# Individual household electric power consumption Data Set

**Rajan Kumar**

**Problem Statement**

Regression: [https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption (https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption)](https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption)

Classification:[https://archive.ics.uci.edu/ml/datasets/Census+Income (https://archive.ics.uci.edu/ml/datasets/Census+Income)](https://archive.ics.uci.edu/ml/datasets/Census+Income)

1. Data ingestion
2. EDA
3. Preprocessing Pickling for the preprocessing object(save the preprocessing model) After preprocessing you have to store data inside MONGODB

You have to load the data from mongo db

4. Model

Regression:linear regression,ridge regression,lasso regression,elastic net, support vector regression

Classification: logistic regression,SVM(kernel)

Hyperparameter tuning is mandatory(GRID SEARCH CV)

5. Evaluation of the model Regression evaluation matrix: R2 and adjusted R2 Classification confusion matrix, ROC AUC score

Data Set Information:

This archive contains 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months). Notes: 1.(global_active_power*1000/60 - sub_metering_1 - sub_metering_2 - sub_metering_3) represents the active energy consumed every minute (in watt hour) in the household by electrical equipment not measured in sub-meterings 1, 2 and 3. 2.The dataset contains some missing values in the measurements (nearly 1,25% of the rows). All calendar timestamps are present in the dataset but for some timestamps, the measurement values are missing: a missing value is represented by the absence of value between two consecutive semi-colon attribute separators. For instance, the dataset shows missing values on April 28, 2007.

Attribute Information:

1.date: Date in format dd/mm/yyyy

2.time: time in format hh:mm:ss

3.global_active_power: household global minute-averaged active power (in kilowatt)

4.global_reactive_power: household global minute-averaged reactive power (in kilowatt)

5.voltage: minute-averaged voltage (in volt)

6.global_intensity: household global minute-averaged current intensity (in ampere)

7.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).

8.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.

9.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.

In [1]:

```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

# 1. Data Ingesion

In [2]:

```
df = pd.read_csv(r"C:\Users\Rajan\household_power_consumption.txt",sep=';')
```

```
C:\Users\Rajan\AppData\Local\Temp\ipykernel_19848\2095378203.py:1: DtypeWarning: Columns (2,3,4,5,6,7) have mixed types. Sp
ecify dtype option on import or set low_memory=False.
  df = pd.read_csv(r"C:\Users\Rajan\household_power_consumption.txt",sep=';')
```

In [3]:

```
df.head()
```

Out[3]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 16/12/2006 | 17:24:00 | 4.216 | 0.418 | 234.840 | 18.400 | 0.000 | 1.000 | 17.0 |
| **1** | 16/12/2006 | 17:25:00 | 5.360 | 0.436 | 233.630 | 23.000 | 0.000 | 1.000 | 16.0 |
| **2** | 16/12/2006 | 17:26:00 | 5.374 | 0.498 | 233.290 | 23.000 | 0.000 | 2.000 | 17.0 |
| **3** | 16/12/2006 | 17:27:00 | 5.388 | 0.502 | 233.740 | 23.000 | 0.000 | 1.000 | 17.0 |
| **4** | 16/12/2006 | 17:28:00 | 3.666 | 0.528 | 235.680 | 15.800 | 0.000 | 1.000 | 17.0 |

In [4]:

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

In [5]:

```
df.shape
```

Out[5]:

```
(2075259, 9)
```

In [6]:

```
df.columns
```

Out[6]:

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

In [7]:

```
#there are multiple blank ? values in the data, need to replace it with np.NaN
```

In [8]:

```
df[df['Global_active_power']=="?"]
```

Out[8]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 |
|---|---|---|---|---|---|---|---|---|---|
| **6839** | 21/12/2006 | 11:23:00 | ? | ? | ? | ? | ? | ? | NaN |
| **6840** | 21/12/2006 | 11:24:00 | ? | ? | ? | ? | ? | ? | NaN |
| **19724** | 30/12/2006 | 10:08:00 | ? | ? | ? | ? | ? | ? | NaN |
| **19725** | 30/12/2006 | 10:09:00 | ? | ? | ? | ? | ? | ? | NaN |
| **41832** | 14/1/2007 | 18:36:00 | ? | ? | ? | ? | ? | ? | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1990185** | 28/9/2010 | 19:09:00 | ? | ? | ? | ? | ? | ? | NaN |
| **1990186** | 28/9/2010 | 19:10:00 | ? | ? | ? | ? | ? | ? | NaN |
| **1990187** | 28/9/2010 | 19:11:00 | ? | ? | ? | ? | ? | ? | NaN |
| **1990188** | 28/9/2010 | 19:12:00 | ? | ? | ? | ? | ? | ? | NaN |
| **2027411** | 24/10/2010 | 15:35:00 | ? | ? | ? | ? | ? | ? | NaN |

25979 rows × 9 columns

In [9]:

```
#replacing ? with Nan value
df.replace("?", np.NaN, inplace= True)
```

In [10]:

```
df[df['Global_active_power']=="?"]
```

Out[10]:

| Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 |
|---|---|---|---|---|---|---|---|---|

In [11]:

```
#Checking null values

df.isna().sum()
#Here we have replaced Nan for '?' for all 25979 rows for all the columns where null values were found
```

Out[11]:

```
Date                         0
Time                         0
Global_active_power      25979
Global_reactive_power    25979
Voltage                  25979
Global_intensity         25979
Sub_metering_1           25979
Sub_metering_2           25979
Sub_metering_3           25979
dtype: int64
```

In [12]:

```
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
```

In [13]:

```
#selecting 40000 sample data
df1 = df.sample(40000)
```

In [14]:

```
df1.head()
```

Out[14]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 |
|---|---|---|---|---|---|---|---|---|---|
| **1413467** | 24/8/2009 | 07:11:00 | 1.600 | 0.262 | 243.170 | 6.600 | 0.000 | 1.000 | 19.0 |
| **406388** | 24/9/2007 | 22:32:00 | 0.238 | 0.000 | 242.520 | 1.000 | 0.000 | 0.000 | 0.0 |
| **155665** | 3/4/2007 | 19:49:00 | 4.530 | 0.000 | 233.660 | 19.400 | 37.000 | 1.000 | 0.0 |
| **1003973** | 12/11/2008 | 22:17:00 | 1.564 | 0.000 | 241.730 | 6.400 | 1.000 | 0.000 | 18.0 |
| **2014931** | 15/10/2010 | 23:35:00 | 1.076 | 0.550 | 244.440 | 4.800 | 0.000 | 1.000 | 11.0 |

In [15]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 1413467 to 17460
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   40000 non-null  object 
 1   Time                   40000 non-null  object 
 2   Global_active_power    39524 non-null  object 
 3   Global_reactive_power  39524 non-null  object 
 4   Voltage                39524 non-null  object 
 5   Global_intensity       39524 non-null  object 
 6   Sub_metering_1         39524 non-null  object 
 7   Sub_metering_2         39524 non-null  object 
 8   Sub_metering_3         39524 non-null  float64
dtypes: float64(1), object(8)
memory usage: 3.1+ MB
```

In [16]:

```
df1['Date'] = pd.to_datetime(df1['Date'])
df1['date'] = df1['Date'].dt.day
df1['month'] = df1['Date'].dt.month
df1['year'] = df1['Date'].dt.year
df1.head()
```

Out[16]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 | date |
|---|---|---|---|---|---|---|---|---|---|---|
| **1413467** | 2009-08-24 | 07:11:00 | 1.600 | 0.262 | 243.170 | 6.600 | 0.000 | 1.000 | 19.0 | 24 |
| **406388** | 2007-09-24 | 22:32:00 | 0.238 | 0.000 | 242.520 | 1.000 | 0.000 | 0.000 | 0.0 | 24 |
| **155665** | 2007-03-04 | 19:49:00 | 4.530 | 0.000 | 233.660 | 19.400 | 37.000 | 1.000 | 0.0 | 4 |
| **1003973** | 2008-12-11 | 22:17:00 | 1.564 | 0.000 | 241.730 | 6.400 | 1.000 | 0.000 | 18.0 | 11 |
| **2014931** | 2010-10-15 | 23:35:00 | 1.076 | 0.550 | 244.440 | 4.800 | 0.000 | 1.000 | 11.0 | 15 |

In [17]:

```
df1['hour'] = pd.to_datetime(df1['Time'],format = '%H:%M:%S').dt.hour
df1['minute'] = pd.to_datetime(df1['Time'],format = '%H:%M:%S').dt.minute
df1.head()
```

Out[17]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 | date |
|---|---|---|---|---|---|---|---|---|---|---|
| **1413467** | 2009-08-24 | 07:11:00 | 1.600 | 0.262 | 243.170 | 6.600 | 0.000 | 1.000 | 19.0 | 24 |
| **406388** | 2007-09-24 | 22:32:00 | 0.238 | 0.000 | 242.520 | 1.000 | 0.000 | 0.000 | 0.0 | 24 |
| **155665** | 2007-03-04 | 19:49:00 | 4.530 | 0.000 | 233.660 | 19.400 | 37.000 | 1.000 | 0.0 | 4 |
| **1003973** | 2008-12-11 | 22:17:00 | 1.564 | 0.000 | 241.730 | 6.400 | 1.000 | 0.000 | 18.0 | 11 |
| **2014931** | 2010-10-15 | 23:35:00 | 1.076 | 0.550 | 244.440 | 4.800 | 0.000 | 1.000 | 11.0 | 15 |

In [18]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 1413467 to 17460
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype         
---  ------                 --------------  -----         
 0   Date                   40000 non-null  datetime64[ns]
 1   Time                   40000 non-null  object        
 2   Global_active_power    39524 non-null  object        
 3   Global_reactive_power  39524 non-null  object        
 4   Voltage                39524 non-null  object        
 5   Global_intensity       39524 non-null  object        
 6   Sub_metering_1         39524 non-null  object        
 7   Sub_metering_2         39524 non-null  object        
 8   Sub_metering_3         39524 non-null  float64       
 9   date                   40000 non-null  int64         
 10  month                  40000 non-null  int64         
 11  year                   40000 non-null  int64         
 12  hour                   40000 non-null  int64         
 13  minute                 40000 non-null  int64         
dtypes: datetime64[ns](1), float64(1), int64(5), object(7)
memory usage: 4.6+ MB
```

In [19]:

```
#converting the data types
df1['Global_active_power']= df1['Global_active_power'].astype(float)
df1['Global_reactive_power']= df1['Global_reactive_power'].astype(float)
df1['Voltage']= df1['Voltage'].astype(float)
df1['Global_intensity']= df1['Global_intensity'].astype(float)
df1['Sub_metering_1']= df1['Sub_metering_1'].astype(float)
df1['Sub_metering_2']= df1['Sub_metering_2'].astype(float)
```

In [20]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 1413467 to 17460
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   40000 non-null  datetime64[ns]
 1   Time                   40000 non-null  object
 2   Global_active_power    39524 non-null  float64
 3   Global_reactive_power  39524 non-null  float64
 4   Voltage                39524 non-null  float64
 5   Global_intensity       39524 non-null  float64
 6   Sub_metering_1         39524 non-null  float64
 7   Sub_metering_2         39524 non-null  float64
 8   Sub_metering_3         39524 non-null  float64
 9   date                   40000 non-null  int64
 10  month                  40000 non-null  int64
 11  year                   40000 non-null  int64
 12  hour                   40000 non-null  int64
 13  minute                 40000 non-null  int64
dtypes: datetime64[ns](1), float64(7), int64(5), object(1)
memory usage: 4.6+ MB
```

In [21]:

```
df1['total_metering'] = df1['Sub_metering_1']+df1['Sub_metering_2']+df1['Sub_metering_3']
df1.head()
```

Out[21]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Sub_metering_2 | Sub_metering_3 | date |
|---|---|---|---|---|---|---|---|---|---|---|
| **1413467** | 2009-08-24 | 07:11:00 | 1.600 | 0.262 | 243.17 | 6.6 | 0.0 | 1.0 | 19.0 | 24 |
| **406388** | 2007-09-24 | 22:32:00 | 0.238 | 0.000 | 242.52 | 1.0 | 0.0 | 0.0 | 0.0 | 24 |
| **155665** | 2007-03-04 | 19:49:00 | 4.530 | 0.000 | 233.66 | 19.4 | 37.0 | 1.0 | 0.0 | 4 |
| **1003973** | 2008-12-11 | 22:17:00 | 1.564 | 0.000 | 241.73 | 6.4 | 1.0 | 0.0 | 18.0 | 11 |
| **2014931** | 2010-10-15 | 23:35:00 | 1.076 | 0.550 | 244.44 | 4.8 | 0.0 | 1.0 | 11.0 | 15 |

In [22]:

```
#dropping unnecessary columns
df2 = df1.drop(columns=['Date','Time','Sub_metering_3','Sub_metering_2','Sub_metering_1'])
```

In [23]:

```
df2.head()
```

Out[23]:

| | Global_active_power | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute | total_metering |
|---|---|---|---|---|---|---|---|---|---|---|
| **1413467** | 1.600 | 0.262 | 243.17 | 6.6 | 24 | 8 | 2009 | 7 | 11 | 20.0 |
| **406388** | 0.238 | 0.000 | 242.52 | 1.0 | 24 | 9 | 2007 | 22 | 32 | 0.0 |
| **155665** | 4.530 | 0.000 | 233.66 | 19.4 | 4 | 3 | 2007 | 19 | 49 | 38.0 |
| **1003973** | 1.564 | 0.000 | 241.73 | 6.4 | 11 | 12 | 2008 | 22 | 17 | 19.0 |
| **2014931** | 1.076 | 0.550 | 244.44 | 4.8 | 15 | 10 | 2010 | 23 | 35 | 12.0 |

In [24]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40000 entries, 1413467 to 17460
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Global_active_power    39524 non-null  float64
 1   Global_reactive_power  39524 non-null  float64
 2   Voltage                39524 non-null  float64
 3   Global_intensity       39524 non-null  float64
 4   date                   40000 non-null  int64  
 5   month                  40000 non-null  int64  
 6   year                   40000 non-null  int64  
 7   hour                   40000 non-null  int64  
 8   minute                 40000 non-null  int64  
 9   total_metering         39524 non-null  float64
dtypes: float64(5), int64(5)
memory usage: 3.4 MB
```

In [25]:

```
df2.describe()
```

Out[25]:

| | Global_active_power | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 39524.000000 | 39524.000000 | 39524.000000 | 39524.000000 | 40000.000000 | 40000.000000 | 40000.000000 | 40000.000000 | 40000.000000 |
| **mean** | 1.099516 | 0.124644 | 240.798057 | 4.662423 | 15.726400 | 6.509950 | 2008.433725 | 11.523050 | 29.657925 |
| **std** | 1.067506 | 0.113641 | 3.237816 | 4.484632 | 8.827598 | 3.437811 | 1.128290 | 6.899805 | 17.301693 |
| **min** | 0.078000 | 0.000000 | 225.510000 | 0.200000 | 1.000000 | 1.000000 | 2006.000000 | 0.000000 | 0.000000 |
| **25%** | 0.310000 | 0.048000 | 238.950000 | 1.400000 | 8.000000 | 4.000000 | 2007.000000 | 6.000000 | 15.000000 |
| **50%** | 0.608000 | 0.100000 | 240.960000 | 2.600000 | 16.000000 | 7.000000 | 2008.000000 | 12.000000 | 30.000000 |
| **75%** | 1.532000 | 0.196000 | 242.850000 | 6.400000 | 23.000000 | 9.000000 | 2009.000000 | 18.000000 | 45.000000 |
| **max** | 9.132000 | 1.190000 | 253.070000 | 39.400000 | 31.000000 | 12.000000 | 2010.000000 | 23.000000 | 59.000000 |

In [26]:

```
df2.describe().T
```

Out[26]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Global_active_power** | 39524.0 | 1.099516 | 1.067506 | 0.078 | 0.310 | 0.608 | 1.532 | 9.132 |
| **Global_reactive_power** | 39524.0 | 0.124644 | 0.113641 | 0.000 | 0.048 | 0.100 | 0.196 | 1.190 |
| **Voltage** | 39524.0 | 240.798057 | 3.237816 | 225.510 | 238.950 | 240.960 | 242.850 | 253.070 |
| **Global_intensity** | 39524.0 | 4.662423 | 4.484632 | 0.200 | 1.400 | 2.600 | 6.400 | 39.400 |
| **date** | 40000.0 | 15.726400 | 8.827598 | 1.000 | 8.000 | 16.000 | 23.000 | 31.000 |
| **month** | 40000.0 | 6.509950 | 3.437811 | 1.000 | 4.000 | 7.000 | 9.000 | 12.000 |
| **year** | 40000.0 | 2008.433725 | 1.128290 | 2006.000 | 2007.000 | 2008.000 | 2009.000 | 2010.000 |
| **hour** | 40000.0 | 11.523050 | 6.899805 | 0.000 | 6.000 | 12.000 | 18.000 | 23.000 |
| **minute** | 40000.0 | 29.657925 | 17.301693 | 0.000 | 15.000 | 30.000 | 45.000 | 59.000 |
| **total_metering** | 39524.0 | 8.989778 | 13.011069 | 0.000 | 0.000 | 1.000 | 18.000 | 126.000 |

In [27]:

```
df2.duplicated()
```

Out[27]:

```
1413467    False
406388     False
155665     False
1003973    False
2014931    False
           ...  
1730198    False
769787     False
1357699    False
740960     False
17460      False
Length: 40000, dtype: bool
```

In [28]:

```
df2.skew()
```

Out[28]:

```
Global_active_power      1.786036
Global_reactive_power    1.287292
Voltage                 -0.325067
Global_intensity         1.844230
date                    -0.000432
month                   -0.007051
year                     0.001029
hour                    -0.004277
minute                  -0.015207
total_metering           2.240503
dtype: float64
```

In [29]:

```
df2['Global_active_power'] = df2['Global_active_power'].fillna(df2['Global_active_power'].median())
df2['Global_reactive_power'] = df2['Global_reactive_power'].fillna(df2['Global_reactive_power'].median())
df2['Voltage'] = df2['Voltage'].fillna(df2['Voltage'].mean())
df2['Global_intensity'] = df2['Global_intensity'].fillna(df2['Global_intensity'].median())
df2['total_metering'] = df2['total_metering'].fillna(df2['total_metering'].median())
```

In [30]:

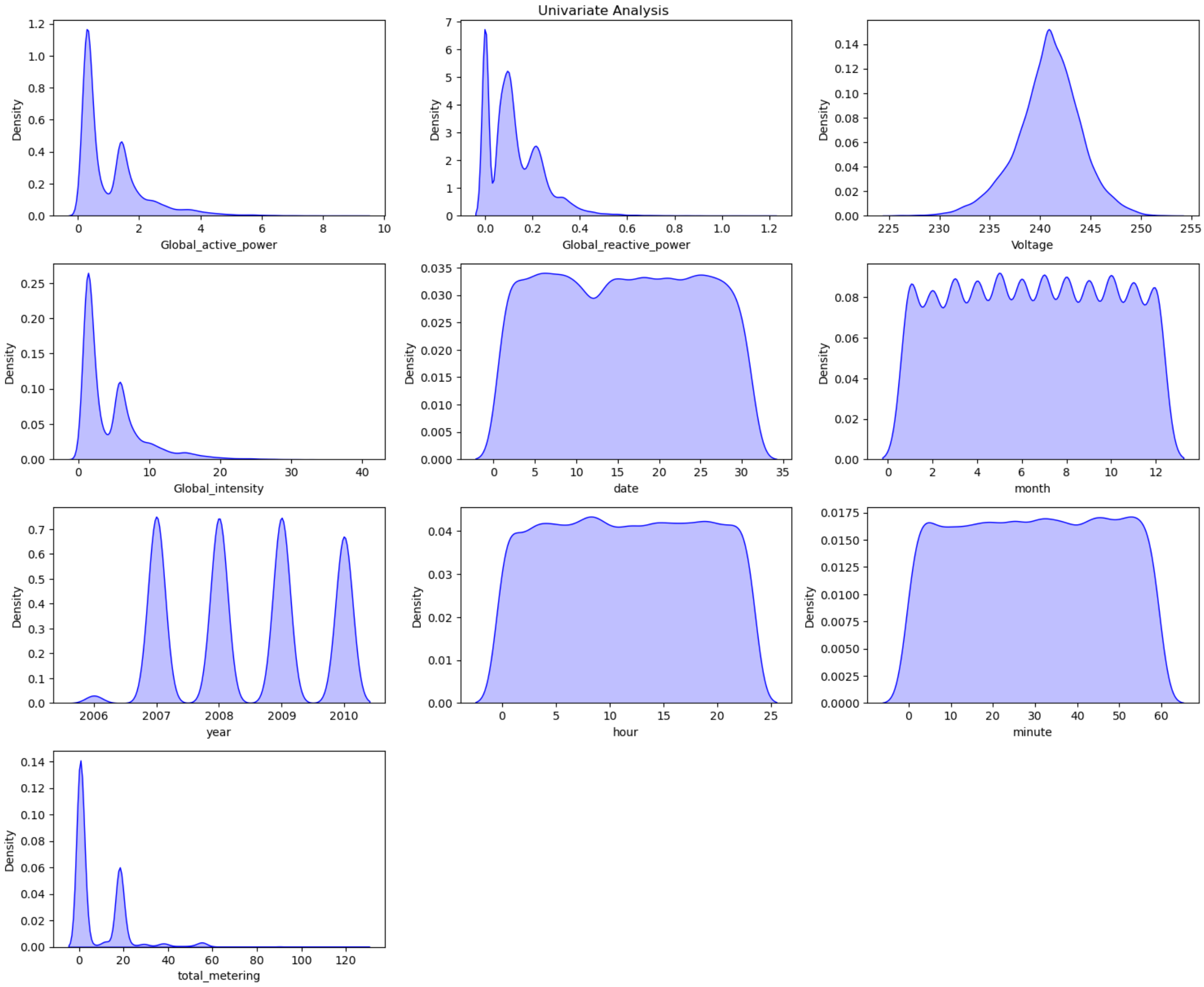
```
df2.head()
```

Out[30]:

| | Global_active_power | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute | total_metering |
|---|---|---|---|---|---|---|---|---|---|---|
| **1413467** | 1.600 | 0.262 | 243.17 | 6.6 | 24 | 8 | 2009 | 7 | 11 | 20.0 |
| **406388** | 0.238 | 0.000 | 242.52 | 1.0 | 24 | 9 | 2007 | 22 | 32 | 0.0 |
| **155665** | 4.530 | 0.000 | 233.66 | 19.4 | 4 | 3 | 2007 | 19 | 49 | 38.0 |
| **1003973** | 1.564 | 0.000 | 241.73 | 6.4 | 11 | 12 | 2008 | 22 | 17 | 19.0 |
| **2014931** | 1.076 | 0.550 | 244.44 | 4.8 | 15 | 10 | 2010 | 23 | 35 | 12.0 |

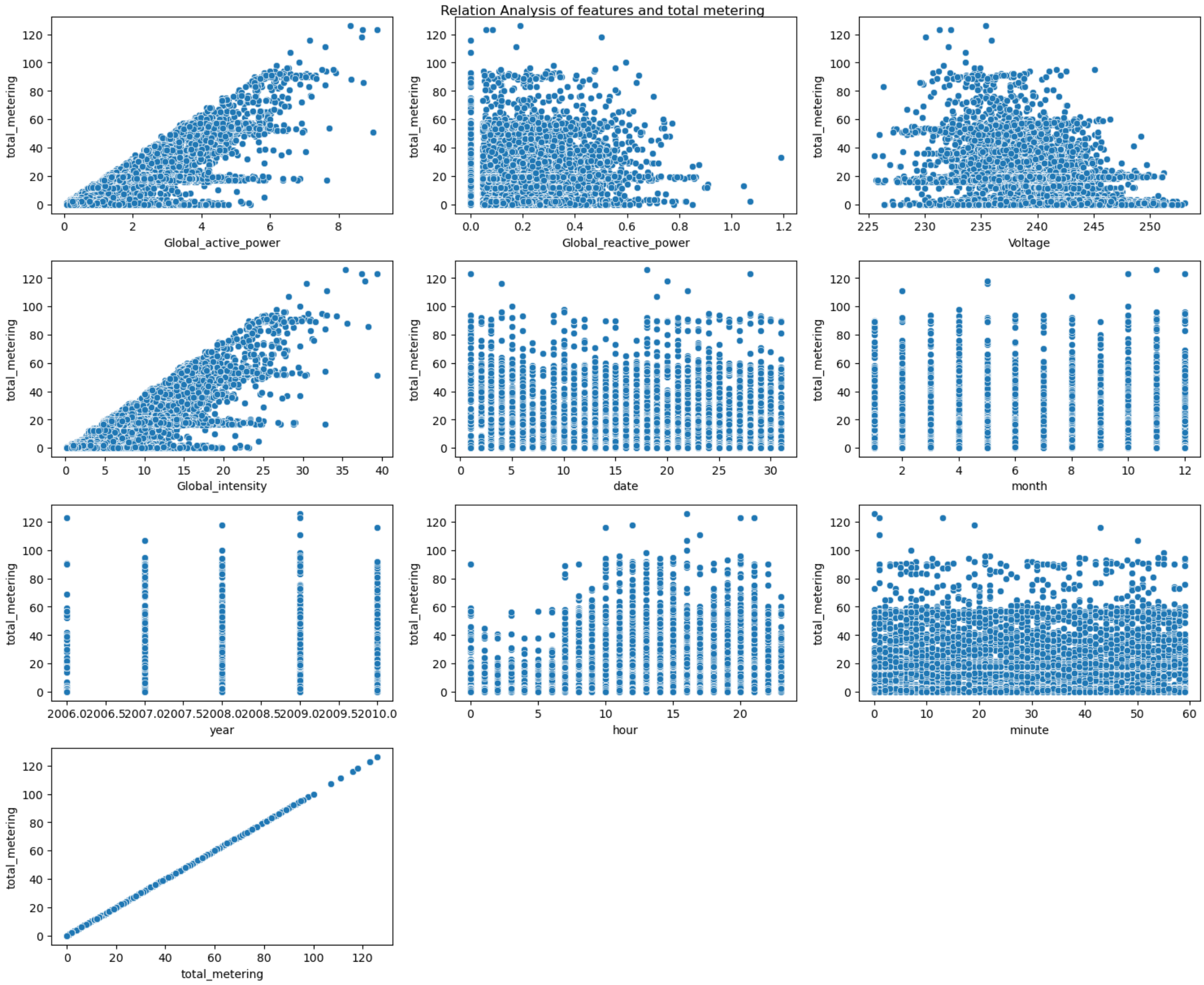
In [32]:

```
plt.figure(figsize=(15,15))
plt.suptitle('Univariate Analysis')

for i in range(0,len(df2.columns)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(x=df2[df2.columns[i]],shade=True, color='b')
    plt.xlabel(df2.columns[i])
    plt.tight_layout()
```



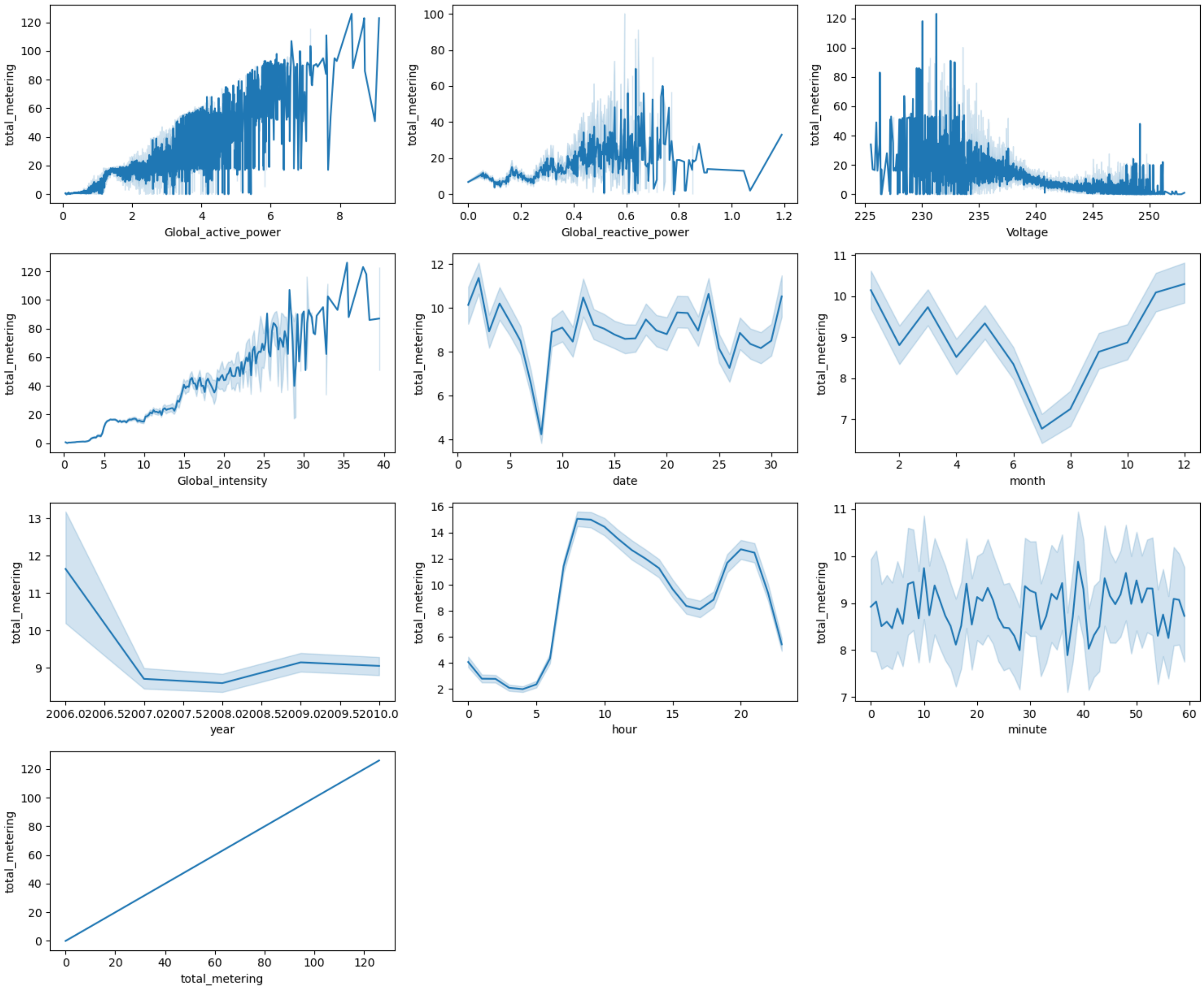
In [36]:

```
plt.figure(figsize=(15,15))
plt.suptitle("Relation Analysis of features and total metering")

for i in range(0,len(df2.columns)):
    plt.subplot(5,3,i+1)
    sns.scatterplot(data = df2, x = df2[df2.columns[i]], y = df2['total_metering'])
    plt.xlabel(df2.columns[i])
    plt.tight_layout()
```



In [39]:

```
plt.figure(figsize=(15,15))

for i in range(0,len(df2.columns)):
    plt.subplot(5,3,i+1)
    sns.lineplot(x = df2[df2.columns[i]], y = df2['total_metering'],data = df2)
    plt.xlabel(df2.columns[i])
    plt.tight_layout()
```




# Observations

1. Reading increases in the morning and then goes dip during after noon, then again rises in evening then dipping during night
2. power consumption gois down till mid year then again rises
3. after 2006 power consumption has been decreased and almost same onwards

In [40]:

```
df2.corr()
#corr() is used to find the pairwise correlation of all columns in the Pandas Dataframe in Python.
#Any NaN values are automatically excluded. Any non-numeric data type or columns in the Dataframe, it is ignored.
```

Out[40]:

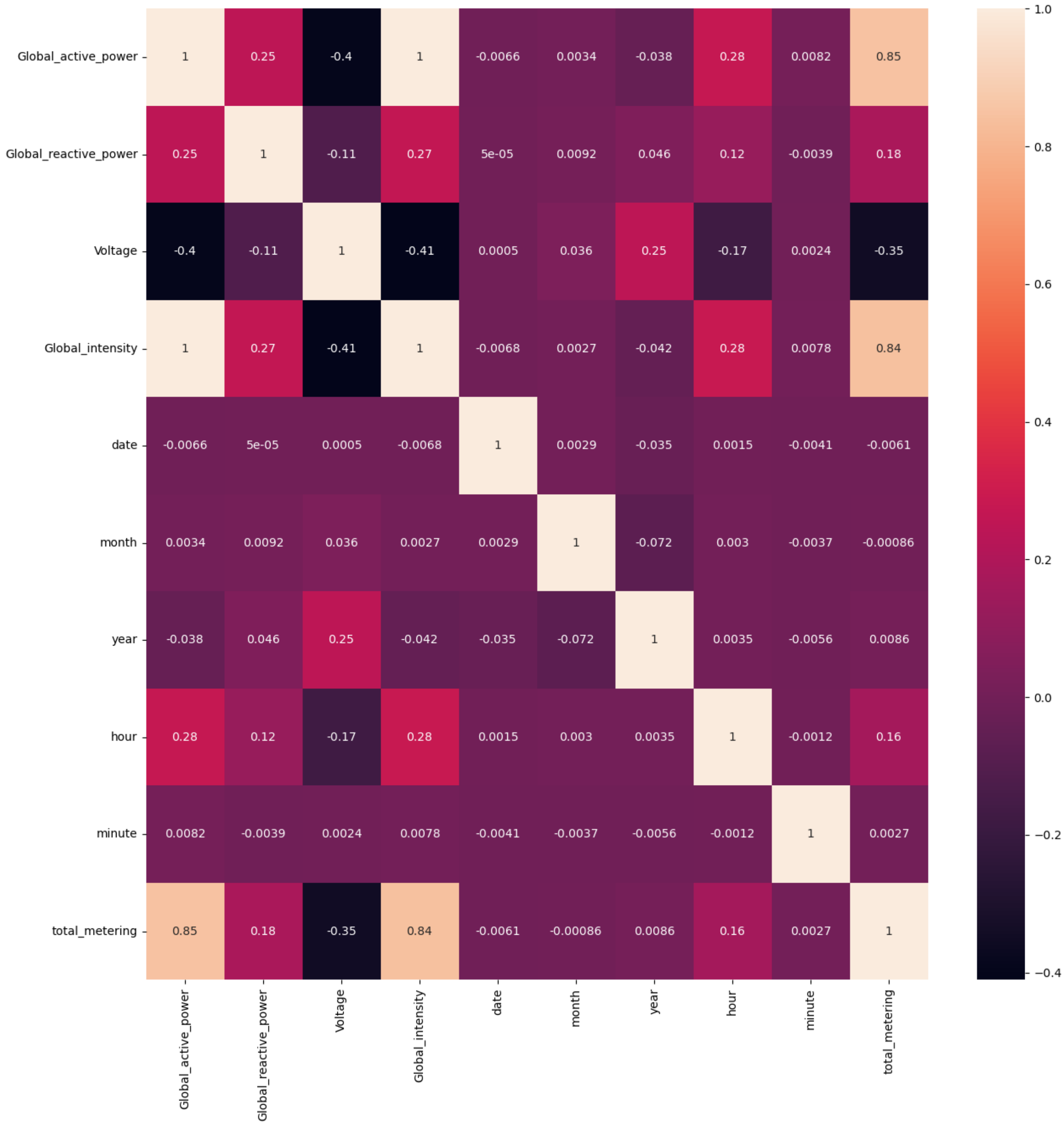
| | Global_active_power | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute | total_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **Global_active_power** | 1.000000 | 0.246635 | -0.398850 | 0.998909 | -0.006595 | 0.003398 | -0.037975 | 0.277497 | 0.008199 | |
| **Global_reactive_power** | 0.246635 | 1.000000 | -0.114163 | 0.265623 | 0.000050 | 0.009208 | 0.045534 | 0.120002 | -0.003875 | |
| **Voltage** | -0.398850 | -0.114163 | 1.000000 | -0.410701 | 0.000498 | 0.036481 | 0.249482 | -0.173845 | 0.002426 | |
| **Global_intensity** | 0.998909 | 0.265623 | -0.410701 | 1.000000 | -0.006777 | 0.002676 | -0.042156 | 0.278240 | 0.007846 | |
| **date** | -0.006595 | 0.000050 | 0.000498 | -0.006777 | 1.000000 | 0.002891 | -0.034848 | 0.001504 | -0.004130 | |
| **month** | 0.003398 | 0.009208 | 0.036481 | 0.002676 | 0.002891 | 1.000000 | -0.072357 | 0.002984 | -0.003686 | |
| **year** | -0.037975 | 0.045534 | 0.249482 | -0.042156 | -0.034848 | -0.072357 | 1.000000 | 0.003483 | -0.005615 | |
| **hour** | 0.277497 | 0.120002 | -0.173845 | 0.278240 | 0.001504 | 0.002984 | 0.003483 | 1.000000 | -0.001227 | |
| **minute** | 0.008199 | -0.003875 | 0.002426 | 0.007846 | -0.004130 | -0.003686 | -0.005615 | -0.001227 | 1.000000 | |
| **total_metering** | 0.847428 | 0.182504 | -0.345242 | 0.844560 | -0.006065 | -0.000856 | 0.008649 | 0.164001 | 0.002703 | |

In [45]:

```
plt.figure(figsize=(15,15))
sns.heatmap(data = df2.corr(),annot=True)
```

Out[45]:

```
<AxesSubplot:>
```




| | Global_active_power | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute | total_metering |
|---|---|---|---|---|---|---|---|---|---|---|
| Global_active_power | 1 | 0.25 | -0.4 | 1 | -0.0066 | 0.0034 | -0.038 | 0.28 | 0.0082 | 0.85 |
| Global_reactive_power | 0.25 | 1 | -0.11 | 0.27 | 5e-05 | 0.0092 | 0.046 | 0.12 | -0.0039 | 0.18 |
| Voltage | -0.4 | -0.11 | 1 | -0.41 | 0.0005 | 0.036 | 0.25 | -0.17 | 0.0024 | -0.35 |
| Global_intensity | 1 | 0.27 | -0.41 | 1 | -0.0068 | 0.0027 | -0.042 | 0.28 | 0.0078 | 0.84 |
| date | -0.0066 | 5e-05 | 0.0005 | -0.0068 | 1 | 0.0029 | -0.035 | 0.0015 | -0.0041 | -0.0061 |
| month | 0.0034 | 0.0092 | 0.036 | 0.0027 | 0.0029 | 1 | -0.072 | 0.003 | -0.0037 | -0.00086 |
| year | -0.038 | 0.046 | 0.25 | -0.042 | -0.035 | -0.072 | 1 | 0.0035 | -0.0056 | 0.0086 |
| hour | 0.28 | 0.12 | -0.17 | 0.28 | 0.0015 | 0.003 | 0.0035 | 1 | -0.0012 | 0.16 |
| minute | 0.0082 | -0.0039 | 0.0024 | 0.0078 | -0.0041 | -0.0037 | -0.0056 | -0.0012 | 1 | 0.0027 |
| total_metering | 0.85 | 0.18 | -0.35 | 0.84 | -0.0061 | -0.00086 | 0.0086 | 0.16 | 0.0027 | 1 |

In [46]:

```
#Handling Multicollinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [47]:

```
vif = pd.DataFrame()
vif['VIF'] = [variance_inflation_factor(df2.values,i) for i in range(len(df2.columns))]
vif['features'] = df2.columns
vif
```

Out[47]:

| | VIF | features |
|---|---|---|
| 0 | 1272.615633 | Global_active_power |
| 1 | 2.925383 | Global_reactive_power |
| 2 | 7587.479285 | Voltage |
| 3 | 1293.244501 | Global_intensity |
| 4 | 4.174021 | date |
| 5 | 4.595565 | month |
| 6 | 7682.076658 | year |
| 7 | 4.226138 | hour |
| 8 | 3.939478 | minute |
| 9 | 5.368753 | total_metering |

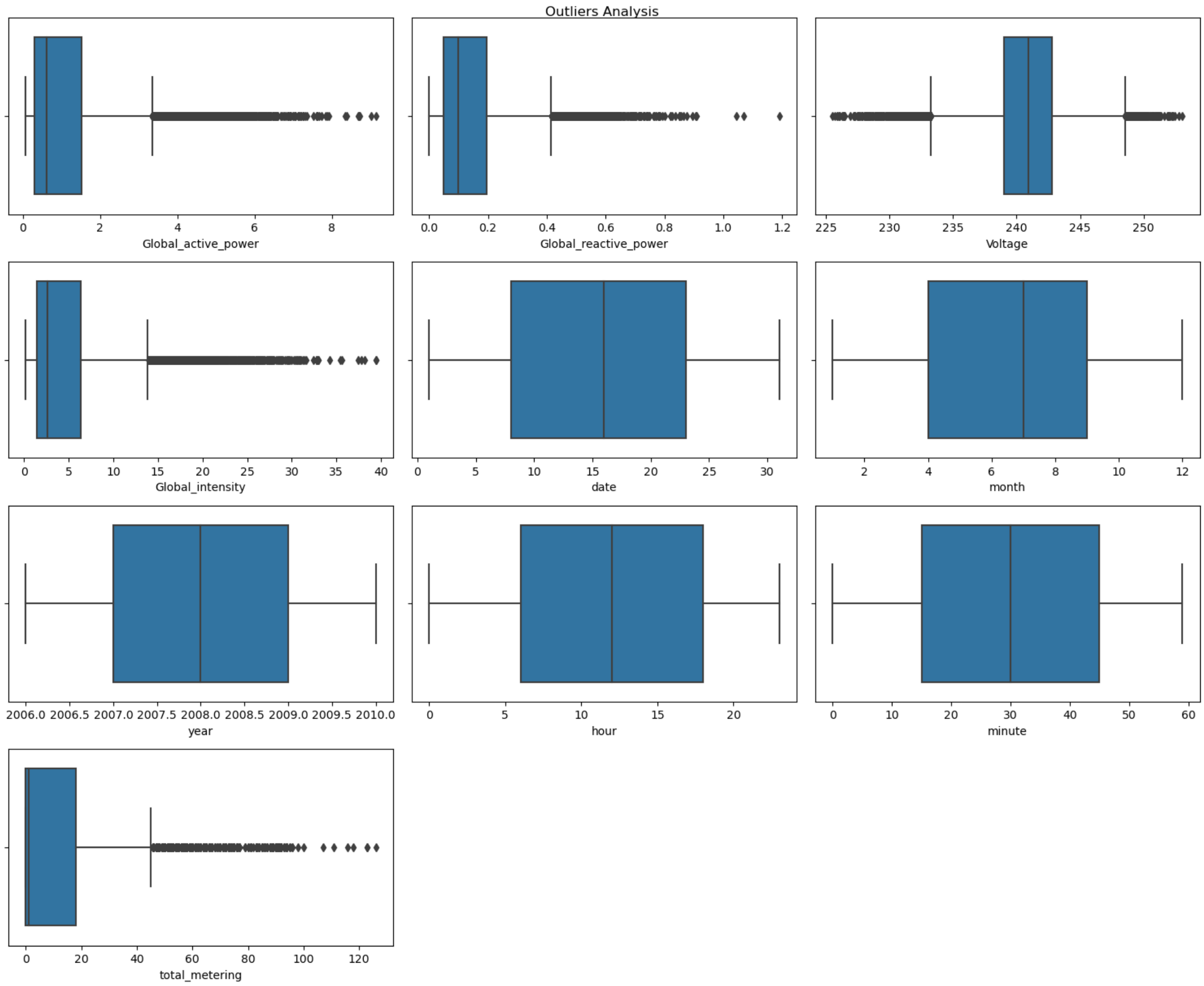
In [ ]:

```
## 'Global_active_power', 'Voltage', 'Global_intensity', and 'year' has VIF value greater than 5, that means these have multicolinearity.
```

## Checking Outliers

In [48]:

```
plt.figure(figsize = (15,15))
plt.suptitle("Outliers Analysis")
for i in range(0,len(df2.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(df2[df2.columns[i]])
    plt.tight_layout()
```




In [53]:

```
#importing winsorizer to handle outliers

from feature_engine.outliers.winsorizer import Winsorizer
```

In [52]:

```
!pip install feature-engine
```

```
Collecting feature-engine
  Downloading feature_engine-1.5.2-py2.py3-none-any.whl (290 kB)
     -------------------------------------- 290.0/290.0 kB 1.5 MB/s eta 0:00:00
Requirement already satisfied: scipy>=1.4.1 in c:\users\rajan\anaconda3\lib\site-packages (from feature-engine) (1.8.1)
Requirement already satisfied: numpy>=1.18.2 in c:\users\rajan\anaconda3\lib\site-packages (from feature-engine) (1.21.5)
Requirement already satisfied: pandas>=1.0.3 in c:\users\rajan\anaconda3\lib\site-packages (from feature-engine) (1.4.4)
Requirement already satisfied: statsmodels>=0.11.1 in c:\users\rajan\anaconda3\lib\site-packages (from feature-engine) (0.1
3.2)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\rajan\anaconda3\lib\site-packages (from feature-engine) (1.
0.2)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\rajan\anaconda3\lib\site-packages (from pandas>=1.0.3->fe
ature-engine) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\rajan\anaconda3\lib\site-packages (from pandas>=1.0.3->feature-engi
ne) (2022.1)
Requirement already satisfied: joblib>=0.11 in c:\users\rajan\anaconda3\lib\site-packages (from scikit-learn>=1.0.0->featur
e-engine) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\rajan\anaconda3\lib\site-packages (from scikit-learn>=1.0.0
->feature-engine) (2.2.0)
Requirement already satisfied: patsy>=0.5.2 in c:\users\rajan\anaconda3\lib\site-packages (from statsmodels>=0.11.1->featur
e-engine) (0.5.2)
Requirement already satisfied: packaging>=21.3 in c:\users\rajan\anaconda3\lib\site-packages (from statsmodels>=0.11.1->fea
ture-engine) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\rajan\anaconda3\lib\site-packages (from packaging>=21.3
->statsmodels>=0.11.1->feature-engine) (3.0.9)
Requirement already satisfied: six in c:\users\rajan\anaconda3\lib\site-packages (from patsy>=0.5.2->statsmodels>=0.11.1->f
eature-engine) (1.16.0)
Installing collected packages: feature-engine
Successfully installed feature-engine-1.5.2
```

In [59]:

```
 winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries or g
                      tail = 'both', # cap left, right or both tails
                      fold = 1.5, # 1.5 times of iqr
                      variables = ['Global_active_power'])

# capping_methods = 'iqr' - 25th quantile & 75th quantile
df2['Global_active_power'] = winsorizer.fit_transform(df2[['Global_active_power']])
```

In [58]:

```
 winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries or g
                      tail = 'both', # cap left, right or both tails
                      fold = 1.5, # 1.5 times of iqr
                      variables = ['Global_reactive_power'])

# capping_methods = 'iqr' - 25th quantile & 75th quantile
df2['Global_reactive_power'] = winsorizer.fit_transform(df2[['Global_reactive_power']])
```

In [61]:

```
 winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries or g
                      tail = 'both', # cap left, right or both tails
                      fold = 1.5, # 1.5 times of iqr
                      variables = ['Voltage'])

# capping_methods = 'iqr' - 25th quantile & 75th quantile
df2['Voltage'] = winsorizer.fit_transform(df2[['Voltage']])
```

In [62]:

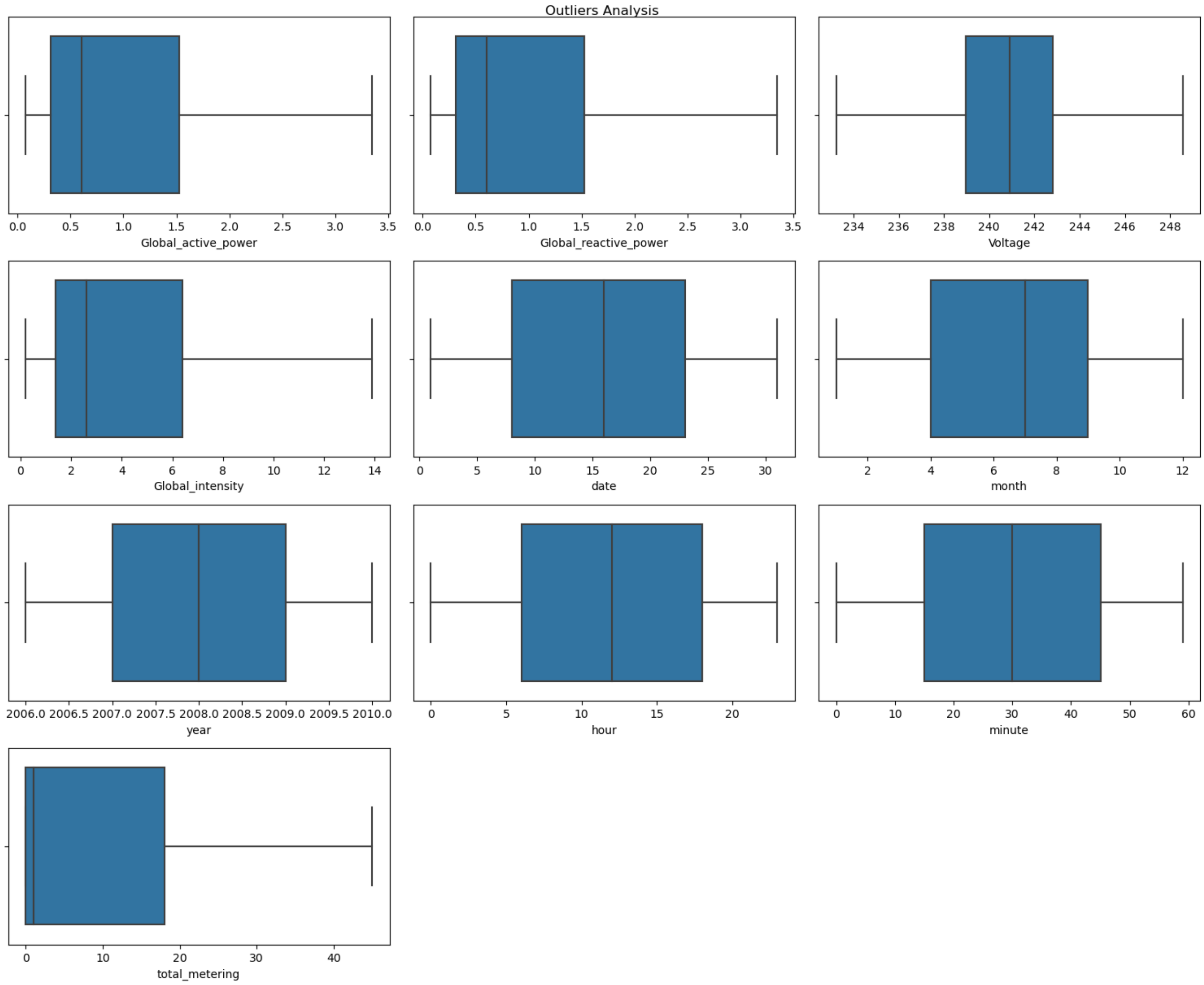
```
 winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries or g
                      tail = 'both', # cap left, right or both tails
                      fold = 1.5, # 1.5 times of iqr
                      variables = ['Global_intensity'])

# capping_methods = 'iqr' - 25th quantile & 75th quantile
df2['Global_intensity'] = winsorizer.fit_transform(df2[['Global_intensity']])
```

In [63]:

```
 winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries or g
                      tail = 'both', # cap left, right or both tails
                      fold = 1.5, # 1.5 times of iqr
                      variables = ['total_metering'])

# capping_methods = 'iqr' - 25th quantile & 75th quantile
df2['total_metering'] = winsorizer.fit_transform(df2[['total_metering']])
```

In [64]:

```
#Now again checking the outliers
plt.figure(figsize = (15,15))
plt.suptitle("Outliers Analysis")
for i in range(0,len(df2.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(df2[df2.columns[i]])
    plt.tight_layout()
```




In [65]:

```
#Save the clean data
df2.to_csv("Power_consumption_data_cleaned.csv")
```

In [68]:

```
#Save the data into pymongo Mongodb
import pymongo
```

In [67]:

```
!pip install pymongo
```

```
Collecting pymongo
  Downloading pymongo-4.3.3-cp39-cp39-win_amd64.whl (382 kB)
     -------------------------------------- 382.5/382.5 kB 2.0 MB/s eta 0:00:00
Collecting dnspython<3.0.0,>=1.16.0
  Using cached dnspython-2.2.1-py3-none-any.whl (269 kB)
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.2.1 pymongo-4.3.3
```

In [69]:

```
client = pymongo.MongoClient("mongodb+srv://root:root@rajancluster0.ub5nt.mongodb.net/?retryWrites=true&w=majority")
db = client.test
```

In [70]:

```
database = client["power_consumption"] #creating a database at mongodb client
```

In [71]:

```
collection = database["power_consumption_data"]
```

In [72]:

```
#need to change this dataframe into dict or json format to store it into mongodb
df2_dict= df2.to_dict(orient= 'records')
```

In [73]:

```
#insert the data in mongodb
collection.insert_many(df2_dict)
```

Out[73]:

```
<pymongo.results.InsertManyResult at 0x1be213697c0>
```

In [74]:

```
#loading the data from mongodb
# fetching data from the collection in mongodb{using find will return all the occurences in the collection}
data = pd.DataFrame(list(collection.find()))
```

In [75]:

```
data
```

Out[75]:

| | _id | Global_active_power | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute | total_metering |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63af1365c0e33e5d26de066f | 1.600 | 1.600 | 243.170 | 6.6 | 24 | 8 | 2009 | 7 | 11 | 20.0 |
| 1 | 63af1365c0e33e5d26de0670 | 0.238 | 0.238 | 242.520 | 1.0 | 24 | 9 | 2007 | 22 | 32 | 0.0 |
| 2 | 63af1365c0e33e5d26de0671 | 3.347 | 3.347 | 233.660 | 13.9 | 4 | 3 | 2007 | 19 | 49 | 38.0 |
| 3 | 63af1365c0e33e5d26de0672 | 1.564 | 1.564 | 241.730 | 6.4 | 11 | 12 | 2008 | 22 | 17 | 19.0 |
| 4 | 63af1365c0e33e5d26de0673 | 1.076 | 1.076 | 244.440 | 4.8 | 15 | 10 | 2010 | 23 | 35 | 12.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 39995 | 63af1365c0e33e5d26dea2aa | 0.262 | 0.262 | 244.400 | 1.0 | 4 | 1 | 2010 | 6 | 2 | 1.0 |
| 39996 | 63af1365c0e33e5d26dea2ab | 1.606 | 1.606 | 233.235 | 6.8 | 6 | 3 | 2008 | 7 | 11 | 17.0 |
| 39997 | 63af1365c0e33e5d26dea2ac | 0.240 | 0.240 | 241.180 | 1.0 | 16 | 7 | 2009 | 13 | 43 | 2.0 |
| 39998 | 63af1365c0e33e5d26dea2ad | 1.702 | 1.702 | 238.490 | 7.2 | 14 | 5 | 2008 | 6 | 44 | 19.0 |
| 39999 | 63af1365c0e33e5d26dea2ae | 1.600 | 1.600 | 241.670 | 6.6 | 28 | 12 | 2006 | 20 | 24 | 0.0 |

40000 rows × 11 columns

In [76]:

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

In [77]:

```
data.head()
```

Out[77]:

| | Global_active_power | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute | total_metering |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1.600 | 1.600 | 243.17 | 6.6 | 24 | 8 | 2009 | 7 | 11 | 20.0 |
| 1 | 0.238 | 0.238 | 242.52 | 1.0 | 24 | 9 | 2007 | 22 | 32 | 0.0 |
| 2 | 3.347 | 3.347 | 233.66 | 13.9 | 4 | 3 | 2007 | 19 | 49 | 38.0 |
| 3 | 1.564 | 1.564 | 241.73 | 6.4 | 11 | 12 | 2008 | 22 | 17 | 19.0 |
| 4 | 1.076 | 1.076 | 244.44 | 4.8 | 15 | 10 | 2010 | 23 | 35 | 12.0 |

In [80]:

```
#Now splitting the data into dependent and independent features
X = data.drop(["total_metering","Global_active_power"], axis = 1)
```

In [81]:

```
X.head()
```

Out[81]:

| | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute |
|---|---|---|---|---|---|---|---|---|
| **0** | 1.600 | 243.17 | 6.6 | 24 | 8 | 2009 | 7 | 11 |
| **1** | 0.238 | 242.52 | 1.0 | 24 | 9 | 2007 | 22 | 32 |
| **2** | 3.347 | 233.66 | 13.9 | 4 | 3 | 2007 | 19 | 49 |
| **3** | 1.564 | 241.73 | 6.4 | 11 | 12 | 2008 | 22 | 17 |
| **4** | 1.076 | 244.44 | 4.8 | 15 | 10 | 2010 | 23 | 35 |

In [82]:

```
y= data['total_metering']
```

In [84]:

```
y.head()
```

Out[84]:

```
0    20.0
1     0.0
2    38.0
3    19.0
4    12.0
Name: total_metering, dtype: float64
```

In [85]:

```
#spliting the data into train and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=10)
```

In [86]:

```
X_train.head()
```

Out[86]:

| | Global_reactive_power | Voltage | Global_intensity | date | month | year | hour | minute |
|---|---|---|---|---|---|---|---|---|
| **4512** | 1.336 | 241.67 | 5.4 | 21 | 2 | 2007 | 15 | 25 |
| **39286** | 1.562 | 242.18 | 6.6 | 3 | 6 | 2009 | 16 | 3 |
| **13026** | 0.312 | 242.90 | 1.4 | 25 | 2 | 2007 | 13 | 35 |
| **5754** | 2.040 | 241.82 | 8.4 | 21 | 6 | 2010 | 22 | 49 |
| **39120** | 0.340 | 243.64 | 1.4 | 19 | 2 | 2008 | 22 | 48 |

In [88]:

```
X_train.shape
```

Out[88]:

```
(30000, 8)
```

In [89]:

```
X_test.shape
```

Out[89]:

```
(10000, 8)
```

In [90]:

```
#Standardising the data

from sklearn.preprocessing import StandardScaler
```

In [91]:

```
scaler = StandardScaler()
```

In [92]:

```
X_train = scaler.fit_transform(X_train)
# Using fit transform to standardise train data
```

In [93]:

```
X_train
```

Out[93]:

```
array([[ 0.31445042,  0.27150791,  0.25539296, ..., -1.27371682,
         0.50351138, -0.27305053],
       [ 0.56152556,  0.43587967,  0.57179316, ...,  0.49952113,
         0.64848183, -1.54239105],
       [-0.80504049,  0.66793392, -0.79927435, ..., -1.27371682,
         0.21357047,  0.30392243],
       ...,
       [-0.77880242,  0.26506195, -0.79927435, ..., -0.38709785,
        -1.38110452,  0.7655008 ],
       [-0.88156819,  0.07812936, -0.90474108, ...,  1.38614011,
        -0.94619316,  0.53471162],
       [ 2.51298188, -2.44707209,  2.496561  , ..., -1.27371682,
         1.08339319, -0.15765594]])
```

In [94]:

```
# using only transform to avoid data leakage
X_test = scaler.transform(X_test)
X_test
```

Out[94]:

```
array([[-0.60388197, -0.10558025, -0.64107425, ...,  1.38614011,
        -1.67104542,  1.40017106],
       [-0.31089021,  0.07812936, -0.32467406, ...,  1.38614011,
        -1.38110452,  0.99628999],
       [ 0.6927159 , -0.25383713,  0.67725989, ..., -0.38709785,
        -0.22134089, -0.56153701],
       ...,
       [-0.74600484,  0.54223786, -0.74654098, ...,  1.38614011,
        -1.23613406,  0.65010621],
       [ 1.25027485, -1.30130423,  1.25732691, ...,  0.49952113,
         1.22836365,  1.57326295],
       [ 0.17670056, -0.52779007,  0.14992623, ...,  0.49952113,
         1.51830455, -0.85002349]])
```

In [95]:

```
#Pickling
## pickling{saving some preprocessing objects/models}
import pickle

# Writing different model files to file
with open('sandardScalar.sav', 'wb') as f:
    pickle.dump(scaler,f)

## here we have pickled the standard scaler object
```

# Linear Regression

In [96]:

```
from sklearn.linear_model import LinearRegression
```

In [97]:

```
## Creating linear regression model
linear_reg = LinearRegression()
```

In [98]:

```
## training the model
linear_reg.fit(X_train, y_train)
```

Out[98]:

```
LinearRegression()
```

In [100]:

```
## coefficients and intercept of the best fit hyperplane
print('1. the coefficients of independent features: {}'.format(linear_reg.coef_))
print('2. intercept of the best fit hyperplane: {}'.format(linear_reg.intercept_))
```

```
1. the coefficients of independent features: [ 25.0176861   -0.75772829 -15.65702081   0.03253776   0.02847474
   0.59926564  -0.90306825  -0.0312098 ]
2. intercept of the best fit hyperplane: 8.486766666666647
```

In [101]:

```
## predicting the test data using the model
y_pred = linear_reg.predict(X_test)
y_pred
```

Out[101]:

```
array([ 5.7329791 ,  7.79718856, 15.38871174, ...,  3.03943   ,
       20.18146759,  9.95497994])
```

## Cost functions

In [102]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

In [104]:

```
print("Mean squared error is {}".format(round(mean_squared_error(y_test, y_pred),2)))
print("Mean squared error is {}".format(round(mean_absolute_error(y_test, y_pred),2)))
print("Root Mean squared error is {}".format(round(np.sqrt(mean_squared_error(y_test, y_pred)),2)))
```

```
Mean squared error is 37.59
Mean squared error is 4.16
Root Mean squared error is 6.13
```

## Validating the model using assumptions of Linear regression

In [105]:

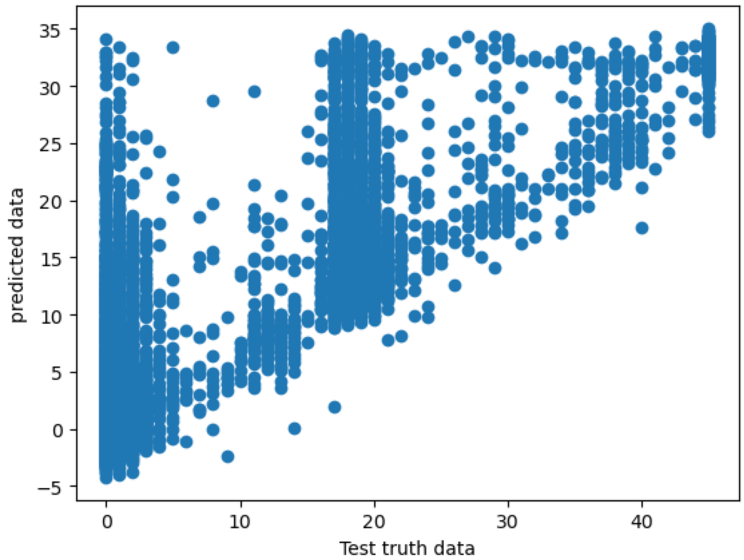
```
#Linear relationship

#Test truth data and predicted data should follow linear relationship {this is an indiaction of a good model}
```

In [106]:

```
plt.scatter(x=y_test, y = y_pred)
plt.xlabel("Test truth data")
plt.ylabel("predicted data")
```

Out[106]:

```
Text(0, 0.5, 'predicted data')
```




In [107]:

```
#Residual Distribution
#Residuals should follow normal distribution
```

In [108]:

```
residual_linear= y_test-y_pred
residual_linear.head()
```

Out[108]:

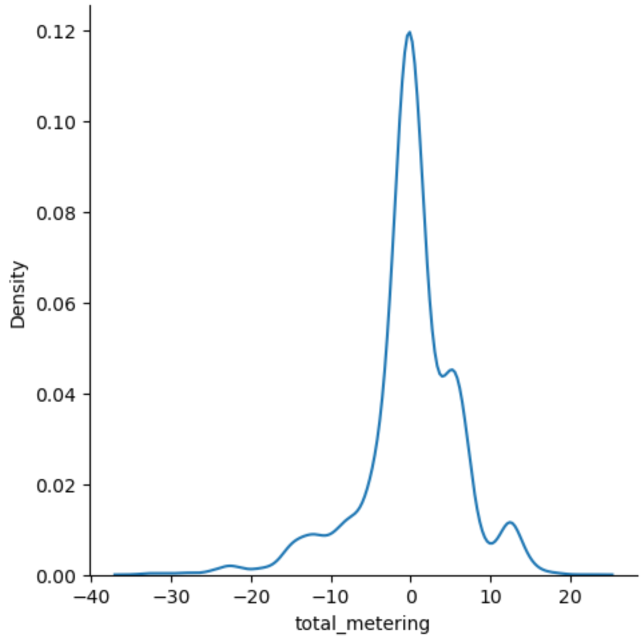
```
26915   -5.732979
37746   -6.797189
15211    2.611288
8556    -1.877028
17307   -0.438493
Name: total_metering, dtype: float64
```

In [109]:

```
sns.displot(x=residual_linear, kind='kde')
#uniform distribution
```

Out[109]:

```
<seaborn.axisgrid.FacetGrid at 0x1be24f4c5e0>
```

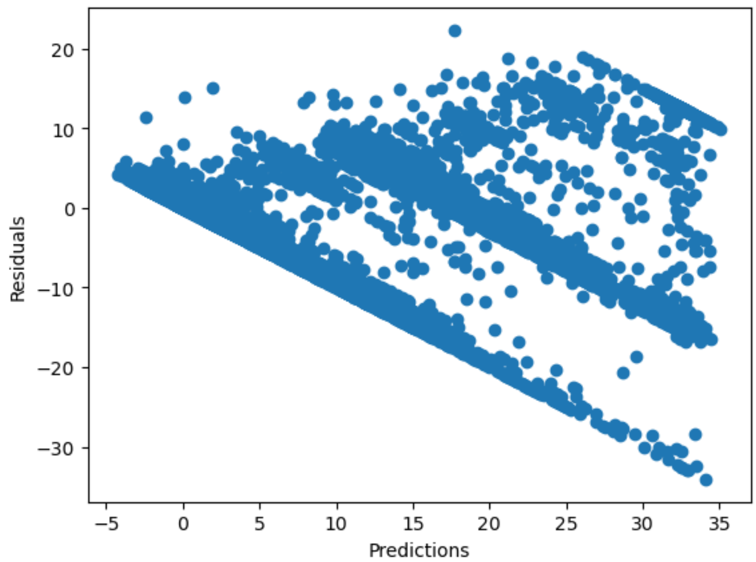



In [110]:

```
#Residuals vs Predictions follow a uniform dstribution.
plt.scatter(x=y_pred, y=residual_linear)
plt.xlabel('Predictions')
plt.ylabel('Residuals')
```

Out[110]:

```
Text(0, 0.5, 'Residuals')
```



In [111]:

```
#Accuracy of the model with train data and with test data
linear_reg.score(X_train,y_train)
```

Out[111]:

```
0.7044669843350699
```

In [112]:

```
linear_reg.score(X_test,y_test)
```

Out[112]:

```
0.7118298627051172
```

## Performance Matrics

In [113]:

```
#R Square and Adjusted R Square values
from sklearn.metrics import r2_score
```

In [114]:

```
r2_score_lr= r2_score(y_test, y_pred)
print("Linear Regression model has {} % accuracy".format(round(r2_score_lr*100,3)))

adjr2_score_lr=1-((1-r2_score_lr)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print("Adjusted R square accuracy is {} percent".format(round(adjr2_score_lr*100,2)))
```

```
Linear Regression model has 71.183 % accuracy
Adjusted R square accuracy is 71.16 percent
```

## Ridge Regression

In [115]:

```
from sklearn.linear_model import Ridge
```

In [116]:

```
## creating ridge regrssion model

ridge_reg= Ridge()
```

In [117]:

```
## training the model

ridge_reg.fit(X_train, y_train)
```

Out[117]:

```
Ridge()
```

In [118]:

```
# Printing co-efficients and intercept of best fit hyperplane

print("coefficient of independent feature is {}".format(ridge_reg.coef_))
print("Intercept of best fit hyperplane is {}".format(ridge_reg.intercept_))
```

```
coefficient of independent feature is [ 24.45310071  -0.75038259 -15.08942304   0.03265248   0.02866776
   0.59959922  -0.9038171   -0.03101933]
Intercept of best fit hyperplane is 8.486766666666647
```

In [120]:

```
# predicting test data

y_pred_r = ridge_reg.predict(X_test)
y_pred_r
```

Out[120]:

```
array([ 5.71100448,  7.79078928, 15.37998875, ...,  3.04250823,
       20.17912124,  9.93540741])
```

In [121]:

```
print("Mean squared error is {}".format(round(mean_squared_error(y_test, y_pred_r),2)))
print("Mean absolute error is {}".format(round(mean_absolute_error(y_test, y_pred_r),2)))
print("Root Mean squared error is {}".format(round(np.sqrt(mean_squared_error(y_test, y_pred_r)),2)))
```

```
Mean squared error is 37.59
Mean absolute error is 4.16
Root Mean squared error is 6.13
```

In [122]:

```
ridge_reg.score(X_train,y_train)
```

Out[122]:

```
0.7044611502884796
```

In [123]:

```
ridge_reg.score(X_test,y_test)
```

Out[123]:

```
0.7118269280575819
```

## Validating model using performace matrics

In [125]:

```
#R Square and Adjusted R Square values
ridge_r2_score=r2_score(y_test, y_pred_r)
print("Ridge regression model has {} % accuracy".format(round(ridge_r2_score*100,3)))

ridge_adjr2_score=1-((1-ridge_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print("Adjusted R square accuracy is {} percent".format(round(ridge_adjr2_score*100,2)))
```

```
Ridge regression model has 71.183 % accuracy
Adjusted R square accuracy is 71.16 percent
```

## Lasso Regression

In [126]:

```
from sklearn.linear_model import Lasso
```

In [127]:

```
# creating Lasso regression model

lasso_reg = Lasso()
lasso_reg
```

Out[127]:

```
Lasso()
```

In [128]:

```
# training the model
lasso_reg.fit(X_train, y_train)
```

Out[128]:

```
Lasso()
```

In [129]:

```
# Printing co-efficients and intercept of best fit hyperplane

print("coefficient of independent feature is {}".format(lasso_reg.coef_))
print("Intercept of best fit hyperplane is {}".format(lasso_reg.intercept_))
```

```
coefficient of independent feature is [ 8.39763511 -0.          0.         -0.         -0.          0.
 -0.         -0.        ]
Intercept of best fit hyperplane is 8.486766666666664
```

In [130]:

```
# predicting the test data
y_pred_lasso = lasso_reg.predict(X_test)
y_pred_lasso
```

Out[130]:

```
array([ 3.41558626,  5.87602416, 14.30394205, ...,  2.22209026,
       18.98611866,  9.97063351])
```

In [132]:

```
print("Mean squared error is {}".format(round(mean_squared_error(y_test, y_pred_lasso),2)))
print("Mean absolute error is {}".format(round(mean_absolute_error(y_test, y_pred_lasso),2)))
print("Root Mean squared error is {}".format(round(np.sqrt(mean_squared_error(y_test, y_pred_lasso)),2)))
```

```
Mean squared error is 40.45
Mean absolute error is 4.38
Root Mean squared error is 6.36
```

In [133]:

```
lasso_reg.score(X_train,y_train)
```

Out[133]:

```
0.6829917229555713
```

In [134]:

```
lasso_reg.score(X_test,y_test)
```

Out[134]:

```
0.6899163895457995
```

## Validating model using performance matrics

In [135]:

```
#R Square and Adjusted R Square values
```

In [136]:

```
lasso_r2_score=r2_score(y_test, y_pred_lasso)
print("Lasso regression model has {} % accuracy".format(round(lasso_r2_score*100,3)))

lasso_adjr2_score=1-((1-lasso_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print("Adjusted R square accuracy is {} percent".format(round(lasso_adjr2_score*100,2)))
```

```
Lasso regression model has 68.992 % accuracy
Adjusted R square accuracy is 68.97 percent
```

## Elastic-Net Regression

In [137]:

```
from sklearn.linear_model import ElasticNet
```

In [138]:

```
# creating Elastic-Net regression model
elastic_reg=ElasticNet()
elastic_reg
```

Out[138]:

```
ElasticNet()
```

In [139]:

```
## training the model

elastic_reg.fit(X_train, y_train)
```

Out[139]:

```
ElasticNet()
```

In [141]:

```
# Printing co-efficients and intercept of best fit hyperplane
print("Co-efficients of independent features is {}".format(elastic_reg.coef_))
print("Intercept of best fit hyper plane is {}".format(elastic_reg.intercept_))
```

```
Co-efficients of independent features is [ 3.52941083 -0.45885996  3.42710755 -0.         -0.          0.00709137
 -0.          0.        ]
Intercept of best fit hyper plane is 8.486766666666659
```

In [142]:

```
elastic_y_pred=elastic_reg.predict(X_test)
elastic_y_pred
```

Out[142]:

```
array([ 4.21666489,  6.25079368, 13.36641879, ...,  3.05635127,
       17.80915349,  9.86995288])
```

In [143]:

```
print("Mean squared error is '{}'".format(round(mean_squared_error(y_test, elastic_y_pred),2)))
print("Mean absolute error is '{}'".format(round(mean_absolute_error(y_test, elastic_y_pred),2)))
print("Root Mean squared error is '{}'".format(round(np.sqrt(mean_squared_error(y_test, elastic_y_pred)),2)))
```

```
Mean squared error is '45.37'
Mean absolute error is '4.95'
Root Mean squared error is '6.74'
```

In [145]:

```
elastic_reg.score(X_train,y_train)
```

Out[145]:

```
0.6488961021055253
```

In [146]:

```
elastic_reg.score(X_test,y_test)
```

Out[146]:

```
0.6522279435849052
```

In [147]:

```
#Validating model using performance matrices
#R Square and Adjusted R Square values
```

In [148]:

```
elastic_reg_r2_score=r2_score(y_test, elastic_y_pred)
print("Elastic-Net regression model has {} % accuracy".format(round(elastic_reg_r2_score*100,3)))

elastic_reg_adj_r2_score=1-((1-elastic_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print("Adjusted R square accuracy is {} percent".format(round(elastic_reg_adj_r2_score*100,2)))
```

```
Elastic-Net regression model has 65.223 % accuracy
Adjusted R square accuracy is 65.19 percent
```

# SVR

In [149]:

```
from sklearn.svm import SVR
```

In [150]:

```
# creating SVR  model
svr = SVR()
svr
```

Out[150]:

```
SVR()
```

In [151]:

```
## training the model

svr.fit(X_train, y_train)
```

Out[151]:

```
SVR()
```

In [152]:

```
## predicting the dependent feature value w.r.t. test data
svr_y_pred= svr.predict(X_test)
svr_y_pred
```

Out[152]:

```
array([ 2.76872985,  6.33201606, 20.90162389, ...,  1.23319072,
       17.61534662,  7.58828703])
```

In [153]:

```
print("Mean squared error is '{}'".format(round(mean_squared_error(y_test, svr_y_pred),2)))
print("Mean absolute error is '{}'".format(round(mean_absolute_error(y_test, svr_y_pred),2)))
print("Root Mean squared error is '{}'".format(round(np.sqrt(mean_squared_error(y_test, svr_y_pred)),2)))
```

```
Mean squared error is '30.76'
Mean absolute error is '3.12'
Root Mean squared error is '5.55'
```

In [154]:

```
svr.score(X_train, y_train)
```

Out[154]:

```
0.7747902423493528
```

In [156]:

```
svr.score(X_test, y_test)
```

Out[156]:

```
0.7641801391093346
```

## Performance Matrix SVR

In [157]:

```
#R Square and Adjusted R Square values
```

In [158]:

```
svr_r2_score=r2_score(y_test, svr_y_pred)
print("SVR model has {} % accuracy".format(round(svr_r2_score*100,3)))

svr_adj_r2_score=1-((1-svr_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print("Adjusted R square accuracy is {} percent".format(round(svr_adj_r2_score*100,2)))
```

```
SVR model has 76.418 % accuracy
Adjusted R square accuracy is 76.4 percent
```

## Apply hyperparameter tuning

In [159]:

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

In [160]:

```
model_params = {
    'Ridge Regression': {
        'model': Ridge(),
        'params' : {
            'alpha': [1,5,10,20]
        }
    },
    'Lasso Regression': {
        'model': Lasso(),
        'params' : {
            'alpha': [1,5,10,20]
        }
    },
    'Elastic-Net Regression' : {
        'model': ElasticNet(),
        'params': {
            'alpha': [1,5,10,20],
            'l1_ratio':[0.5,1,1.5,2]
        }
    },
    'SVR':{
        'model': SVR(),
        'params':{
            'kernel': ['linear', 'poly', 'sigmoid', 'rbf'],
            'C':[1,5,10,20]
        }
    }
}
```

In [161]:

```
model_params.items()
```

Out[161]:

```
dict_items([('Ridge Regression', {'model': Ridge(), 'params': {'alpha': [1, 5, 10, 20]}}), ('Lasso Regression', {'model': L
asso(), 'params': {'alpha': [1, 5, 10, 20]}}), ('Elastic-Net Regression', {'model': ElasticNet(), 'params': {'alpha': [1,
5, 10, 20], 'l1_ratio': [0.5, 1, 1.5, 2]}}), ('SVR', {'model': SVR(), 'params': {'kernel': ['linear', 'poly', 'sigmoid', 'r
bf'], 'C': [1, 5, 10, 20]}})])
```

In [162]:

```
##scaling the independent features before fitting it inside grid object{to simplify the calculation part}
X1= scaler.fit_transform(X)
```

In [*]:

```
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, n_jobs=-1,return_train_score=False)
    clf.fit(X1, y)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

In [*]:

```
#Conclusion
#The SVR model with 'rbf kernel' is the best model for this household power consumption data
```