In [2]:
```
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import metrics
from scipy.stats import zscore
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

# Classification

from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

# Regression

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

In [3]:
```
df = pd.read_csv("global-power-plant-database_IND.csv")
df
```

Out[3]:

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

907 rows × 27 columns

In [5]:
```
df.head()
```

Out[5]:

ıeration_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh
NaN	NaN	NaN	NaN	
NaN	NaN	NaN	NaN	
NaN	NaN	NaN	NaN	
617.789264	843.747	886.004428	663.774500	626.2
3035.550000	5916.370	6243.000000	5385.579736	7279.0

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

Out[6]:
```
(907, 27)
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['country', 'country_long', 'name', 'gppd_idnr', 'capacity_mw',
       'latitude', 'longitude', 'primary_fuel', 'other_fuel1', 'other_fuel2',
       'other_fuel3', 'commissioning_year', 'owner', 'source', 'url',
       'geolocation_source', 'wepp_id', 'year_of_capacity_data',
       'generation_gwh_2013', 'generation_gwh_2014', 'generation_gwh_2015',
       'generation_gwh_2016', 'generation_gwh_2017', 'generation_gwh_2018',
       'generation_gwh_2019', 'generation_data_source',
       'estimated_generation_gwh'],
      dtype='object')
```

In [9]:
```
df.dtypes
```

Out[9]:
```
country                    object
country_long               object
name                       object
gppd_idnr                  object
capacity_mw               float64
latitude                  float64
longitude                 float64
primary_fuel               object
other_fuel1                object
other_fuel2                object
other_fuel3               float64
commissioning_year        float64
owner                      object
source                     object
url                        object
geolocation_source         object
wepp_id                   float64
year_of_capacity_data     float64
generation_gwh_2013       float64
generation_gwh_2014       float64
generation_gwh_2015       float64
generation_gwh_2016       float64
generation_gwh_2017       float64
generation_gwh_2018       float64
generation_gwh_2019       float64
generation_data_source     object
estimated_generation_gwh  float64
dtype: object
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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
```

In [14]:
```
df = df.drop(["other_fuel1", "other_fuel2", "other_fuel3", "owner", "wepp_id", "g
```

In [15]:
```
df
```

Out[15]:

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

907 rows × 13 columns

In [16]:
```
df.shape
```

Out[16]: (907, 13)

In [17]:
```
df.nunique().to_frame("Unique Values")
```

Out[17]:

	Unique Values
country	1
country_long	1
name	907
gppd_idnr	907
capacity_mw	361
latitude	836
longitude	827
primary_fuel	8
commissioning_year	73
source	191
url	304
geolocation_source	3
year_of_capacity_data	1

In [18]:
```
df = df.drop(["country", "country_long", "year_of_capacity_data", "name", "gppd_i
```

In [19]:
```
print(df.shape)
df.head()
```

```
(907, 7)
```

Out[19]:

	capacity_mw	latitude	longitude	primary_fuel	commissioning_year	source	geolocation_sou
0	2.5	28.1839	73.2407	Solar	2011.0	National Renewable Energy Laboratory	National Renewa Energy Laborat
1	98.0	24.7663	74.6090	Coal	NaN	Ultratech Cement ltd	W
2	39.2	21.9038	69.3732	Wind	NaN	CDM	W
3	135.0	23.8712	91.3602	Gas	2004.0	Central Electricity Authority	W
4	1800.0	21.9603	82.4091	Coal	2015.0	Central Electricity Authority	W

In [20]:
```
df.isnull().sum()
```

Out[20]:
```
capacity_mw            0
latitude              46
longitude             46
primary_fuel           0
commissioning_year    380
source                 0
geolocation_source     19
dtype: int64
```

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 907 entries, 0 to 906
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   capacity_mw         907 non-null    float64
 1   latitude            861 non-null    float64
 2   longitude           861 non-null    float64
 3   primary_fuel        907 non-null    object 
 4   commissioning_year  527 non-null    float64
 5   source              907 non-null    object 
 6   geolocation_source  888 non-null    object 
dtypes: float64(4), object(3)
memory usage: 49.7+ KB
```

In [22]:
```
df.describe()
```

Out[22]:

	capacity_mw	latitude	longitude	commissioning_year
count	907.000000	861.000000	861.000000	527.000000
mean	326.223755	21.197918	77.464907	1997.091082
std	590.085456	6.239612	4.939316	17.082868
min	0.000000	8.168900	68.644700	1927.000000
25%	16.725000	16.773900	74.256200	1988.000000
50%	59.200000	21.780000	76.719500	2001.000000
75%	385.250000	25.512400	79.440800	2012.000000
max	4760.000000	34.649000	95.408000	2018.000000

In [23]:
```
# getting list of object data type column names
object_datatype = []
for x in df.dtypes.index:
    if df.dtypes[x] == 'object':
        object_datatype.append(x)
print(f"Object Data Type Columns are: ", object_datatype)


# getting the list of float data type column names
float_datatype = []
for x in df.dtypes.index:
    if df.dtypes[x] == 'float64':
        float_datatype.append(x)
print(f"Float Data Type Columns are: ", float_datatype)
```

```
Object Data Type Columns are:  ['primary_fuel', 'source', 'geolocation_source']
Float Data Type Columns are:  ['capacity_mw', 'latitude', 'longitude', 'commiss
ioning_year']
```

We have successfully bifurcated the object datatype column names and float data type column names. Since we do not have to worry about outliers and skewness in categorical columns we can use this separated column names to pre process only on numerical continous columns.

In [24]:
```
# filling missing data for continous values with mean
df["latitude"].fillna(df["latitude"].mean(),inplace=True)
df["longitude"].fillna(df["longitude"].mean(),inplace=True)

# filling missing data for categorical values with mode
df["commissioning_year"].fillna(df["commissioning_year"].mode()[0],inplace=True)
df["geolocation_source"].fillna(df["geolocation_source"].mode()[0],inplace=True)
```

Since we had to take care of the missing data I have chosen to fill the null values in continous data column with it's mean and the null values for categorical data column with it's mode information.

In [25]:
```
df.isnull().sum()
```

Out[25]:
```
capacity_mw           0
latitude              0
longitude             0
primary_fuel          0
commissioning_year    0
source                0
geolocation_source    0
dtype: int64
```

In [26]:
```
for col in object_datatype:
    print(col)
    print(df[col].value_counts())
    print("="*120)
```

```
primary_fuel
Coal       258
Hydro      251
Solar      127
Wind       123
Gas         69
Biomass     50
Oil         20
Nuclear      9
Name: primary_fuel, dtype: int64
========================================================================================================================
source
Central Electricity Authority                                519
CDM                                                          124
Lancosola                                                     10
National Renewable Energy Laboratory                           8
National Thermal Power Corporation (NTPC)                      6
                                                            ...
Harsha Engineers Limited                                       1
Godawari Energy ltd                                            1
Sunkon Energy Private Limited                                  1
West Bengal Energy Development Corporation Limited (WBEDCL)    1
Yashwantrao Krishna ssk                                        1
Name: source, Length: 191, dtype: int64
========================================================================================================================
geolocation_source
WRI                                  784
Industry About                       119
National Renewable Energy Laboratory   4
Name: geolocation_source, dtype: int64
========================================================================================================================
```

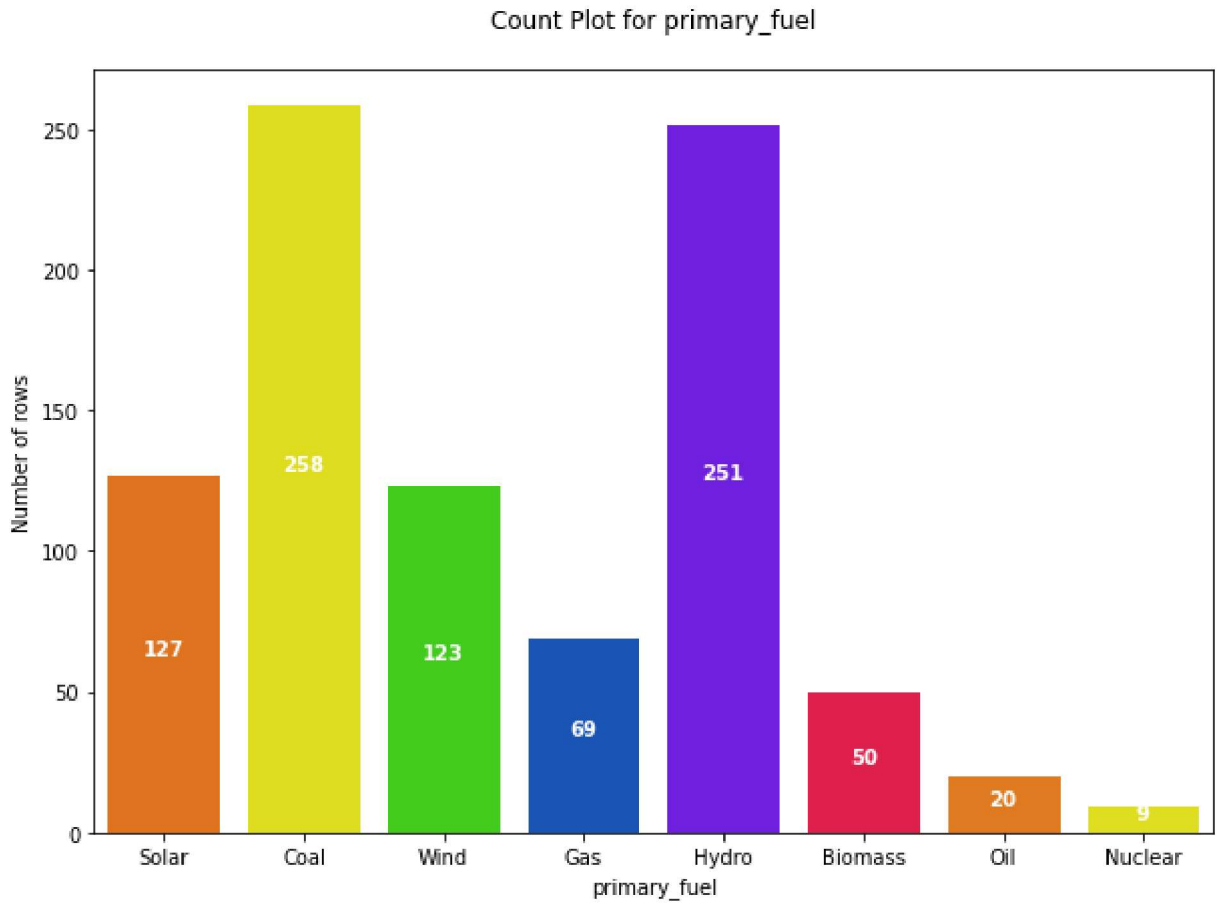
In the above cell I am taking a look at the object data type columns so as to check how many categories each of them hold and how many values/rows are populated with that data

Visualization of the dataset

In [27]:
```
df.columns
```

Out[27]:
```
Index(['capacity_mw', 'latitude', 'longitude', 'primary_fuel',
       'commissioning_year', 'source', 'geolocation_source'],
      dtype='object')
```

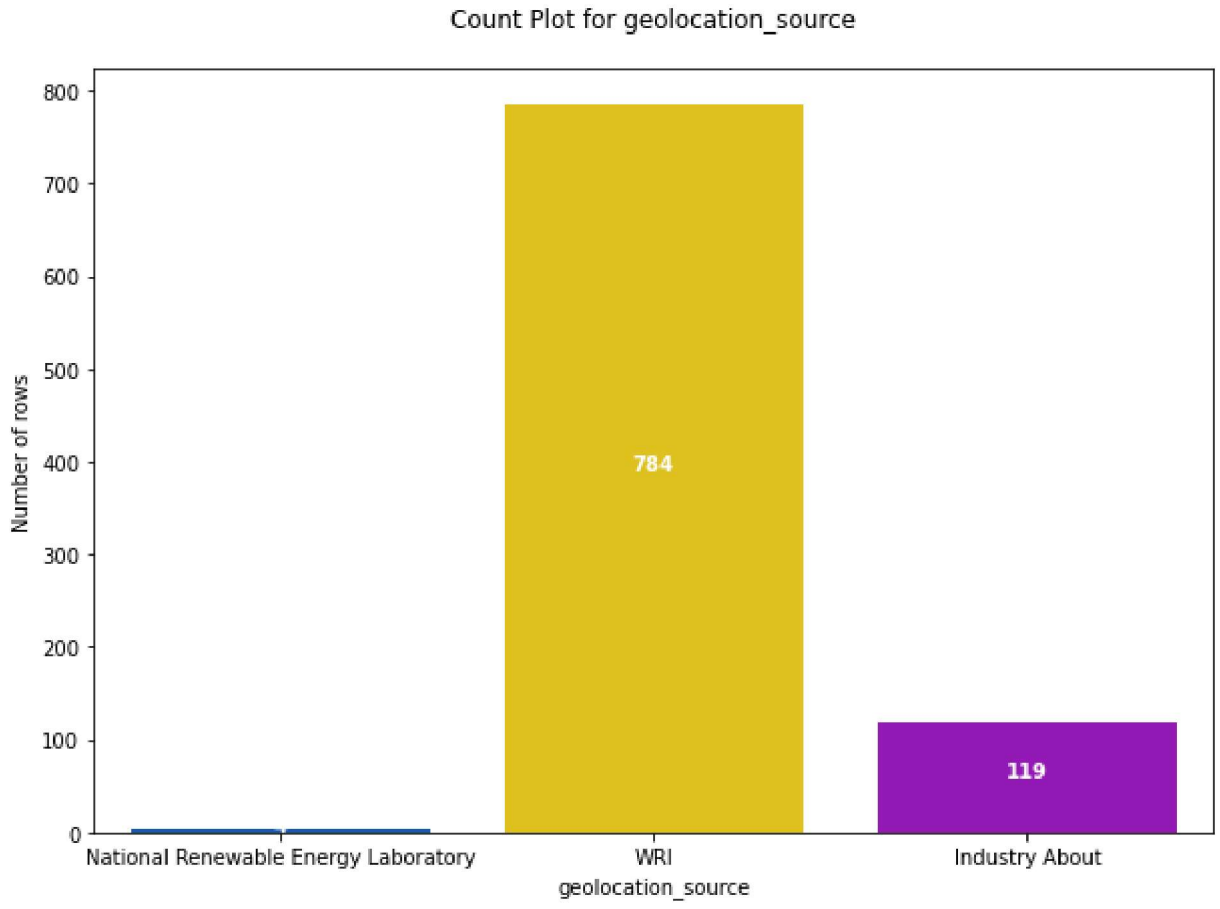
In [28]:
```
try:
    plt.figure(figsize=(10,7))
    col_name = 'primary_fuel'
    values = df[col_name].value_counts()
    index = 0
    ax = sns.countplot(df[col_name], palette="prism")

    for i in ax.get_xticklabels():
        ax.text(index, values[i.get_text()]/2, values[i.get_text()],
                horizontalalignment="center", fontweight='bold', color='w')
        index += 1

    plt.title(f"Count Plot for {col_name}\n")
    plt.ylabel(f"Number of rows")
    plt.show()

except Exception as e:
    pass
```



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

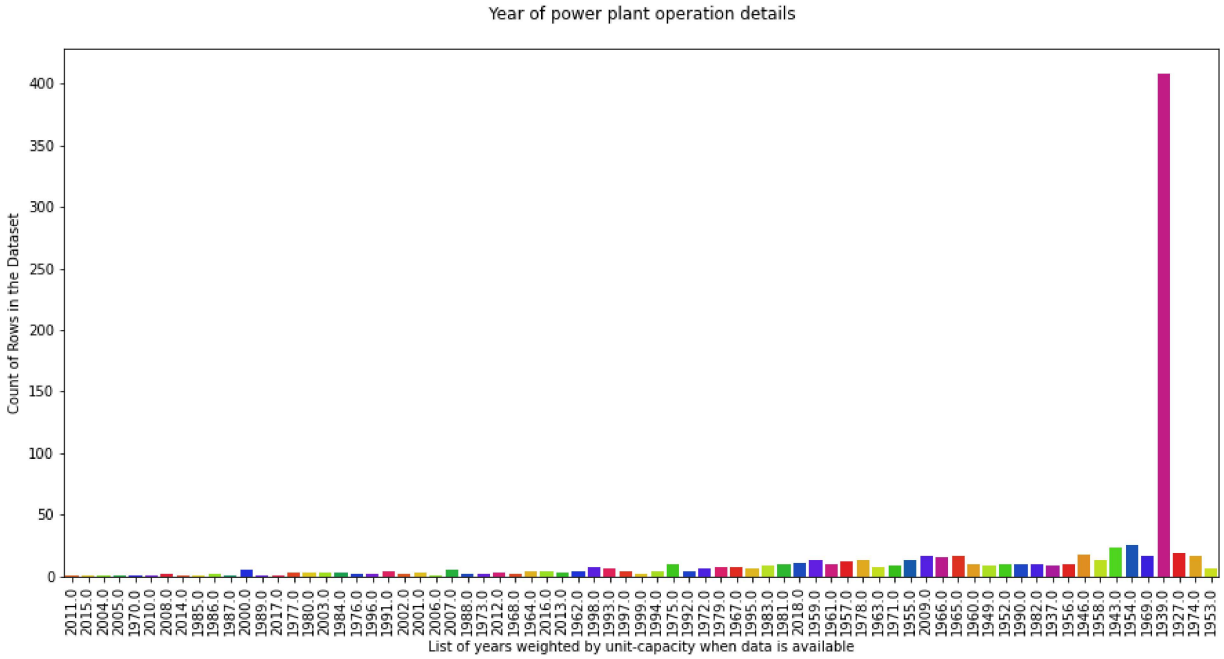
However when we will be considering "primary_fuel" as our target label then this is impose a class imbalance issue while trying to create a classification model and therefore will need to be treated accordingly

In [29]:
```
try:
    plt.figure(figsize=(10,7))
    col_name = 'geolocation_source'
    values = df[col_name].value_counts()
    index = 0
    ax = sns.countplot(df[col_name], palette="prism")

    for i in ax.get_xticklabels():
        ax.text(index, values[i.get_text()]/2, values[i.get_text()],
                horizontalalignment="center", fontweight='bold', color='w')
        index += 1

    plt.title(f"Count Plot for {col_name}\n")
    plt.ylabel(f"Number of rows")
    plt.show()

except Exception as e:
    pass
```



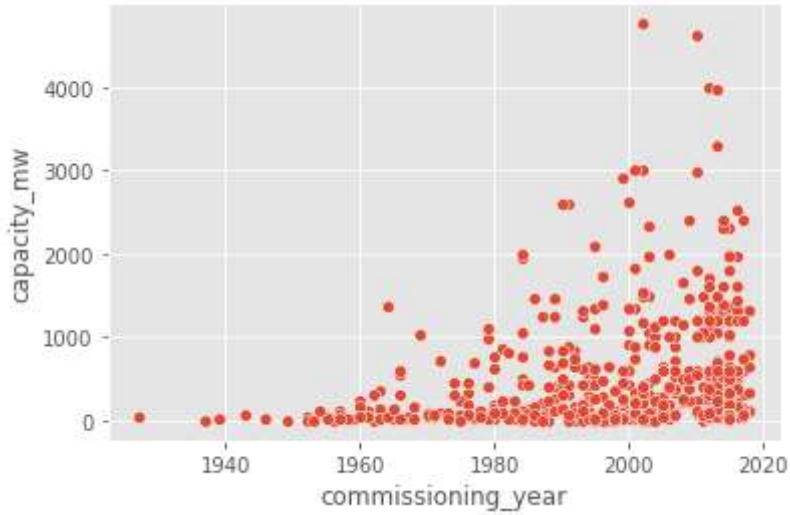
In the above count plot for "geolocation_source" column we see that the highest data value is covered by WRI option and the least value which seems quite negligible has been accumulated by National Renewable Energy Laboratory.

In [30]:
```
plt.figure(figsize=(15,7))
values = list(df['commissioning_year'].unique())
diag = sns.countplot(df["commissioning_year"], palette="prism")
diag.set_xticklabels(labels=values, rotation=90)
plt.title("Year of power plant operation details\n")
plt.xlabel("List of years weighted by unit-capacity when data is available")
plt.ylabel("Count of Rows in the Dataset")
plt.show()
```



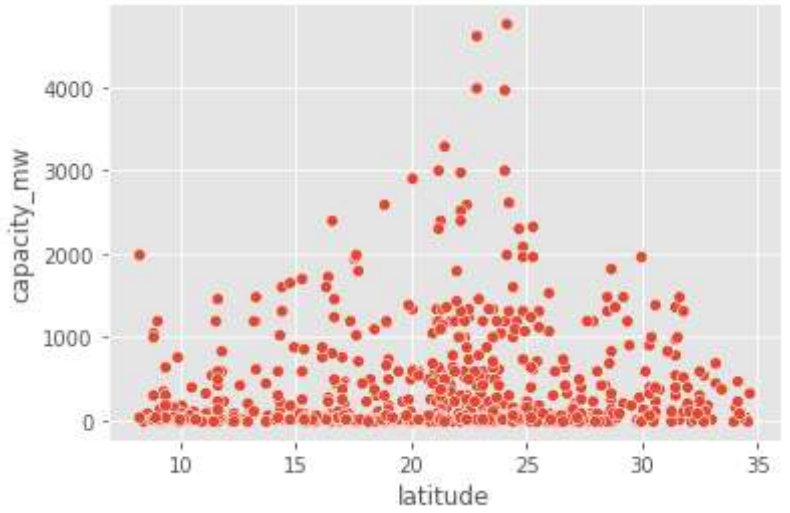
In the above count plot we can see the list of years as to when the power plant data was made available. Since we had missing values in the "commissioning_year" column we replaced it with the mode wherein the year 1954 covered the most rows in our dataset compared to all the other years.

In [31]:
```
plt.style.use('ggplot')
sns.scatterplot(x = "commissioning_year", y = "capacity_mw", data = df)
plt.show()
```



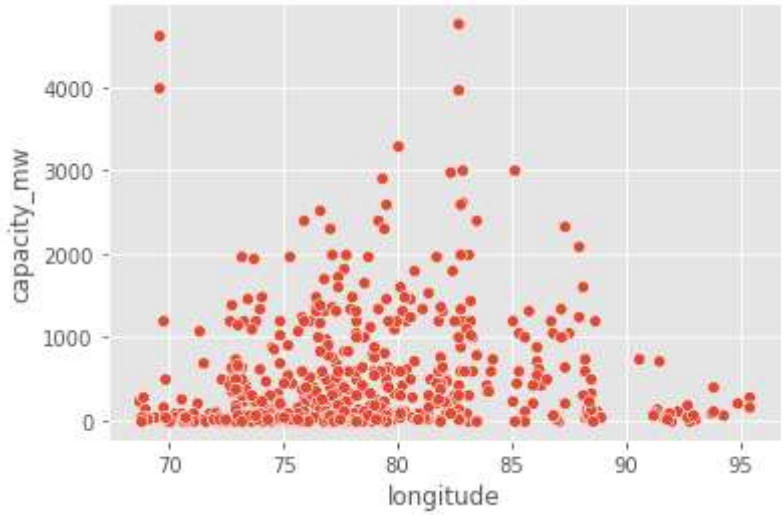
In the above scatter plot we can see that the electricity generating capacity measured in mega watts was pretty low in the olden times as compared to recent years due to the invention of efficient fuel types used for power generation.

In [33]:
```
sns.scatterplot(x = "latitude", y = "capacity_mw", data = df)
plt.show()
```



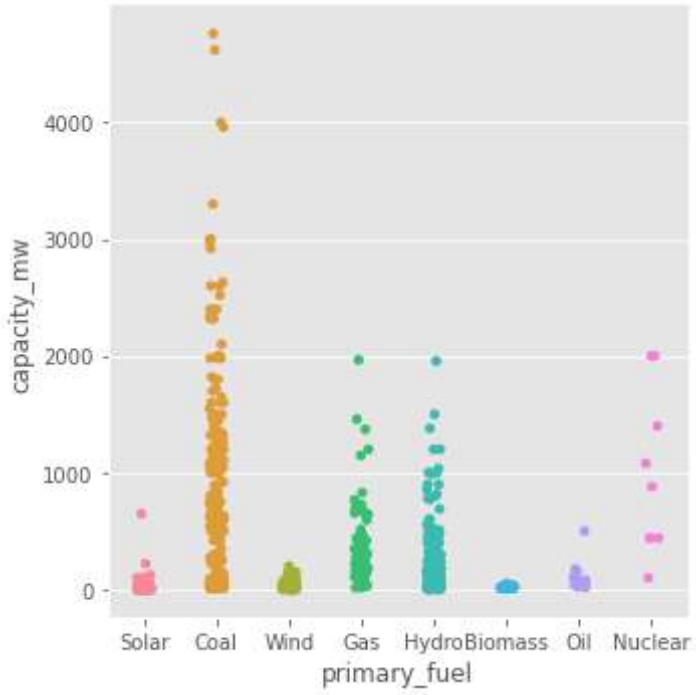
In the above scatter plot we can see that the latitude geolocation for electricty generation is highest measuring the mega watts between 20 to 25 decimal degrees.

In [34]:
```
sns.scatterplot(x = "longitude", y = "capacity_mw", data = df)
plt.show()
```



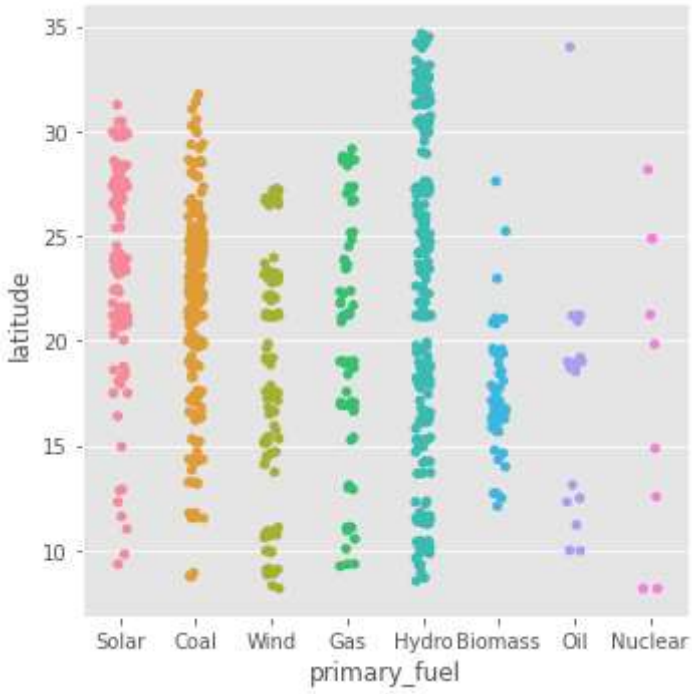
In the above scatter plot we can see that the longitude geolocation for electricty generation is highest measuring the mega watts around 70 and then again between 80-85 decimal degrees.

In [35]:
```
plt.style.use('seaborn-pastel')
sns.catplot(x = "primary_fuel", y = "capacity_mw", data = df)
plt.show()
```



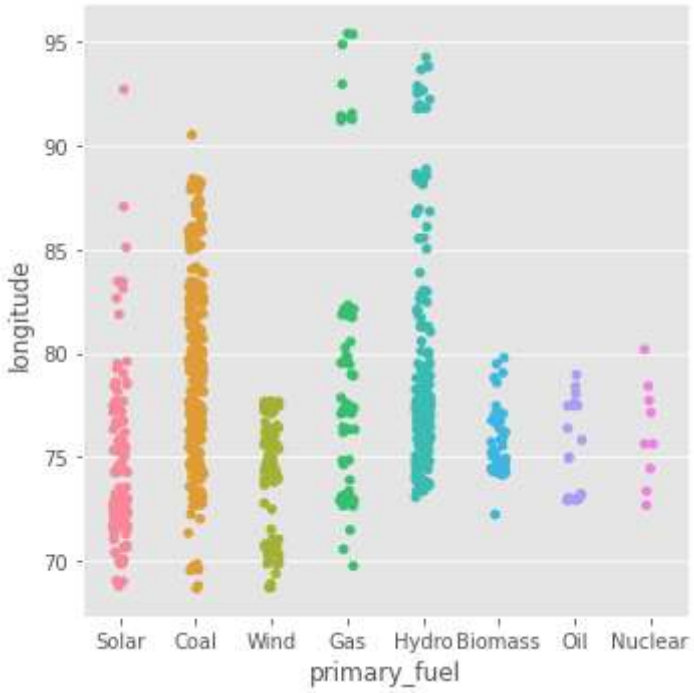
In the above categorical plot we can see that coal is the highest contender when it comes to generating electricity and biomass is used the least.

In [36]:
```
sns.catplot(x = "primary_fuel", y = "latitude", data = df)
plt.show()
```



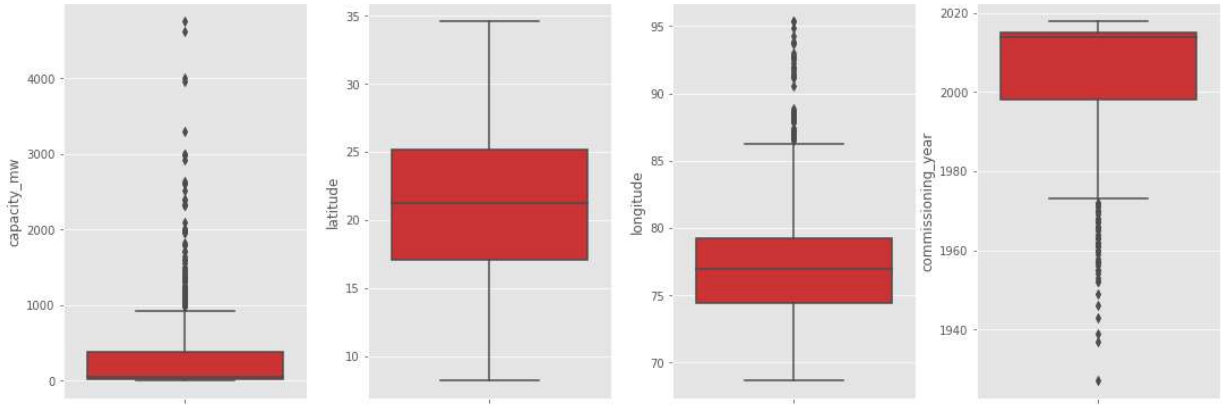
In the above categorical plot we can see that hydro fuel type is distributed across all the latitude values considering the water bodies that help in generating electricity at a power plant.

In [37]:
```
sns.catplot(x = "primary_fuel", y = "longitude", data = df)
plt.show()
```

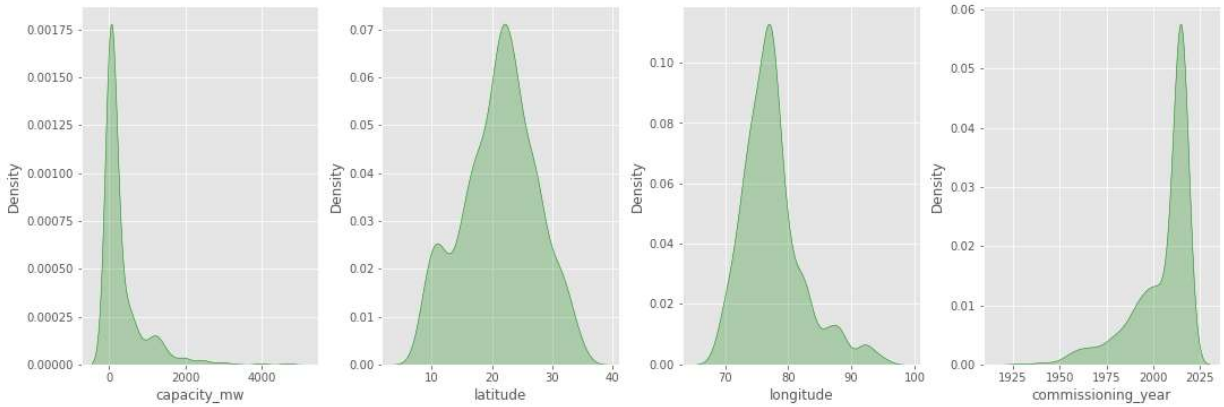


In the above categorical plot we can see that longitude wise decimal degress have quite a splitting data between coal and hydro considering the land lock areas rely on coal for electricity generation and costal area rely mostly on the hydro fuel type. Then we have fuel types solar and wind that are quite comprising when targeted across the longitude values.

In [38]:
```
fig, ax = plt.subplots(ncols=4, nrows=1, figsize=(15,5))
index = 0
ax = ax.flatten()
for col, value in df[float_datatype].items():
    sns.boxplot(y=col, data=df, ax=ax[index], palette="Set1")
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.4, h_pad=1.0)
plt.show()
```

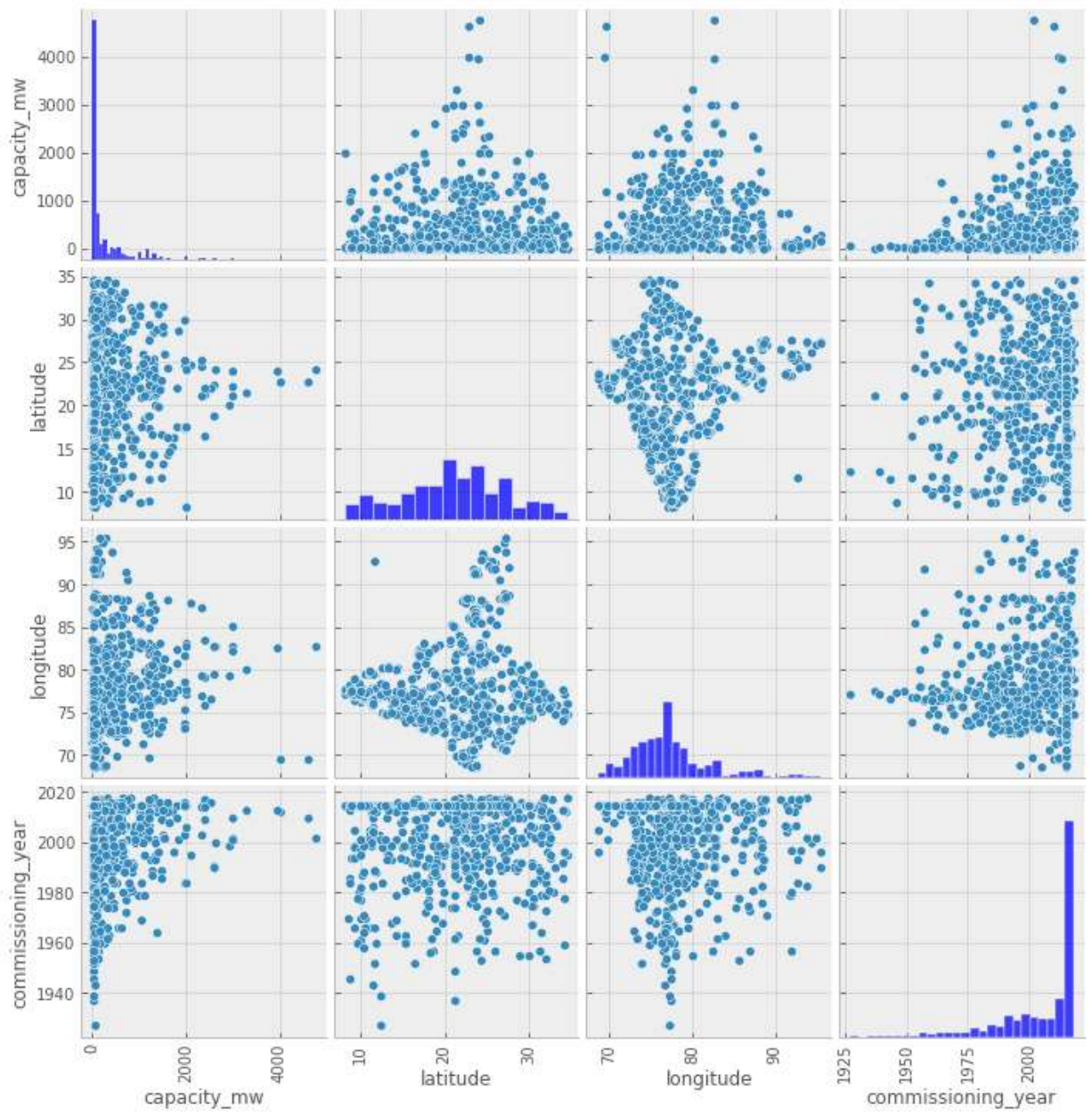


In [39]:
```
fig, ax = plt.subplots(ncols=4, nrows=1, figsize=(15,5))
index = 0
ax = ax.flatten()
for col, value in df[float_datatype].items():
    sns.distplot(value, ax=ax[index], hist=False, color="g", kde_kws={"shade": Tr
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.4, h_pad=1.0)
plt.show()
```



In the above distribution plot created for our numerical data columns we see skewness and that will need to be treated before we can proceed with our model building process.

In [40]:
```
plt.style.use('bmh')
g = sns.pairplot(df)
for ax in g.axes.flat:
    ax.tick_params("x", labelrotation=90)
plt.show()
```



Encoding all the object datatype columns

In [41]:
```
# Label Encoder

le = LabelEncoder()
df["primary_fuel"] = le.fit_transform(df["primary_fuel"])
df.head()
```

Out[41]:

	capacity_mw	latitude	longitude	primary_fuel	commissioning_year	source	geolocation_sou
0	2.5	28.1839	73.2407	6	2011.0	National Renewable Energy Laboratory	National Renewa Energy Laborat
1	98.0	24.7663	74.6090	1	2015.0	Ultratech Cement ltd	V
2	39.2	21.9038	69.3732	7	2015.0	CDM	V
3	135.0	23.8712	91.3602	2	2004.0	Central Electricity Authority	V
4	1800.0	21.9603	82.4091	1	2015.0	Central Electricity Authority	V

In [42]:
```
# Ordinal Encoder

oe = OrdinalEncoder()
df['geolocation_source'] = oe.fit_transform(df['geolocation_source'].values.resha
df['source'] = oe.fit_transform(df['source'].values.reshape(-1,1))
df.head()
```

Out[42]:

	capacity_mw	latitude	longitude	primary_fuel	commissioning_year	source	geolocation_source
0	2.5	28.1839	73.2407	6	2011.0	109.0	1.0
1	98.0	24.7663	74.6090	1	2015.0	174.0	2.0
2	39.2	21.9038	69.3732	7	2015.0	21.0	2.0
3	135.0	23.8712	91.3602	2	2004.0	22.0	2.0
4	1800.0	21.9603	82.4091	1	2015.0	22.0	2.0

I am using the Ordinal Encoder to convert all the categorical feature columns from object datatype to numerical datatype. I could have used one hot encoding but considering that "source" column has lots of unique values the number of columns would have increased a lot so I felt ordinal encoding was a better option here.

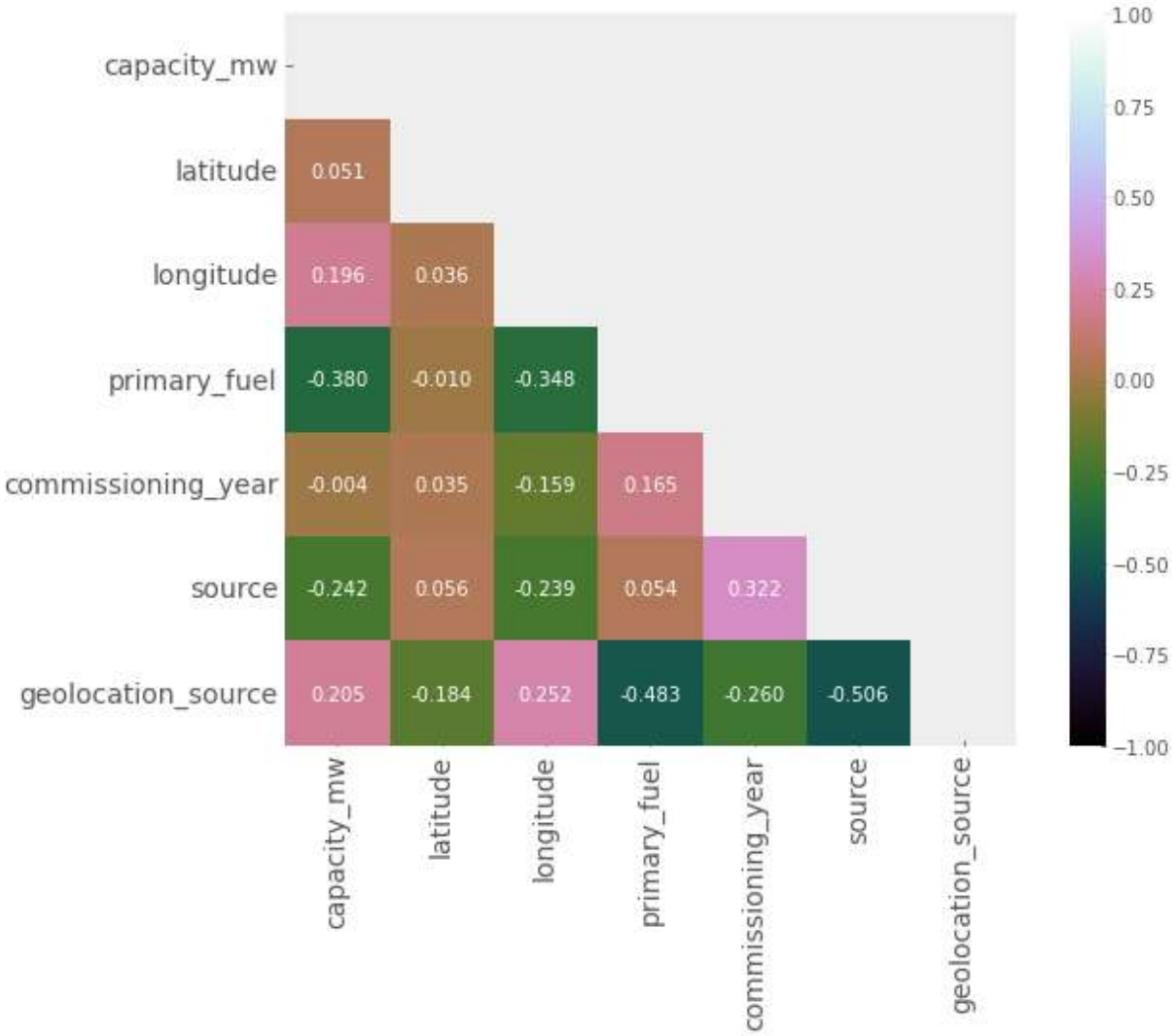
Correlation using a Heatmap

Positive correlation - A correlation of +1 indicates a perfect positive correlation, meaning that both variables move in the same direction together.

Negative correlation - A correlation of –1 indicates a perfect negative correlation, meaning that as one variable goes up, the other goes down.

In [43]:

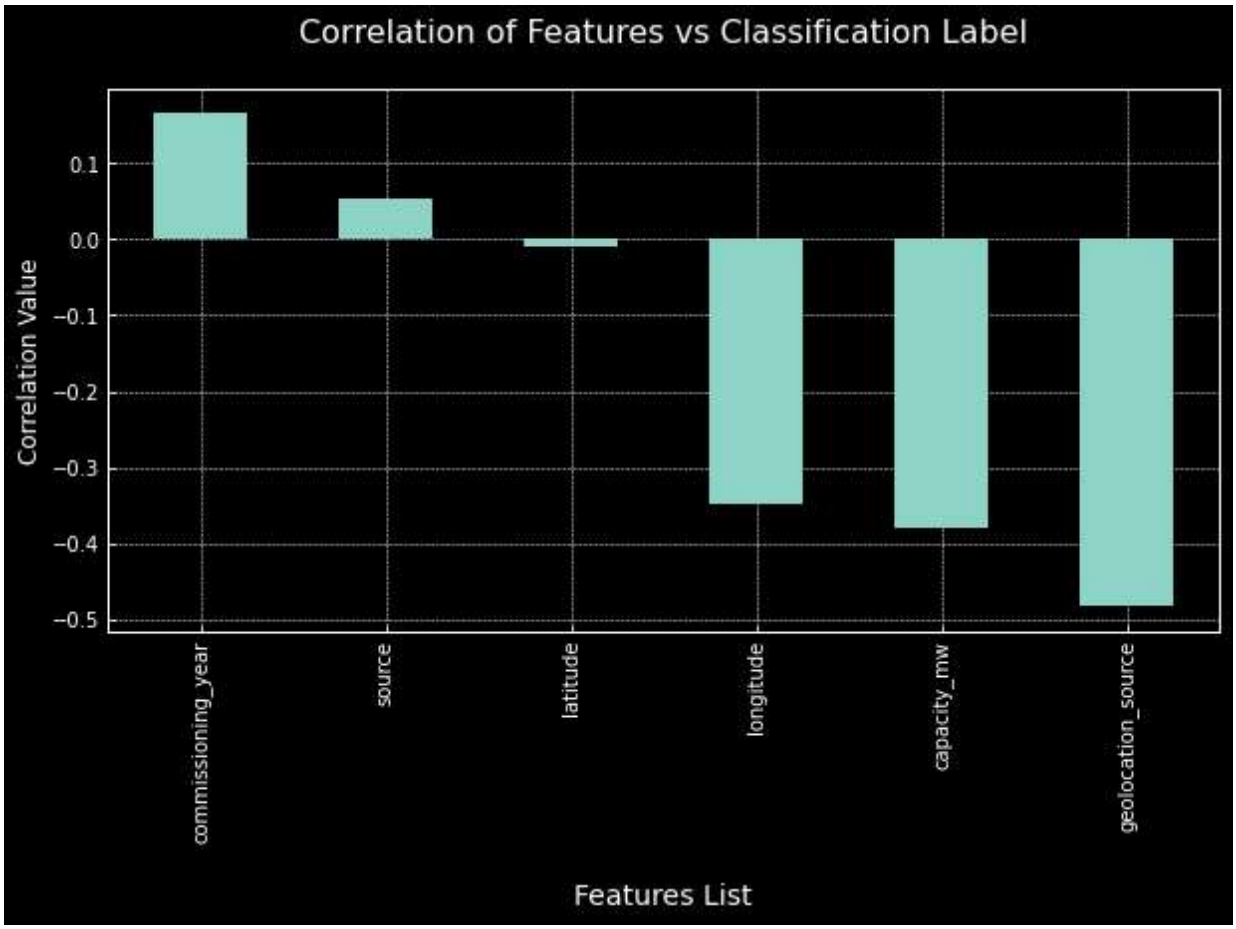
```
upper_triangle = np.triu(df.corr())
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, square=True, fmt='0.3f',
            annot_kws={'size':10}, cmap="cubehelix", mask=upper_triangle)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



In the above heatmap we can see that our target labels "capacity_mw" and "primary_fuel" has both positive and negative correlations with the remaining feature columns. Also we see very less or negligible amount of multi colinearity so we will not have to worry about it. Since the one's which are reflecting the value are inter dependent on those feature columns and I intend to retain and keep them.

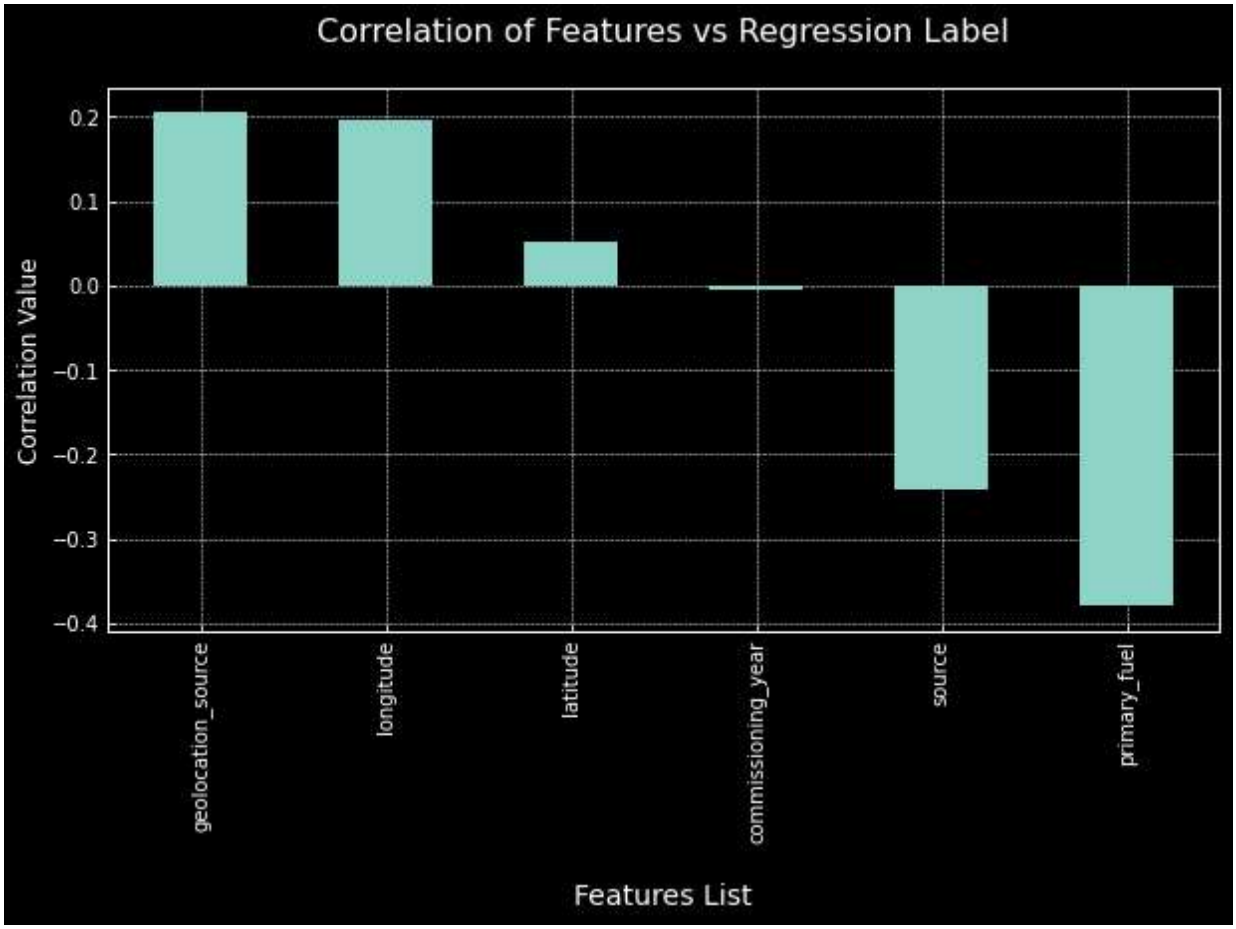
Correlation Bar Plot comparing features with our labels

In [44]:
```
plt.style.use('dark_background')
df_corr = df.corr()
plt.figure(figsize=(10,5))
df_corr['primary_fuel'].sort_values(ascending=False).drop('primary_fuel').plot.ba
plt.title("Correlation of Features vs Classification Label\n", fontsize=16)
plt.xlabel("\nFeatures List", fontsize=14)
plt.ylabel("Correlation Value", fontsize=12)
plt.show()
```



When we consider our classification label "primary_fuel" then we can see the remaining columns that are the features have 2 of them with positive correlation and 4 of them with neagtive correlation.

In [45]:
```
df_corr = df.corr()
plt.figure(figsize=(10,5))
df_corr['capacity_mw'].sort_values(ascending=False).drop('capacity_mw').plot.bar(
plt.title("Correlation of Features vs Regression Label\n", fontsize=16)
plt.xlabel("\nFeatures List", fontsize=14)
plt.ylabel("Correlation Value", fontsize=12)
plt.show()
```



When we consider our regression label "capacity_mw" then we can see the remaining columns that are the features have 4 of them with positive correlation and 2 of them with neagtive correlation.

Using Z Score to remove outliers

In [46]:
```
z = np.abs(zscore(df))
threshold = 3
df1 = df[(z<3).all(axis = 1)]

print ("Shape of the dataframe before removing outliers: ", df.shape)
print ("Shape of the dataframe after removing outliers: ", df1.shape)
print ("Percentage of data loss post outlier removal: ", (df.shape[0]-df1.shape[0

df=df1.copy() # reassigning the changed dataframe name to our original dataframe
```

```
Shape of the dataframe before removing outliers:  (907, 7)
Shape of the dataframe after removing outliers:  (838, 7)
Percentage of data loss post outlier removal:  7.6074972436604185
```

I have used the Z score method to remove the outliers since the IQR method was making me lose way more than 10 percent of data which I could not have afforded to lose.

In [47]:
```
df.skew()
```

Out[47]:
```
capacity_mw           1.967086
latitude             -0.112601
longitude             0.903442
primary_fuel          0.418559
commissioning_year   -1.500521
source                1.792245
geolocation_source   -2.112259
dtype: float64
```

The skew method we see that there are columns present in our dataset that are above the acceptable range of +/-0.5 skewness value.

Using Log Transform to fix skewness

In [49]:
```
for col in float_datatype:
    if df.skew().loc[col]>0.55:
        df[col]=np.log1p(df[col])
```

Splitting the dataset into 2 variables namely 'X' and 'Y' for feature and classification label

In [50]:
```
X = df.drop('primary_fuel', axis=1)
Y = df['primary_fuel']
```

I have bifurcated the dataset into features and classification label where X represents all the feature columns and Y represents the classification target label column.

Resolving the class imbalance issue in our label

column

In [51]:
```
Y.value_counts()
```

Out[51]:
```
1    234
3    222
7    123
6    121
2     64
0     45
5     20
4      9
Name: primary_fuel, dtype: int64
```

Listing the values of our classification label column to count the number of rows occupied by each category. This indicates class imbalance that we will need to fix by using the oversampling method.

In [53]:
```
# adding samples to make all the categorical label values same

oversample = SMOTE()
X, Y = oversample.fit_resample(X, Y)
```

SMOTE is the over sampling mechanism that we are using to ensure that all the categories present in our target label have the same value.

In [54]:
```
Y.value_counts()
```

Out[54]:
```
6    234
1    234
7    234
2    234
3    234
0    234
5    234
4    234
Name: primary_fuel, dtype: int64
```

After applying over sampling we are once again listing the values of our label column to cross verify the updated information. Here we see that we have successfully resolved the class imbalance problem and now all the categories have same data ensuring that the classification machine learning model does not get biased towards one category.

Feature Scaling

In [55]:
```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()
```

Out[55]:

	capacity_mw	latitude	longitude	commissioning_year	source	geolocation_source
0	-1.742858	1.465891	-0.851943	0.396714	1.389474	-1.196188
1	0.168833	0.863691	-0.454639	0.722341	2.853693	0.369121
2	-0.346647	0.359302	-2.015826	0.722341	-0.592852	0.369121
3	0.350450	0.705969	3.898953	-0.173134	-0.570326	0.369121
4	1.828074	0.369258	1.681295	0.722341	-0.570326	0.369121

Finding best random state for building Classification Models

In [56]:
```
maxAccu=0
maxRS=0

for i in range(1, 500):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran
    lr=LogisticRegression()
    lr.fit(X_train, Y_train)
    pred = lr.predict(X_test)
    acc_score = (accuracy_score(Y_test, pred))*100

    if acc_score>maxAccu:
        maxAccu=acc_score
        maxRS=i

print("Best accuracy score is", maxAccu,"on Random State", maxRS)
```

```
Best accuracy score is 75.21367521367522 on Random State 370
```

Machine Learning Model for Classification with Evaluation Metrics

In [57]:
```
# Classification Model Function

def classify(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran

    # Training the model
    model.fit(X_train, Y_train)

    # Predicting Y_test
    pred = model.predict(X_test)

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

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

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

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

In [58]:
```
# Logistic Regression

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

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

           0       0.81      0.81      0.81        69
           1       0.60      0.53      0.56        47
           2       0.38      0.28      0.32        36
           3       0.64      0.48      0.55        77
           4       0.72      0.91      0.81        64
           5       0.55      0.58      0.57        55
           6       1.00      1.00      1.00        57
           7       0.79      0.98      0.88        63

    accuracy                           0.72       468
   macro avg       0.69      0.70      0.69       468
weighted avg       0.71      0.72      0.71       468

Accuracy Score: 72.00854700854701
Cross Validation Score: 68.10994652406417

Accuracy Score - Cross Validation Score is 3.8986004844828415
```

In [59]:
```
# Support Vector Classifier

model=SVC(C=1.0, kernel='rbf', gamma='auto', random_state=42)
classify(model, X, Y)
```

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

           0       0.97      0.84      0.90        69
           1       0.74      0.66      0.70        47
           2       0.56      0.42      0.48        36
           3       0.92      0.71      0.80        77
           4       0.77      1.00      0.87        64
           5       0.83      0.89      0.86        55
           6       1.00      1.00      1.00        57
           7       0.79      1.00      0.88        63

    accuracy                           0.84       468
   macro avg       0.82      0.82      0.81       468
weighted avg       0.84      0.84      0.83       468

Accuracy Score: 83.76068376068376
Cross Validation Score: 79.48748663101604

Accuracy Score - Cross Validation Score is 4.273197129667722
```

In [60]:
```
# Decision Tree Classifier

model=DecisionTreeClassifier(random_state=21, max_depth=15)
classify(model, X, Y)
```

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

           0       0.98      0.90      0.94        69
           1       0.64      0.68      0.66        47
           2       0.60      0.75      0.67        36
           3       0.88      0.77      0.82        77
           4       0.92      0.92      0.92        64
           5       0.84      0.87      0.86        55
           6       0.98      1.00      0.99        57
           7       0.98      1.00      0.99        63

    accuracy                           0.87       468
   macro avg       0.85      0.86      0.86       468
weighted avg       0.88      0.87      0.87       468

Accuracy Score: 86.96581196581197
Cross Validation Score: 85.68570409982173

Accuracy Score - Cross Validation Score is 1.2801078659902316
```

In [61]:
```
# Random Forest Classifier

model=RandomForestClassifier(max_depth=15, random_state=111)
classify(model, X, Y)
```

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

           0       1.00      0.94      0.97        69
           1       0.77      0.79      0.78        47
           2       0.72      0.72      0.72        36
           3       0.93      0.81      0.86        77
           4       0.88      1.00      0.93        64
           5       0.88      0.95      0.91        55
           6       1.00      1.00      1.00        57
           7       1.00      1.00      1.00        63

    accuracy                           0.91       468
   macro avg       0.90      0.90      0.90       468
weighted avg       0.91      0.91      0.91       468

Accuracy Score: 91.02564102564102
Cross Validation Score: 90.27793226381462

Accuracy Score - Cross Validation Score is 0.7477087618264022
```

In [62]:
```
# K Neighbors Classifier

model=KNeighborsClassifier(n_neighbors=15)
classify(model, X, Y)
```

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

           0       0.92      0.86      0.89        69
           1       0.79      0.57      0.67        47
           2       0.51      0.53      0.52        36
           3       0.93      0.56      0.70        77
           4       0.76      0.97      0.85        64
           5       0.75      0.93      0.83        55
           6       0.97      1.00      0.98        57
           7       0.79      0.98      0.88        63

    accuracy                           0.81       468
   macro avg       0.80      0.80      0.79       468
weighted avg       0.83      0.81      0.80       468

Accuracy Score: 81.19658119658119
Cross Validation Score: 79.38081996434939

Accuracy Score - Cross Validation Score is 1.8157612322318073
```

In [63]:
```
# Extra Trees Classifier

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

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

           0       0.98      0.94      0.96        69
           1       0.78      0.77      0.77        47
           2       0.71      0.75      0.73        36
           3       0.96      0.83      0.89        77
           4       0.89      0.98      0.93        64
           5       0.92      0.98      0.95        55
           6       1.00      1.00      1.00        57
           7       0.98      1.00      0.99        63

    accuracy                           0.92       468
   macro avg       0.90      0.91      0.90       468
weighted avg       0.92      0.92      0.92       468

Accuracy Score: 91.66666666666666
Cross Validation Score: 91.23950089126559

Accuracy Score - Cross Validation Score is 0.4271657754010647
```

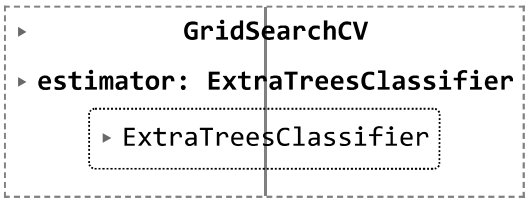
Hyper parameter tuning on the best Classification ML Model

In [64]:
```
# Choosing Extra Trees Classifier

fmod_param = {'criterion' : ["gini", "entropy"],
              'n_jobs' : [2, 1, -1],
              'min_samples_split' : [2, 3, 4],
              'max_depth' : [20, 25, 30],
              'random_state' : [42, 45, 111]
             }
```

In [65]:
```
GSCV = GridSearchCV(ExtraTreesClassifier(), fmod_param, cv=5)
```

In [66]:
```
GSCV.fit(X_train,Y_train)
```

Out[66]:



In [67]:
```
GSCV.best_params_
```

Out[67]:
```
{'criterion': 'entropy',
 'max_depth': 30,
 'min_samples_split': 2,
 'n_jobs': 2,
 'random_state': 111}
```

In [68]:
```
Final_Model = ExtraTreesClassifier(criterion="gini", max_depth=30, min_samples_sp
Classifier = Final_Model.fit(X_train, Y_train)
fmod_pred = Final_Model.predict(X_test)
fmod_acc = (accuracy_score(Y_test, fmod_pred))*100
print("Accuracy score for the Best Model is:", fmod_acc)
```

```
Accuracy score for the Best Model is: 91.45299145299145
```

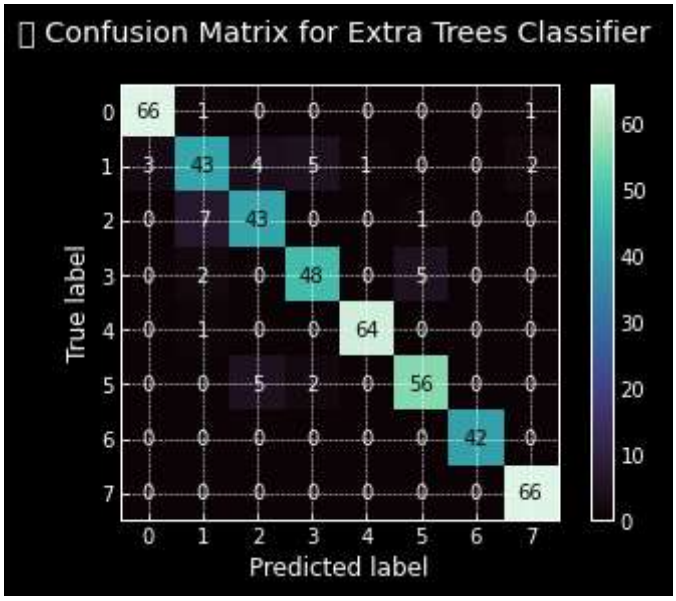
AUC ROC Curve for multi class label

In [69]:
```
y_prob = Classifier.predict_proba(X_test)

macro_roc_auc_ovo = roc_auc_score(Y_test, y_prob, multi_class="ovo", average="mac
weighted_roc_auc_ovo = roc_auc_score(Y_test, y_prob, multi_class="ovo", average="
macro_roc_auc_ovr = roc_auc_score(Y_test, y_prob, multi_class="ovr", average="mac
weighted_roc_auc_ovr = roc_auc_score(Y_test, y_prob, multi_class="ovr", average="
print("One-vs-One ROC AUC scores:\n{:.6f} (macro),\n{:.6f} "
      "(weighted by prevalence)"
      .format(macro_roc_auc_ovo, weighted_roc_auc_ovo))
print("="*40)
print("One-vs-Rest ROC AUC scores:\n{:.6f} (macro),\n{:.6f} "
      "(weighted by prevalence)"
      .format(macro_roc_auc_ovr, weighted_roc_auc_ovr))
```

```
One-vs-One ROC AUC scores:
0.993792 (macro),
0.993967 (weighted by prevalence)
========================================
One-vs-Rest ROC AUC scores:
0.993976 (macro),
0.994174 (weighted by prevalence)
```

Confusion Matrix

In [70]:
```
class_names = df.columns
metrics.plot_confusion_matrix(Classifier, X_test, Y_test, cmap='mako')
plt.title('\t Confusion Matrix for Extra Trees Classifier \n')
plt.show()
```



Saving the best Classification ML model

In [71]:
```
filename = "FinalModel_Classification_E04.pkl"
joblib.dump(Final_Model, filename)
```

Out[71]:
```
['FinalModel_Classification_E04.pkl']
```

Splitting the dataset into 2 variables namely 'X' and 'Y' for feature and regression label

In [72]:
```
X = df.drop('capacity_mw', axis=1)
Y = df['capacity_mw']
```

I have separated the dataset into features and regression label where X represents all the feature columns and Y represents the regression target label column

Feature Scaling

In [73]:
```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head() # Displaying all the features after applying scaling technique to avoid
```

Out[73]:

	latitude	longitude	primary_fuel	commissioning_year	source	geolocation_source
0	1.155327	-0.905807	1.174653	0.384586	1.578700	-1.059632
1	0.598448	-0.567933	-0.986874	0.671188	3.125148	0.400848
2	0.132019	-1.895590	1.606958	0.671188	-0.514953	0.400848
3	0.452596	3.134428	-0.554568	-0.116966	-0.491161	0.400848
4	0.141226	1.248497	-0.986874	0.671188	-0.491161	0.400848

Finding the best random state for building Regression Models

In [74]:
```
maxAccu=0
maxRS=0

for i in range(1, 1000):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran
    lr=LinearRegression()
    lr.fit(X_train, Y_train)
    pred = lr.predict(X_test)
    r2 = r2_score(Y_test, pred)

    if r2>maxAccu:
        maxAccu=r2
        maxRS=i

print("Best R2 score is", maxAccu,"on Random State", maxRS)
```

```
Best R2 score is 0.5342507568882502 on Random State 672
```

In [75]:
```
# Regression Model Function

def reg(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, ran

    # Training the model
    model.fit(X_train, Y_train)

    # Predicting Y_test
    pred = model.predict(X_test)

    # RMSE - a lower RMSE score is better than a higher one
    rmse = mean_squared_error(Y_test, pred, squared=False)
    print("RMSE Score is:", rmse)

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

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

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

I have built a regression function that splits the training and testing features and labels, then trains the model, predicts the label, calculates the RMSE score, generates the R2 score, calculates the Cross Validation score and finally finds the difference between the R2 score and Cross Validation score.

In [77]:
```
# Linear Regression Model

model=LinearRegression()
reg(model, X, Y)
```

```
RMSE Score is: 1.316047753228102
R2 Score is: 50.667731774233914
Cross Validation Score: 41.93834317486533
R2 Score - Cross Validation Score is 8.729388599368583
```

In [78]:
```
# Ridge Regression

model=Ridge(alpha=1e-2, normalize=True)
reg(model, X, Y)
```

```
RMSE Score is: 1.3163445478521831
R2 Score is: 50.64547847385627
Cross Validation Score: 41.964875678989415
R2 Score - Cross Validation Score is 8.680602794866857
```

In [80]:
```
# Support Vector Regression

model=SVR(C=1.0, epsilon=0.2, kernel='poly', gamma='auto')
reg(model, X, Y)
```

```
RMSE Score is: 1.1781920726054165
R2 Score is: 60.461518991102125
Cross Validation Score: 48.73928224288554
R2 Score - Cross Validation Score is 11.722236748216588
```

In [81]:
```
# Decision Tree Regressor

model=DecisionTreeRegressor(criterion="poisson", random_state=111)
reg(model, X, Y)
```

```
RMSE Score is: 1.361301122160665
R2 Score is: 47.21674201609639
Cross Validation Score: 37.41863008912064
R2 Score - Cross Validation Score is 9.79811192697575
```

In [82]:
```
# Random Forest Regressor

model=RandomForestRegressor(max_depth=2, max_features="sqrt")
reg(model, X, Y)
```

```
RMSE Score is: 1.2832607719806859
R2 Score is: 53.095163654972445
Cross Validation Score: 45.80481137312858
R2 Score - Cross Validation Score is 7.290352281843866
```

In [83]:
```
# Gradient Boosting Regressor

model=GradientBoostingRegressor(loss='quantile', n_estimators=200, max_depth=5)
reg(model, X, Y)
```

```
RMSE Score is: 1.6652132429375566
R2 Score is: 21.018113987391995
Cross Validation Score: 4.648026036660049
R2 Score - Cross Validation Score is 16.370087950731946
```

In [84]:
```
# Ada Boost Regressor

model=AdaBoostRegressor(n_estimators=300, learning_rate=1.05, random_state=42)
reg(model, X, Y)
```

```
RMSE Score is: 1.1671359208810121
R2 Score is: 61.200095287034806
Cross Validation Score: 54.91379554379362
R2 Score - Cross Validation Score is 6.286299743241187
```

In [85]:
```
# Extra Trees Regressor

model=ExtraTreesRegressor(n_estimators=200, max_features='sqrt', n_jobs=6)
reg(model, X, Y)
```

```
RMSE Score is: 1.0704725966164155
R2 Score is: 67.3608457877167
Cross Validation Score: 62.76554110636287
R2 Score - Cross Validation Score is 4.595304681353831
```

In [86]:
```
# XGB Regressor

model=XGBRegressor()
reg(model, X, Y)
```

```
RMSE Score is: 1.1700265189512444
R2 Score is: 61.007669002390315
Cross Validation Score: 56.768431000290434
R2 Score - Cross Validation Score is 4.239238002099881
```

Hyper parameter tuning on the best Regression ML Model

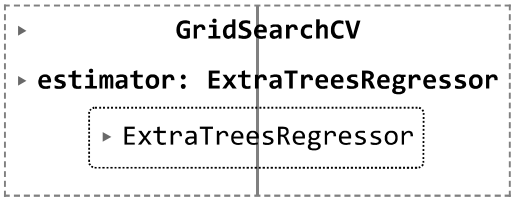
In [87]:
```
# Choosing Extra Trees Regressor

fmod_param = {'criterion' : ['mse', 'mae'],
              'n_estimators' : [100, 200],
              'min_samples_split' : [2, 3],
              'random_state' : [42, 135],
              'n_jobs' : [-1, 1]
             }
```

After comparing all the regression models I have selected Extra Trees Regressor as my best regression model and have listed down it's parameters above referring the sklearn webpage. I chose Extra Trees Regressor model becuase I feel among all the other regression models it was able to give me a better R2 score and a decent cross validation score for the same

In [88]:
```
GSCV = GridSearchCV(ExtraTreesRegressor(), fmod_param, cv=5)
```

In [89]:
```
GSCV.fit(X_train,Y_train)
```

Out[89]:



In [90]:
```
GSCV.best_params_
```

Out[90]:
```
{'criterion': 'mse',
 'min_samples_split': 3,
 'n_estimators': 200,
 'n_jobs': 1,
 'random_state': 42}
```

In [91]:
```
Final_Model = ExtraTreesRegressor(criterion='mse', min_samples_split=3, n_estimat
Classifier = Final_Model.fit(X_train, Y_train)
fmod_pred = Final_Model.predict(X_test)
fmod_r2 = r2_score(Y_test, fmod_pred)*100
print("R2 score for the Best Model is:", fmod_r2)
```

```
R2 score for the Best Model is: 68.09997386756837
```

Saving the best Regression ML model

In [92]:
```
filename = "FinalModel_Regression_E04.pkl"
joblib.dump(Final_Model, filename)
```

Out[92]:
```
['FinalModel_Regression_E04.pkl']
```

In [ ]:
```

```