Household Power Consumption

Life cycle of Machine learning Project

- Understanding the Problem Statement
- Data Ingestion
- Data Cleaning
- Exploratory data analysis
- Data Pre-Processing
- Model Training
- Choose best model
- Hypertuning the model

1. Problem statement.

- The dataSet consists of 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months).
- Using this data we need to predict the energy consumption of the houshold.

2. Data Collection.

- The Dataset is collected from Website named, UCI Machine Learning Repository.
- Link for dataset - https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption (https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption)
- This archive contains 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months).

Importing important libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pickle
from pandas_profiling import ProfileReport
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, Lasso , ElasticNet, RidgeCV , LassoC
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc
import scikitplot as skl
sns.set()
```

3. Data Ingestion

Loading Dataset

In [2]:
```
df = pd.read_csv(r'C:\Users\bolt0\Data science\house electricity dataset\hou
```

In [3]:
```
df.head()
```

Out[3]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub
0	16/12/2006	17:24:00	4.216	0.418	234.840	18.400	
1	16/12/2006	17:25:00	5.360	0.436	233.630	23.000	
2	16/12/2006	17:26:00	5.374	0.498	233.290	23.000	
3	16/12/2006	17:27:00	5.388	0.502	233.740	23.000	
4	16/12/2006	17:28:00	3.666	0.528	235.680	15.800	

In [4]:
```
1 df.sample(5)
```

Out[4]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity
2042637	4/11/2010	05:21:00	0.37	0.106	250.02	1.6
1274297	19/5/2009	15:41:00	0.248	0.000	241.860	1.000
2040832	2/11/2010	23:16:00	2.622	0.202	244.91	10.6
597001	4/2/2008	07:25:00	2.970	0.000	238.420	12.400
88178	15/2/2007	23:02:00	1.722	0.148	245.300	7.000

Shape and basic info about the data:

In [5]:
```
1 df.shape
```

Out[5]:
```
(2075259, 9)
```

In [6]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075259 entries, 0 to 2075258
Data columns (total 9 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Date                   object
 1   Time                   object
 2   Global_active_power    object
 3   Global_reactive_power  object
 4   Voltage                object
 5   Global_intensity       object
 6   Sub_metering_1         object
 7   Sub_metering_2         object
 8   Sub_metering_3         float64
dtypes: float64(1), object(8)
memory usage: 142.5+ MB
```

Features information:

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

Out[7]:
```
Index(['Date', 'Time', 'Global_active_power', 'Global_reactive_power',
       'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3'],
      dtype='object')
```

There are 9 columns or features in the dataset:

- **Date**: Date in format dd/mm/yyyy
- **Time**: time in format hh:mm:ss
- **Global_active_power**: household global minute-averaged active power (in kilowatt)
- **Global_reactive_power**: household global minute-averaged reactive power (in kilowatt)
- **Voltage**: minute-averaged voltage (in volt)
- **Global_intensity**: household global minute-averaged current intensity (in ampere)
- **Sub_metering_1**: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered).
- **Sub_metering_2**: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
- **Sub_metering_3**: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

Since the dataset is too large so we will select a sample of the dataset having 100000 values and use it for all the study

In [8]:
```
1 df_sample = df.sample(100000)
```

In [9]:
```
1 df_sample
```

Out[9]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity
1416665	26/8/2009	12:29:00	0.242	0.140	242.720	1.200
487078	19/11/2007	23:22:00	0.224	0.000	245.690	1.000
254186	11/6/2007	05:50:00	0.290	0.214	243.100	1.400
813061	3/7/2008	08:25:00	1.288	0.116	238.690	5.400
1778354	4/5/2010	16:38:00	0.244	0.000	237.930	1.000
...	...	...	...	...	...	..
1987921	27/9/2010	05:25:00	?	?	?	?
1764380	24/4/2010	23:44:00	1.692	0.344	241.400	7.000
1303497	8/6/2009	22:21:00	2.954	0.142	239.130	12.200
1160059	1/3/2009	07:43:00	0.262	0.054	243.480	1.200
704825	19/4/2008	04:29:00	0.274	0.000	244.960	1.200

100000 rows × 9 columns

Resetting the index of the sample dataframe

In [10]:
```
1 df_sample.reset_index(inplace = True , drop = True )
```

In [11]:
```
df_sample
```

Out[11]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity
0	26/8/2009	12:29:00	0.242	0.140	242.720	1.200
1	19/11/2007	23:22:00	0.224	0.000	245.690	1.000
2	11/6/2007	05:50:00	0.290	0.214	243.100	1.400
3	3/7/2008	08:25:00	1.288	0.116	238.690	5.400
4	4/5/2010	16:38:00	0.244	0.000	237.930	1.000
...	...	...	...	...	...	...
99995	27/9/2010	05:25:00	?	?	?	?
99996	24/4/2010	23:44:00	1.692	0.344	241.400	7.000
99997	8/6/2009	22:21:00	2.954	0.142	239.130	12.200
99998	1/3/2009	07:43:00	0.262	0.054	243.480	1.200
99999	19/4/2008	04:29:00	0.274	0.000	244.960	1.200

100000 rows × 9 columns

4. Data Cleaning

Data include 'nan' , ' ' , ',' and '?' as a string. We need to convert both to numpy nan.

In [12]:
```
df_sample.replace('?', np.nan ,inplace = True)
df_sample.replace(',', np.nan ,inplace = True)
df_sample.replace(' ', np.nan ,inplace = True)
df_sample.replace('nan', np.nan ,inplace = True)
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075259 entries, 0 to 2075258
Data columns (total 9 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   Date                   object 
 1   Time                   object 
 2   Global_active_power    object 
 3   Global_reactive_power  object 
 4   Voltage                object 
 5   Global_intensity       object 
 6   Sub_metering_1         object 
 7   Sub_metering_2         object 
 8   Sub_metering_3         float64
dtypes: float64(1), object(8)
memory usage: 142.5+ MB
```

Also we now create seperate columns for day,month,year,hour and minute from the Date and Time Columns

In [14]:
```
df_sample['Date'] = pd.to_datetime(df_sample['Date'])

df_sample['Day'] = df_sample['Date'].dt.day
df_sample['Month'] = df_sample['Date'].dt.month
df_sample['Year'] = df_sample['Date'].dt.year
```

In [15]:
```
df_sample['Hour'] = pd.to_datetime(df_sample['Time'], format='%H:%M:%S').dt.
df_sample['Minutes'] = pd.to_datetime(df_sample['Time'], format='%H:%M:%S').
```

Converting Datatype to float

In [16]:
```
for item in df_sample.columns:
    if df_sample[str(item)].dtype == 'O' and item != 'Time':
        df_sample[str(item)] = df_sample[str(item)].astype(float)
```

In [17]:
```
df_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Date                   100000 non-null  datetime64[ns]
 1   Time                   100000 non-null  object
 2   Global_active_power    98731 non-null   float64
 3   Global_reactive_power  98731 non-null   float64
 4   Voltage                98731 non-null   float64
 5   Global_intensity       98731 non-null   float64
 6   Sub_metering_1         98731 non-null   float64
 7   Sub_metering_2         98731 non-null   float64
 8   Sub_metering_3         98731 non-null   float64
 9   Day                    100000 non-null  int64
 10  Month                  100000 non-null  int64
 11  Year                   100000 non-null  int64
 12  Hour                   100000 non-null  int64
 13  Minutes                100000 non-null  int64
dtypes: datetime64[ns](1), float64(7), int64(5), object(1)
memory usage: 10.7+ MB
```

Null Values

In [18]:
```
df_sample.isnull().sum()
```

Out[18]:
```
Date                       0
Time                       0
Global_active_power     1269
Global_reactive_power   1269
Voltage                 1269
Global_intensity        1269
Sub_metering_1          1269
Sub_metering_2          1269
Sub_metering_3          1269
Day                        0
Month                      0
Year                       0
Hour                       0
Minutes                    0
dtype: int64
```

There are 1226 null values in the dataset , Since the dataset has mostly float values so we will replace the null values by the median of the dataset.

In [19]:
```
for item in df_sample.columns:
    if df_sample[str(item)].isnull().sum() > 0:
        df_sample[str(item)] = df_sample[str(item)].fillna(df_sample[str(ite
```

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

Out[20]:
```
Date                    0
Time                    0
Global_active_power     0
Global_reactive_power   0
Voltage                 0
Global_intensity        0
Sub_metering_1          0
Sub_metering_2          0
Sub_metering_3          0
Day                     0
Month                   0
Year                    0
Hour                    0
Minutes                 0
dtype: int64
```

We have succesfully filled all the null values. Now we check for any duplicated rows.

Duplicated entries

In [21]:
```
df_sample.duplicated().sum()
```

Out[21]:
```
0
```

There are **0** duplicate entries in the dataframe.

Creating Column For Total Metering

In [22]:
```
df_sample['Total_metering'] = df_sample['Sub_metering_1']+df_sample['Sub_met
```

In [23]:
```
df_sample.head()
```

Out[23]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_mete
0	2009-08-26	12:29:00	0.242	0.140	242.72	1.2	
1	2007-11-19	23:22:00	0.224	0.000	245.69	1.0	
2	2007-11-06	05:50:00	0.290	0.214	243.10	1.4	
3	2008-03-07	08:25:00	1.288	0.116	238.69	5.4	
4	2010-04-05	16:38:00	0.244	0.000	237.93	1.0	

In [24]:
```
df_sample.columns
```

Out[24]:
```
Index(['Date', 'Time', 'Global_active_power', 'Global_reactive_power',
       'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3', 'Day', 'Month', 'Year', 'Hour', 'Minutes',
       'Total_metering'],
      dtype='object')
```

Dropping non relevant columns:

In [25]:
```
df_sample.drop(columns = ['Date' ,'Time','Sub_metering_1', 'Sub_metering_2',
```

In [26]:
```
df_sample.columns
```

Out[26]:
```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'Day', 'Month', 'Year', 'Hour', 'Minutes',
       'Total_metering'],
      dtype='object')
```

5. EDA

Description of the data

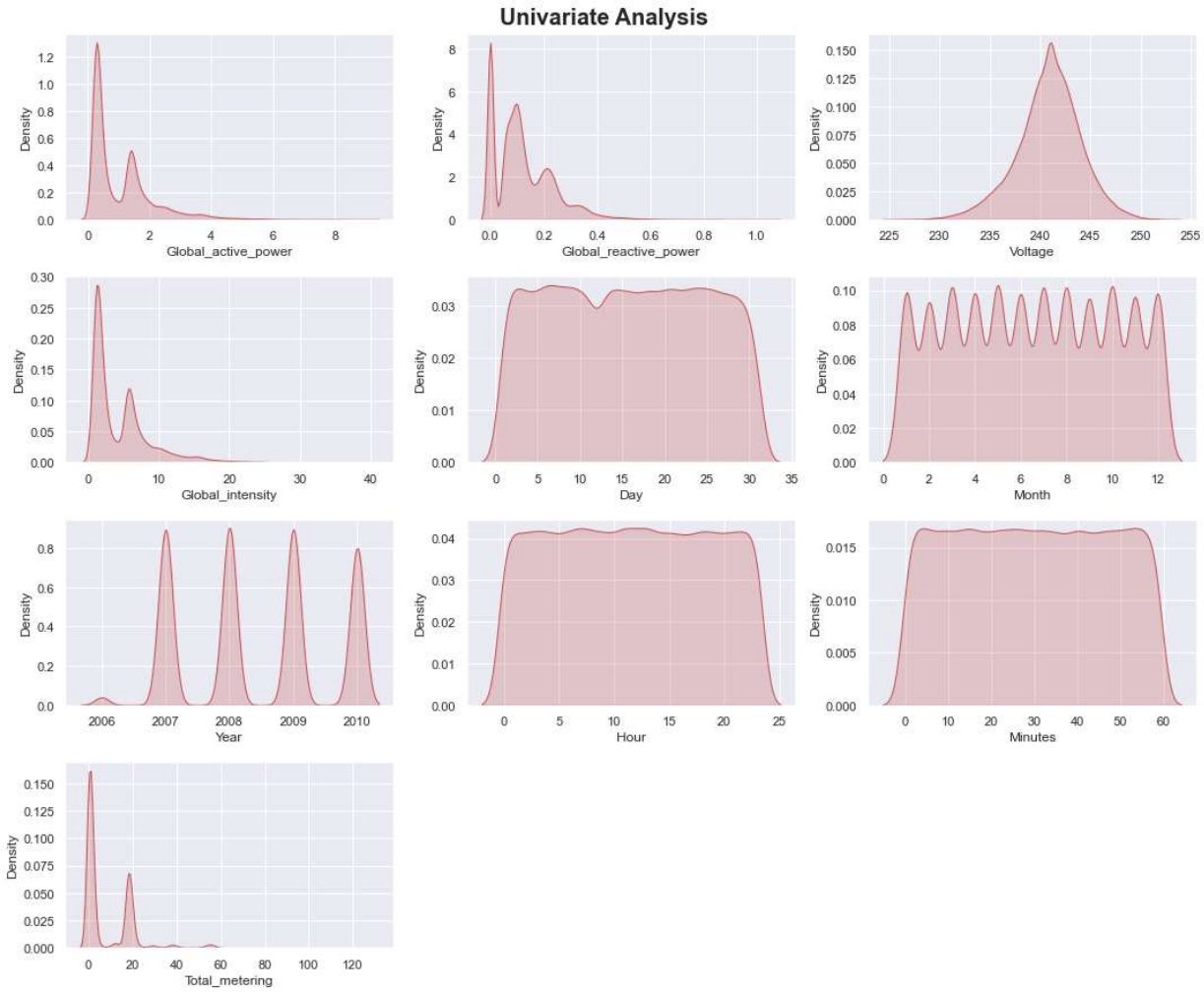
In [27]:
```
1 df_sample.describe().T
```

Out[27]:

	count	mean	std	min	25%	50%	75%
Global_active_power	100000.0	1.083286	1.048443	0.076	0.310	0.61	1.518
Global_reactive_power	100000.0	0.122654	0.111197	0.000	0.048	0.10	0.190
Voltage	100000.0	240.842737	3.216013	225.320	239.040	241.00	242.850
Global_intensity	100000.0	4.592018	4.408322	0.200	1.400	2.60	6.400
Day	100000.0	15.728650	8.832073	1.000	8.000	16.00	23.000
Month	100000.0	6.503830	3.439919	1.000	4.000	7.00	10.000
Year	100000.0	2008.430940	1.127503	2006.000	2007.000	2008.00	2009.000
Hour	100000.0	11.492410	6.909566	0.000	6.000	11.00	17.000
Minutes	100000.0	29.438280	17.373622	0.000	14.000	29.00	45.000
Total_metering	100000.0	8.766390	12.760959	0.000	0.000	1.00	18.000

In [28]:

```
plt.figure(figsize=(15,15))
plt.suptitle("Univariate Analysis", fontsize=20, fontweight= 'bold')
for i in range(0, len(df_sample.columns)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(x=df_sample[df_sample.columns[i]],shade=True, color='r')
    plt.xlabel(df_sample.columns[i])
    plt.tight_layout()

```



observation:

Global_active_power Power is distributed betweem 0 to 8. Most of the power distributed between 0 to 2. Distribution is not normal

Global reactive power Reactive power is distributed between 0 to 0.8. most of the power distributed between 0 to 0.2.

Voltage Voltage is distributed between 230 to 250. most of the voltage distributed between 230 to 250. Voltage is distributed normally

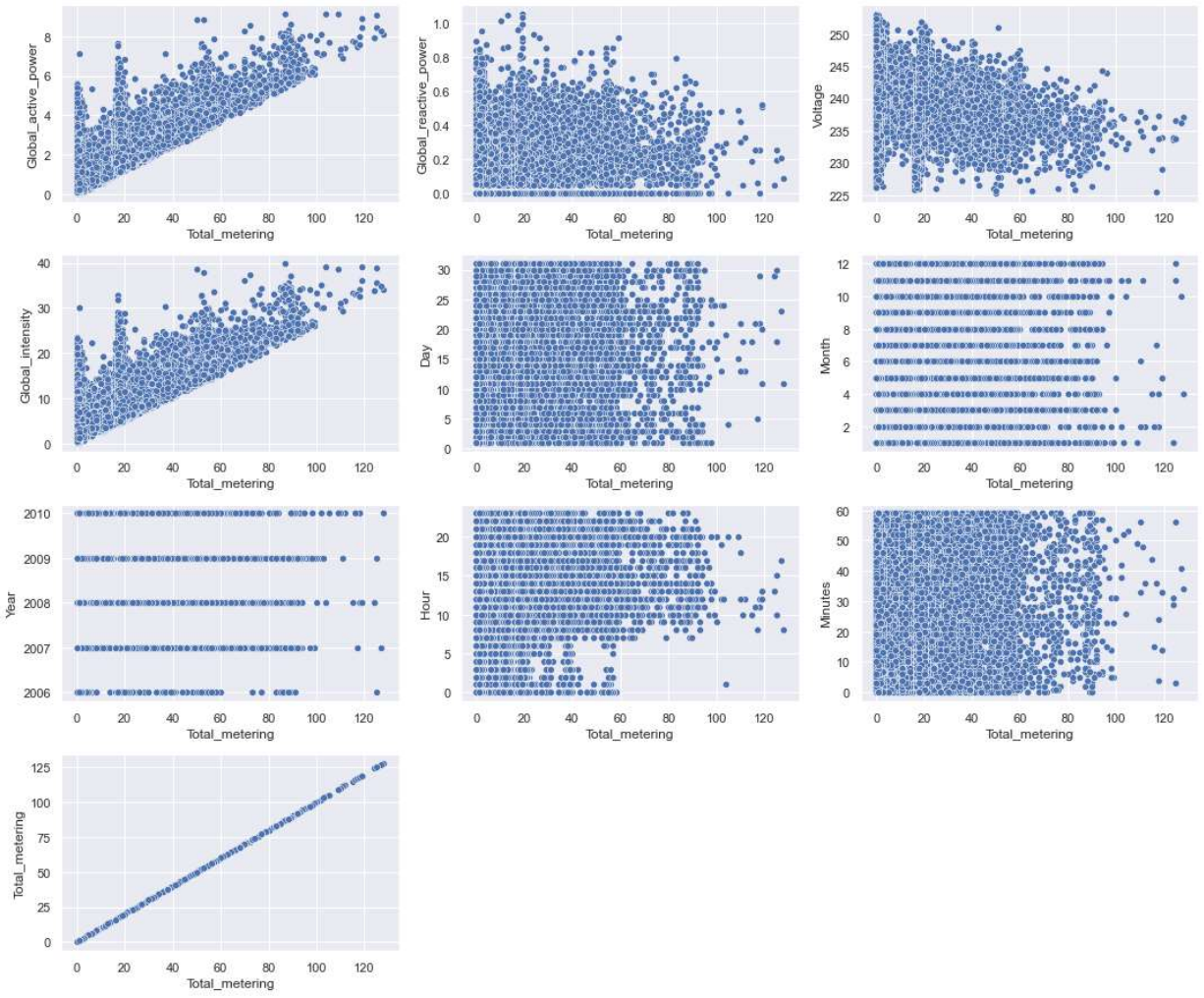
Global intensity Intensity is distributed between 0 to 20. Most of the intensity distributrd between 0 to 10.

Total metering Metering is distributed between 0 to 60 most of the distribution is between 0 to 25.

Relation of Total Metering with other features

In [29]:

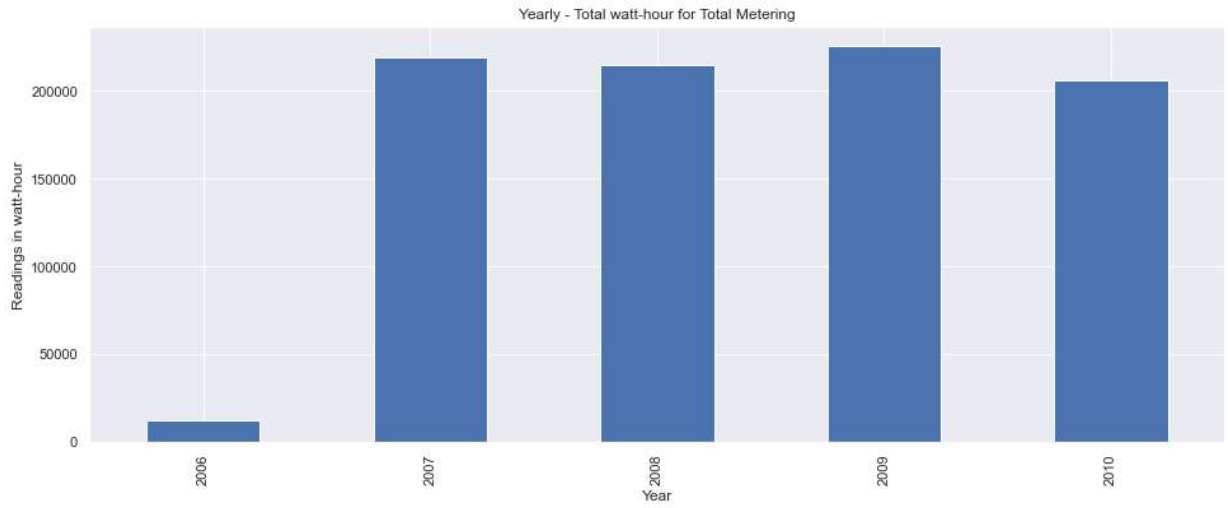
```
plt.figure(figsize=(15,15))
for i in range(0, len(df_sample.columns)):
    plt.subplot(5,3,i+1)
    sns.scatterplot(x=df_sample['Total_metering'], y=df_sample[df_sample.col
    plt.ylabel(df_sample.columns[i])
    plt.xlabel('Total_metering')
    plt.tight_layout()
```



Year wise total metering

In [30]:
```
1 df_sample.groupby(df_sample.Year)['Total_metering'].sum().plot(kind="bar",xl
```

Out[30]:
```
<AxesSubplot:title={'center':'Yearly - Total watt-hour for Total Metering'}, xl
abel='Year', ylabel='Readings in watt-hour'>
```



observation:

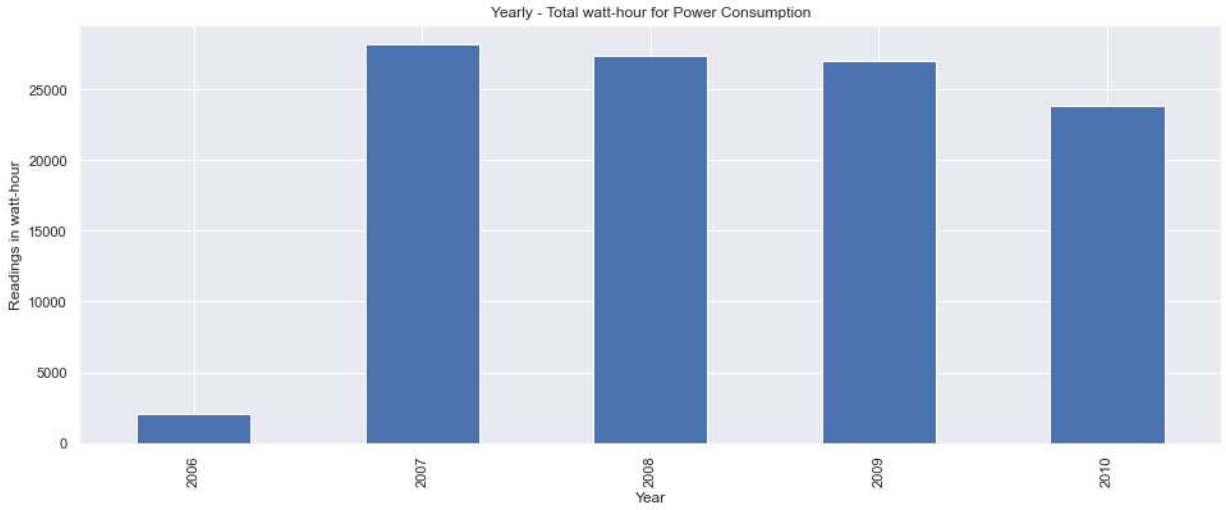
The total metering was max in the year 2009.

The yearly total metering was minimum for year 2006.

Year wise total power consumption

In [31]:
```
1 df_sample.groupby(df_sample.Year)['Global_active_power'].sum().plot(kind="ba
```

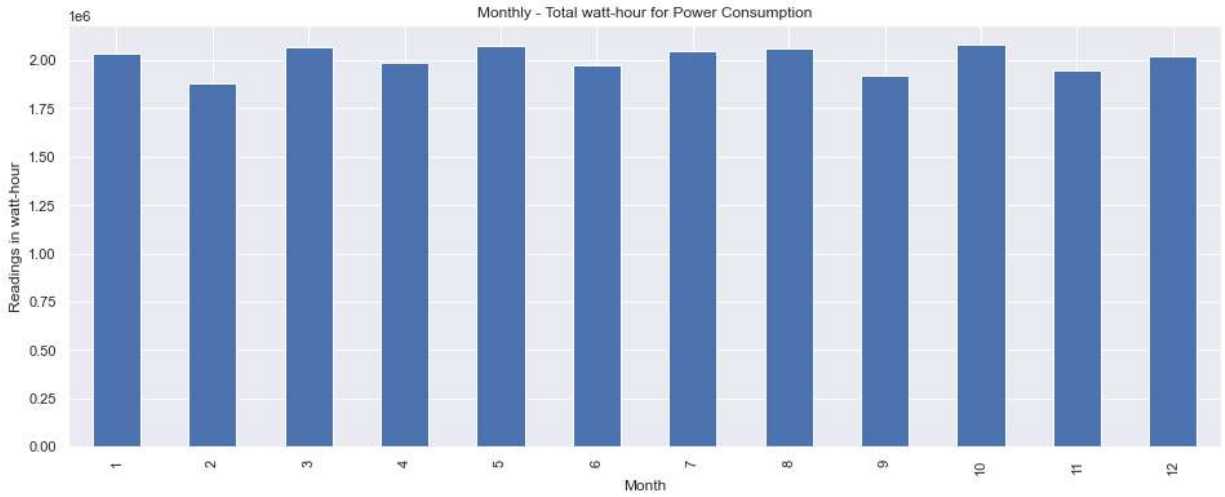
Out[31]:
```
<AxesSubplot:title={'center':'Yearly - Total watt-hour for Power Consumption'},
xlabel='Year', ylabel='Readings in watt-hour'>
```



Monthly - Total Voltage

In [32]:
```
1 df_sample.groupby(df_sample.Month)['Voltage'].sum().plot(kind="bar",xlabel='
```

Out[32]:
```
<AxesSubplot:title={'center':'Monthly - Total watt-hour for Power Consumptio
n'}, xlabel='Month', ylabel='Readings in watt-hour'>
```

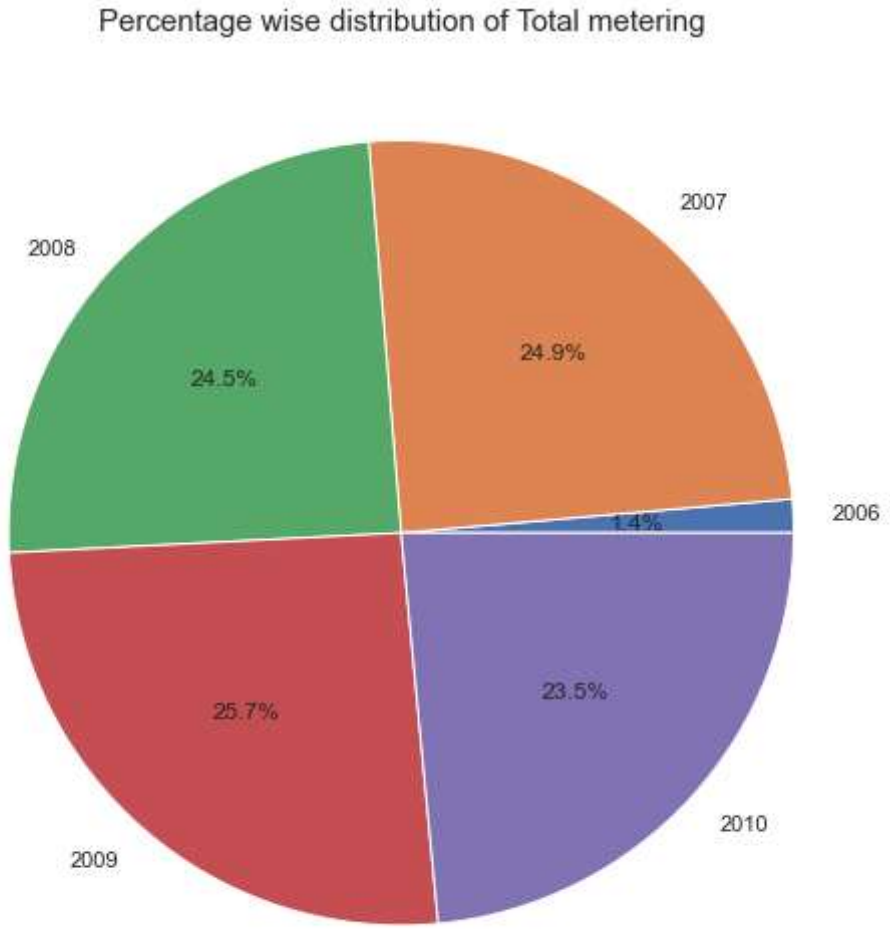


observation:

The monthly distribution of voltage is almost same for all the months.

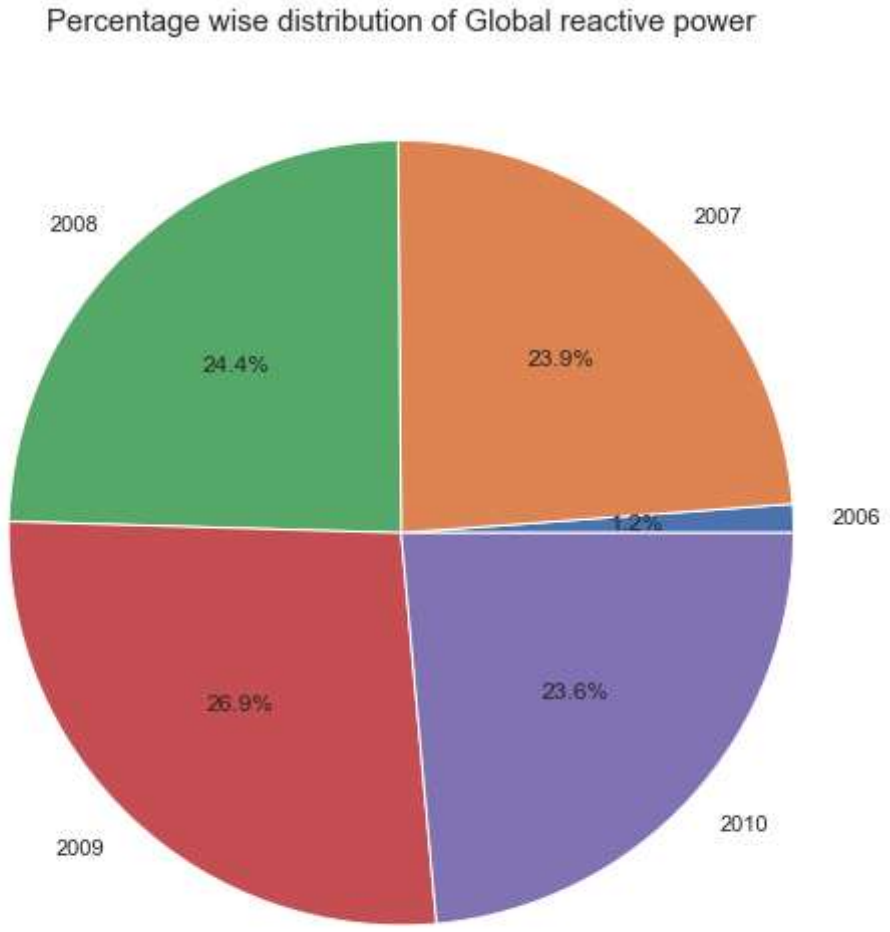
In [33]:

```
1 plt.figure(figsize=(9,9))
2 year_labels = [2006,2007,2008,2009,2010]
3 plt.pie(x=df_sample.groupby(df_sample['Year'])['Total_metering'].sum(),autop
4 plt.title("Percentage wise distribution of Total metering",fontsize=15)
5 plt.show()
```



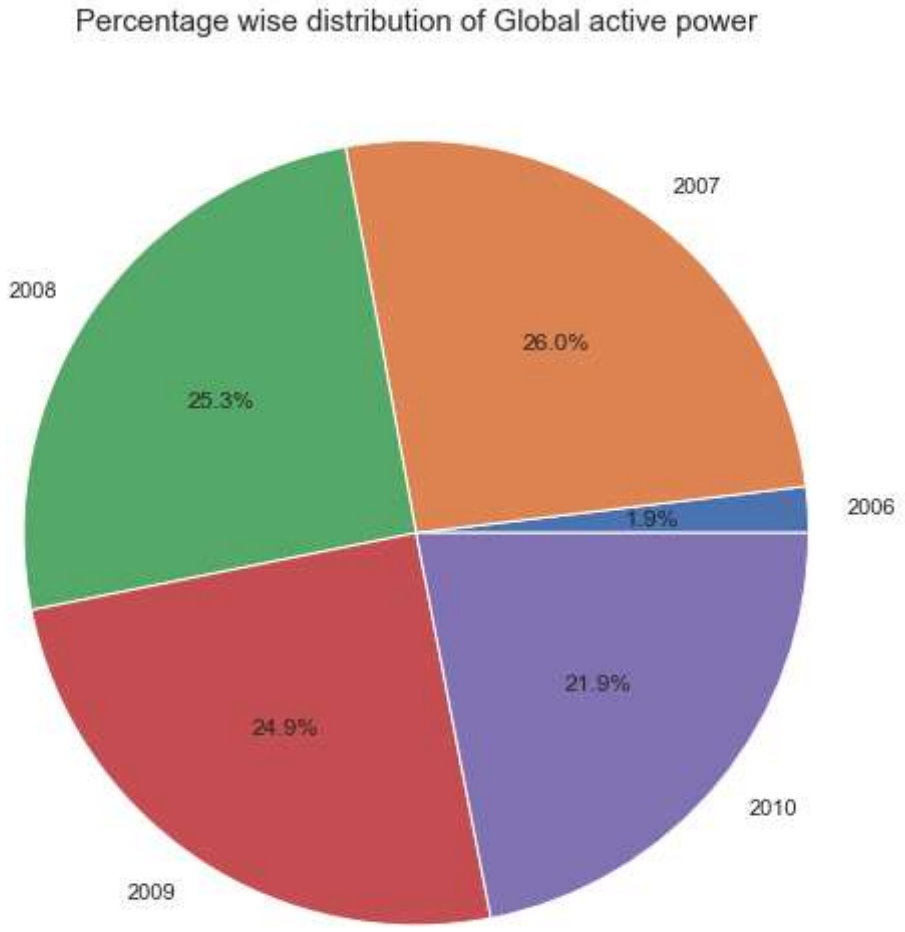
In [34]:

```
plt.figure(figsize=(9,9))
year_labels = [2006,2007,2008,2009,2010]
plt.pie(x=df_sample.groupby(df_sample['Year'])['Global_reactive_power'].sum(
plt.title("Percentage wise distribution of Global reactive power",fontsize=1
plt.show()
```



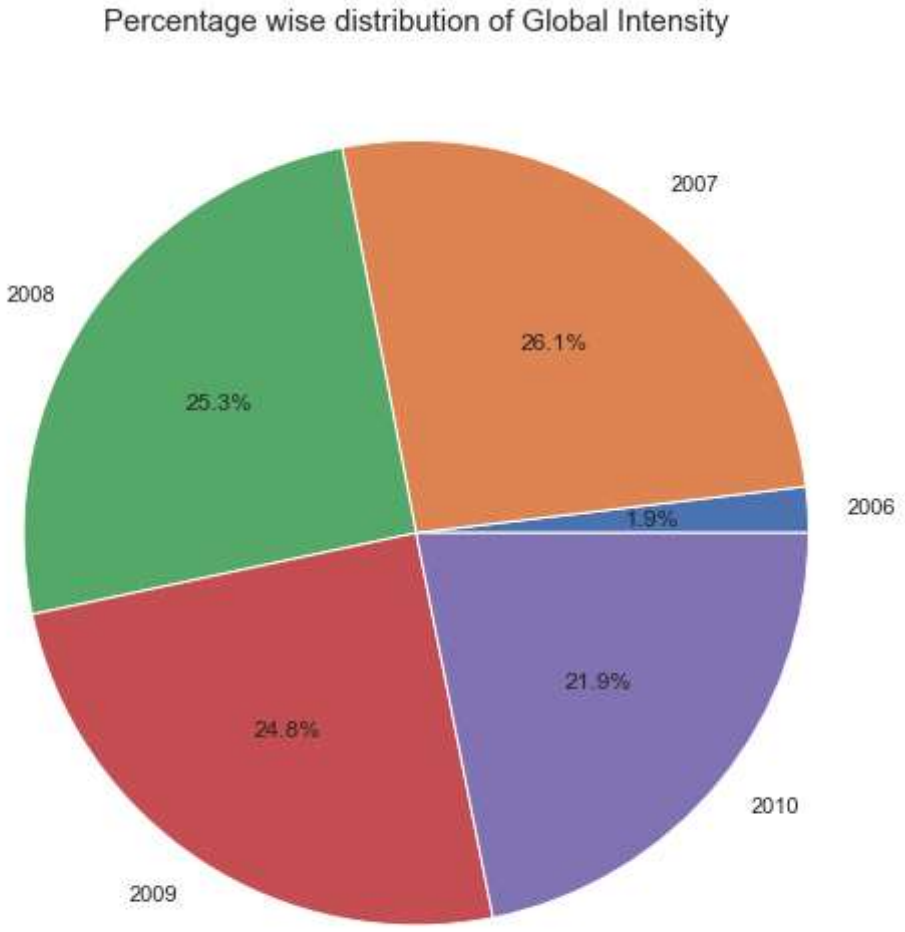
In [35]:

```
plt.figure(figsize=(9,9))
year_labels = [2006,2007,2008,2009,2010]
plt.pie(x=df_sample.groupby(df_sample['Year'])['Global_active_power'].sum(),
plt.title("Percentage wise distribution of Global active power",fontsize=15)
plt.show()
```



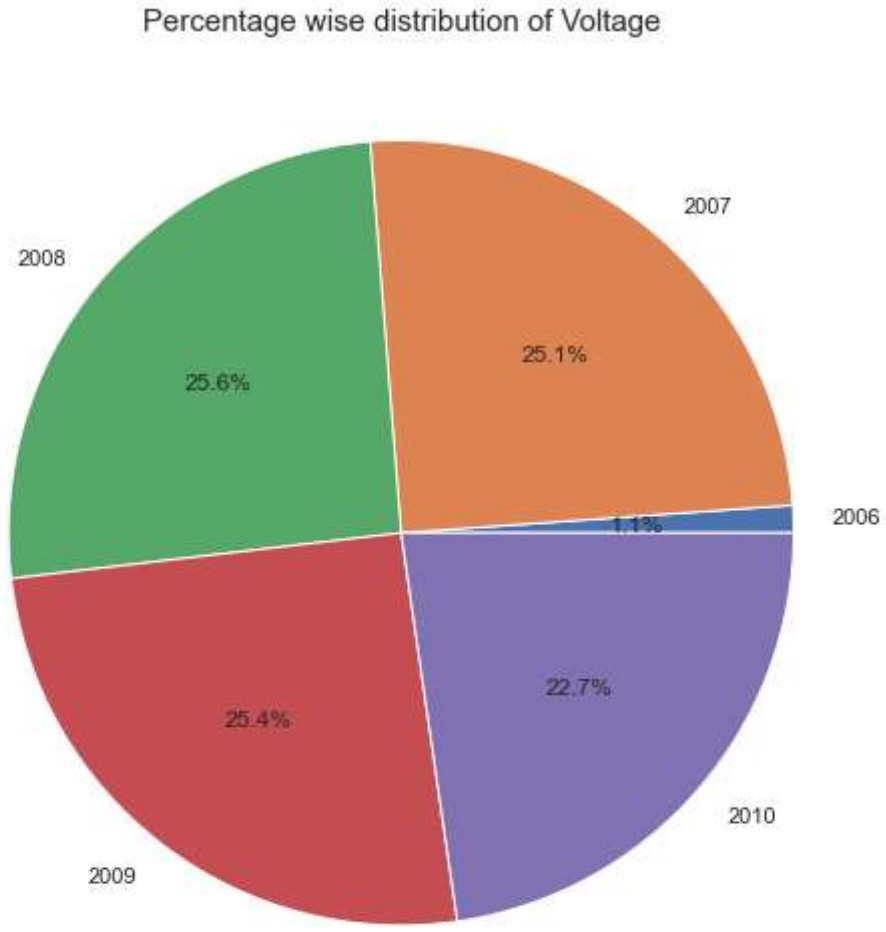
In [36]:

```
1 plt.figure(figsize=(9,9))
2 year_labels = [2006,2007,2008,2009,2010]
3 plt.pie(x=df_sample.groupby(df_sample['Year'])['Global_intensity'].sum(),aut
4 plt.title("Percentage wise distribution of Global Intensity",fontsize=15)
5 plt.show()
```



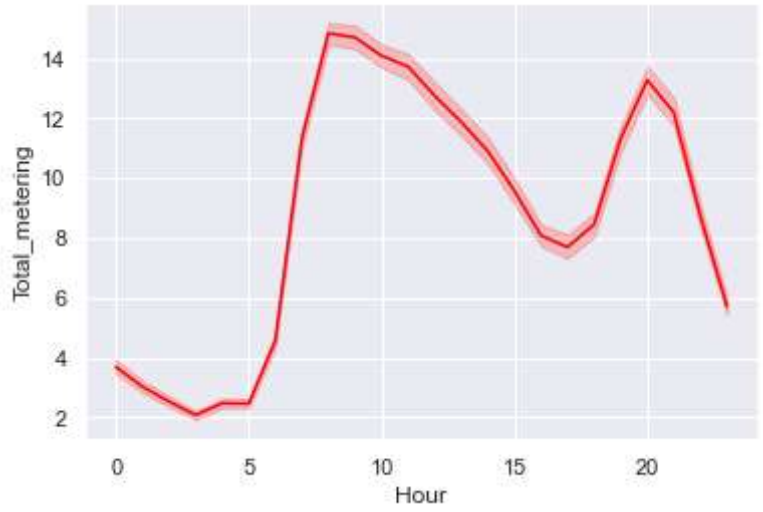
In [37]:

```
plt.figure(figsize=(9,9))
year_labels = [2006,2007,2008,2009,2010]
plt.pie(x=df_sample.groupby(df_sample['Year'])['Voltage'].sum(),autopct="%1.
plt.title("Percentage wise distribution of Voltage",fontsize=15)
plt.show()
```



In [38]:
```
1 sns.lineplot(x='Hour', y='Total_metering', data=df_sample, color = 'red')
```

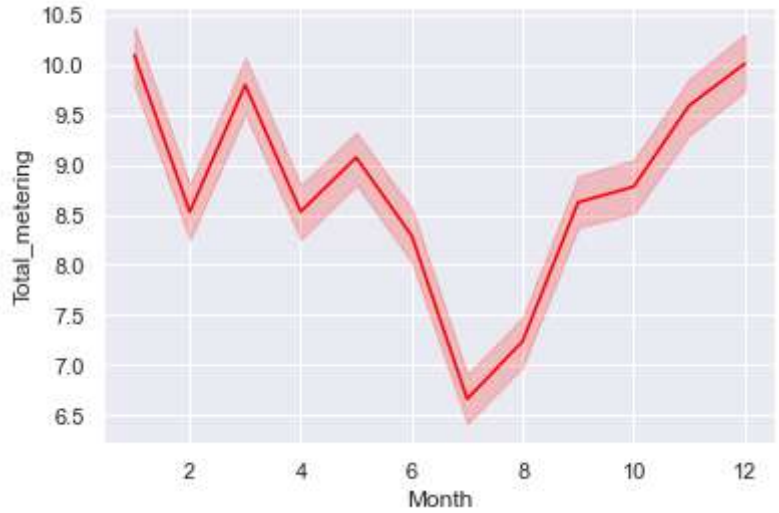
Out[38]: <AxesSubplot:xlabel='Hour', ylabel='Total_metering'>



observation:
Total metering id maximum between 5 to 10 hour

In [39]:
```
1 sns.lineplot(x='Month', y='Total_metering', data=df_sample, color = 'red')
```

Out[39]: <AxesSubplot:xlabel='Month', ylabel='Total_metering'>

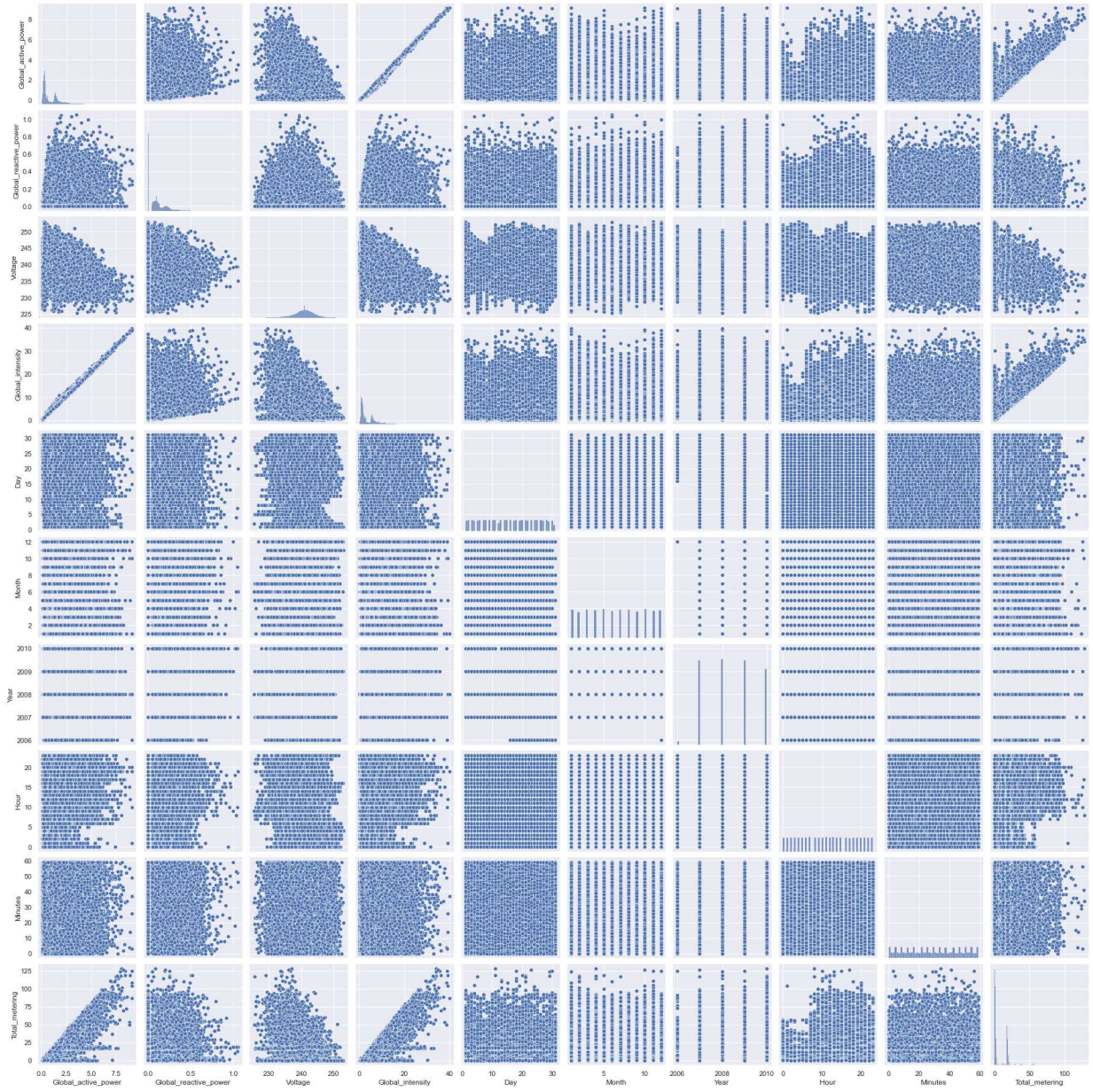


observation:
The max power consumption is in January.
In july there is least power consumption

Pairplot

In [40]:
```
1 sns.pairplot(df_sample)
```

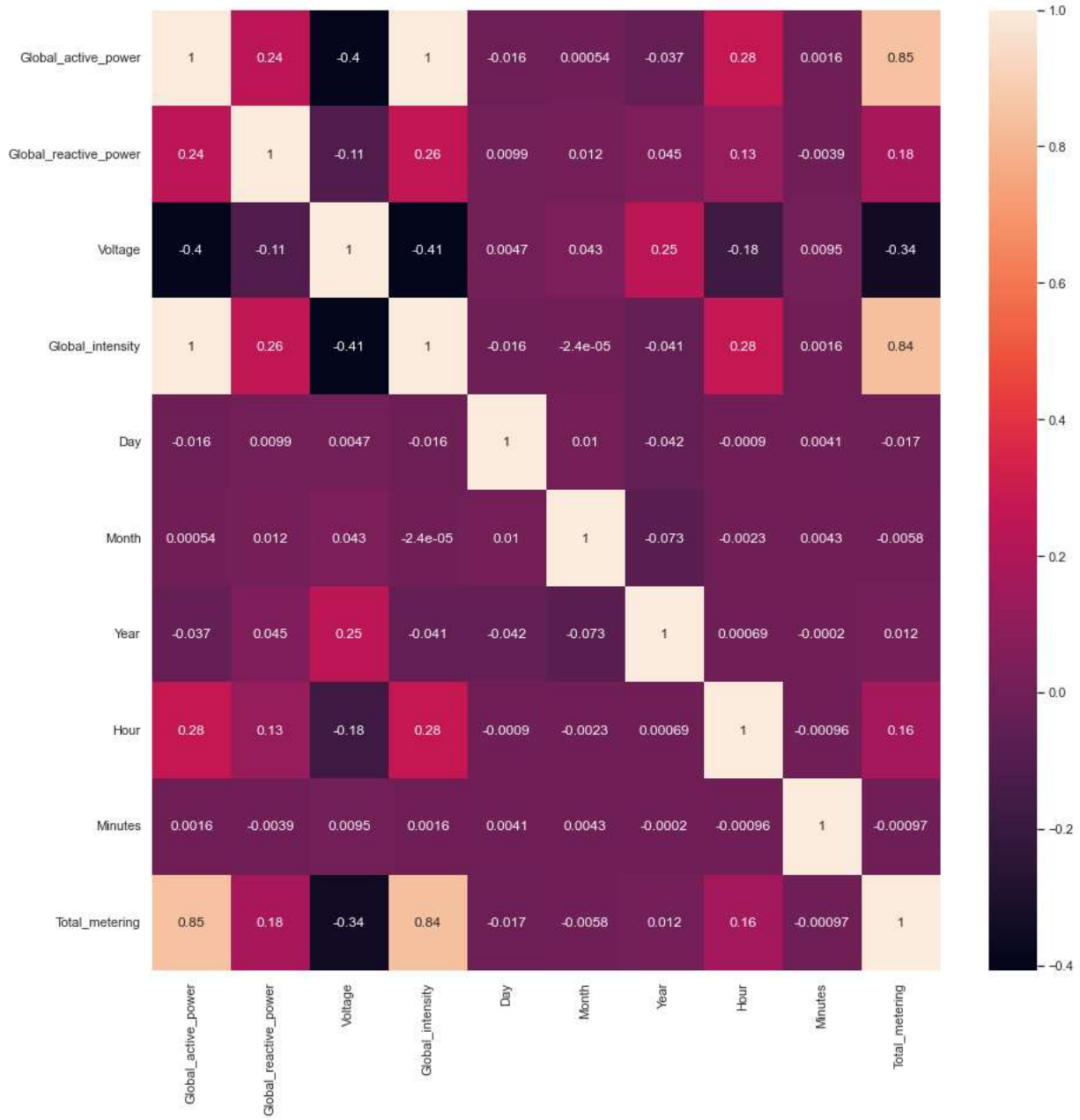
Out[40]: <seaborn.axisgrid.PairGrid at 0x164d5c58cd0>



Checking correlation

In [41]:

```
plt.figure(figsize=(15,15))
sns.heatmap(data=df_sample.corr(), annot=True);
```



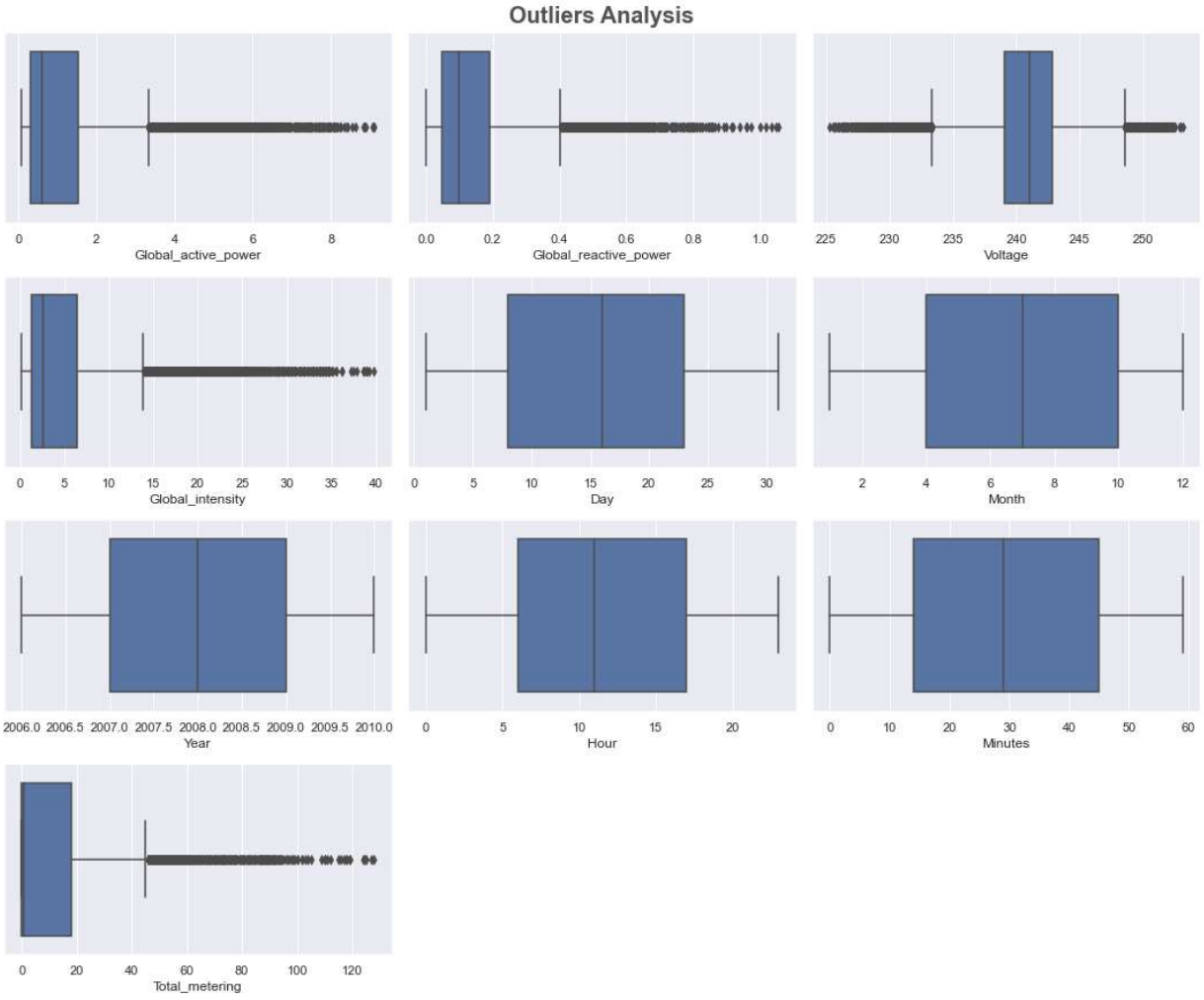
observation:

Global active power and Global intensity are highly correlated

Checking for outliers

In [42]:

```
plt.figure(figsize=(15,15))
plt.suptitle("Outliers Analysis",fontsize = 20, fontweight = 'bold', alpha=
for i in range(0, len(df_sample.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(df_sample[df_sample.columns[i]])
    plt.tight_layout()
```



Handling Outliers

In [ ]:

```
'''
from feature_engine.outliers.winsorizer import Winsorizer
for item in df_sample.columns:
    winsorizer = Winsorizer(capping_method = 'iqr',tail = 'both', fold = 1.5
    # capping_methods = 'iqr' - 25th quantile & 75th quantile
    df_sample[str(item)] = winsorizer.fit_transform(df_sample[[str(item)]])
    '''
```

In [ ]:
```
'''
plt.figure(figsize=(15,15))
plt.suptitle("Boxplot after handling Outliers",fontsize = 20, fontweight = '
for i in range(0, len(df_sample.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(df_sample[df_sample.columns[i]])
    plt.tight_layout()
'''
```

Saving this Cleaned data

In [43]:
```
df_sample.to_csv("cleaned_power_consumption_data.csv")
```

Uploading data to mongodb server

In [44]:
```
import pymongo
```

In [45]:
```
client = pymongo.MongoClient('mongodb+srv://ankitmongo:mongo123@cluster0.ict
```

In [53]:
```
database = client['Household_power_consumption']
collection = database['Power_consumption_collection']
data_dict = df_sample.to_dict("Records")
collection.insert_many(data_dict)
```

Out[53]:
```
<pymongo.results.InsertManyResult at 0x164d5783730>
```

Loading the data from MongoDb server

In [54]:
```
db = client.Household_power_consumption
collection = db.Power_consumption_collection
df_new = pd.DataFrame(list(collection.find()))
```

In [55]:
```
df_new.head()
```

Out[55]:

	_id	Global_active_power	Global_reactive_power	Voltage	Global_intensity
0	638628aeffd7a90490ec8da9	0.242	0.140	242.72	1.2
1	638628aeffd7a90490ec8daa	0.224	0.000	245.69	1.0
2	638628aeffd7a90490ec8dab	0.290	0.214	243.10	1.4
3	638628aeffd7a90490ec8dac	1.288	0.116	238.69	5.4
4	638628aeffd7a90490ec8dad	0.244	0.000	237.93	1.0

In [56]:
```
df_new.shape
```

Out[56]: (100000, 11)

In [57]:
```
## dropping id column
df_new.drop(columns=['_id'],axis=1,inplace=True)
```

In [58]:
```
df_new.head()
```

Out[58]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Day	Month	Year	Hour
0	0.242	0.140	242.72	1.2	26	8	2009	12
1	0.224	0.000	245.69	1.0	19	11	2007	23
2	0.290	0.214	243.10	1.4	6	11	2007	5
3	1.288	0.116	238.69	5.4	7	3	2008	8
4	0.244	0.000	237.93	1.0	5	4	2010	16

6. Preprocessing

Input and Output variables

In [59]:
```
X = df_new.drop(['Total_metering','Day','Month','Year','Hour','Minutes'], ax
y = df_new.Total_metering          #output variable
```

In [60]:
```
X.head()
```

Out[60]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity
0	0.242	0.140	242.72	1.2
1	0.224	0.000	245.69	1.0
2	0.290	0.214	243.10	1.4
3	1.288	0.116	238.69	5.4
4	0.244	0.000	237.93	1.0

In [61]:
```
1 y
```

Out[61]:
```
0         1.0
1         0.0
2         0.0
3        18.0
4         1.0
         ...
99995     1.0
99996     1.0
99997    38.0
99998     0.0
99999     1.0
Name: Total_metering, Length: 100000, dtype: float64
```

Splitting train and test data

In [62]:
```
1 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33,rando
2 X_train.shape, X_test.shape
```

Out[62]:
```
((67000, 4), (33000, 4))
```

In [63]:
```
1 X_train.head()
```

Out[63]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity
53461	0.240	0.000	242.92	1.0
82804	0.384	0.224	240.88	1.8
95065	2.250	0.060	238.67	9.8
81769	1.450	0.068	238.63	6.0
7312	0.450	0.370	240.57	2.4

In [64]:
```
1 X_test.head()
```

Out[64]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity
23257	0.490	0.312	240.96	2.6
49344	6.624	0.218	235.78	28.6
18582	0.322	0.098	246.09	1.4
2870	0.970	0.130	240.16	4.0
80932	0.340	0.246	242.37	1.6

In [65]:
```
1 y_train
```

Out[65]:
```
53461     0.0
82804     0.0
95065     0.0
81769    18.0
7312      2.0
         ...
7329     29.0
52136     1.0
91038     1.0
98914     0.0
94853     1.0
Name: Total_metering, Length: 67000, dtype: float64
```

In [66]:
```
1 y_test
```

Out[66]:
```
23257     1.0
49344    56.0
18582     0.0
2870      0.0
80932     2.0
         ...
10983    19.0
21071    63.0
4034      0.0
65819    38.0
72346    18.0
Name: Total_metering, Length: 33000, dtype: float64
```

Feaure Scaling

Standardization

In [67]:
```
1 def scaler_standard(X_train, X_test):
2     scaler = StandardScaler()
3     X_train_scaled = scaler.fit_transform(X_train)
4     X_test_scaled = scaler.transform(X_test)
5 
6     return X_train_scaled, X_test_scaled
```

In [68]:
```
1 X_train_scaled, X_test_scaled = scaler_standard(X_train, X_test)
```

In [69]:
```
1 X_train_scaled
```

Out[69]:
```
array([[-0.80338542, -1.09946841,  0.63956178, -0.81399359],
       [-0.66555994,  0.91060949,  0.00740938, -0.63187798],
       [ 1.1204286 , -0.56105469, -0.67742238,  1.18927815],
       ...,
       [-0.62536084,  0.39014289,  0.28939893, -0.63187798],
       [-0.71533025,  0.03120041,  2.0123241 , -0.72293579],
       [-0.64258902, -0.11237659,  1.13536758, -0.63187798]])
```

In [70]:
```
1 X_test_scaled
```

Out[70]:
```
array([[-5.64105070e-01,  1.70028295e+00,  3.21996731e-02,
        -4.49762367e-01],
       [ 5.30687762e+00,  8.56768115e-01, -1.57297161e+00,
         5.46899505e+00],
       [-7.24901466e-01, -2.20059332e-01,  1.62187703e+00,
        -7.22935786e-01],
       ...,
       [-7.19158737e-01, -4.69384263e-03,  3.85461308e-01,
        -7.22935786e-01],
       [ 3.28735146e+00,  6.23455501e-01, -9.25325284e-01,
         3.28360770e+00],
       [ 2.05420538e-01, -6.86684559e-01, -5.34878214e-01,
         1.87642278e-01]])
```

Variance Inflation Factor

In [71]:
```
1 vif = pd.DataFrame()
2 vif["vif"] = [variance_inflation_factor(X_train_scaled,i) for i in range(X_t
3 vif["Features"] = X_train.columns
4 #let's check the values
5 vif
```

Out[71]:

	vif	Features
0	577.192346	Global_active_power
1	1.314498	Global_reactive_power
2	1.296658	Voltage
3	589.539461	Global_intensity

7. Model

Linear Regression

In [72]:

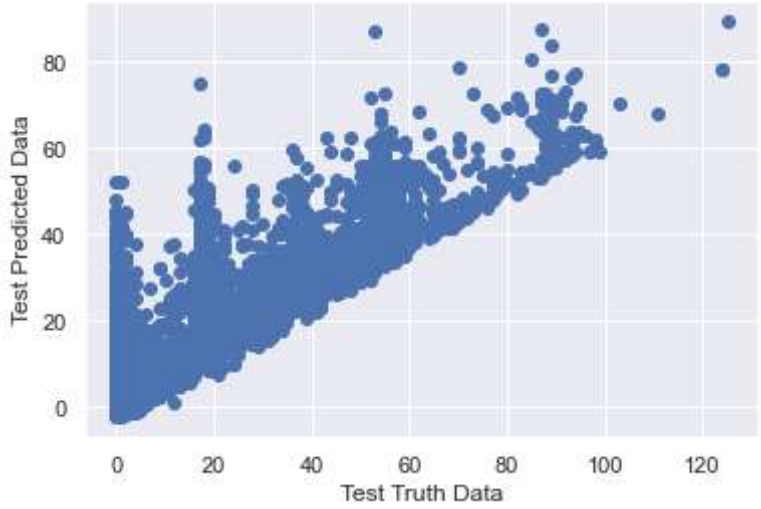
```
from sklearn.metrics import mean_absolute_error, r2_score , mean_squared_err

lreg = LinearRegression()
lreg.fit(X_train_scaled, y_train)

# Prediction
lreg_pred = lreg.predict(X_test_scaled)

# Performance Metrics
mae = mean_absolute_error(y_test, lreg_pred)
r2 = r2_score(y_test, lreg_pred)
adjusted_r2 = 1 - (1-r2)*(len(y_test)-1)/(len(y_test)-X_test_scaled.shape[1]
mse = mean_absolute_error(y_test, lreg_pred)
rmse = np.sqrt(mean_squared_error(y_test,lreg_pred))

print("Linear Regression")
print("The regression Coefficient are : " , lreg.coef_)
print("The Intercept  is : " , lreg.intercept_)
print ("MAE value: {:.4f}".format(mae))
print ("MSE value: {:.4f}".format(mse))
print(" RMSE value : ".format(rmse))
print ("R2 Score value: {:.4f}".format(r2))
print ("Adjusted R2 Score value: {:.4f}".format(adjusted_r2))
```

```
Linear Regression
The regression Coefficient are :  [ 22.26986162  -0.10287995  -0.26293484 -11.5
7091945]
The Intercept  is :  8.741641791044772
MAE value: 4.2946
MSE value: 4.2946
 RMSE value :
R2 Score value: 0.7177
Adjusted R2 Score value: 0.7176
```

Validating Assumptions

In [73]:
```
# 1. linearity
plt.scatter(y_test,lreg_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[73]:
```
Text(0, 0.5, 'Test Predicted Data')
```

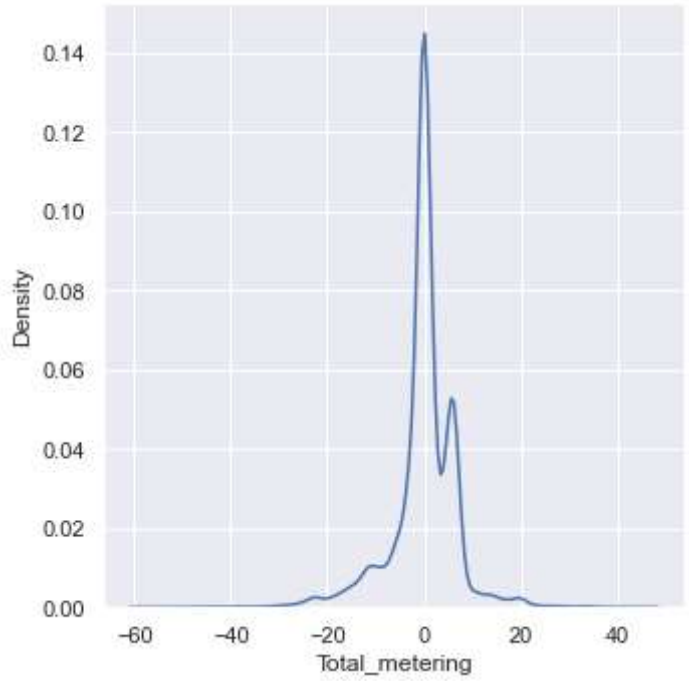


In [74]:
```
# 2. residuals
residuals=y_test- lreg_pred
residuals
```

Out[74]:
```
23257    -0.199873
49344    -7.969216
18582    -0.559410
2870     -7.976580
80932     1.417268
             ...
10983     6.545270
21071    10.845925
4034     -0.990240
65819    -6.135303
72346     6.643581
Name: Total_metering, Length: 33000, dtype: float64
```

In [75]:
```
1 sns.displot(residuals,kind="kde")
```

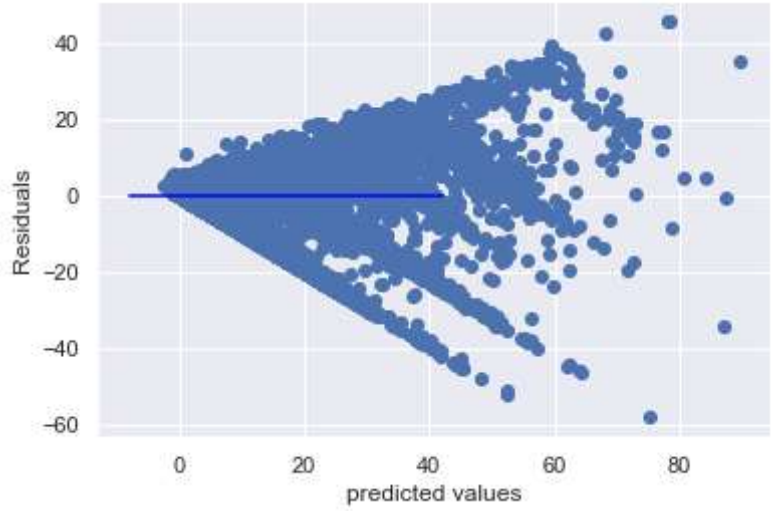
Out[75]: <seaborn.axisgrid.FacetGrid at 0x164884e7d30>



In [76]:

```
# 3. homoscedasticity
plt.scatter(lreg_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

Out[76]: <AxesSubplot:xlabel='predicted values', ylabel='Residuals'>



Ridge Regression Model

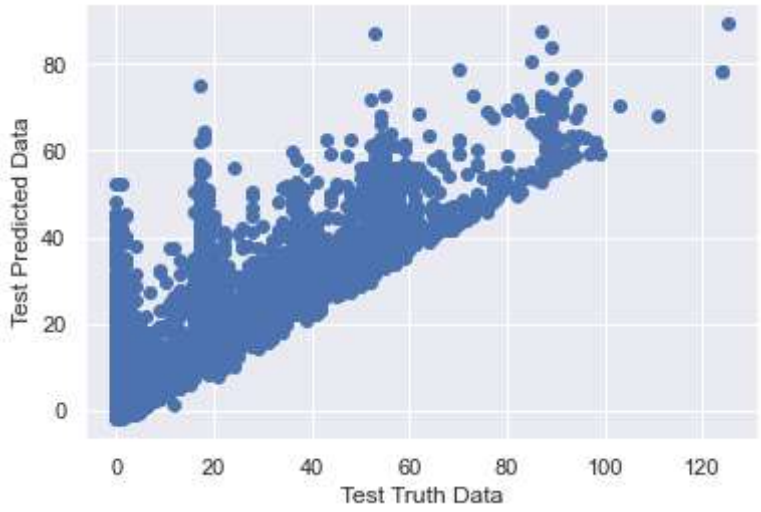
In [77]:

```
# Fitting Data into model
ridge = Ridge()
ridge.fit(X_train_scaled, y_train)

# Predicting Test Data
ridge_pred = ridge.predict(X_test_scaled)

# Performance Metrics
mae = mean_absolute_error(y_test, ridge_pred)
r2 = r2_score(y_test, ridge_pred)
adjusted_r2 = 1 - (1-r2)*(len(y_test)-1)/(len(y_test)-X_test_scaled.shape[1]
mse = mean_absolute_error(y_test, ridge_pred)
rmse = np.sqrt(mean_squared_error(y_test,ridge_pred))

print("Ridge Regression")
print("The regression Coefficient are : " , ridge.coef_)
print("The Intercept  is : " , ridge.intercept_)
print ("MAE value: {:.4f}".format(mae))
print ("MSE value: {:.4f}".format(mse))
print(" RMSE value : ".format(rmse))
print ("R2 Score value: {:.4f}".format(r2))
print ("Adjusted R2 Score value: {:.4f}".format(adjusted_r2))
```

```
Ridge Regression
The regression Coefficient are :  [ 21.98237015  -0.10879691  -0.25906394 -11.2
8046707]
The Intercept  is :  8.741641791044772
MAE value: 4.2948
MSE value: 4.2948
 RMSE value :
R2 Score value: 0.7177
Adjusted R2 Score value: 0.7176
```

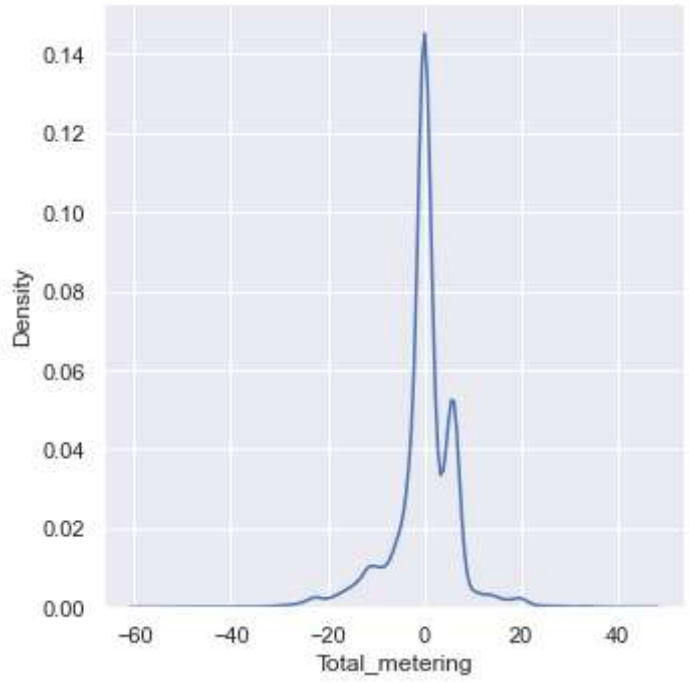
Validating Assumptions

In [78]:
```
## 1. linearity
plt.scatter(y_test,ridge_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
sns.set(rc={'figure.figsize':(10,8)})
```



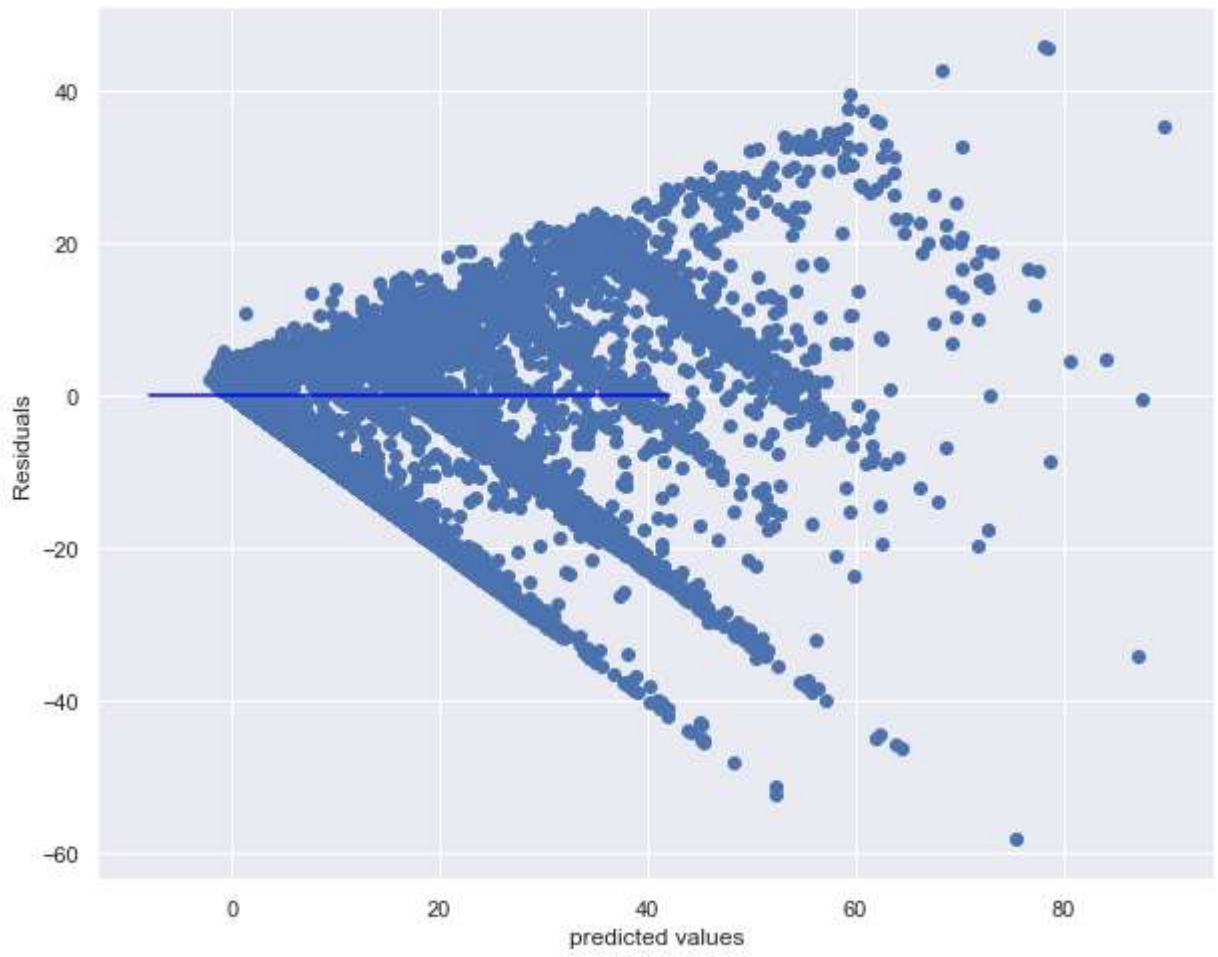
In [79]:
```
#2.residuals
residuals=y_test- ridge_pred
sns.displot(residuals,kind="kde")
```

Out[79]: <seaborn.axisgrid.FacetGrid at 0x164f5ec8400>



In [80]:
```
# 3. homoscedesacity
plt.scatter(ridge_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

Out[80]: <AxesSubplot:xlabel='predicted values', ylabel='Residuals'>



Lasso Regression Model

In [81]:

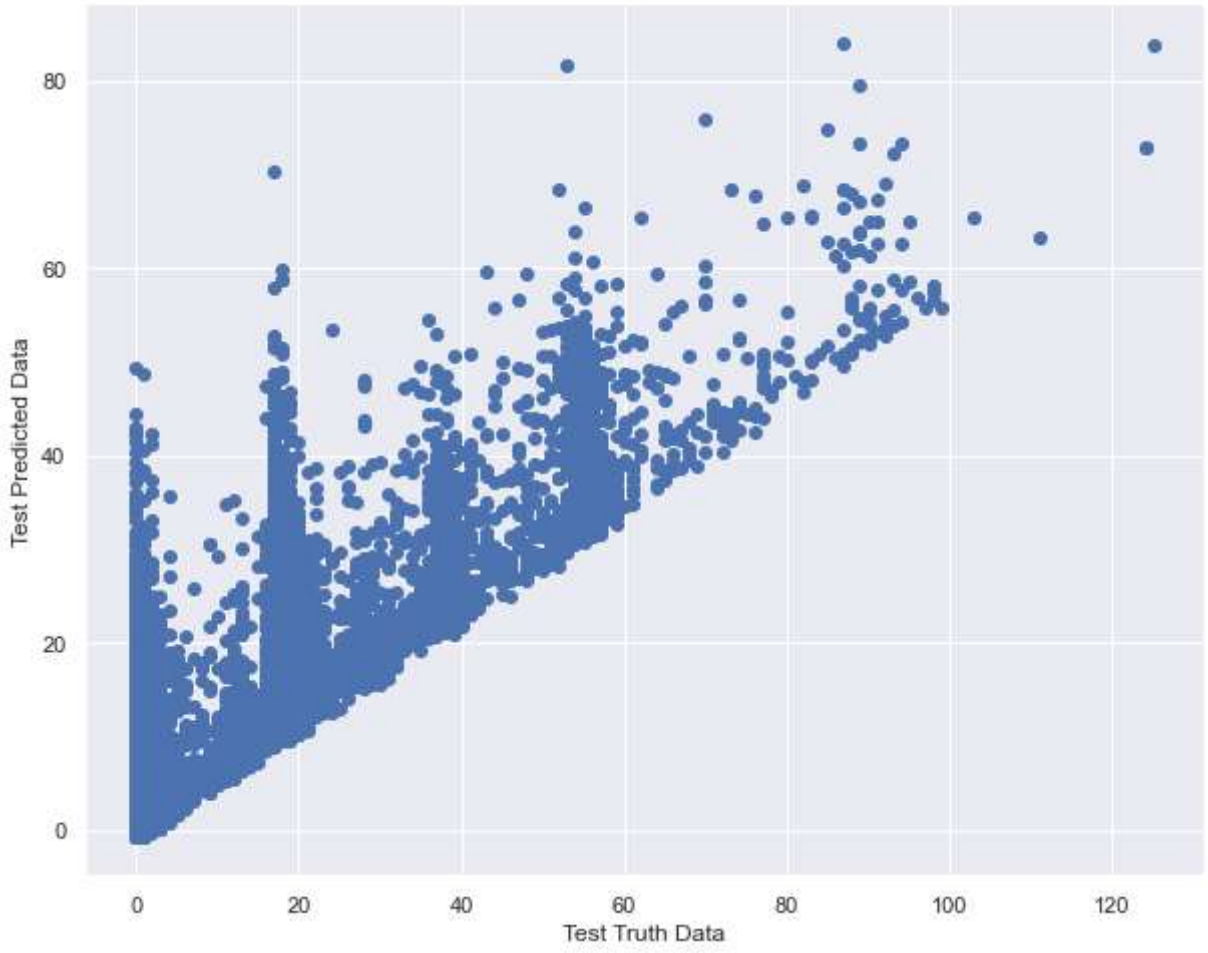
```
lasso = Lasso()

#Fitting Data into model

lasso.fit(X_train_scaled, y_train)

# Predicting Test Data
lasso_pred = lasso.predict(X_test_scaled)

#Performance Metrics
mae = mean_absolute_error(y_test, lasso_pred)
r2 = r2_score(y_test, lasso_pred)
adjusted_r2 = 1 - (1-r2)*(len(y_test)-1)/(len(y_test)-X_test_scaled.shape[1]
mse = mean_absolute_error(y_test, lasso_pred)
rmse = np.sqrt(mean_squared_error(y_test,lasso_pred))

print("Lasso Regression")
print("The regression Coefficient are : " , lasso.coef_)
print("The Intercept  is : " , lasso.intercept_)
print ("MAE value: {:.4f}".format(mae))
print ("MSE value: {:.4f}".format(mse))
print(" RMSE value : ".format(rmse))
print ("R2 Score value: {:.4f}".format(r2))
print ("Adjusted R2 Score value: {:.4f}".format(adjusted_r2))
```

```
Lasso Regression
The regression Coefficient are :  [ 9.79064854 -0.         -0.          0.
]
The Intercept  is :  8.741641791044774
MAE value: 4.5015
MSE value: 4.5015
 RMSE value :
R2 Score value: 0.7096
Adjusted R2 Score value: 0.7096
```

Checking Assumptions for Lasso model

In [82]:
```
# 1 .linearity
plt.scatter(y_test,lasso_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[82]: Text(0, 0.5, 'Test Predicted Data')

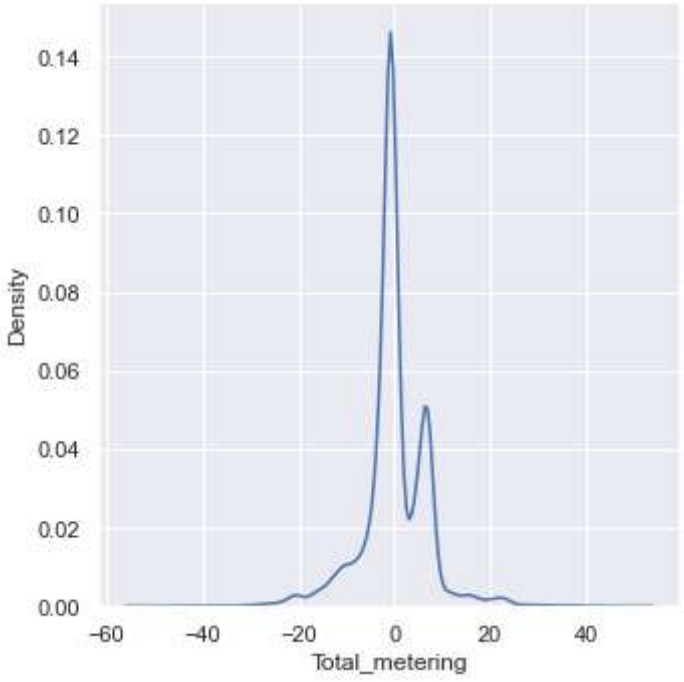


In [83]:
```
# 2. residuals
residuals=y_test- lasso_pred
residuals
```

Out[83]:
```
23257    -2.218687
49344    -4.699415
18582    -1.644386
2870     -7.716690
80932     0.186939
            ...
10983     7.253849
21071    15.063668
4034     -1.700611
65819    -2.926945
72346     7.247158
Name: Total_metering, Length: 33000, dtype: float64
```

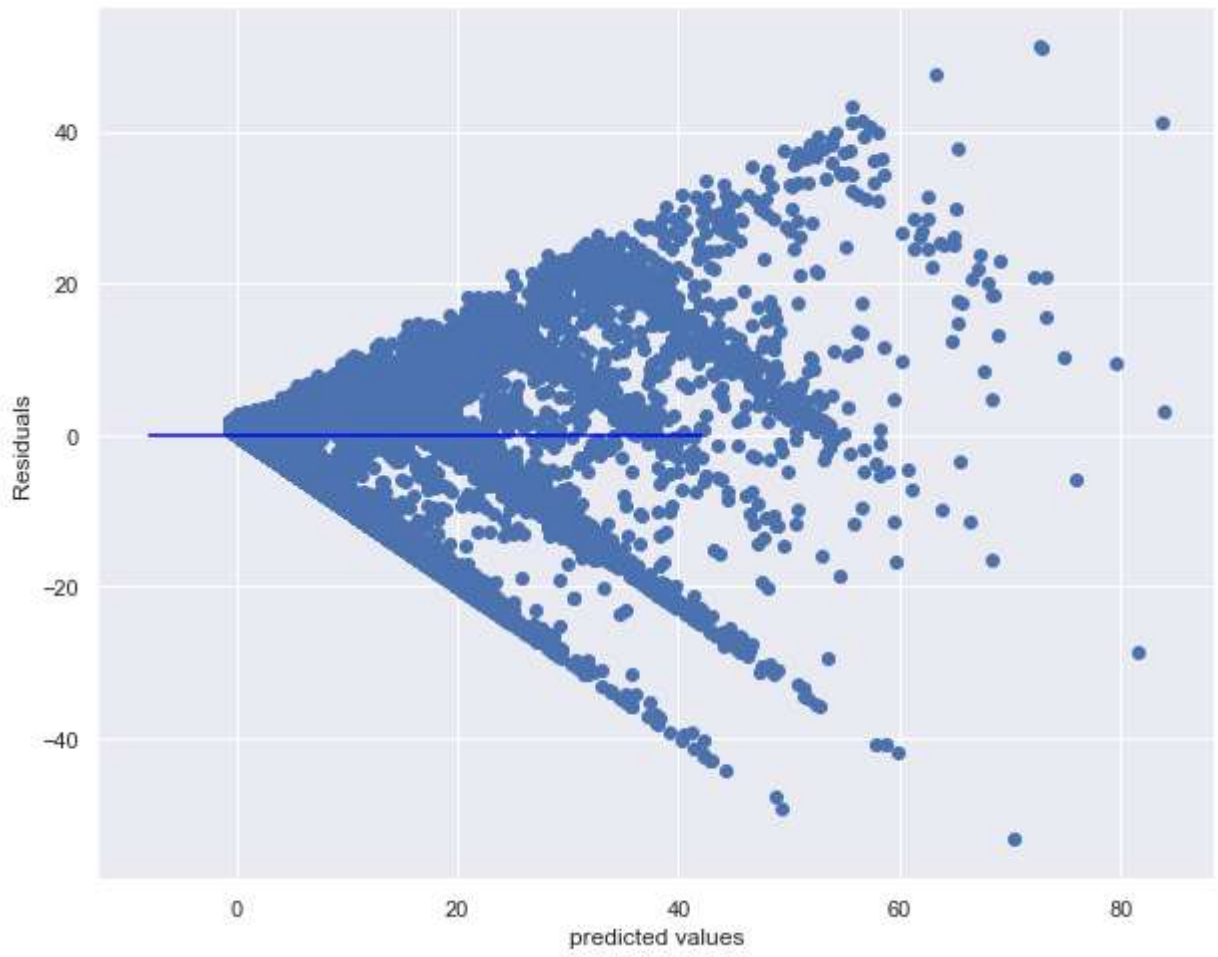
In [84]:
```
sns.displot(residuals,kind="kde")
```

Out[84]:
```
<seaborn.axisgrid.FacetGrid at 0x164fc4cf670>
```



In [85]:
```
# 3. Homoscedasticity
plt.scatter(lasso_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

Out[85]: <AxesSubplot:xlabel='predicted values', ylabel='Residuals'>



ElasticNet Regression Model

In [86]:

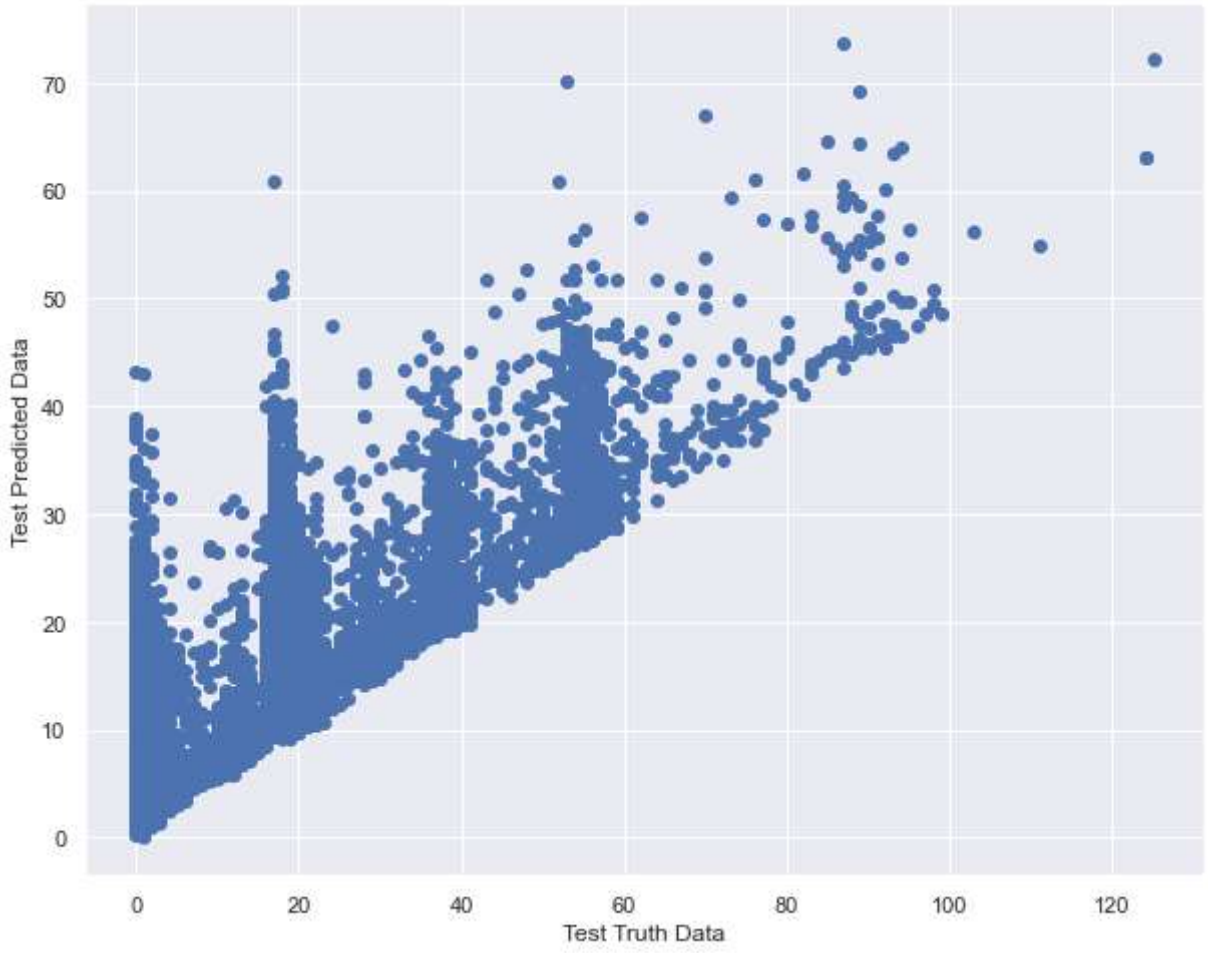
```
elsc = ElasticNet()

# Fitting Data into Model
elsc.fit(X_train_scaled, y_train)

# Predicting Test data

elsc_pred = elsc.predict(X_test_scaled)

# Performance metrics

mae = mean_absolute_error(y_test, elsc_pred)
r2 = r2_score(y_test, elsc_pred)
adjusted_r2 = 1 - (1-r2)*(len(y_test)-1)/(len(y_test)-X_test_scaled.shape[1]
mse = mean_absolute_error(y_test, elsc_pred)
rmse = np.sqrt(mean_squared_error(y_test,elsc_pred))

print("Elastic Net Regression")
print("The regression Coefficient are : " , elsc.coef_)
print("The Intercept  is : " , elsc.intercept_)
print ("MAE value: {:.4f}".format(mae))
print ("MSE value: {:.4f}".format(mse))
print(" RMSE value : ".format(rmse))
print ("R2 Score value: {:.4f}".format(r2))
print ("Adjusted R2 Score value: {:.4f}".format(adjusted_r2))
```

```
Elastic Net Regression
The regression Coefficient are :  [ 4.08916676  0.         -0.39787958  4.00499
64 ]
The Intercept  is :  8.741641791044774
MAE value: 5.1032
MSE value: 5.1032
 RMSE value :
R2 Score value: 0.6747
Adjusted R2 Score value: 0.6747
```

Checking Assumptions:

In [87]:
```
# Linarity
plt.scatter(y_test,elsc_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[87]: Text(0, 0.5, 'Test Predicted Data')

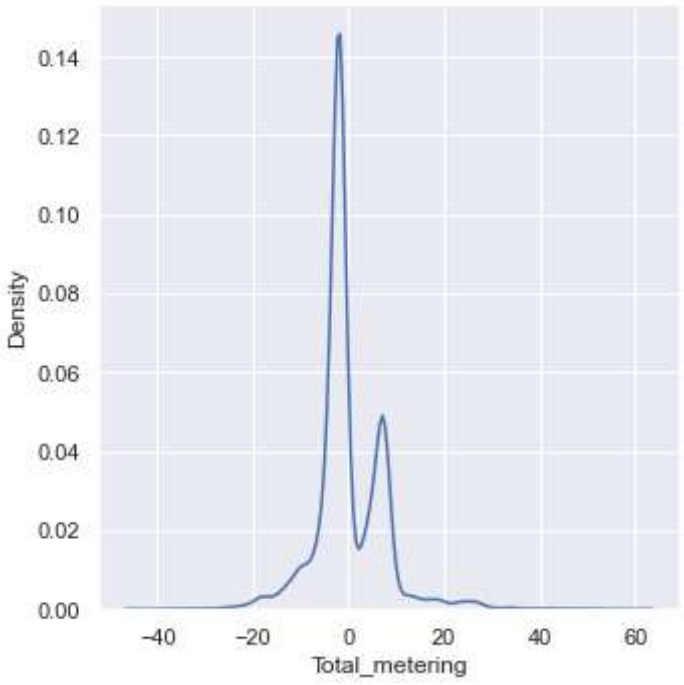


In [88]:
```
# residual distribution
residuals=y_test- elsc_pred
residuals
```

Out[88]:
```
23257    -3.620814
49344     3.028492
18582    -2.236732
2870     -7.874489
80932    -0.948179
           ...
10983     7.901345
21071    21.660026
4034     -2.752159
65819     2.296825
72346     7.454036
Name: Total_metering, Length: 33000, dtype: float64
```

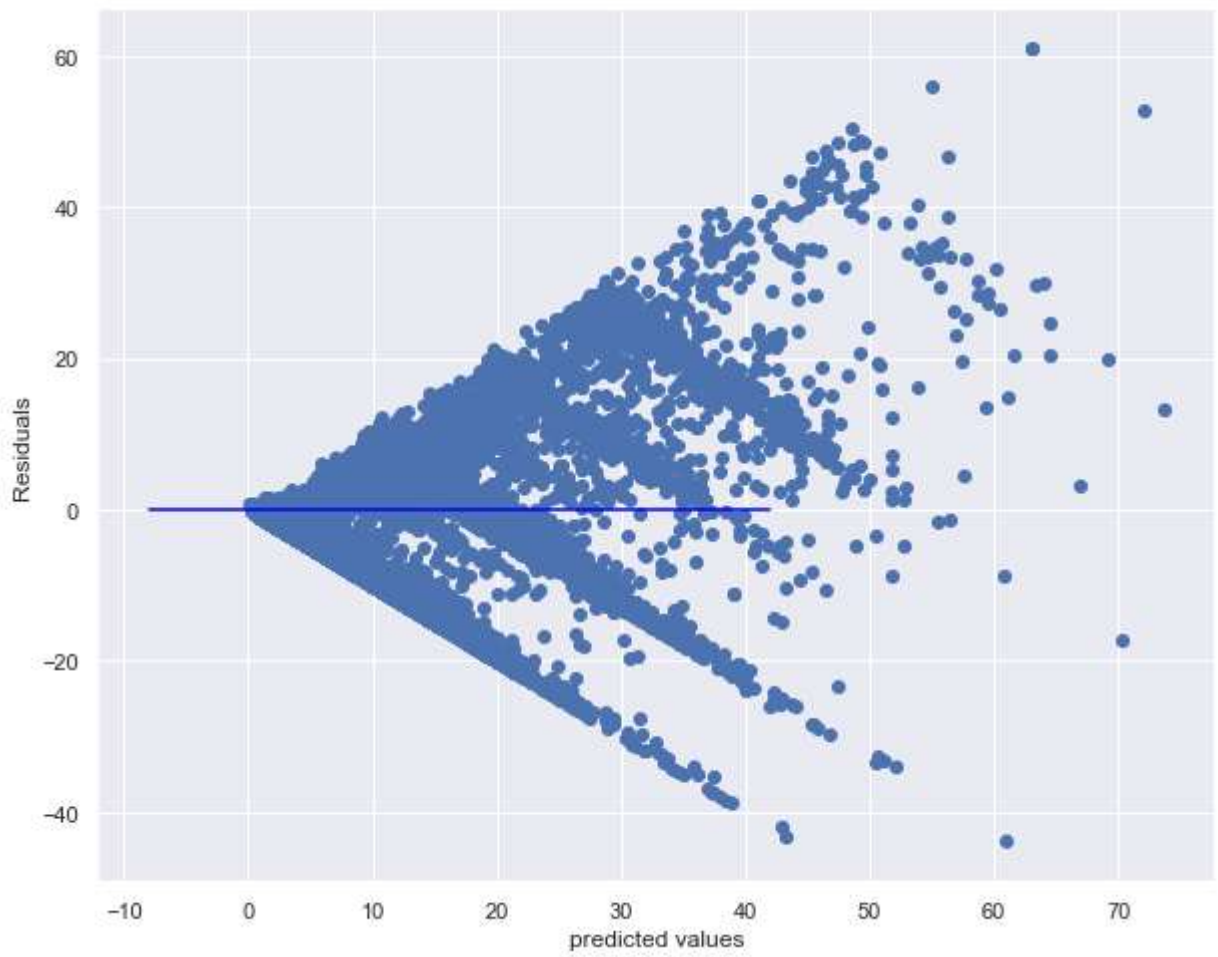
In [89]:
```
sns.displot(residuals,kind="kde")
```

Out[89]:
```
<seaborn.axisgrid.FacetGrid at 0x164fffb8c40>
```



In [90]:
```
# homoscedestacity
plt.scatter(elsc_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

Out[90]: <AxesSubplot:xlabel='predicted values', ylabel='Residuals'>



SVR Model

In [91]:
```
from sklearn.svm import SVR
```

In [94]:

```
svr = SVR(kernel='linear')

# Fitting Data into Model
svr.fit(X_train_scaled, y_train)

# Predicting Test data

svr_pred = svr.predict(X_test_scaled)

# Performance metrics

mae = mean_absolute_error(y_test, svr_pred)
r2 = r2_score(y_test, svr_pred)
adjusted_r2 = 1 - (1-r2)*(len(y_test)-1)/(len(y_test)-X_test_scaled.shape[1]
mse = mean_absolute_error(y_test, svr_pred)
rmse = np.sqrt(mean_squared_error(y_test,svr_pred))

print("SVR Model")
print("The regression Coefficient are : " , svr.coef_)
print("The Intercept  is : " , svr.intercept_)
print ("MAE value: {:.4f}".format(mae))
print ("MSE value: {:.4f}".format(mse))
print(" RMSE value : ".format(rmse))
print ("R2 Score value: {:.4f}".format(r2))
print ("Adjusted R2 Score value: {:.4f}".format(adjusted_r2))
```

```
SVR Model
The regression Coefficient are :  [[ 24.87838427  -0.13918674  -0.33761088 -13.
20534558]]
The Intercept  is :  [9.62432896]
MAE value: 4.2539
MSE value: 4.2539
 RMSE value :
R2 Score value: 0.7057
Adjusted R2 Score value: 0.7056
```

Observation:

Adjusted R squared score of different models is:

- Simple Linear Regression Model - **71.76%**
- Ridge Regression Model - **71.76%**
- Lasso Regression Model - **70.96%**
- ElasticNet Regression Model - **67.47%**
- SVC Model - **70.56%**

So the model with best Adjusted R sqr. score is **Simple Linear Regression Model**.

8. HyperParameter Tuning

Now we perform hyper parameter tuning to find the best parameters for SVC model using GridSearchCV

In [95]:
```
from sklearn.model_selection import GridSearchCV
param= {'kernel':["linear","rbf","poly","sigmoid"]}

grid_svr= GridSearchCV(estimator = svr, param_grid=param, cv=5,n_jobs=-1)
grid_svr.fit(X_train_scaled,y_train)

grid_svr.best_params_
```

Out[95]:
```
{'kernel': 'rbf'}
```

Using the best parameters creating new model

In [96]:
```
svr_model2=SVR( kernel='rbf')   # fitting the new model with best parameters
svr_model2.fit(X_train_scaled,y_train)   # new predictions
```

Out[96]:
```
SVR()
```

In [97]:
```
y_pred_new=svr_model2.predict(X_test_scaled)
y_pred_new
```

Out[97]:
```
array([ 1.87737617, 62.69220944,  0.25986639, ...,  0.31298476,
       54.06431366, 16.45824968])
```

In [99]:
```
# Performance metrics

mae = mean_absolute_error(y_test, y_pred_new)
r2 = r2_score(y_test, y_pred_new)
adjusted_r2 = 1 - (1-r2)*(len(y_test)-1)/(len(y_test)-X_test_scaled.shape[1]
mse = mean_absolute_error(y_test, y_pred_new)
rmse = np.sqrt(mean_squared_error(y_test,y_pred_new))

print("SVR hyperparameter Tuned model")
# print("The regression Coefficient are : " , svr_model2.coef_)
print("The Intercept  is : " , svr_model2.intercept_)
print ("MAE value: {:.4f}".format(mae))
print ("MSE value: {:.4f}".format(mse))
print(" RMSE value : ".format(rmse))
print ("R2 Score value: {:.4f}".format(r2))
print ("Adjusted R2 Score value: {:.4f}".format(adjusted_r2))
```

```
SVR hyperparameter Tuned model
The Intercept  is :  [28.22321035]
MAE value: 3.2458
MSE value: 3.2458
 RMSE value :
R2 Score value: 0.7270
Adjusted R2 Score value: 0.7269
```

After Hyper Parameter Tuning we are able to improve the Adjusted R squared value to **72.69%**.

Thank you

-Performed By: Ankit Dubey