     388
generation_gwh_2013       524
generation_gwh_2014       507
generation_gwh_2015       483
generation_gwh_2016       471
generation_gwh_2017       465
generation_data_source    458
estimated_generation_gwh  908
dtype: int64

----------------------
```

There is null value

Information about dataset

In [5]:
```
print('------------------------------------------------------\n')
print(df.info())
print('------------------------------------------------------')
```

```
------------------------------------------------------

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 908 entries, 0 to 907
Data columns (total 25 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   country                   908 non-null    object 
 1   country_long              908 non-null    object 
 2   name                      908 non-null    object 
 3   gppd_idnr                 908 non-null    object 
 4   capacity_mw               908 non-null    float64
 5   latitude                  862 non-null    float64
 6   longitude                 862 non-null    float64
 7   primary_fuel              908 non-null    object 
 8   other_fuel1               199 non-null    object 
 9   other_fuel2               1 non-null      object 
 10  other_fuel3               0 non-null      float64
 11  commissioning_year        528 non-null    float64
 12  owner                     342 non-null    object 
 13  source                    908 non-null    object 
 14  url                       908 non-null    object 
 15  geolocation_source        889 non-null    object 
 16  wepp_id                   0 non-null      float64
 17  year_of_capacity_data     520 non-null    float64
 18  generation_gwh_2013       384 non-null    float64
 19  generation_gwh_2014       401 non-null    float64
 20  generation_gwh_2015       425 non-null    float64
 21  generation_gwh_2016       437 non-null    float64
 22  generation_gwh_2017       443 non-null    float64
 23  generation_data_source    450 non-null    object 
 24  estimated_generation_gwh  0 non-null      float64
dtypes: float64(13), object(12)
memory usage: 177.5+ KB
None
------------------------------------------------------
```

Categorical data present in our data set

Statistics of Data

In [6]:
```
df.describe()
```

Out[6]:

	capacity_mw	latitude	longitude	other_fuel3	commissioning_year	wepp_id	year_of_ca
count	908.000000	862.000000	862.000000	0.0	528.000000	0.0	
mean	321.046378	21.196189	77.447848	NaN	1996.876894	NaN	
std	580.221767	6.248627	4.907260	NaN	17.047817	NaN	
min	0.000000	8.168900	68.644700	NaN	1927.000000	NaN	
25%	16.837500	16.771575	74.258975	NaN	1988.000000	NaN	
50%	60.000000	21.778300	76.719250	NaN	2000.000000	NaN	
75%	388.125000	25.516375	79.441475	NaN	2011.250000	NaN	
max	4760.000000	34.649000	95.408000	NaN	2018.000000	NaN	

Outliers are present in our data set

Analysis of Null value

In [7]:
```python
print('----------------------\n')
print(df.isnull().sum())
print('\n----------------------')
```

```
----------------------

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                 907
other_fuel3                 908
commissioning_year          380
owner                       566
source                        0
url                           0
geolocation_source           19
wepp_id                     908
year_of_capacity_data       388
generation_gwh_2013         524
generation_gwh_2014         507
generation_gwh_2015         483
generation_gwh_2016         471
generation_gwh_2017         465
generation_data_source      458
estimated_generation_gwh    908
dtype: int64

----------------------
```

In [8]:
```python
df['other_fuel1'].value_counts()
```

Out[8]:
```
Oil             196
Gas               2
Cogeneration      1
Name: other_fuel1, dtype: int64
```

In [9]:
```python
df['other_fuel2'].value_counts()
```

Out[9]:
```
Oil    1
Name: other_fuel2, dtype: int64
```

In [10]:
```python
df['other_fuel3'].value_counts()
```

Out[10]:
```
Series([], Name: other_fuel3, dtype: int64)
```

Approach : We drop above other_fuel2 and other_fuel3 column because maximun null value and keep other_fuel1 and fill null value with mode

In [11]:
```
df['latitude'].value_counts()
```

Out[11]:
```
24.1917    3
19.0004    3
16.5697    2
23.4639    2
13.2450    2
          ..
20.9099    1
17.2387    1
23.5594    1
27.3426    1
16.5973    1
Name: latitude, Length: 837, dtype: int64
```

In [12]:
```
df['longitude'].value_counts()
```

Out[12]:
```
71.6917    4
71.6918    3
75.8988    3
72.8983    3
81.2875    3
          ..
80.1264    1
76.1137    1
74.6447    1
86.0970    1
79.5748    1
Name: longitude, Length: 828, dtype: int64
```

Approach : We fill above longitude and latitude columns with mean

In [13]:
```
df['commissioning_year'].value_counts()
```

Out[13]:
```
2013.0    28
2015.0    26
2012.0    23
2016.0    21
2014.0    17
          ..
1958.0     1
1949.0     1
1954.0     1
1956.0     1
1927.0     1
Name: commissioning_year, Length: 73, dtype: int64
```

Approach: We fill commissioning_year column with mode

In [14]:
```
df['owner'].value_counts()
```

Out[14]:
```
Acc Acc ltd                                 4
Sterling Agro Industries ltd.               4
Jk Cement ltd                               4
Tata Power Solar Systems Limited (TPREL)    3
Ujaas Energy Limited                        3
                                           ..
Gm Energy ltd                               1
Dcm & chem                                  1
REI Agro Limited                            1
National And paper                          1
Spr Pvt ltd                                 1
Name: owner, Length: 280, dtype: int64
```

Approach : We fill owner column with mode

In [15]:
```
df['geolocation_source'].value_counts()
```

Out[15]:
```
WRI                                     766
Industry About                          119
National Renewable Energy Laboratory      4
Name: geolocation_source, dtype: int64
```

Approach : We fill geolocation_source column with mode

In [16]:
```
df['wepp_id'].value_counts()
```

Out[16]:
```
Series([], Name: wepp_id, dtype: int64)
```

Approach : We drop wepp_id column because maximum null value

In [17]:
```
df['year_of_capacity_data'].value_counts()
```

Out[17]:
```
2018.0    520
Name: year_of_capacity_data, dtype: int64
```

Approach : We fill year_of_capacity_data column with bfill beacuse only one data in it

In [18]:
```
df['generation_gwh_2013'].value_counts()
```

Out[18]:
```
0.00000        21
14881.88000     1
42.49645        1
2036.00000      1
97.73885        1
               ..
7229.33000      1
657.21740       1
507.89775       1
8556.42400      1
8211.00000      1
Name: generation_gwh_2013, Length: 364, dtype: int64
```

In [19]:
```
df['generation_gwh_2014'].value_counts()
```

Out[19]:
```
0.00000       28
6803.31250     1
4735.13000     1
145.81400      1
2022.57000     1
              ..
6224.00000     1
268.48085      1
1255.73200     1
164.32425      1
1153.65300     1
Name: generation_gwh_2014, Length: 374, dtype: int64
```

In [20]:
```
df['generation_gwh_2015'].value_counts()
```

Out[20]:
```
0.00000       28
5837.76600     1
1297.97750     1
8076.81050     1
1.09395        1
              ..
2636.86400     1
665.19730      1
1516.36010     1
741.86205      1
7130.50700     1
Name: generation_gwh_2015, Length: 398, dtype: int64
```

In [21]:
```
df['generation_gwh_2016'].value_counts()
```

Out[21]:
```
0.00000       31
8470.57000     2
1511.00000     2
7.31325        1
94.85500       1
              ..
433.84800      1
283.74811      1
259.94375      1
403.96000      1
307.87290      1
Name: generation_gwh_2016, Length: 405, dtype: int64
```

In [22]:
```
df['generation_gwh_2017'].value_counts()
```

Out[22]:
```
0.00000       33
170.08530      2
344.35955      1
2265.47000     1
59.43135       1
              ..
214.48220      1
272.73945      1
2887.00000     1
12.73600       1
158.73235      1
Name: generation_gwh_2017, Length: 410, dtype: int64
```

Approach : We fill above generation_gwh columns with mean

In [23]:
```
df['generation_data_source'].value_counts()
```

Out[23]:
```
Central Electricity Authority    450
Name: generation_data_source, dtype: int64
```

Approach : We fill above generation_data_source columns with bfill because only one data in it

In [24]:
```
df['estimated_generation_gwh'].value_counts()
```

Out[24]:
```
Series([], Name: estimated_generation_gwh, dtype: int64)
```

Approach : We drop estimated_generation_gwh columns because no data in it

Drop Unwanted column

In [25]:
```
col = ['estimated_generation_gwh', 'wepp_id', 'other_fuel2', 'other_fuel3', 'url'
```

In [26]:
```
df = df.drop(col, axis = 1)
df.head(2)
```

Out[26]:

	country_long	name	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commission
0	India	ACME Solar Tower	2.5	28.1839	73.2407	Solar	NaN	
1	India	ADITYA CEMENT WORKS	98.0	24.7663	74.6090	Coal	NaN	

In [27]:
```
print('After droping no of Rows and Columns  ----->', df.shape )
```

```
After droping no of Rows and Columns  -----> (908, 18)
```

Fill NaN

In [28]:
```
df = df.apply(lambda x:x.fillna(x.mean())if x.dtype == 'float' else x.fillna(x.va
```

In [29]:
```
print('----------------------\n')
print(df.isnull().sum())
print('\n----------------------')
```

```
----------------------

country_long              0
name                      0
capacity_mw               0
latitude                  0
longitude                 0
primary_fuel              0
other_fuel1               0
commissioning_year        0
owner                     0
source                    0
geolocation_source        0
year_of_capacity_data     0
generation_gwh_2013       0
generation_gwh_2014       0
generation_gwh_2015       0
generation_gwh_2016       0
generation_gwh_2017       0
generation_data_source    0
dtype: int64

----------------------
```

There is no null value left

Analysis of data respect to fuel type

In [30]:
```
df.head(2)
```

Out[30]:

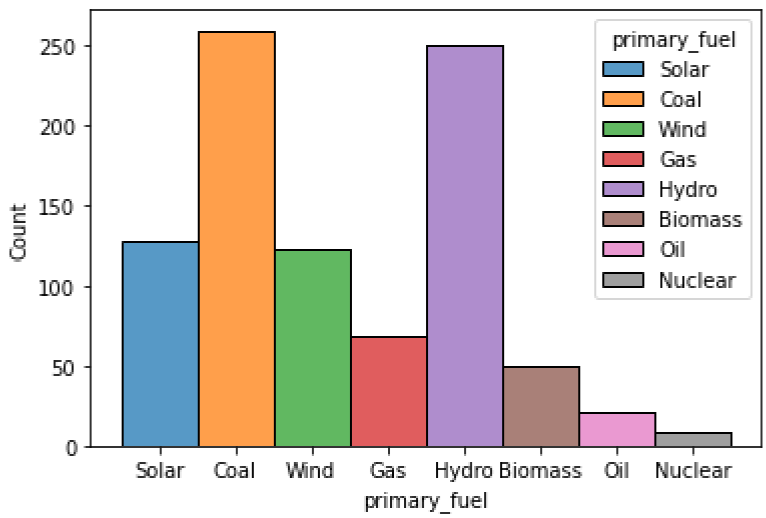
	country_long	name	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commission
0	India	ACME Solar Tower	2.5	28.1839	73.2407	Solar	Oil	201
1	India	ADITYA CEMENT WORKS	98.0	24.7663	74.6090	Coal	Oil	199

Fule Type column

In [31]:
```
df['primary_fuel'].value_counts()
```

Out[31]:
```
Coal        259
Hydro       250
Solar       127
Wind        123
Gas          69
Biomass      50
Oil          21
Nuclear       9
Name: primary_fuel, dtype: int64
```

In [32]:
```
sns.histplot(binwidth=0.5, x="primary_fuel", hue="primary_fuel", data=df, stat="c
plt.show()
```



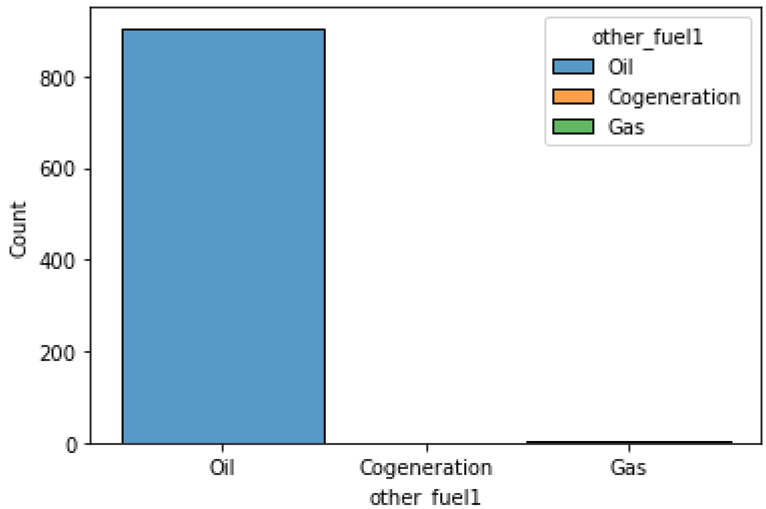
Cole type fuel used highest and Nuclear type fuel used least

Other Fuel Type column

In [33]:
```
df['other_fuel1'].value_counts()
```

Out[33]:
```
Oil             905
Gas               2
Cogeneration      1
Name: other_fuel1, dtype: int64
```

In [34]:
```
sns.histplot(binwidth=0.5, x="other_fuel1", hue="other_fuel1", data=df, stat="cou
plt.show()
```



Oil fuel used most of all fuel

Commission Year column

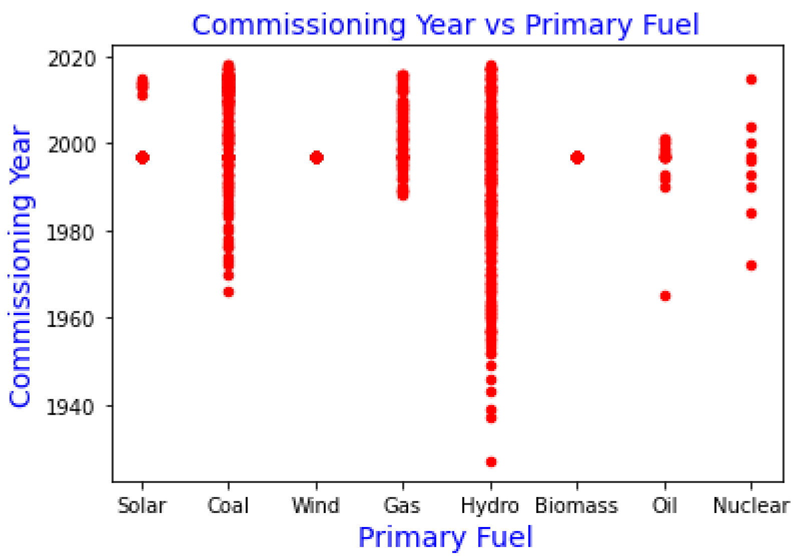
In [35]:
```
df['commissioning_year'].value_counts()
```

Out[35]:
```
1996.876894    380
2013.000000     28
2015.000000     26
2012.000000     23
2016.000000     21
              ...
1949.000000      1
1958.000000      1
1954.000000      1
1956.000000      1
1927.000000      1
Name: commissioning_year, Length: 74, dtype: int64
```

In [36]:
```
y = df.groupby('commissioning_year')['primary_fuel'].sum()
y
```

Out[36]:
```
commissioning_year
1927.0                                                   Hydro
1937.0                                                   Hydro
1939.0                                                   Hydro
1943.0                                                   Hydro
1946.0                                                   Hydro
                                ...
2014.0    CoalCoalCoalCoalCoalGasSolarSolarCoalCoalSolar...
2015.0    CoalCoalSolarHydroCoalCoalGasCoalCoalHydroHydr...
2016.0    CoalCoalCoalGasCoalCoalHydroGasGasCoalCoalCoal...
2017.0    CoalCoalHydroHydroCoalCoalHydroHydroHydroCoalC...
2018.0                                           HydroCoalCoal
Name: primary_fuel, Length: 74, dtype: object
```

In [37]:
```
df.plot.scatter(x = 'primary_fuel', y= 'commissioning_year', c = 'r')
plt.ylabel('Commissioning Year',fontsize = 14, color = 'b')
plt.xlabel('Primary Fuel',fontsize = 14, color = 'b')
plt.title('Commissioning Year vs Primary Fuel',fontsize = 14, color = 'b')
plt.show()
```



Above plot shows early years 1927 Hydro and Coal Fuel used but Now a days we used Solar and Gas Fuel most

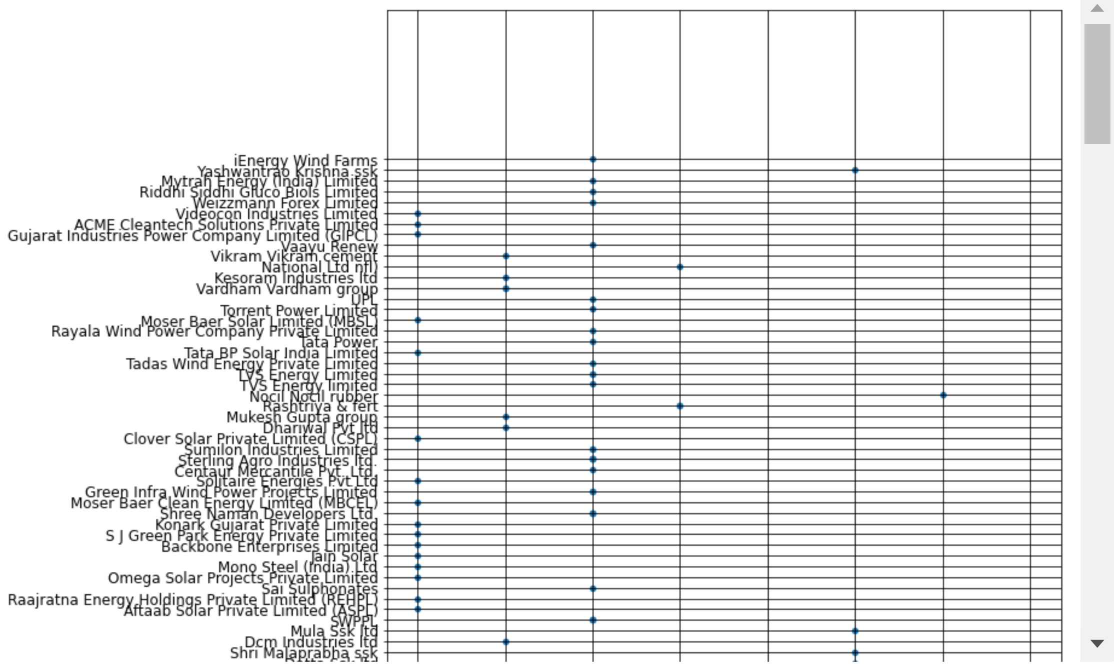
In [38]:
```python
df['owner'].value_counts()
```

Out[38]:
```
Acc Acc ltd                                  570
Sterling Agro Industries ltd.                  4
Jk Cement ltd                                  4
Tata Power Solar Systems Limited (TPREL)       3
Ujaas Energy Limited                           3
                                            ... 
Gm Energy ltd                                  1
Dcm & chem                                     1
REI Agro Limited                               1
National And paper                             1
Spr Pvt ltd                                    1
Name: owner, Length: 280, dtype: int64
```

In [39]:
```python
o = df.groupby('owner')['primary_fuel'].sum()
o
```

Out[39]:
```
owner
ACME Cleantech Solutions Private Limited            Solar
ACME Solar Energy                                   Solar
AES                                                  Wind
AEW Infratech Private Limited                       Solar
Abellon CleanEnergy Limited                         Solar
                                                   ...  
West Coast Paper Mills Ltd.                           Gas
Yashwantrao Krishna ssk                           Biomass
Ym Ssk ltd                                        Biomass
Zamil New Delhi Infrastructure Private Limited      Solar
iEnergy Wind Farms                                   Wind
Name: primary_fuel, Length: 280, dtype: object
```

```
In [40]: df.plot.scatter(y = 'owner', x = 'primary_fuel',figsize = (10,50), rot = 360, for
plt.grid(c = 'black')
plt.show()
```

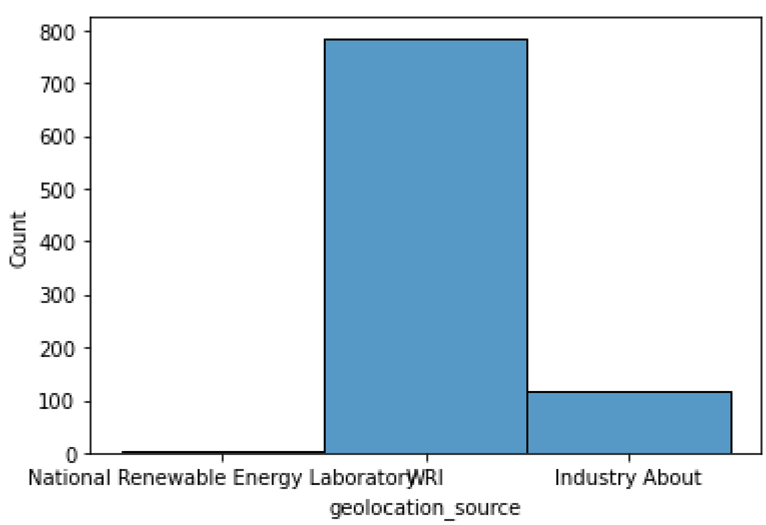


Above plot show most of the company used solar energy

Geolocation Source column

```
In [41]: g = df['geolocation_source'].value_counts()
```

```
In [42]: sns.histplot(binwidth=0.5, x="geolocation_source", data=df, stat="count", multipl
plt.show()
```

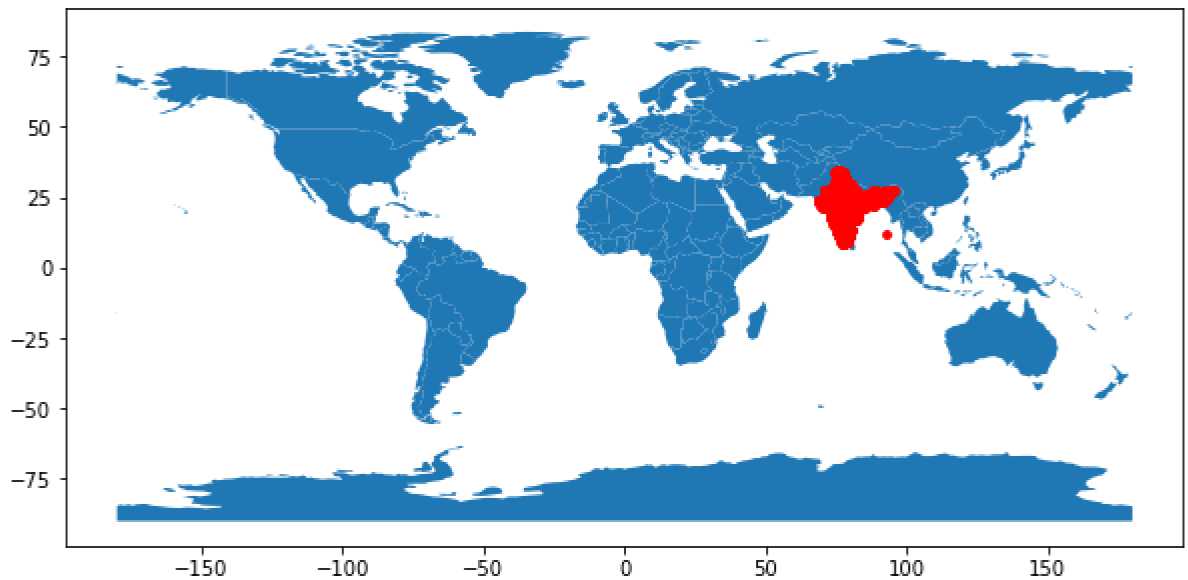


Above plot shows WRI highest among all

Co-ordinates

In [43]:
```
import geopandas as gpd
from shapely.geometry import Point, Polygon
import descartes
from geopandas import GeoDataFrame
from pyproj import CRS
```

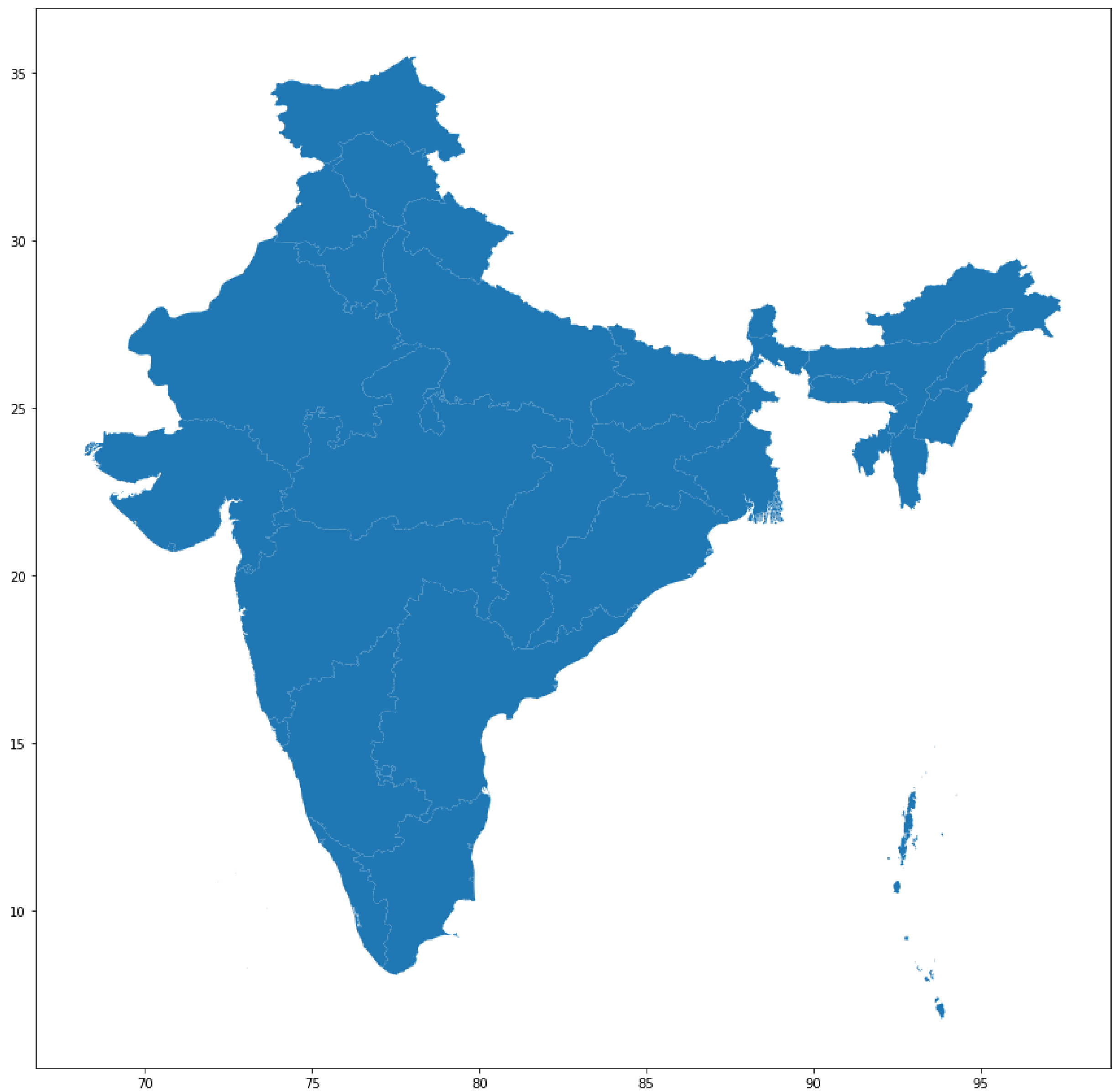
In [44]:
```
geometry = [Point(xy) for xy in zip(df['longitude'], df['latitude'])]
gdf = GeoDataFrame(df, geometry=geometry)

#this is a simple map that goes with geopandas
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
gdf.plot(ax=world.plot(figsize=(10, 6)), marker='o', color='red', markersize=15);
```



Above plot shows all cordinate belong to indian

In [45]:
```
indmap = gpd.read_file(r"C:\Users\Kushal Arya\Desktop\csv file\map.shx")
fig,ax = plt.subplots(figsize = (15,15))
indmap.plot(ax = ax)
```

Out[45]: <AxesSubplot:>



In [46]:
```
crs=CRS('EPSG:4326').to_proj4()
```

In [47]:
```
geometry = [Point(xy) for xy in zip(df['longitude'], df['latitude'])]
geometry[:3]
```

Out[47]:
```
[<shapely.geometry.point.Point at 0x1733acc1130>,
 <shapely.geometry.point.Point at 0x1733acc1250>,
 <shapely.geometry.point.Point at 0x1733acc1340>]
```

In [48]:
```
geo_df = gpd.GeoDataFrame(df, crs = crs, geometry = geometry)
geo_df.head()
```

Out[48]:

	country_long	name	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissic
0	India	ACME Solar Tower	2.5	28.1839	73.2407	Solar	Oil	2(
1	India	ADITYA CEMENT WORKS	98.0	24.7663	74.6090	Coal	Oil	1(
2	India	AES Saurashtra Windfarms	39.2	21.9038	69.3732	Wind	Oil	1(
3	India	AGARTALA GT	135.0	23.8712	91.3602	Gas	Oil	2(
4	India	AKALTARA TPP	1800.0	21.9603	82.4091	Coal	Oil	2(

In [49]:
```
try:
    fig, ax = plt.subplots(figsize = (15,15))
    indmap.plot(ax = ax, alpha = 0.4, color = 'grey')
    geo_df[geo_df['primary_fuel']].plot(ax = ax, markersize = 20, color = 'b', ma
    geo_df[geo_df['primary_fuel']].plot(ax = ax, markersize = 20, color = '^', ma
    plt.legend(prop = {'size': 15})
except:
    pass
```

In [50]:
```
import plotly
import plotly.graph_objects as go
import chart_studio.plotly as py
from plotly.offline import iplot
```

In [51]:
```
data = dict(type = 'choropleth',
         locations = df['geometry'],
         locationmode = 'country names',
         z = df['primary_fuel'],
         text = df['geometry'],
         colorscale = 'YlGnBu',
         autocolorscale = False,
        )
layout = dict(geo = {'scope':'asia'})

diagram = go.Figure(data = [data], layout = layout)
iplot(diagram)
```

In [52]:
```
from geopy.geocoders import Nominatim
```

In [53]:
```
# initialize Nominatim API
geolocator = Nominatim(user_agent="geoapiExercises")
```

In [54]:
```
# Latitude & Longitude input
Latitude = "28.1839"
Longitude = "73.2407"

location = geolocator.reverse(Latitude+","+Longitude)

address = location.raw['address']

# traverse the data
city = address.get('city', '')
state = address.get('state', '')
country = address.get('country', '')
code = address.get('country_code')
zipcode = address.get('postcode')
print('City : ', city)
print('State : ', state)
print('Country : ', country)
print('Zip Code : ', zipcode)
```

```
City :
State :  Rajasthan
Country :  India
Zip Code :  None
```

Drop Columns

In [55]:
```
col = ['geometry', 'generation_data_source', 'source', 'owner', 'name', 'country_
```

In [56]:
```
df = df.drop(col, axis = 1)
df.head(2)
```

Out[56]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	year_of_capacit
0	2.5	28.1839	73.2407	Solar	Oil	2011.000000	
1	98.0	24.7663	74.6090	Coal	Oil	1996.876894	

In [57]:
```
print('After droping no of Rows and Columns  ----->', df.shape )
```

```
After droping no of Rows and Columns  -----> (908, 12)
```

Encoding columns

In [58]:
```
le = LabelEncoder()
```

In [59]:
```
df['primary_fuel'] = le.fit_transform(df['primary_fuel'])
```

In [60]:
```
df['other_fuel1'] = le.fit_transform(df['other_fuel1'])
```

In [61]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 908 entries, 0 to 907
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   capacity_mw             908 non-null    float64
 1   latitude                908 non-null    float64
 2   longitude               908 non-null    float64
 3   primary_fuel            908 non-null    int32  
 4   other_fuel1             908 non-null    int32  
 5   commissioning_year      908 non-null    float64
 6   year_of_capacity_data   908 non-null    float64
 7   generation_gwh_2013     908 non-null    float64
 8   generation_gwh_2014     908 non-null    float64
 9   generation_gwh_2015     908 non-null    float64
 10  generation_gwh_2016     908 non-null    float64
 11  generation_gwh_2017     908 non-null    float64
dtypes: float64(10), int32(2)
memory usage: 78.2 KB
```

In [62]:
```
df.head(2)
```

Out[62]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	year_of_capacit
0	2.5	28.1839	73.2407	6	2	2011.000000	
1	98.0	24.7663	74.6090	1	2	1996.876894	

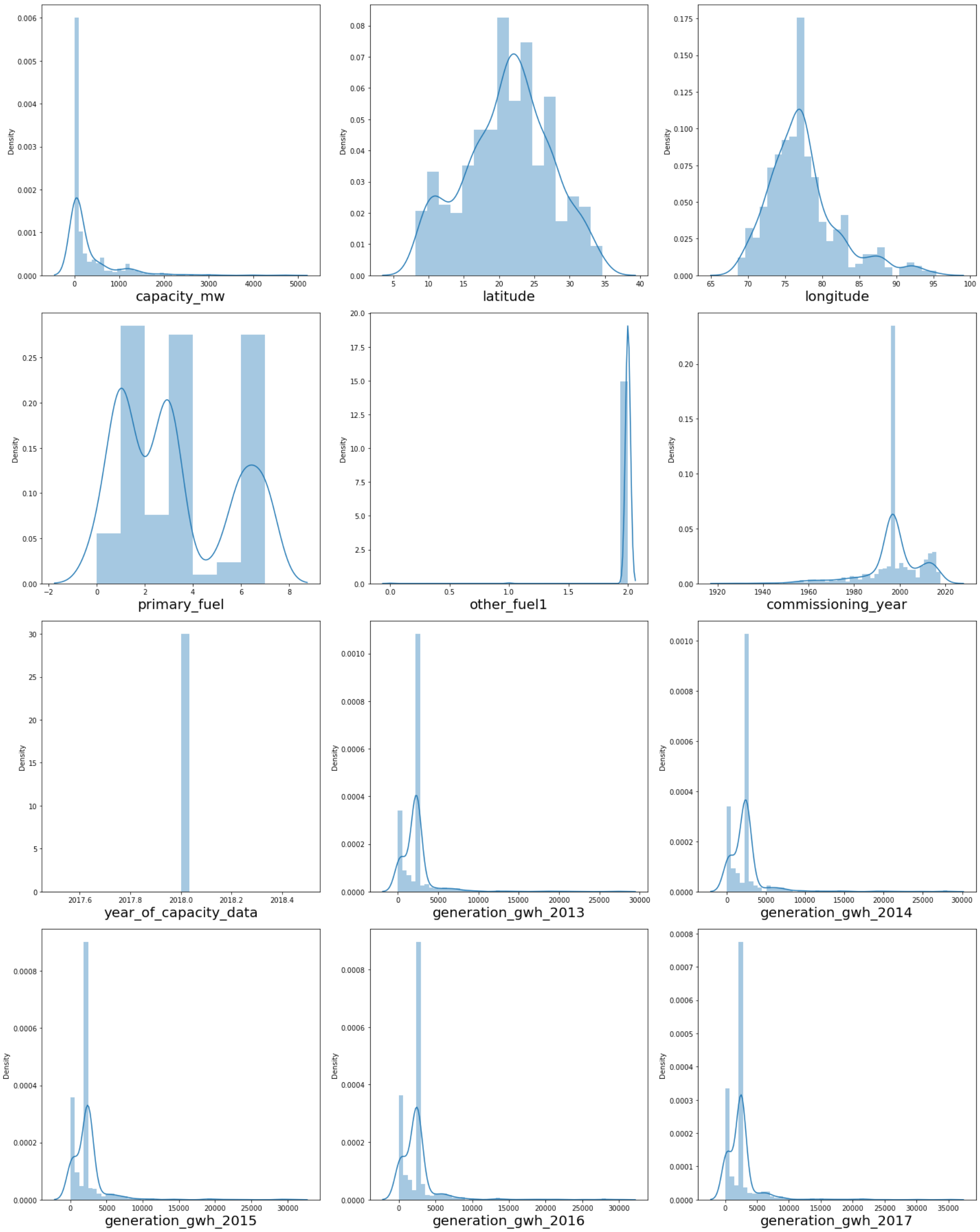
All columns are encoded

Data distribution and checking outliers and skewness

In [63]:

```python
print('---------------------')
print('Distribution Plot :- ')
print('---------------------')

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

for column in df:
    if plotnumber <=12:
        ax = plt.subplot(4,3, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column, fontsize = 20)
    plotnumber +=1
plt.tight_layout()
```

```
---------------------
Distribution Plot :-
---------------------
```


Density
capacity_mw
latitude
longitude
primary_fuel
other_fuel1
commissioning_year
year_of_capacity_data
generation_gwh_2013
generation_gwh_2014
generation_gwh_2015
generation_gwh_2016
generation_gwh_2017

In [64]:
```
df.skew()
```

Out[64]:
```
capacity_mw               3.193257
latitude                 -0.147391
longitude                 1.129836
primary_fuel              0.471141
other_fuel1             -20.464435
commissioning_year       -1.383330
year_of_capacity_data     0.000000
generation_gwh_2013       5.241491
generation_gwh_2014       5.041961
generation_gwh_2015       5.367370
generation_gwh_2016       5.071758
generation_gwh_2017       5.111938
dtype: float64
```

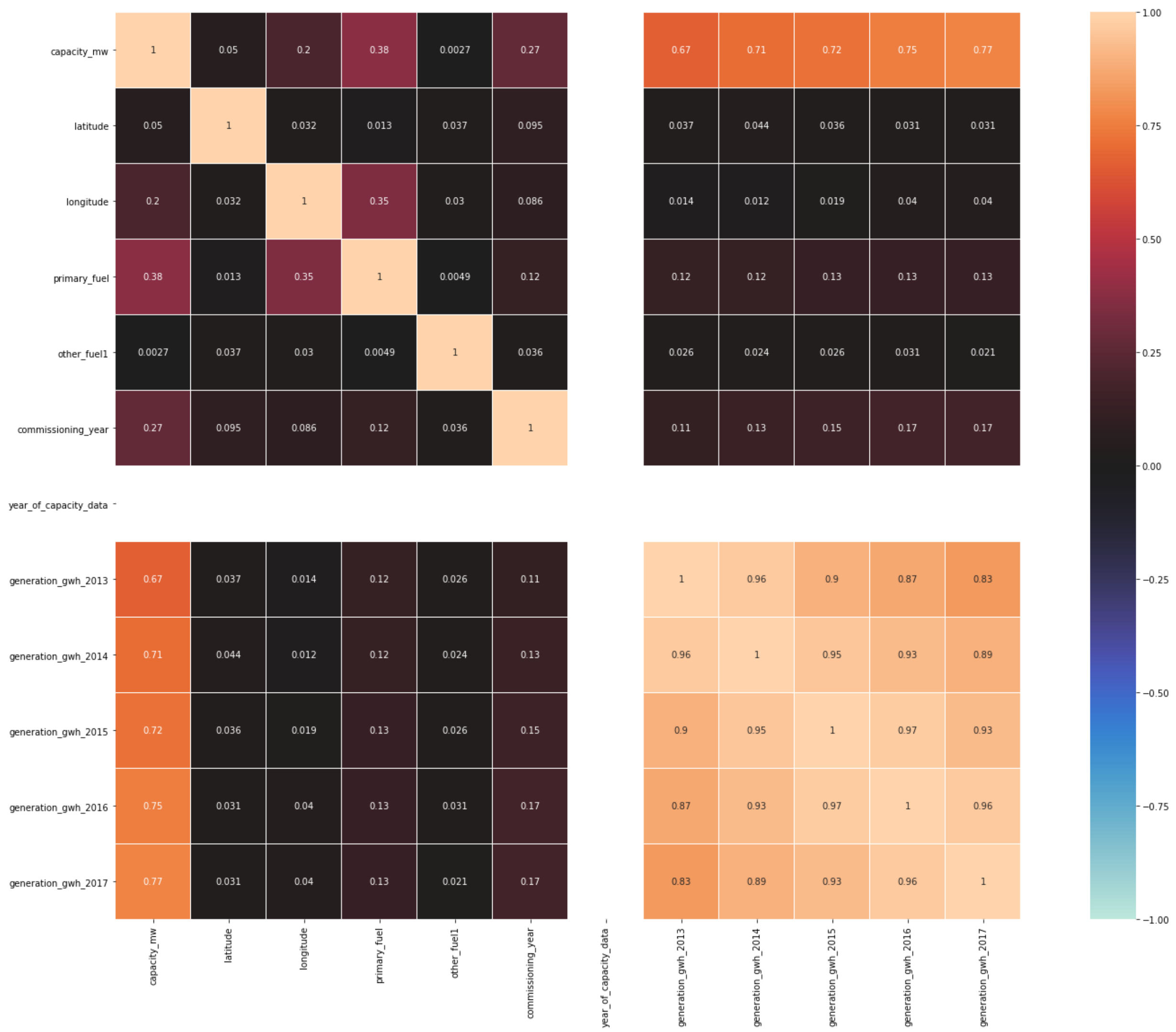
Data has outliers and skewness

Corelation of Feature vs Label using Heat map

In [65]:

```
print('-----------')
print('Heat Map :-')
print('-----------')
df_corr = df.corr().abs()

plt.figure(figsize = (22,16))
sns.heatmap(df_corr, vmin = -1, annot = True, square = True, center = 0, fmt = '.
plt.tight_layout()
```

```
-----------
Heat Map :-
-----------
```



generation_gwh_2013,generation_gwh_2014 higest relation

Removing Outliers using Zscore

In [66]:
```
# with std 3 Lets see the stats

z_score = zscore(df[['capacity_mw', 'longitude', 'generation_gwh_2013', 'generati
abs_z_score = np.abs(z_score)

filtering_entry = (abs_z_score < 3).all(axis = 1)

df = df[filtering_entry]

df.describe()
```

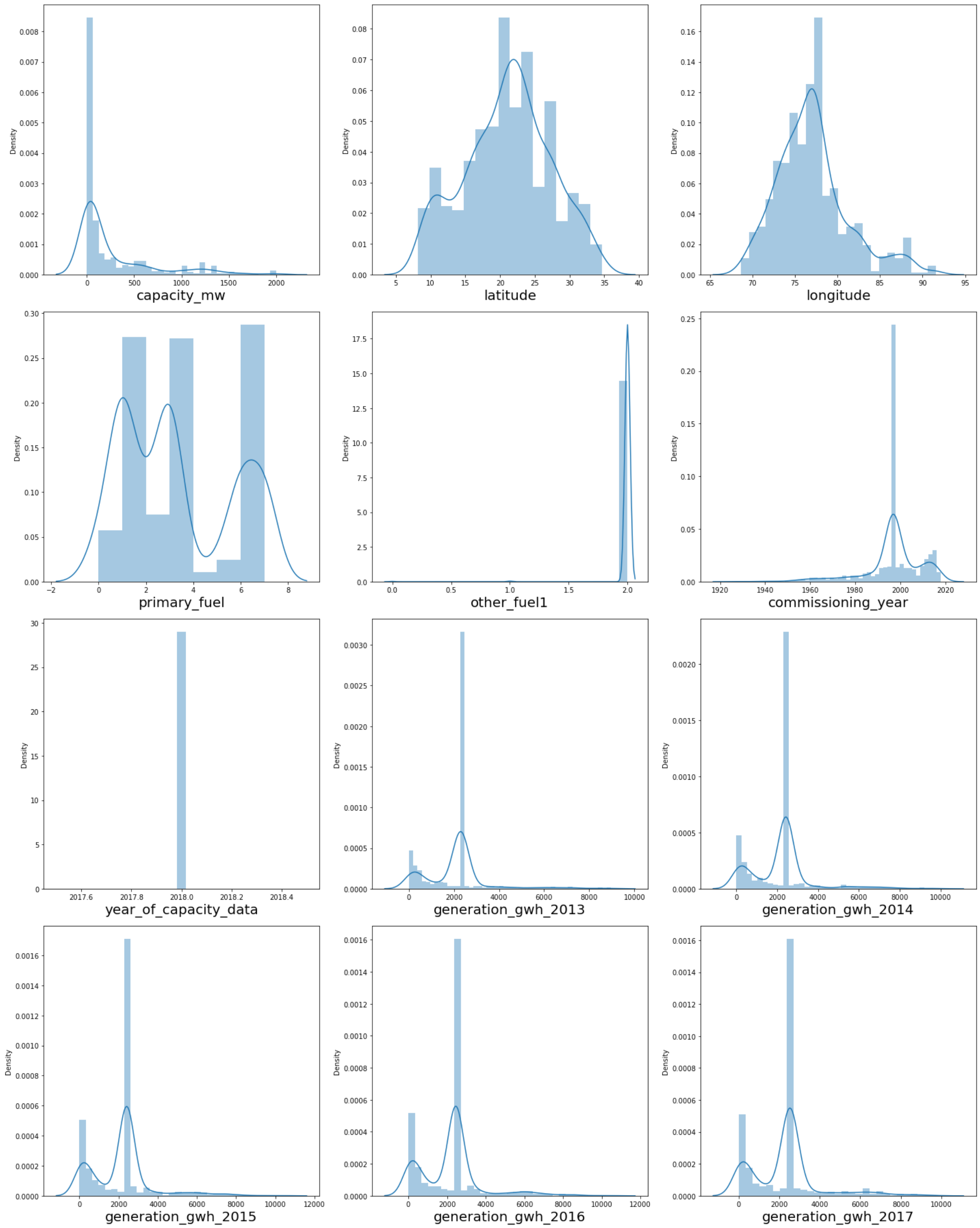
Out[66]:

	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	commissioning_year	year_c
count	867.000000	867.000000	867.000000	867.000000	867.000000	867.000000	
mean	262.860970	21.093784	77.025306	3.267589	1.995386	1996.778942	
std	416.438061	6.157370	4.200192	2.303707	0.083109	12.996286	
min	0.000000	8.168900	68.644700	0.000000	0.000000	1927.000000	
25%	16.500000	16.899050	74.318150	1.000000	2.000000	1996.876894	
50%	50.400000	21.196189	76.737000	3.000000	2.000000	1996.876894	
75%	317.500000	25.134100	78.906250	6.000000	2.000000	2002.000000	
max	2000.000000	34.649000	91.565000	7.000000	2.000000	2018.000000	

Checking Outliers and skewness removed or not

In [67]:
```
# Let' see outliers are removed in columns or not.
print('---------------------')
print('Distribution Plot :- ')
print('---------------------')

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

for column in df:
    if plotnumber <=12:
        ax = plt.subplot(4,3, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column, fontsize = 20)
    plotnumber +=1
plt.tight_layout()
```

```
---------------------
Distribution Plot :- 
---------------------
```



Outliers are removed

Spliting Dataset into features and labels

In [68]:

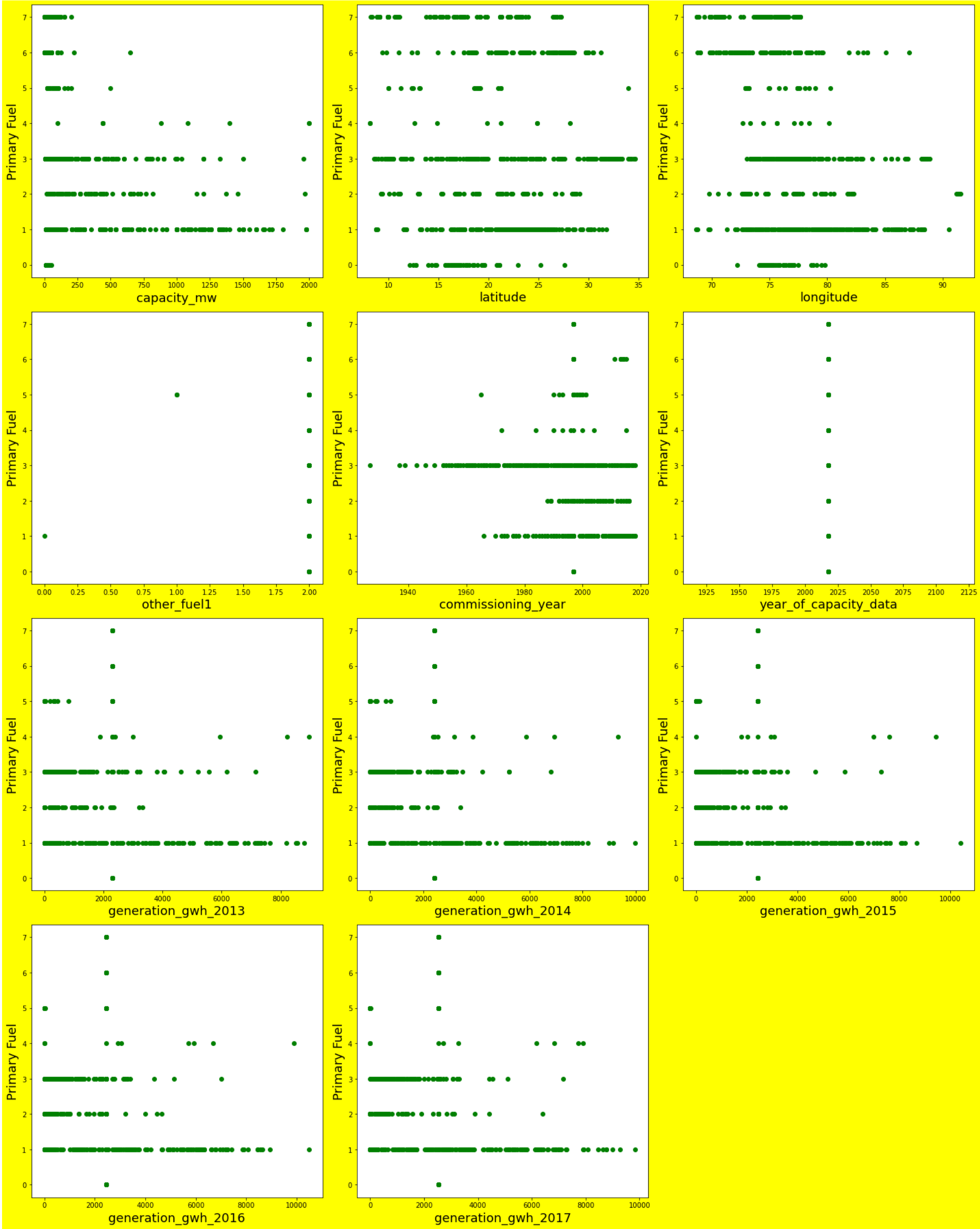
```
x = df.drop('primary_fuel', axis = 1)
y = df. primary_fuel
print('Data has been splited')
```

```
Data has been splited
```

In [69]:

```
# Let' see relation between features and labels.
print('------------------------')
print('Scatter Plot :-')
print('------------------------')

plt.figure(figsize = (20,25), facecolor = 'yellow')
plotnumber = 1
for column in x:
    if plotnumber <=12:
        ax = plt.subplot(4,3, plotnumber)
        plt.scatter(x[column],y, c = 'g')
        plt.xlabel(column, fontsize = 18)
        plt.ylabel('Primary Fuel', fontsize = 18)
    plotnumber += 1
plt.tight_layout()
```

```
------------------------
Scatter Plot :-
------------------------
```



Features are related to label

Checking for class imbalance

In [70]:
```
df['primary_fuel'].value_counts()
```

Out[70]:
```
1    237
3    236
6    126
7    123
2     65
0     50
5     21
4      9
Name: primary_fuel, dtype: int64
```

Class are not balanced

Handling Class Imbalance

In [71]:
```
from imblearn.over_sampling import RandomOverSampler
ros  = RandomOverSampler()
x_over, y_over = ros.fit_resample(x, y)
```

In [72]:
```
print('---------------------')
print('Class are balanced :-')
print('---------------------')
print(y_over.value_counts())
print('---------------------')
```

```
---------------------
Class are balanced :-
---------------------
0    237
1    237
2    237
3    237
4    237
5    237
6    237
7    237
Name: primary_fuel, dtype: int64
---------------------
```

Data Scaling

In [73]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled
```

Out[73]:
```
array([[-0.62557022,  1.15214913, -0.90157571, ...,  0.20188562,
         0.19589199,  0.19330316],
       [-0.39611203,  0.5967866 , -0.57561684, ...,  0.20188562,
         0.19589199,  0.19330316],
       [-0.537391  ,  0.13162823, -1.82289847, ...,  0.20188562,
         0.19589199,  0.19330316],
       ...,
       [-0.57030804, -0.94542669, -0.34404144, ...,  0.20188562,
         0.19589199,  0.19330316],
       [-0.4393607 ,  0.52913754, -0.78079727, ...,  0.20188562,
         0.19589199,  0.19330316],
       [-0.59193237, -1.81340815,  0.1075556 , ...,  0.20188562,
         0.19589199,  0.19330316]])
```

Data has been scaled

Split data into train and test. Model will be bulit on training data and tested on test data

In [74]:
```
x_train, x_test, y_train, y_test = train_test_split(x_over, y_over, test_size = 0
print('Data has been splited.')
```

```
Data has been splited.
```

Model Bulding

Decision Tree model instantiaing, training and evaluating

In [76]:
```
DT = DecisionTreeClassifier()
DT.fit(x_train, y_train)
y_pred = DT.predict(x_test)
```

In [77]:
```
print('-------------------------------------------------------------')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('------------------------------------------------------------\n')
```

```
-------------------------------------------------------------

Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        62
           1       0.82      0.66      0.73        61
           2       0.90      1.00      0.95        53
           3       0.85      0.70      0.76        56
           4       0.94      1.00      0.97        67
           5       0.93      1.00      0.96        52
           6       0.94      0.99      0.96        68
           7       0.89      0.93      0.91        55

    accuracy                           0.91       474
   macro avg       0.90      0.91      0.90       474
weighted avg       0.90      0.91      0.90       474

------------------------------------------------------------
```

Conclusion : Decision Tree model has 91% score

Cross Validation score to check if the model is overfitting

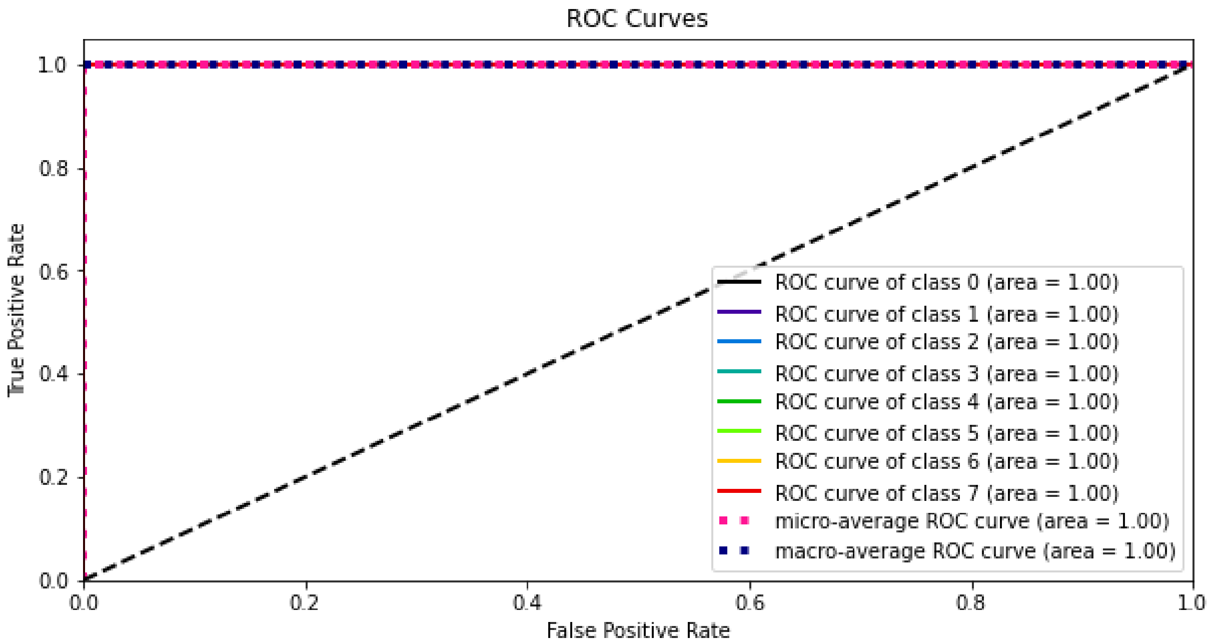
In [81]:
```
cv = cross_val_score(DT, x, y, cv = 5)
print('Cross Validation score of Decision Tree model --->', cv.mean())
```

```
Cross Validation score of Decision Tree model ---> 0.7047172945319249
```

Conclusion : Decision Tree model has 70% Cross Validation score

ROC, AUC Curve

In [85]:
```
prob = DT.predict_proba(x_test) # calculating probability
skplt.metrics.plot_roc(y_pred,prob, figsize = (10,5))
plt.show()
```



XGBoost model instantiaing, training and evaluating

In [96]:
```
xgb = xgb.XGBClassifier(eval_metric = 'mlogloss')
xgb.fit(x_train, y_train)
y_pred = xgb.predict(x_test)
```

In [97]:
```
print('-------------------------------------------------------------')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('------------------------------------------------------------\n')
```

```
-------------------------------------------------------------

Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        62
           1       0.82      0.75      0.79        61
           2       0.86      0.96      0.91        53
           3       0.92      0.79      0.85        56
           4       0.99      1.00      0.99        67
           5       0.98      1.00      0.99        52
           6       0.97      0.96      0.96        68
           7       0.93      0.95      0.94        55

    accuracy                           0.93       474
   macro avg       0.92      0.93      0.92       474
weighted avg       0.93      0.93      0.92       474

------------------------------------------------------------
```

Conclusion : XGBoost model has 93% score

Cross Validation score to check if the model is overfitting

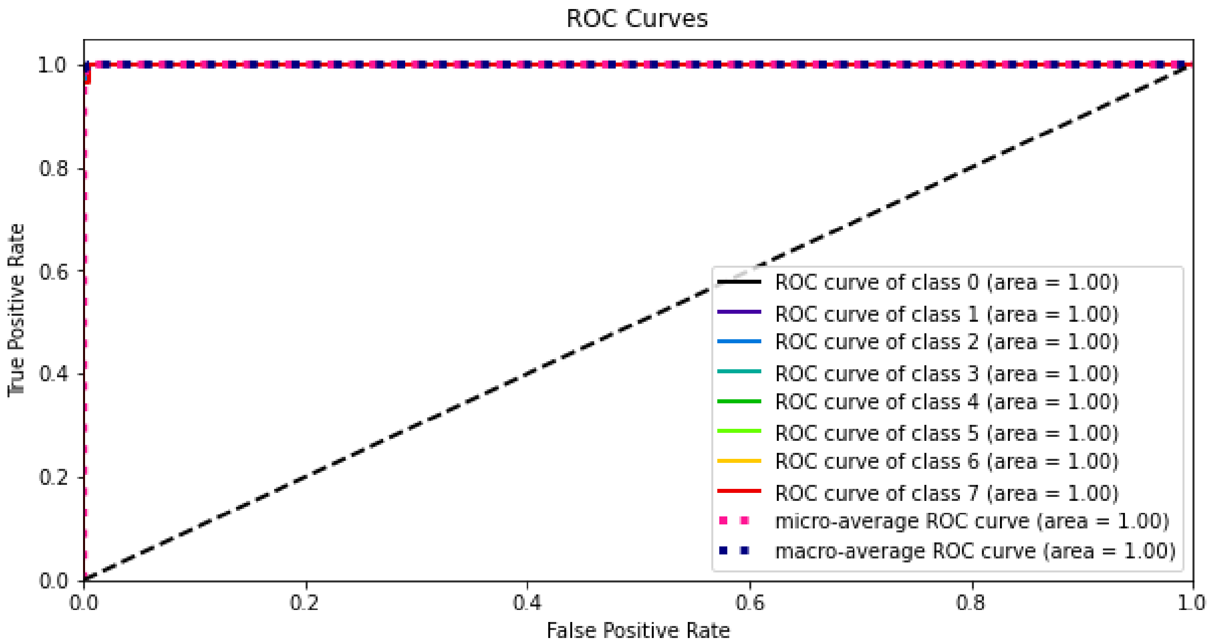
In [98]:
```
cv = cross_val_score(xgb, x, y, cv = 5)
print('Cross Validation score of Ada Boost model --->', cv.mean())
```

```
Cross Validation score of Ada Boost model ---> 0.778566208225367
```

Conclusion : XGBoost model has 77% Cross Validation score

ROC, AUC Curve

In [100]:
```
prob = xgb.predict_proba(x_test) # calculating probability
skplt.metrics.plot_roc(y_pred,prob, figsize = (10,5))
plt.show()
```



Knn model instantiaing, training and evaluating

In [101]:
```
Knn = KNeighborsClassifier()
Knn.fit(x_train, y_train)
y_pred = Knn.predict(x_test)
```

In [102]:
```
print('-------------------------------------------------------------')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('------------------------------------------------------------\n')
```

```
-------------------------------------------------------------

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.97      0.85        62
           1       0.67      0.51      0.58        61
           2       0.69      0.81      0.75        53
           3       0.72      0.46      0.57        56
           4       0.91      1.00      0.95        67
           5       0.85      1.00      0.92        52
           6       0.91      0.63      0.75        68
           7       0.54      0.67      0.60        55

    accuracy                           0.76       474
   macro avg       0.76      0.76      0.74       474
weighted avg       0.76      0.76      0.75       474

------------------------------------------------------------
```

Conclusion : KNN model has 76% score

Cross Validation score to check if the model is overfitting

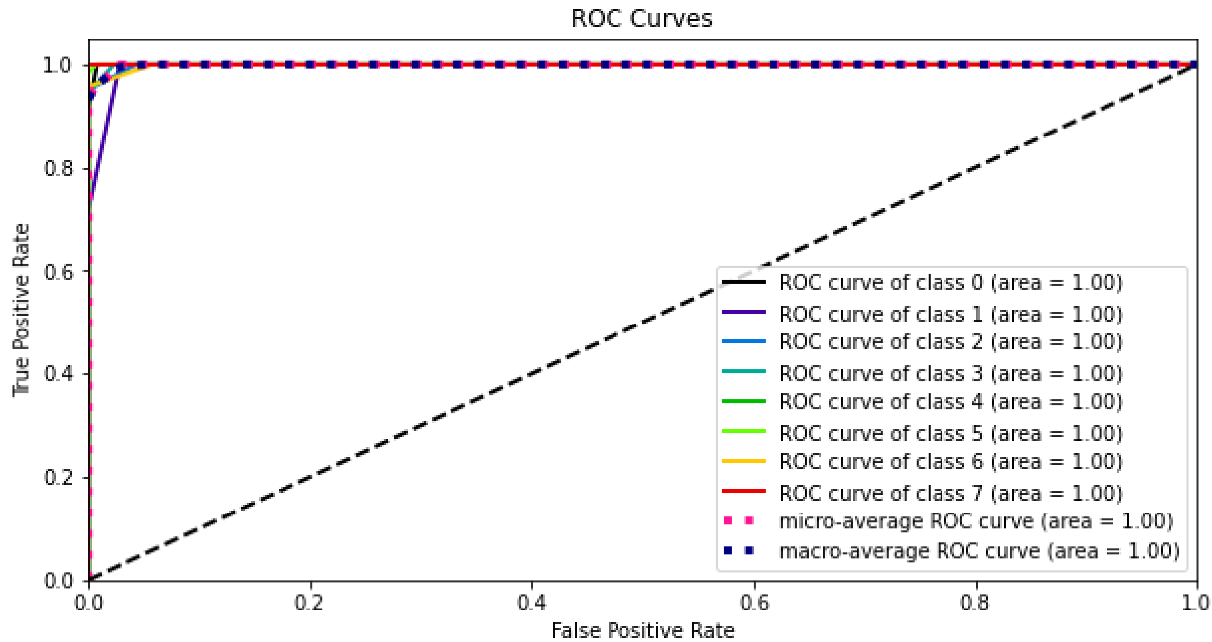
In [103]:
```
cv = cross_val_score(Knn, x, y, cv = 5)
print('Cross Validation score of Ada Boost model --->', cv.mean())
```

```
Cross Validation score of Ada Boost model ---> 0.604391734768454
```

Conclusion : Knn model has 60% Cross Validation score

ROC, AUC Curve

In [104]:
```
prob = Knn.predict_proba(x_test) # calculating probability
skplt.metrics.plot_roc(y_pred,prob, figsize = (10,5))
plt.show()
```



Random Forest model instantiaing, training and evaluating

In [105]:
```
Rn = RandomForestClassifier()
Rn.fit(x_train, y_train)
y_pred = Rn.predict(x_test)
```

In [106]:
```
print('-------------------------------------------------------------')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('------------------------------------------------------------\n')
```

```
-------------------------------------------------------------

Classification Report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        62
           1       0.82      0.80      0.81        61
           2       0.89      0.96      0.93        53
           3       0.89      0.75      0.82        56
           4       1.00      1.00      1.00        67
           5       0.96      1.00      0.98        52
           6       0.97      0.97      0.97        68
           7       0.91      0.91      0.91        55

    accuracy                           0.93       474
   macro avg       0.92      0.92      0.92       474
weighted avg       0.93      0.93      0.92       474

------------------------------------------------------------
```

Conclusion : Random Forest model has 93% score

Cross Validation score to check if the model is overfitting

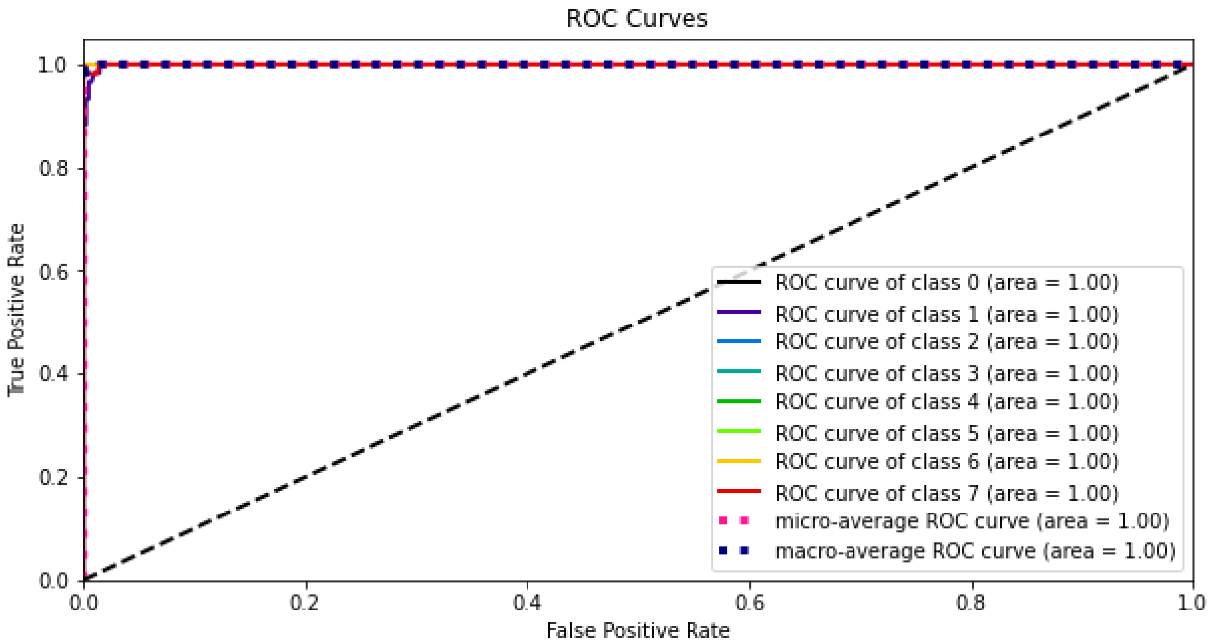
In [107]:
```
cv = cross_val_score(Rn, x, y, cv = 5)
print('Cross Validation score of Ada Boost model --->', cv.mean())
```

```
Cross Validation score of Ada Boost model ---> 0.7854893362567272
```

Conclusion : Random Forest model has 78% Cross Validation score

ROC, AUC Curve

In [109]:
```
prob = Rn.predict_proba(x_test) # calculating probability
skplt.metrics.plot_roc(y_pred,prob, figsize = (10,5))
plt.show()
```



Looking CV score we found Random Forest has best model so we do Hyperparameter Tuning on it

In [115]:
```
param_grid = {'n_estimators': [100,200,300,400,500],
              'max_features': ['auto', 'squrt'],
              'max_depth': [5, 10, 15, 20, 25, 30],
              'min_samples_split': [2, 5, 10, 15, 100],
              'min_samples_leaf': [1, 2, 5, 10]}
```

In [116]:
```
grid_search = GridSearchCV(estimator = Rn, param_grid = param_grid, cv = 5,n_jobs
```

In [117]:
```
grid_search.fit(x_train, y_train)
```

Out[117]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [5, 10, 15, 20, 25, 30],
                         'max_features': ['auto', 'squrt'],
                         'min_samples_leaf': [1, 2, 5, 10],
                         'min_samples_split': [2, 5, 10, 15, 100],
                         'n_estimators': [100, 200, 300, 400, 500]})
```

In [118]:
```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_s
plit': 2, 'n_estimators': 400}
```

In [120]:
```
hRn = RandomForestClassifier(max_depth = 20, max_features = 'auto', min_samples_l
hRn.fit(x_train, y_train)
hRn.score(x_test, y_test)
```

Out[120]:
```
0.919831223628692
```

In [122]:
```
y_pred = hRn.predict(x_test)
```

In [123]:
```
print('-------------------------------------------------------------')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----------------------------------------------------------\n')
```

```
-------------------------------------------------------------

Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        62
           1       0.80      0.77      0.78        61
           2       0.88      0.96      0.92        53
           3       0.91      0.73      0.81        56
           4       1.00      1.00      1.00        67
           5       0.96      1.00      0.98        52
           6       0.97      0.97      0.97        68
           7       0.89      0.91      0.90        55

    accuracy                           0.92       474
   macro avg       0.92      0.92      0.92       474
weighted avg       0.92      0.92      0.92       474

-----------------------------------------------------------
```

After Hyperparameter Tuning model accuracy score 92%.

Saving The Model

In [124]:
```
# saving the model to the Local file system
filename = 'Global Power Plant Project (Fuel Type).pickle'
pickle.dump(hRn, open(filename, 'wb'))
```

Final Conclusion : Random Forest is our best model.

In [ ]:
```
```