GLOBAL POWER PLANT DATABASE PROJECT

In This dataset we will make two predictions .

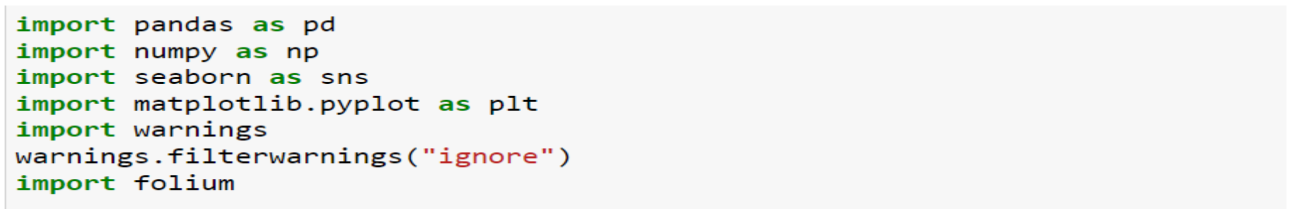
1. Primary Fuel

2. capacity_mw

In this blog , I will go through the whole process of creating machine learning models on the famous GLOBAL POWER PLANT dataset .

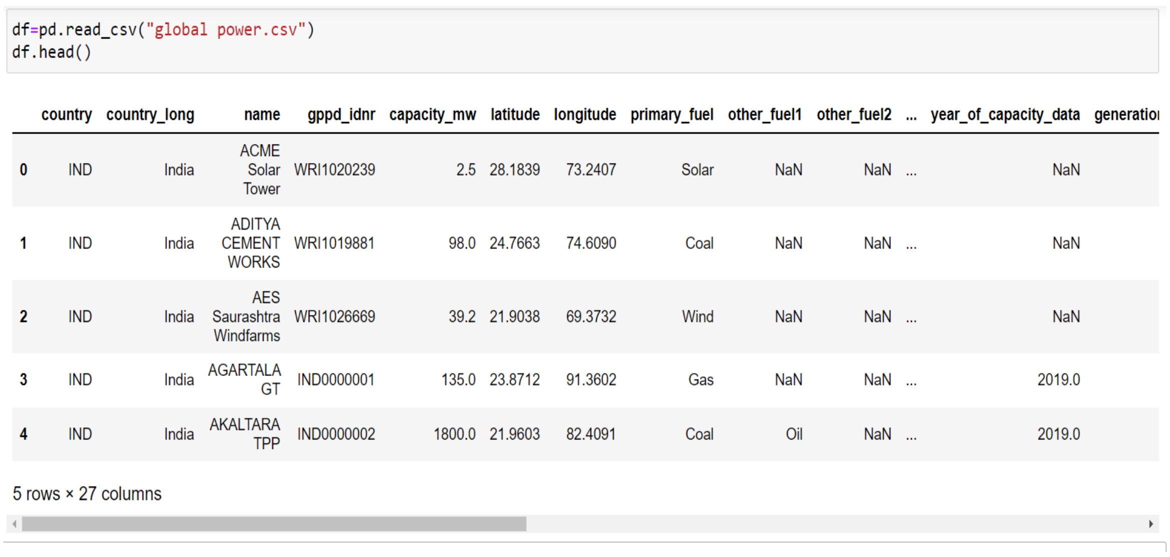
The Global Power Plant Database is a comprehensive, open source database of power plants around the world. It centralizes power plant data to make it easier to navigate, compare and draw insights for one's own analysis. The database covers approximately 35,000 power plants from 167 countries and includes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

Importing the Libraries



```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import folium
```

Uploading the csv file of the dataset



```
df=pd.read_csv("global power.csv")
df.head()
```

	country	country_long	name	gppd_idnr	capacity_mw	latitude	longitude	primary_fuel	other_fuel1	other_fuel2	...	year_of_capacity_data	generatio
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	

5 rows × 27 columns

here we can see there are 907 rows and 27 features in the dataset .

Key attributes of the database :

country (text): 3 character country code corresponding to the ISO 3166-1 alpha-3 specification

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

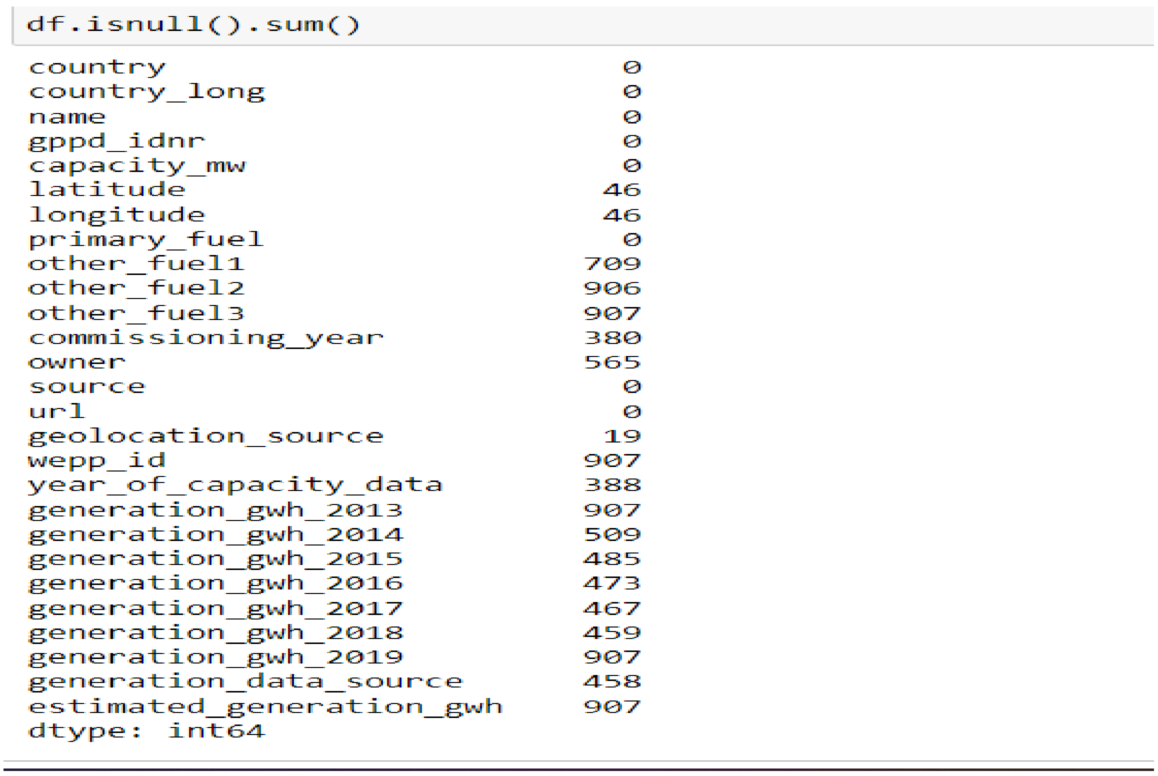
generation_gwh_2017 (number): electricity generation in gigawatt-hours reported for the year 2017

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

Data Explore / analysis (EDA)



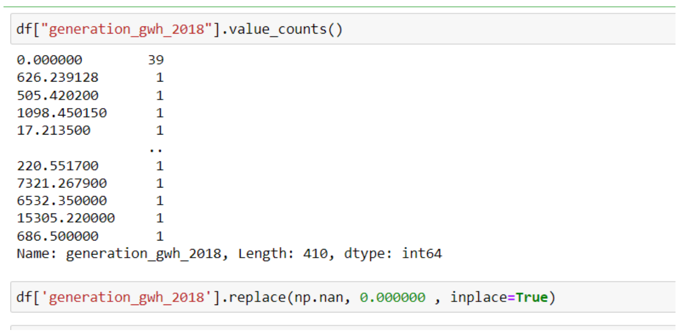
```
df.isnull().sum()
```

```
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
```

A lot of missing values are there in the data set . In the next step we have to remove some unimportant collumns and missing values.

"estimated_generation_gwh","owner","year_of_capacity_data","commissioning_year","generation_data_source","generation_gwh_2019", "wepp_id" ,"generation_gwh_2013" ,"other_fuel2" ,"other_fuel3" and "other_fuel1" are some unimportant features that we have removed from the data set.

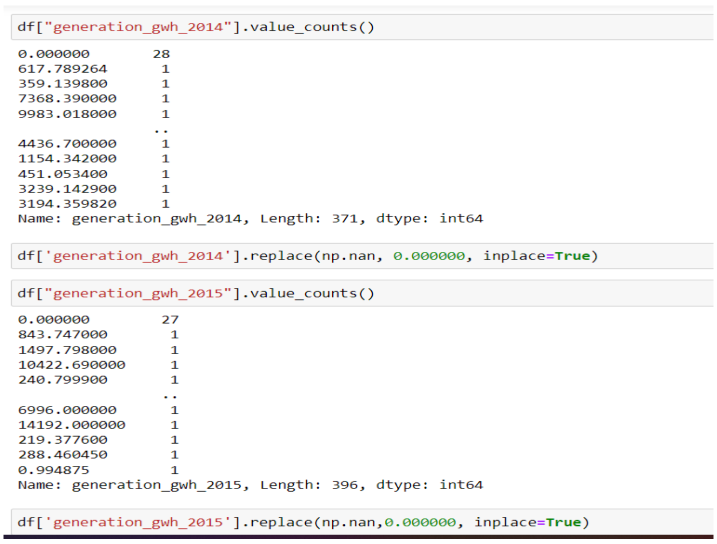
we have to take care all the missing values , so that there will be no missing values in the dataset .



```
df["generation_gwh_2018"].value_counts()
```

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

```
df['generation_gwh_2018'].replace(np.nan, 0.000000 , inplace=True)
```



```
df["generation_gwh_2014"].value_counts()
```

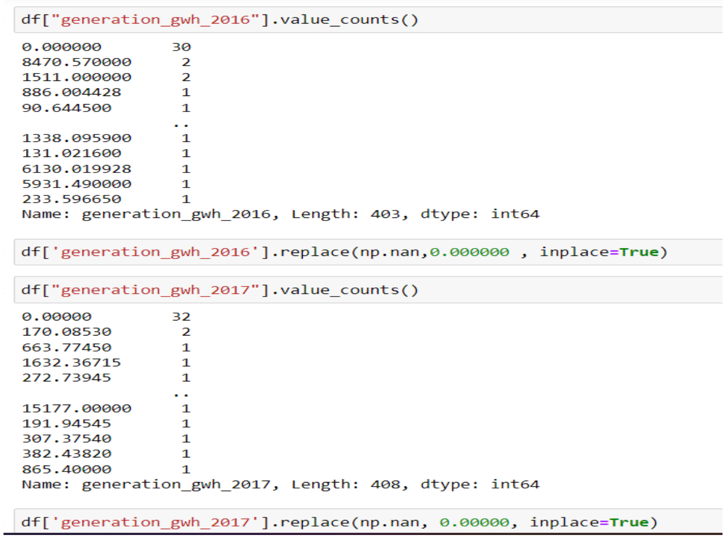
```
0.000000        28
617.789264       1
359.139800       1
7368.390000      1
9983.018000      1
                ..
4436.700000      1
1154.342000      1
451.053400       1
3239.142900      1
3194.359820      1
Name: generation_gwh_2014, Length: 371, dtype: int64
```

```
df['generation_gwh_2014'].replace(np.nan, 0.000000, inplace=True)
```

```
df["generation_gwh_2015"].value_counts()
```

```
0.000000         27
843.747000        1
1497.798000       1
10422.690000      1
240.799900        1
                 ..
6996.000000       1
14192.000000      1
219.377600        1
288.460450        1
0.994875          1
Name: generation_gwh_2015, Length: 396, dtype: int64
```

```
df['generation_gwh_2015'].replace(np.nan,0.000000, inplace=True)
```



```
df["generation_gwh_2016"].value_counts()
```

```
0.000000        30
8470.570000      2
1511.000000      2
886.004428       1
90.644500        1
                ..
1338.095900      1
131.021600       1
6130.019928      1
5931.490000      1
233.596650       1
Name: generation_gwh_2016, Length: 403, dtype: int64
```

```
df['generation_gwh_2016'].replace(np.nan,0.000000 , inplace=True)
```

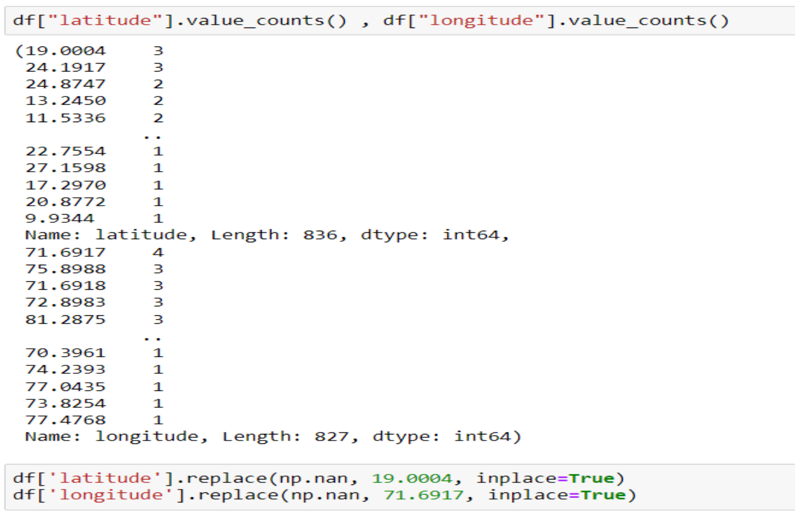
```
df["generation_gwh_2017"].value_counts()
```

```
0.00000         32
170.08530        2
663.77450        1
1632.36715       1
272.73945        1
                ..
15177.00000      1
191.94545        1
307.37540        1
382.43820        1
865.40000        1
Name: generation_gwh_2017, Length: 408, dtype: int64
```

```
df['generation_gwh_2017'].replace(np.nan, 0.00000, inplace=True)
```

In most of features we have replaed the nan values with most frequent values of those features.

Now we have to handle the missing values of Latitude and Longitude features .Here also we have replaced the missing values with most frequent values.

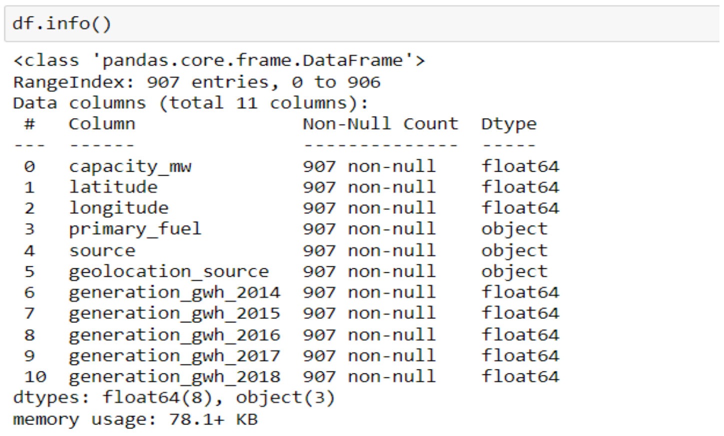


```
df["latitude"].value_counts() , df["longitude"].value_counts()
```

```
(19.0004     3
 24.1917     3
 24.8747     2
 13.2450     2
 11.5336     2
            ..
 22.7554     1
 27.1598     1
 17.2970     1
 20.8772     1
 9.9344      1
 Name: latitude, Length: 836, dtype: int64,
 71.6917     4
 75.8988     3
 71.6918     3
 72.8983     3
 81.2875     3
            ..
 70.3961     1
 74.2393     1
 77.0435     1
 73.8254     1
 77.4768     1
 Name: longitude, Length: 827, dtype: int64)
```

```
df['latitude'].replace(np.nan, 19.0004, inplace=True)
df['longitude'].replace(np.nan, 71.6917, inplace=True)
```

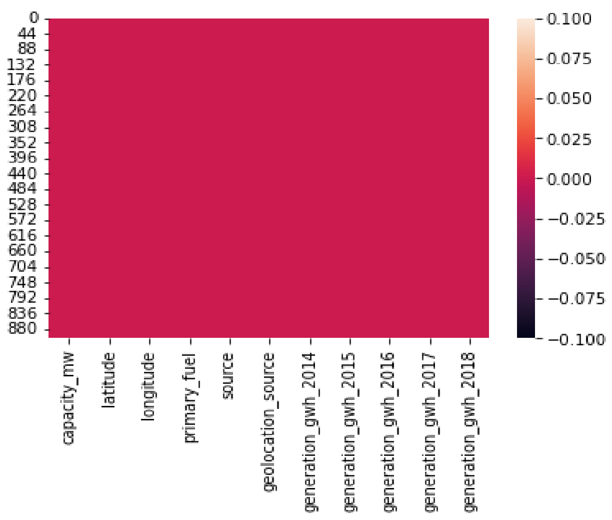
Now there is no missing values in the dataset .



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 907 entries, 0 to 906
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   capacity_mw          907 non-null    float64
 1   latitude             907 non-null    float64
 2   longitude            907 non-null    float64
 3   primary_fuel         907 non-null    object 
 4   source               907 non-null    object 
 5   geolocation_source   907 non-null    object 
 6   generation_gwh_2014  907 non-null    float64
 7   generation_gwh_2015  907 non-null    float64
 8   generation_gwh_2016  907 non-null    float64
 9   generation_gwh_2017  907 non-null    float64
 10  generation_gwh_2018  907 non-null    float64
dtypes: float64(8), object(3)
memory usage: 78.1+ KB
```

From above information we can see that after removing all unimportant features , missing values now there are 11 features in the dataset . Among them three features are object features . Now we will encode this object Features .



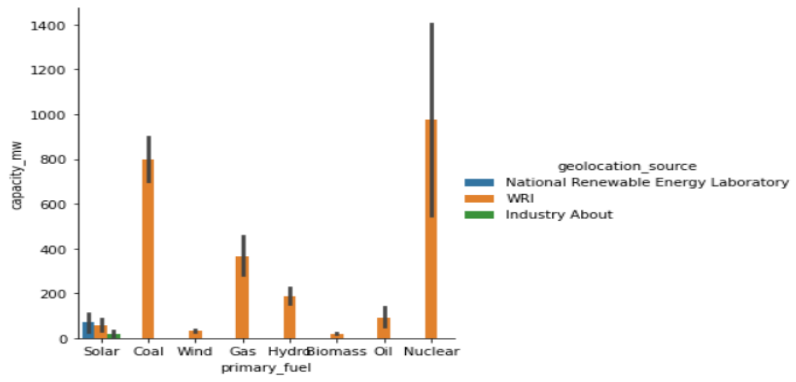
From this above heatmap we can visualise that there is no missing values in the dataset.

now we will be doing some univariate and multivariate analysis and will visualise the same through graphs.

```
plt.figure(figsize=(10,8))
sns.catplot(x="primary_fuel",y="capacity_mw",hue="geolocation_source", data = df,kind ="bar")
```

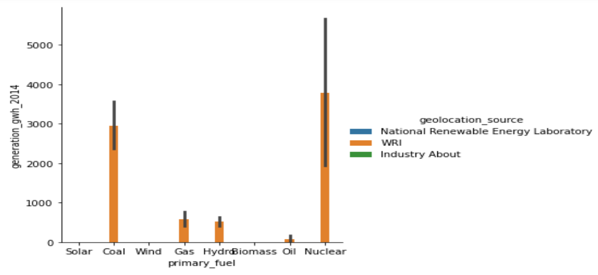
```
<seaborn.axisgrid.FacetGrid at 0x147e4fe5a90>
```

```
<Figure size 720x576 with 0 Axes>
```



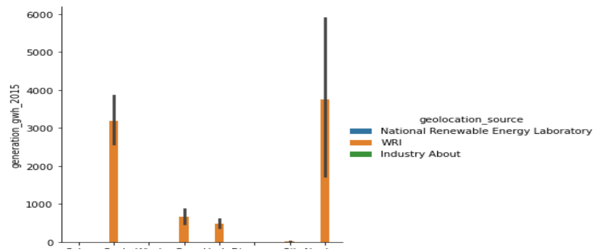
from the above visualisation we can see that electrical generating capacity is maximum in case of nuclear power plants and minimum in case of solar power plants and for the maximum power plants geolocation_source is WRI. Only in the solar power plants we can see the three kind of geolocation sources.

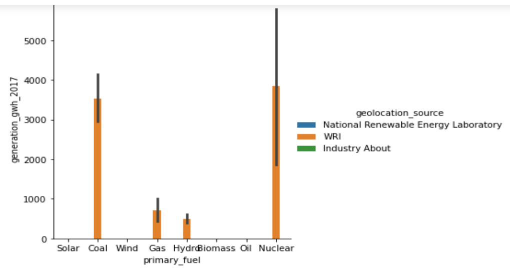
it is clearly shown that neauclear power plant is the first and then coal power plant comes in the second position in generating electrical generating capacity from 2014 to 2018.



```
sns.catplot(x="primary_fuel",y="generation_gwh_2015",hue="geolocation_source", data = df,kind ="bar")
```

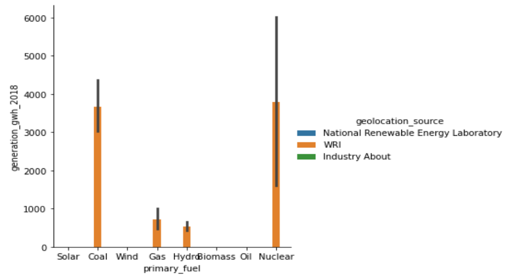
```
<seaborn.axisgrid.FacetGrid at 0x147e21ac9a0>
```

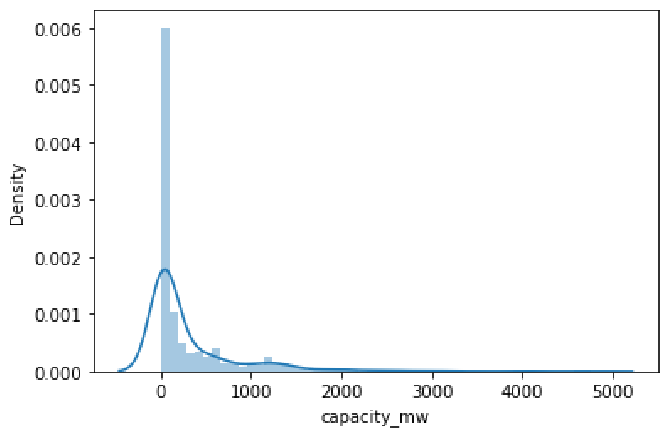




```
sns.catplot(x="primary_fuel",y="generation_gwh_2018",hue="geolocation_source", data = df,kind ="bar")
```

```
<seaborn.axisgrid.FacetGrid at 0x147e51035b0>
```





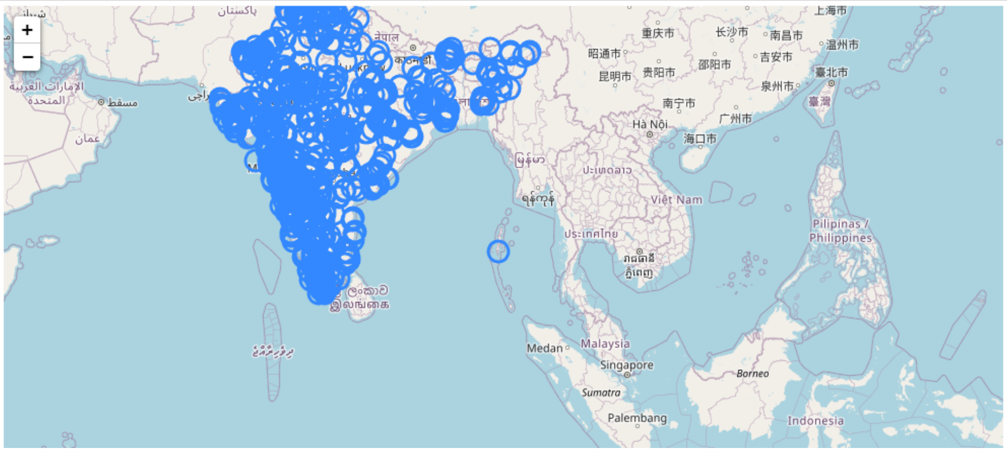
We can see the dist plot of capacity mw , it is clearly shown that the dataset is skewed.

```
map_powerplant = folium.Map( location=[19.0004,71.6917])
```

```
df.apply(lambda row:folium.CircleMarker(location=[row["latitude"], row["longitude"]] ).add_to(map_powerplant), axis=1)
```

```
0      <folium.vector_layers.CircleMarker object at 0...
1      <folium.vector_layers.CircleMarker object at 0...
2      <folium.vector_layers.CircleMarker object at 0...
3      <folium.vector_layers.CircleMarker object at 0...
4      <folium.vector_layers.CircleMarker object at 0...
                             ...
902    <folium.vector_layers.CircleMarker object at 0...
903    <folium.vector_layers.CircleMarker object at 0...
904    <folium.vector_layers.CircleMarker object at 0...
905    <folium.vector_layers.CircleMarker object at 0...
906    <folium.vector_layers.CircleMarker object at 0...
Length: 907, dtype: object
```

```
map_powerplant
```



Here we have installed folium and geopy to visualise the map and the location of the power plants using latitude and longitude values.

Now we will use ordinal encoder to encode the categorical columns .

```
import sklearn
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
```

here we are using ordinal encoder to encode the catagorical features.

```
for i in df.columns:
    if df[i].dtypes=="object":
        df[i]= oe.fit_transform(df[i].values.reshape(-1,1))
df
```

	capacity_mw	latitude	longitude	primary_fuel	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_
0	2.5	28.1839	73.2407	6.0	109.0	1.0	0.000000	0.000000	0.000000	
1	98.0	24.7663	74.6090	1.0	174.0	2.0	0.000000	0.000000	0.000000	
2	39.2	21.9038	69.3732	7.0	21.0	2.0	0.000000	0.000000	0.000000	
3	135.0	23.8712	91.3602	2.0	22.0	2.0	617.789264	843.747000	886.004428	6
4	1800.0	21.9603	82.4091	1.0	22.0	2.0	3035.550000	5916.370000	6243.000000	53
...	...	...	...	...	...	...	...	...	...	
902	1600.0	16.2949	77.3568	1.0	22.0	2.0	0.000000	0.994875	233.596650	8
903	3.0	12.8932	78.1654	6.0	77.0	0.0	0.000000	0.000000	0.000000	
904	25.5	15.2758	75.5811	7.0	21.0	2.0	0.000000	0.000000	0.000000	
905	80.0	24.3500	73.7477	1.0	59.0	2.0	0.000000	0.000000	0.000000	
906	16.5	9.9344	77.4768	7.0	21.0	2.0	0.000000	0.000000	0.000000	

907 rows × 11 columns

Now we will see the statistical description and the heatmap of correlation between various features of the dataset.

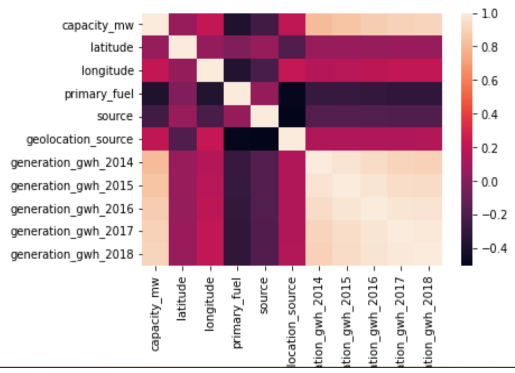
```
df.describe()
```

	capacity_mw	latitude	longitude	primary_fuel	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016
count	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000	907.000000
mean	326.223755	21.086467	77.172109	3.206174	43.847850	1.733186	1067.106713	1129.781446	1180.909147
std	590.085456	6.098262	4.976401	2.280652	44.642818	0.677151	2926.078576	3105.552487	3131.100641
min	0.000000	8.168900	68.644700	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	16.725000	17.072000	73.811550	1.000000	22.000000	2.000000	0.000000	0.000000	0.000000
50%	59.200000	21.281800	76.493800	3.000000	22.000000	2.000000	0.000000	0.000000	0.000000
75%	385.250000	25.176450	79.206100	6.000000	29.500000	2.000000	556.562750	598.926350	670.039394
max	4760.000000	34.649000	95.408000	7.000000	190.000000	2.000000	28127.000000	30539.000000	30015.000000

maximum features are catagorical here so nothing much to observe in the description .

```
sns.heatmap(df.corr())
```

```
<AxesSubplot:>
```



maximum features are catagorical here so nothing much to observe in the description

```
x1=df.drop(["latitude" ,"longitude","primary_fuel"],axis =1)
y1=df["primary_fuel"]
x2=df.drop(["latitude" ,"longitude","capacity_mw"],axis =1)
y2=df["capacity_mw"]
```

here we have splitted our data set ,, we have to predict two features firstly we will predict primary fuel , it will be a classification model and secondly we have to predict capacity_mw , it is a regression model.

```
from sklearn.preprocessing import power_transform
xt=power_transform(x1, method="yeo-johnson")
xt=pd.DataFrame(xt)
xt.columns = x1.columns
xt
```

	capacity_mw	source	geolocation_source	generation_gwh_2014	generation_gwh_2015	generation_gwh_2016	generation_gwh_2017	generation_gwh_2018
0	-1.695767	1.582861	-2.345400	-0.813809	-0.853614	-0.872976	-0.878131	-0.883788
1	0.193694	2.036658	0.396062	-0.813809	-0.853614	-0.872976	-0.878131	-0.883788
2	-0.292337	-0.471879	0.396062	-0.813809	-0.853614	-0.872976	-0.878131	-0.883788
3	0.360998	-0.403248	0.396062	1.174182	1.168585	1.140233	1.069326	1.038383
4	1.649503	-0.403248	0.396062	1.380682	1.432242	1.423083	1.390376	1.420200
...	...	...	...	...	...	...	...	...
902	1.593509	-0.403248	0.396062	-0.813809	-0.523935	0.892411	1.115671	1.055150
903	-1.615576	1.211010	-2.530510	-0.813809	-0.853614	-0.872976	-0.878131	-0.883788
904	-0.522773	-0.471879	0.396062	-0.813809	-0.853614	-0.872976	-0.878131	-0.883788
905	0.086915	0.904767	0.396062	-0.813809	-0.853614	-0.872976	-0.878131	-0.883788
906	-0.755891	-0.471879	0.396062	-0.813809	-0.853614	-0.872976	-0.878131	-0.883788

907 rows × 8 columns

Here we have used power transform Yeo_johnson method to transform the dataset and remove skewness.

```
import statsmodels.api as sm
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

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

```
calvif=calc_vif(xt)
print(calvif)
calvif["V_I_F"].sort_values(ascending=False)
```

```
            variables      V_I_F
0         capacity_mw   3.059555
1              source   1.273636
2   geolocation_source   1.418044
3  generation_gwh_2014   7.690201
4  generation_gwh_2015  16.219644
5  generation_gwh_2016  23.855960
6  generation_gwh_2017  14.245340

5    23.855960
4    16.219644
6    14.245340
3     7.690201
0     3.059555
2     1.418044
1     1.273636
Name: V_I_F, dtype: float64
```

Here we have imported statsmodel , scipy library and varience _ inflation_factor to check if there is any multicollinearity present in the dataset and we can see that VIF is more than 10 in generation growth of various years lie 2015, 2016 , 2017 ,, so we have to remove some features of high VIF values.

```
calvif=calc_vif(xp1)
print(calvif)
calvif["V_I_F"].sort_values(ascending=False)
```

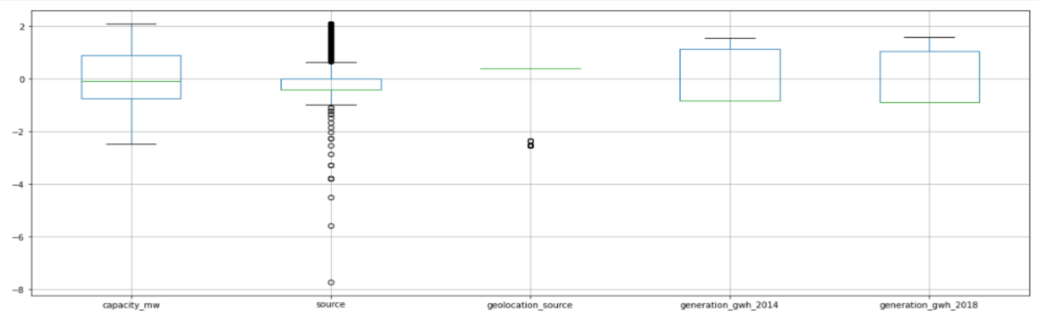
```
            variables     V_I_F
0         capacity_mw  2.895870
1              source  1.273936
2   geolocation_source  1.413136
3  generation_gwh_2014  3.658066
4  generation_gwh_2018  4.996538

4    4.996538
3    3.658066
0    2.895870
2    1.413136
1    1.273936
Name: V_I_F, dtype: float64
```

after removing some features , where high multi coliniarity was present , now the next step is scaling the data set.

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
xc=sc.fit_transform(xp1)
xc=pd.DataFrame(xc)
xc.columns=xp1.columns
xc
```

	capacity_mw	source	geolocation_source	generation_gwh_2014	generation_gwh_2018
0	-1.695767	1.582861	-2.345400	-0.813809	-0.883788
1	0.193694	2.036658	0.396062	-0.813809	-0.883788
2	-0.292337	-0.471879	0.396062	-0.813809	-0.883788
3	0.360998	-0.403248	0.396062	1.174182	1.038383
4	1.649503	-0.403248	0.396062	1.380682	1.420200
...	...	...	...	...	...
902	1.593509	-0.403248	0.396062	-0.813809	1.055150
903	-1.615576	1.211010	-2.530510	-0.813809	-0.883788
904	-0.522773	-0.471879	0.396062	-0.813809	-0.883788
905	0.086915	0.904767	0.396062	-0.813809	-0.883788

After removing the features with high VIF values now we can see that VIF values of remaining features are below 10 and then we have used standard scaler to scale the dataset .



though we can see that outliers are present , but maximum features are catagorical so not worry here.

```
from imblearn.over_sampling import SMOTE
smote=SMOTE()
xtrainw,ytrainw=smote.fit_resample(xc,y1)
```

here we have used smote method to balance the dataset.

```
ytrainw.value_counts()
```

```
6.0    258
1.0    258
7.0    258
2.0    258
3.0    258
0.0    258
5.0    258
4.0    258
Name: primary_fuel, dtype: int64
```

We can see in the above boxplots , that some outliers are present in the dataset , but we can ignore it because maximum features are categorical in nature.

Then we have used smote technique to balance the dataset .

Now the dataset is ready for model creation and using of different ML algorithms for making predictions .

```
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score , confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
```

Here we have imported various classification models as our first prediction model is primary fuel and it is a classification model .

```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=i)
    knc=KNeighborsClassifier()
    knc.fit(x_train,y_train)
    predknc=knc.predict(x_test)
    acc=accuracy_score(y_test,predknc)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.8595641646489104 on random state 152
```

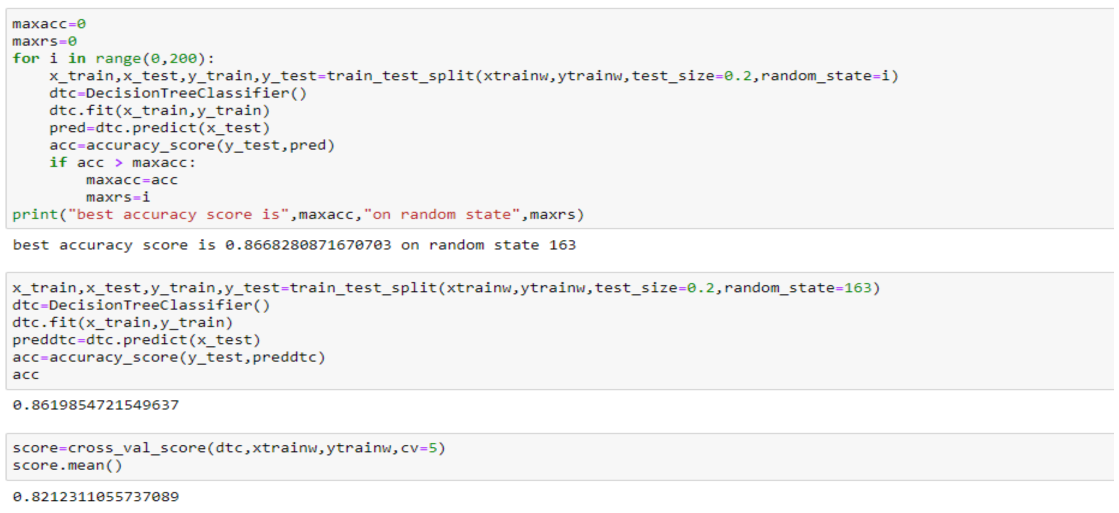
```
x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=152)
knc=KNeighborsClassifier()
knc.fit(x_train,y_train)
predknc=knc.predict(x_test)
acc=accuracy_score(y_test,predknc)
acc
```

```
0.8595641646489104
```

```
score=cross_val_score(knc,xtrainw,ytrainw,cv=5)
score.mean()
```

```
0.8144373398528408
```

At first we have used k_neighbours classifier and at random state 152 it has given the best score that is around 86% where the cross validation score for this model is 82 % , that is also very near to the model score .



```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=i)
    dtc=DecisionTreeClassifier()
    dtc.fit(x_train,y_train)
    pred=dtc.predict(x_test)
    acc=accuracy_score(y_test,pred)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.8668280871670703 on random state 163
```

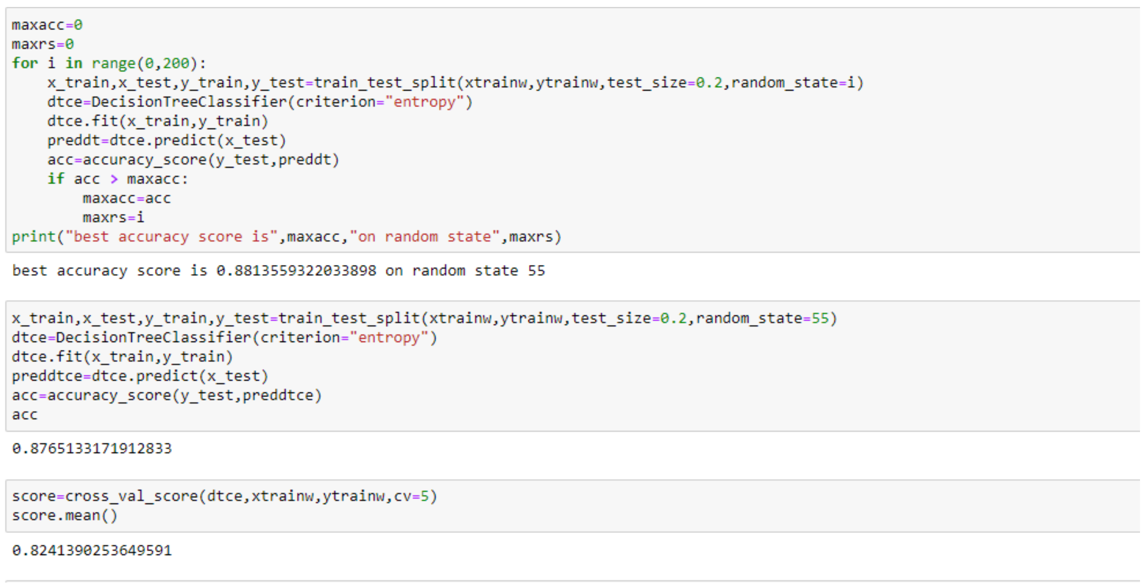
```
x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=163)
dtc=DecisionTreeClassifier()
dtc.fit(x_train,y_train)
preddtc=dtc.predict(x_test)
acc=accuracy_score(y_test,preddtc)
acc
```

```
0.8619854721549637
```

```
score=cross_val_score(dtc,xtrainw,ytrainw,cv=5)
score.mean()
```

```
0.8212311055737089
```

Now we have used decision tree classifier and at random state 163 it has given the best score that is around 86% where the cross validation score for this model is 82 % , that is also very near to the model score . here wehave not mentioned the criterion , so by default the criterion is genny .



```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=i)
    dtce=DecisionTreeClassifier(criterion="entropy")
    dtce.fit(x_train,y_train)
    preddt=dtce.predict(x_test)
    acc=accuracy_score(y_test,preddt)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.8813559322033898 on random state 55
```

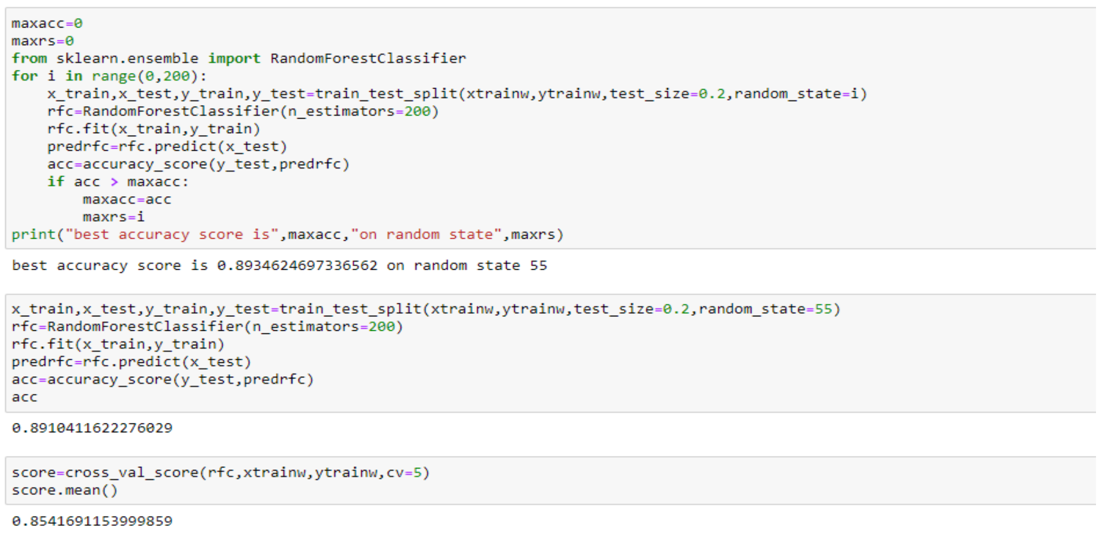
```
x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=55)
dtce=DecisionTreeClassifier(criterion="entropy")
dtce.fit(x_train,y_train)
preddtce=dtce.predict(x_test)
acc=accuracy_score(y_test,preddtce)
acc
```

```
0.8765133171912833
```

```
score=cross_val_score(dtce,xtrainw,ytrainw,cv=5)
score.mean()
```

```
0.8241390253649591
```

Now we have used decision tree classifier where the criterion is entropy and at random state 55 it has given the best score that is around 88% where the cross validation score for this model is around 88 % , that is also very near to the model score .



```
maxacc=0
maxrs=0
from sklearn.ensemble import RandomForestClassifier
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=i)
    rfc=RandomForestClassifier(n_estimators=200)
    rfc.fit(x_train,y_train)
    predrfc=rfc.predict(x_test)
    acc=accuracy_score(y_test,predrfc)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.8934624697336562 on random state 55
```

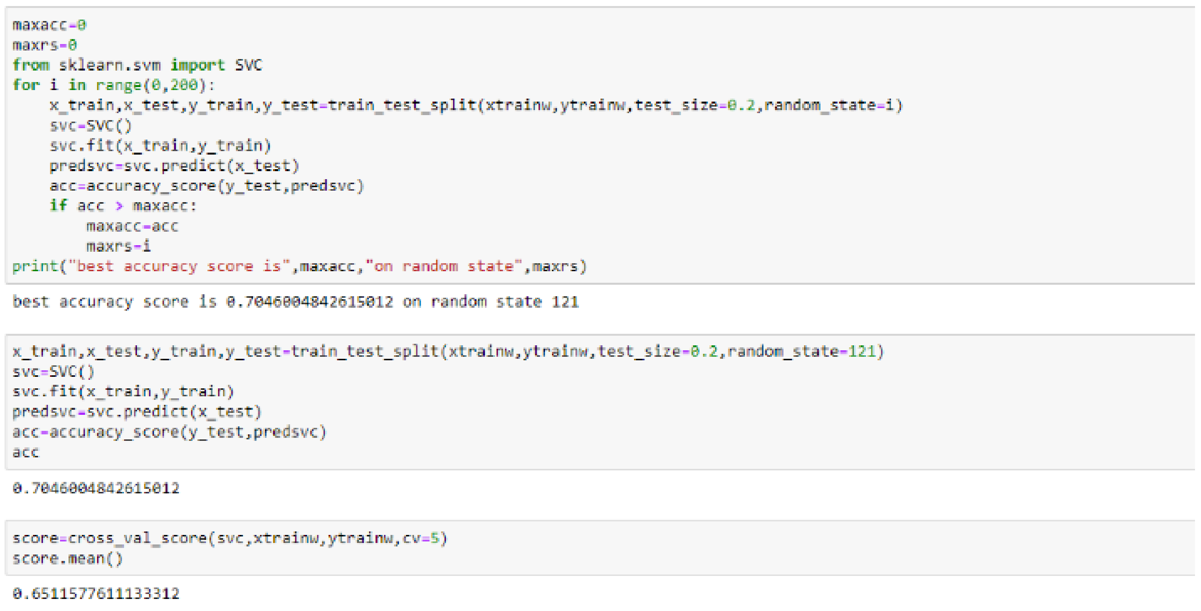
```
x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=55)
rfc=RandomForestClassifier(n_estimators=200)
rfc.fit(x_train,y_train)
predrfc=rfc.predict(x_test)
acc=accuracy_score(y_test,predrfc)
acc
```

```
0.8910411622276029
```

```
score=cross_val_score(rfc,xtrainw,ytrainw,cv=5)
score.mean()
```

```
0.8541691153999859
```

Now we have used Random Forest classifier where the criterion is entropy and at random state 55 it has given the best score that is around 89% where the cross validation score for this model is around 85 % , that is also very near to the model score .



```
maxacc=0
maxrs=0
from sklearn.svm import SVC
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=i)
    svc=SVC()
    svc.fit(x_train,y_train)
    predsvc=svc.predict(x_test)
    acc=accuracy_score(y_test,predsvc)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.7046004842615012 on random state 121
```

```
x_train,x_test,y_train,y_test=train_test_split(xtrainw,ytrainw,test_size=0.2,random_state=121)
svc=SVC()
svc.fit(x_train,y_train)
predsvc=svc.predict(x_test)
acc=accuracy_score(y_test,predsvc)
acc
```

```
0.7046004842615012
```

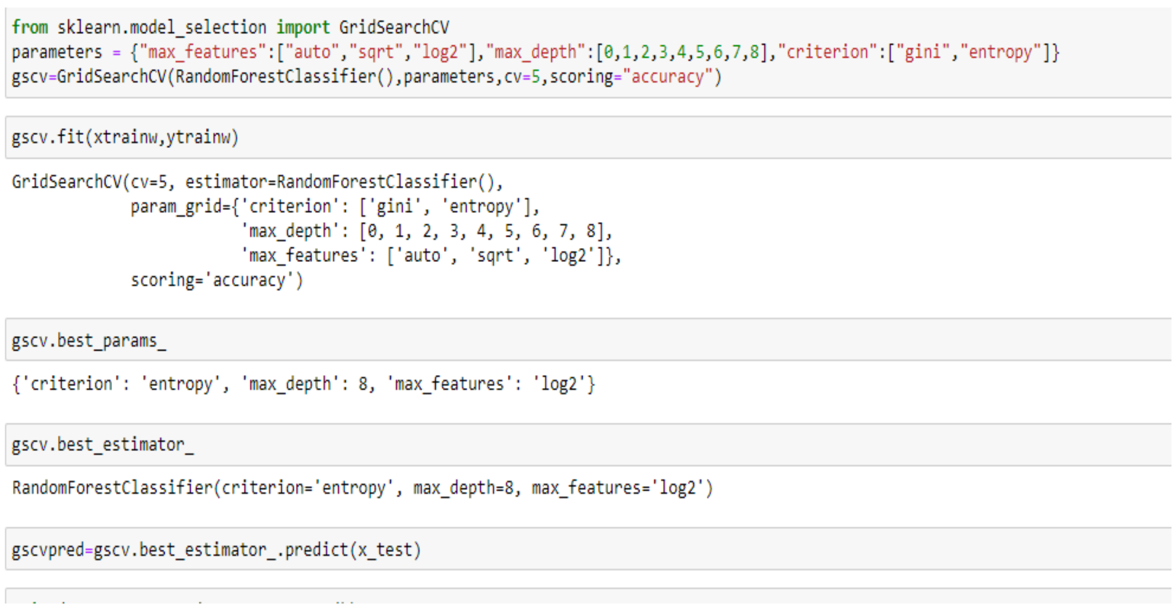
```
score=cross_val_score(svc,xtrainw,ytrainw,cv=5)
score.mean()
```

```
0.6511577611133312
```

Now we have used SupportVector classifier where the criterion is entropy and at random state 121 it has given the best score that is around 70% where the cross validation score for this model is around 65 % , that is also very near to the model score . SVC has given verry poor results in comparison to other models.

from various models random forest classifier has given the best result and the cross validation score is comparatively good for this model so we have decided to use this model for hyper tuning and getting the score

now we will do hyper tunning for random forest classifier model and for that we will import gridsearch cv then we will check the best parametes and best estimators for the same.



```
from sklearn.model_selection import GridSearchCV
parameters = {"max_features":["auto","sqrt","log2"],"max_depth":[0,1,2,3,4,5,6,7,8],"criterion":["gini","entropy"]}
gscv=GridSearchCV(RandomForestClassifier(),parameters,cv=5,scoring="accuracy")
```

```
gscv.fit(xtrainw,ytrainw)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [0, 1, 2, 3, 4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2']},
             scoring='accuracy')
```

```
gscv.best_params_
```

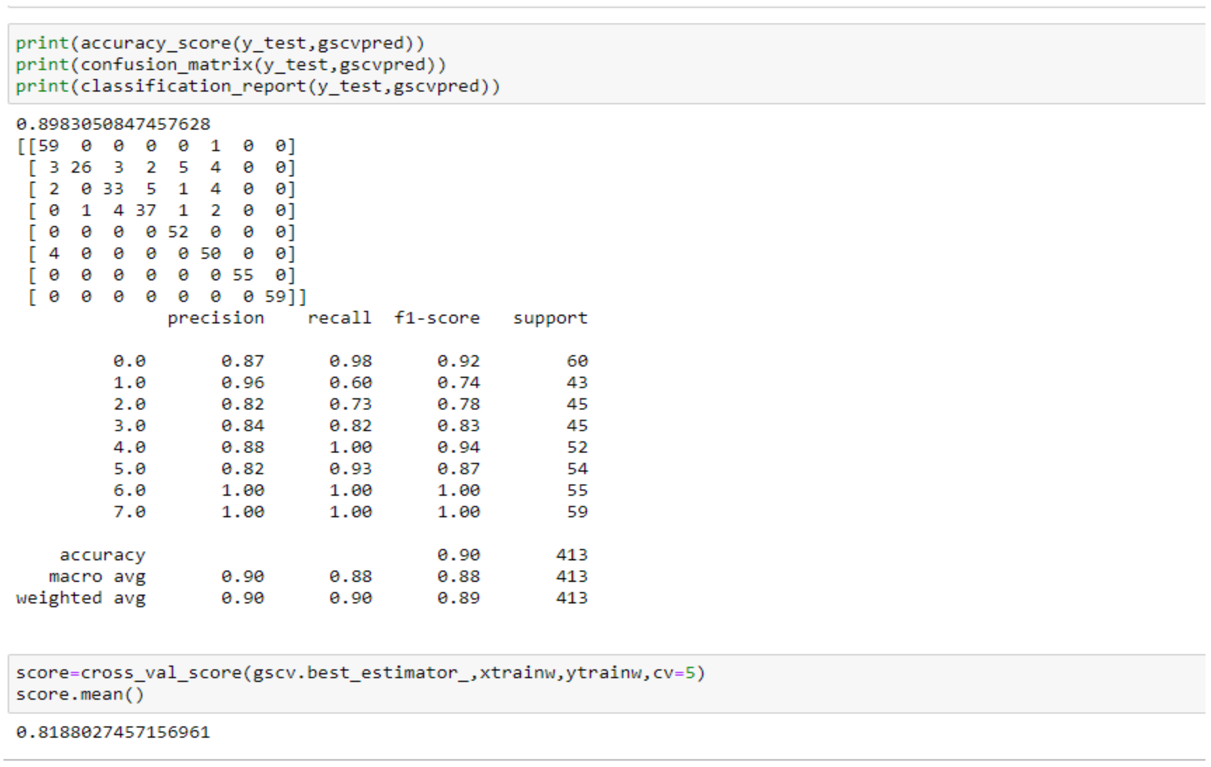
```
{'criterion': 'entropy', 'max_depth': 8, 'max_features': 'log2'}
```

```
gscv.best_estimator_
```

```
RandomForestClassifier(criterion='entropy', max_depth=8, max_features='log2')
```

```
gscvpred=gscv.best_estimator_.predict(x_test)
```

in the first model the accuracy score is around 90 % and cross validation score is around 82% .



```
print(accuracy_score(y_test,gscvpred))
print(confusion_matrix(y_test,gscvpred))
print(classification_report(y_test,gscvpred))
```

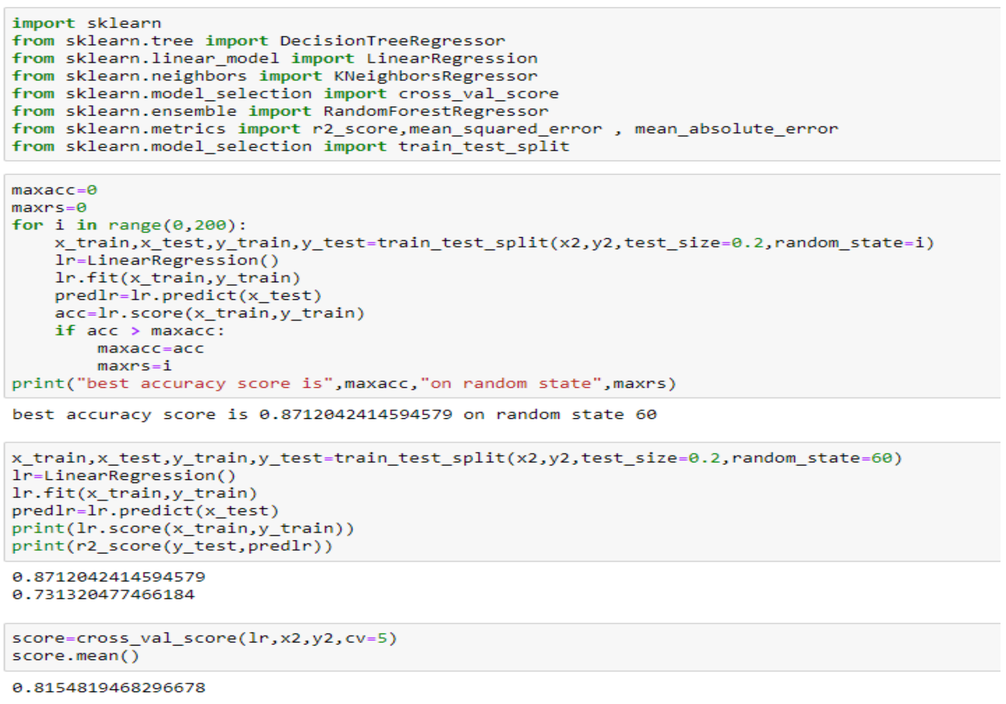
```
0.8983050847457628
[[59  0  0  0  0  1  0  0]
 [ 3 26  3  2  5  4  0  0]
 [ 2  0 33  5  1  4  0  0]
 [ 0  1  4 37  1  2  0  0]
 [ 0  0  0  0 52  0  0  0]
 [ 4  0  0  0  0 50  0  0]
 [ 0  0  0  0  0  0 55  0]
 [ 0  0  0  0  0  0  0 59]]
              precision    recall  f1-score   support

         0.0       0.87      0.98      0.92        60
         1.0       0.96      0.60      0.74        43
         2.0       0.82      0.73      0.78        45
         3.0       0.84      0.82      0.83        45
         4.0       0.88      1.00      0.94        52
         5.0       0.82      0.93      0.87        54
         6.0       1.00      1.00      1.00        55
         7.0       1.00      1.00      1.00        59

    accuracy                           0.90       413
   macro avg       0.90      0.88      0.88       413
weighted avg       0.90      0.90      0.89       413
```

```
score=cross_val_score(gscv.best_estimator_,xtrainw,ytrainw,cv=5)
score.mean()
```

```
0.8188027457156961
```

Here we can also check the confusion matrix , precision value , recall value and f1 scoreas well.

Now we have to use different regression model for the prediction of capacity_mw.



```
import sklearn
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score,mean_squared_error , mean_absolute_error
from sklearn.model_selection import train_test_split
```

```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=i)
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    predlr=lr.predict(x_test)
    acc=lr.score(x_train,y_train)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.8712042414594579 on random state 60
```

```
x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=60)
lr=LinearRegression()
lr.fit(x_train,y_train)
predlr=lr.predict(x_test)
print(lr.score(x_train,y_train))
print(r2_score(y_test,predlr))
```

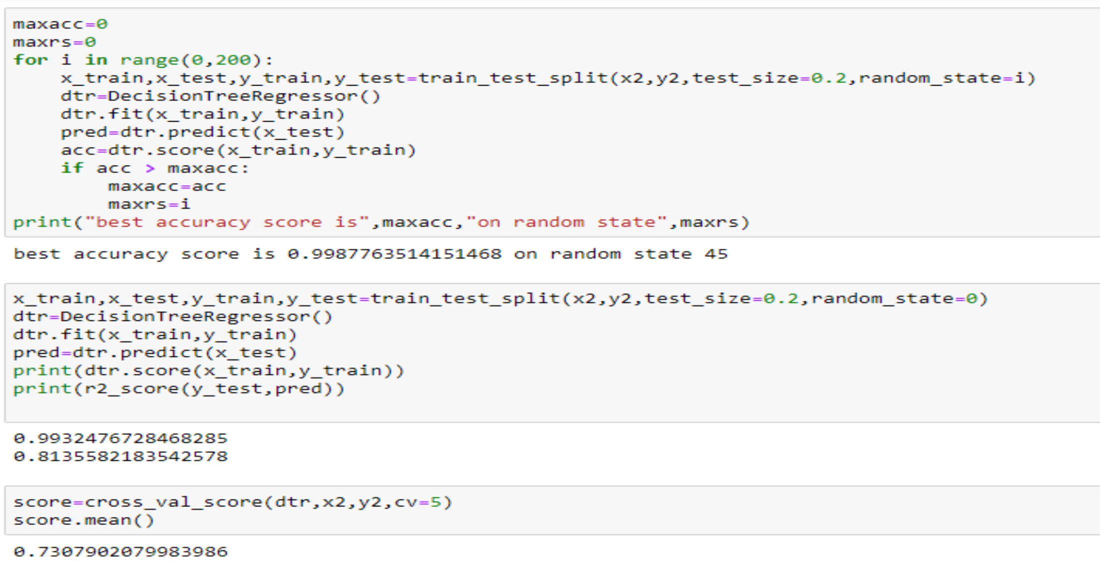
```
0.8712042414594579
0.731320477466184
```

```
score=cross_val_score(lr,x2,y2,cv=5)
score.mean()
```

```
0.8154819468296678
```

Here we have imported different regression models .

First we have used Linear Regression model and at random state 60 it has given the best score that is around 87% where the cross validation score for this model is around 82 % , that is also very near to the model score



```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=i)
    dtr=DecisionTreeRegressor()
    dtr.fit(x_train,y_train)
    pred=dtr.predict(x_test)
    acc=dtr.score(x_train,y_train)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.9987763514151468 on random state 45
```

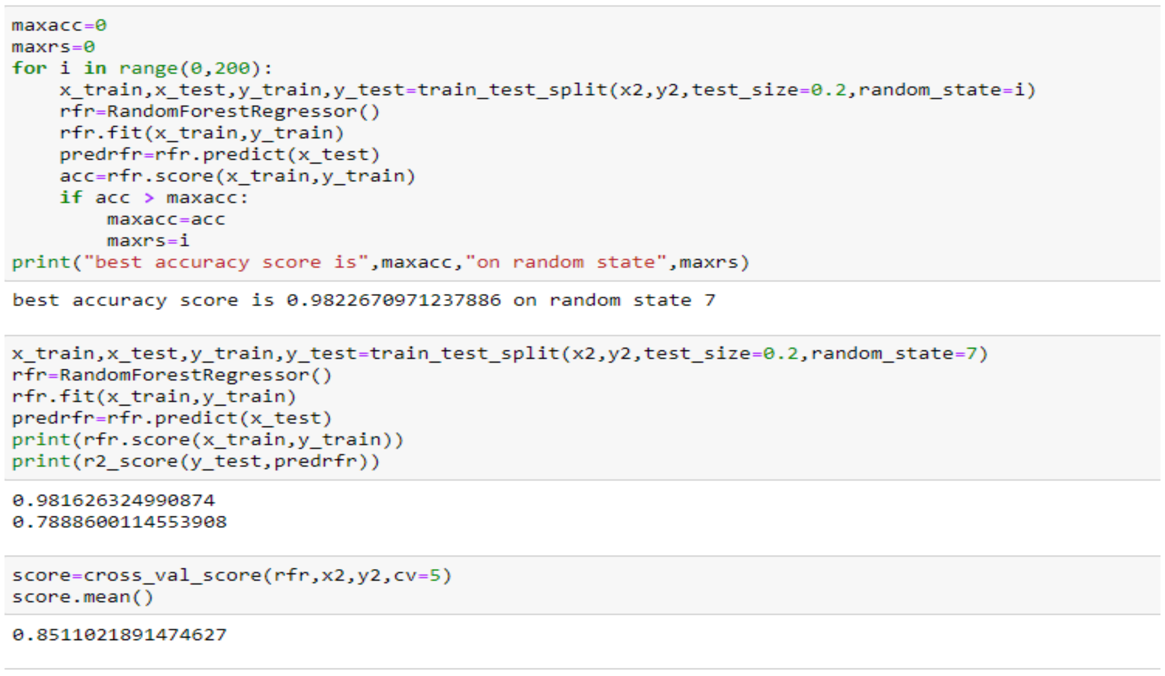
```
x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=0)
dtr=DecisionTreeRegressor()
dtr.fit(x_train,y_train)
pred=dtr.predict(x_test)
print(dtr.score(x_train,y_train))
print(r2_score(y_test,pred))
```

```
0.9932476728468285
0.8135582183542578
```

```
score=cross_val_score(dtr,x2,y2,cv=5)
score.mean()
```

```
0.7307902079983986
```

Now we have used Decision tree Regression model and at random state 0 it has given the best score that is around 99% where the cross validation score for this model is around 73 % .



```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=i)
    rfr=RandomForestRegressor()
    rfr.fit(x_train,y_train)
    predrfr=rfr.predict(x_test)
    acc=rfr.score(x_train,y_train)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.9822670971237886 on random state 7
```

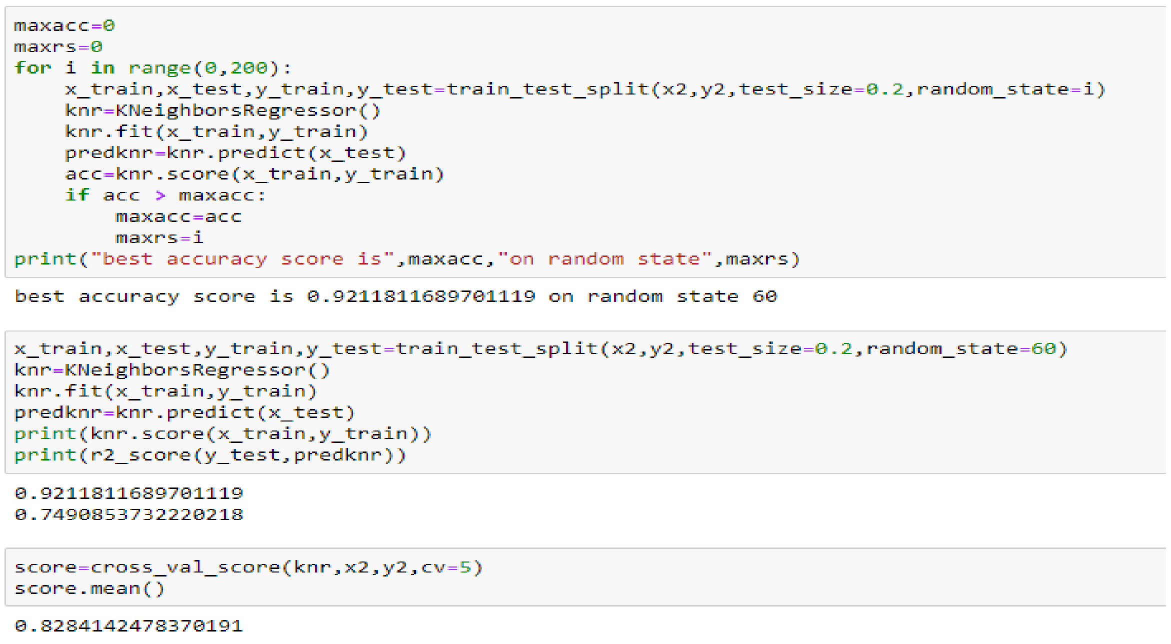
```
x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=7)
rfr=RandomForestRegressor()
rfr.fit(x_train,y_train)
predrfr=rfr.predict(x_test)
print(rfr.score(x_train,y_train))
print(r2_score(y_test,predrfr))
```

```
0.981626324990874
0.7888600114553908
```

```
score=cross_val_score(rfr,x2,y2,cv=5)
score.mean()
```

```
0.8511021891474627
```

Now we have used Random Forest Regression model and at random state 7 it has given the best score that is around 98% where the cross validation score for this model is around 85 % , that is also very near to the model score .



```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=i)
    knr=KNeighborsRegressor()
    knr.fit(x_train,y_train)
    predknr=knr.predict(x_test)
    acc=knr.score(x_train,y_train)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.9211811689701119 on random state 60
```

```
x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=60)
knr=KNeighborsRegressor()
knr.fit(x_train,y_train)
predknr=knr.predict(x_test)
print(knr.score(x_train,y_train))
print(r2_score(y_test,predknr))
```

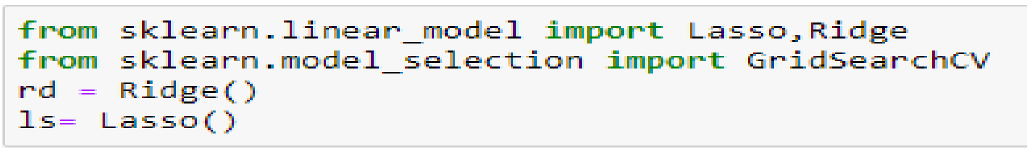
```
0.9211811689701119
0.7490853732220218
```

```
score=cross_val_score(knr,x2,y2,cv=5)
score.mean()
```

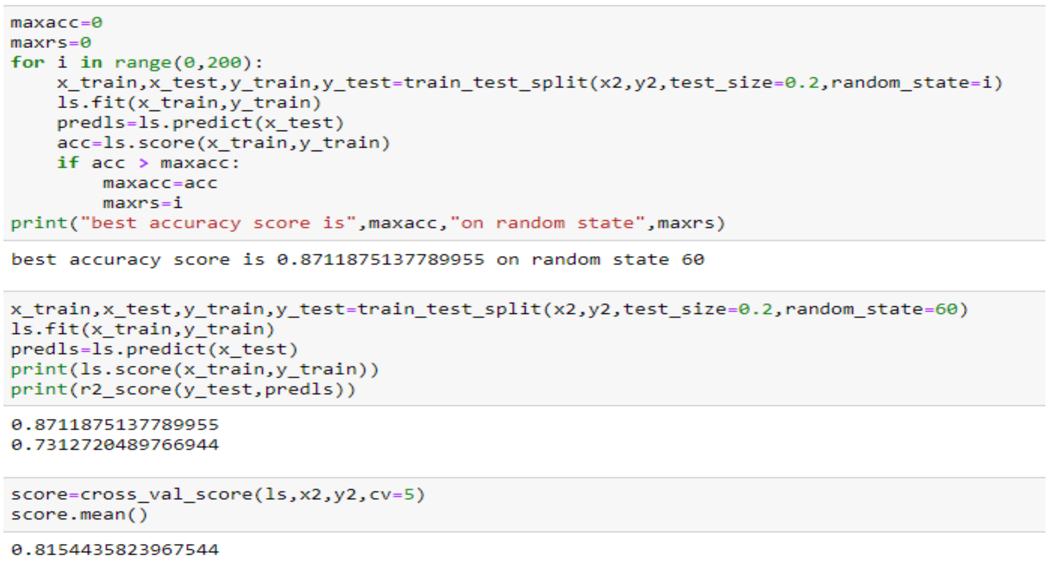
```
0.8284142478370191
```

Now we have used KNeighbous Regression model and at random state 60 it has given the best score that is around 92% where the cross validation score for this model is around 82 % , that is also very near to the model score , but r2 score is poor ,75% .

Now we will import Lasso and Ridge regression to check whether the the model is over fitted or underfitted it will give the better scores.



```
from sklearn.linear_model import Lasso,Ridge
from sklearn.model_selection import GridSearchCV
rd = Ridge()
ls= Lasso()
```



```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=i)
    ls.fit(x_train,y_train)
    predls=ls.predict(x_test)
    acc=ls.score(x_train,y_train)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.8711875137789955 on random state 60
```

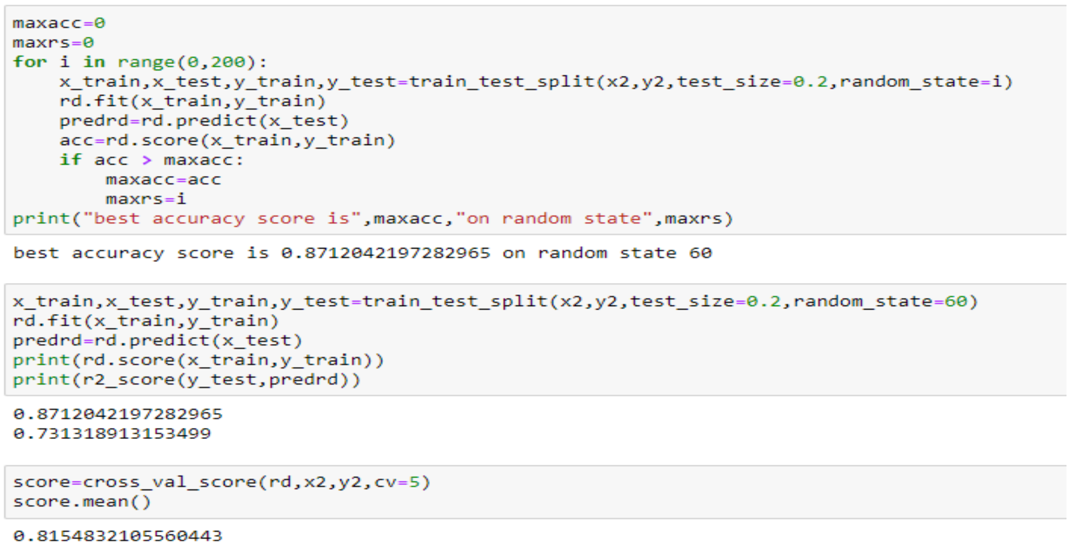
```
x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=60)
ls.fit(x_train,y_train)
predls=ls.predict(x_test)
print(ls.score(x_train,y_train))
print(r2_score(y_test,predls))
```

```
0.8711875137789955
0.7312720489766944
```

```
score=cross_val_score(ls,x2,y2,cv=5)
score.mean()
```

```
0.8154435823967544
```

At random state 60 Lasso Regression has given the best score that is around 87% where the cross validation score for this model is around 82 % , that is also very near to the model score , but r2 score is poor ,73% .



```
maxacc=0
maxrs=0
for i in range(0,200):
    x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=i)
    rd.fit(x_train,y_train)
    predrd=rd.predict(x_test)
    acc=rd.score(x_train,y_train)
    if acc > maxacc:
        maxacc=acc
        maxrs=i
print("best accuracy score is",maxacc,"on random state",maxrs)
```

```
best accuracy score is 0.8712042197282965 on random state 60
```

```
x_train,x_test,y_train,y_test=train_test_split(x2,y2,test_size=0.2,random_state=60)
rd.fit(x_train,y_train)
predrd=rd.predict(x_test)
print(rd.score(x_train,y_train))
print(r2_score(y_test,predrd))
```

```
0.8712042197282965
0.731318913153499
```

```
score=cross_val_score(rd,x2,y2,cv=5)
score.mean()
```

```
0.8154832105560443
```

At random state 60 Ridge Regression has given the best score that is around 87% where the cross validation score for this model is around 82 % , that is also very near to the model score , but r2 score is poor ,73% .

Random Forest

What is Random Forest ?

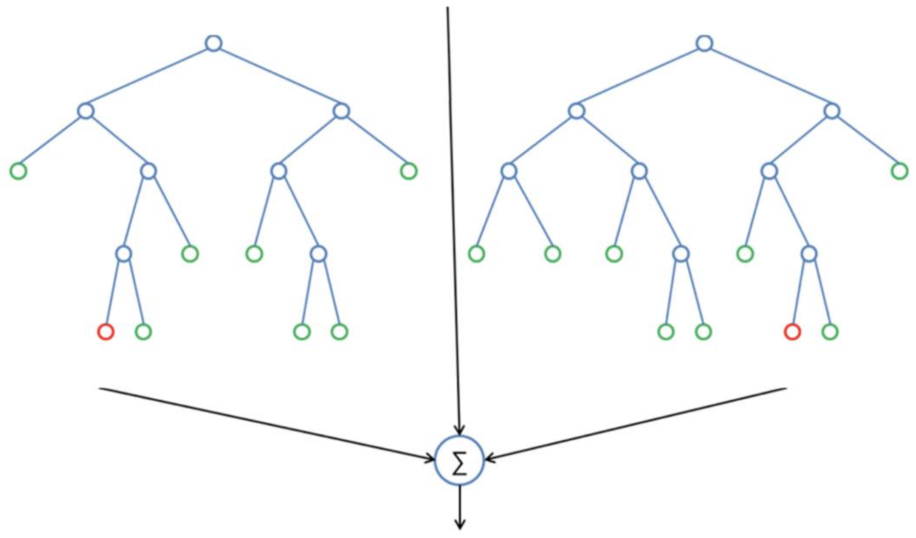
Random Forest is a supervised learning algorithm. Like you can already see from it's name, it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. With a few exceptions a random-forest classifier has all the hyperparameters of a decision-tree classifier and also all the hyperparameters of a bagging classifier, to control the ensemble itself.

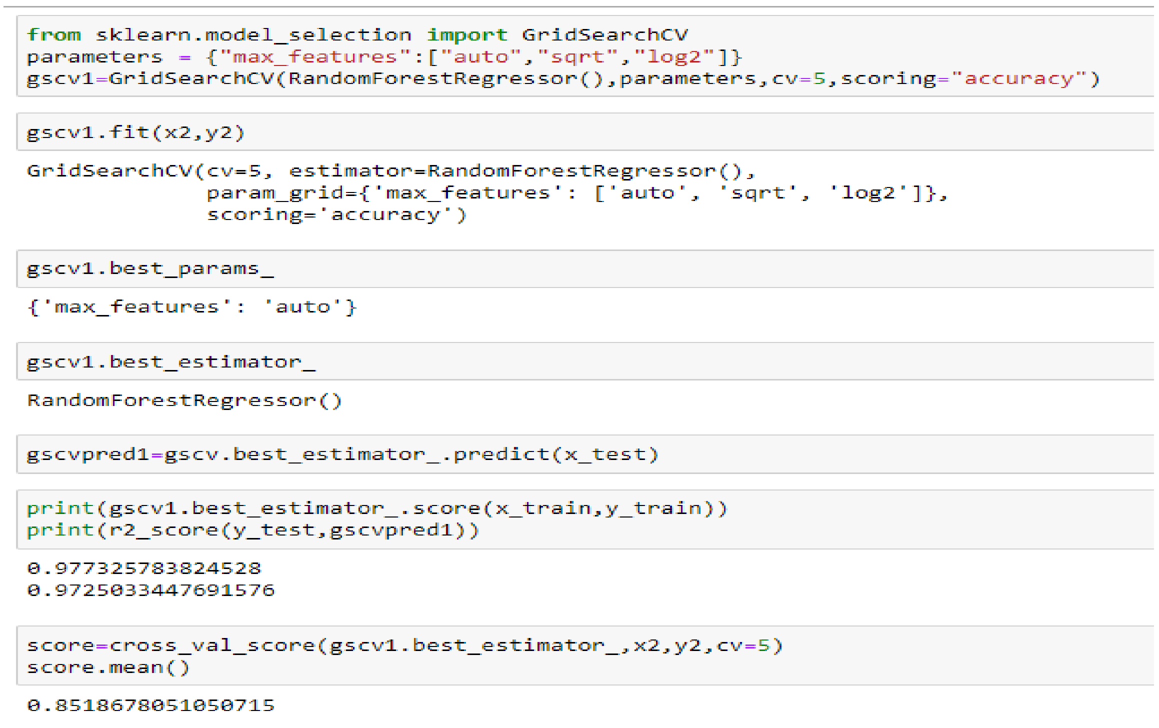
The random-forest algorithm brings extra randomness into the model, when it is growing the trees. Instead of searching for the best feature while splitting a node, it searches for the best feature among a random subset of features. This process creates a wide diversity, which generally results in a better model. Therefore when you are growing a tree in random forest, only a random subset of the features is considered for splitting a node. You can even make trees more random, by using random thresholds on top of it, for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Below you can see how a random forest would look like with two trees:



From various models random forest regressor has given the best result and the cross validation score is comparatively good for this model so we have decided to use this model for hyper tuning and getting the score .

For that we will import gridsearch cv then we will check the best parametes and best estimators for the same.



```
from sklearn.model_selection import GridSearchCV
parameters = {"max_features":["auto","sqrt","log2"]}
gscv1=GridSearchCV(RandomForestRegressor(),parameters,cv=5,scoring="accuracy")
```

```
gscv1.fit(x2,y2)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(),
             param_grid={'max_features': ['auto', 'sqrt', 'log2']},
             scoring='accuracy')
```

```
gscv1.best_params_
```

```
{'max_features': 'auto'}
```

```
gscv1.best_estimator_
```

```
RandomForestRegressor()
```

```
gscvpred1=gscv.best_estimator_.predict(x_test)
```

```
print(gscv1.best_estimator_.score(x_train,y_train))
print(r2_score(y_test,gscvpred1))
```

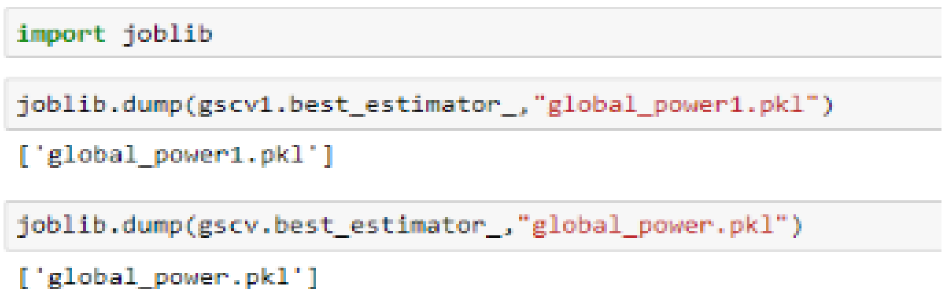
```
0.977325783824528
0.9725033447691576
```

```
score=cross_val_score(gscv1.best_estimator_,x2,y2,cv=5)
score.mean()
```

```
0.8518678051050715
```

Here we can see that our model score is around 98% , r2 score is 97 % and te cross validation score is 85%.

Now we will import joblib for deploying our models.



```
import joblib
```

```
joblib.dump(gscv1.best_estimator_,"global_power1.pkl")
```

```
['global_power1.pkl']
```

```
joblib.dump(gscv.best_estimator_,"global_power.pkl")
```

```
['global_power.pkl']
```