Household Power Consumption Dataset

```
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 warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df = pd.read_csv(r"C:\Users\sahil\Documents\ineuron\dataset\household_power_consumption\hou
```

In [3]:

```
df.shape
```

Out[3]:

```
(2075259, 9)
```

In [4]:

```
df.head()
```

Out[4]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity
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 [5]:

```
# Selecting random 50000 sample data
```

In [6]:

```
data = df.sample(50000)
```

In [7]:

```
data.shape
```

Out[7]:

```
(50000, 9)
```

In [8]:

```
data.head()
```

Out[8]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_inten
239228	31/5/2007	20:32:00	1.312	0.000	236.290	5.
1623366	17/1/2010	01:30:00	0.244	0.000	244.290	1.
1252711	4/5/2009	15:55:00	0.404	0.204	241.870	1.
1392782	9/8/2009	22:26:00	0.448	0.288	239.460	2.
1661227	12/2/2010	08:31:00	1.426	0.000	241.990	5.

In [9]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 239228 to 1691695
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   50000 non-null  object 
 1   Time                   50000 non-null  object 
 2   Global_active_power    50000 non-null  object 
 3   Global_reactive_power  50000 non-null  object 
 4   Voltage                50000 non-null  object 
 5   Global_intensity       50000 non-null  object 
 6   Sub_metering_1         50000 non-null  object 
 7   Sub_metering_2         50000 non-null  object 
 8   Sub_metering_3         49379 non-null  float64
dtypes: float64(1), object(8)
memory usage: 3.8+ MB
```

In [10]:

```
import datetime  as dt
```

In [11]:

```
# Seperating date, month and Year
```

In [12]:

```
data['Date'] = pd.to_datetime(data['Date'])
```

In [13]:

```
data['date'] = data['Date'].dt.day
```

In [14]:

```
data['month'] = data['Date'].dt.month
```

In [15]:

```
data['year'] = data['Date'].dt.year
```

In [16]:

```
data.year.unique()
```

Out[16]:

```
array([2007, 2010, 2009, 2008, 2006], dtype=int64)
```

In [17]:

```
# Separating Hours, Minutes and seconds
```

In [18]:

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

In [19]:

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

In [20]:

```
data['Seconds'] = pd.to_datetime(data['Time'], format='%H:%M:%S').dt.second
```

In [21]:

```
# Converting data types & replacing special characters
```

In [22]:

```
data['Global_active_power'] = data['Global_active_power'].replace("?","")
```

In [23]:

```
data['Global_active_power'] = data['Global_active_power'].replace("'",np.nan)
```

In [24]:

```
data['Global_active_power'] = data['Global_active_power'].replace(" ",np.nan)
```

In [25]:

```
data['Global_active_power'] = data['Global_active_power'].replace("",np.nan)
```

In [26]:

```
data['Global_active_power'] = data['Global_active_power'].astype('float64')
```

In [27]:

```
data['Global_active_power'] = data['Global_active_power'].fillna(data['Global_active_power'
```

In [28]:

```
data['Global_active_power'].isna().sum()
```

Out[28]:

```
0
```

In [29]:

```
data['Global_reactive_power'] = data['Global_reactive_power'].replace('?',np.nan)
```

In [30]:

```
data['Global_reactive_power'] = data['Global_reactive_power'].astype(float)
```

In [31]:

```
data['Global_reactive_power'].isna().sum()
```

Out[31]:

```
621
```

In [32]:

```
data['Global_reactive_power'] = data['Global_reactive_power'].fillna(data['Global_reactive_
```

In [33]:

```
data['Global_reactive_power'].isna().sum()
```

Out[33]:

```
0
```

In [34]:

```
data['Voltage'] = data['Voltage'].replace('?',np.nan)
```

In [35]:

```
data['Voltage'] = data['Voltage'].astype(float)
```

In [36]:

```
data['Voltage'].isna().sum()
```

Out[36]:

```
621
```

In [37]:

```
data['Voltage'] = data['Voltage'].fillna(data['Voltage'].mean())
```

In [38]:

```
data['Voltage'].isna().sum()
```

Out[38]:

```
0
```

In [39]:

```
data['Global_intensity'] = data['Global_intensity'].replace('?',np.nan)
```

In [40]:

```
data['Global_intensity'] =  data['Global_intensity'].astype('float')
```

In [41]:

```
data['Global_intensity'] = data['Global_intensity'].fillna(data['Global_intensity'].mean())
```

In [42]:

```
data['Sub_metering_1'] = data['Sub_metering_1'].replace('?',np.nan)
```

In [43]:

```
data['Sub_metering_1'] = data['Sub_metering_1'].astype('float')
```

In [44]:

```
data['Sub_metering_1'] = data['Sub_metering_1'].fillna(data['Sub_metering_1'].mean())
```

In [45]:

```
data['Sub_metering_2'] = data['Sub_metering_2'].replace('?',np.nan)
```

In [46]:

```
data['Sub_metering_3'] = data['Sub_metering_3'].replace('?',np.nan)
```

In [47]:

```
data['Sub_metering_2'] = data['Sub_metering_2'].astype('float')
```

In [48]:

```
data['Sub_metering_3'] = data['Sub_metering_3'].astype('float')
```

In [49]:

```
data['Sub_metering_2'] = data['Sub_metering_2'].fillna(data['Sub_metering_2'].mean())
```

In [50]:

```
data['Sub_metering_3'] = data['Sub_metering_3'].fillna(data['Sub_metering_3'].mean())
```

In [51]:

```
data['Total_metering'] = data['Sub_metering_1']+data['Sub_metering_2']+data['Sub_metering_3
```

In [52]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 239228 to 1691695
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   50000 non-null  datetime64[ns]
 1   Time                   50000 non-null  object
 2   Global_active_power    50000 non-null  float64
 3   Global_reactive_power  50000 non-null  float64
 4   Voltage                50000 non-null  float64
 5   Global_intensity       50000 non-null  float64
 6   Sub_metering_1         50000 non-null  float64
 7   Sub_metering_2         50000 non-null  float64
 8   Sub_metering_3         50000 non-null  float64
 9   date                   50000 non-null  int64
 10  month                  50000 non-null  int64
 11  year                   50000 non-null  int64
 12  hour                   50000 non-null  int64
 13  Minutes                50000 non-null  int64
 14  Seconds                50000 non-null  int64
 15  Total_metering         50000 non-null  float64
dtypes: datetime64[ns](1), float64(8), int64(6), object(1)
memory usage: 6.5+ MB
```

In [53]:

```
new_data = data.drop(columns=['Date','Time','Sub_metering_1','Sub_metering_2','Sub_metering
```

In [54]:

```
new_data.columns
```

Out[54]:

```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'date', 'month', 'year', 'hour', 'Minutes',
       'Seconds', 'Total_metering'],
      dtype='object')
```

In [55]:

```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 239228 to 1691695
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Global_active_power    50000 non-null  float64
 1   Global_reactive_power  50000 non-null  float64
 2   Voltage                50000 non-null  float64
 3   Global_intensity       50000 non-null  float64
 4   date                   50000 non-null  int64  
 5   month                  50000 non-null  int64  
 6   year                   50000 non-null  int64  
 7   hour                   50000 non-null  int64  
 8   Minutes                50000 non-null  int64  
 9   Seconds                50000 non-null  int64  
 10  Total_metering         50000 non-null  float64
dtypes: float64(5), int64(6)
memory usage: 4.6 MB
```

In [56]:

```
new_data.describe()
```

Out[56]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	date
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.00000C
mean	1.088157	0.123146	240.849140	4.613492	15.77916C
std	1.049785	0.111551	3.237009	4.412993	8.817956
min	0.078000	0.000000	223.200000	0.200000	1.00000C
25%	0.310000	0.048000	239.030000	1.400000	8.00000C
50%	0.614000	0.102000	240.970000	2.800000	16.00000C
75%	1.514500	0.192000	242.870000	6.400000	23.00000C
max	9.994000	1.096000	252.970000	43.000000	31.00000C

In [57]:

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

In [58]:

```
new_data.columns
```

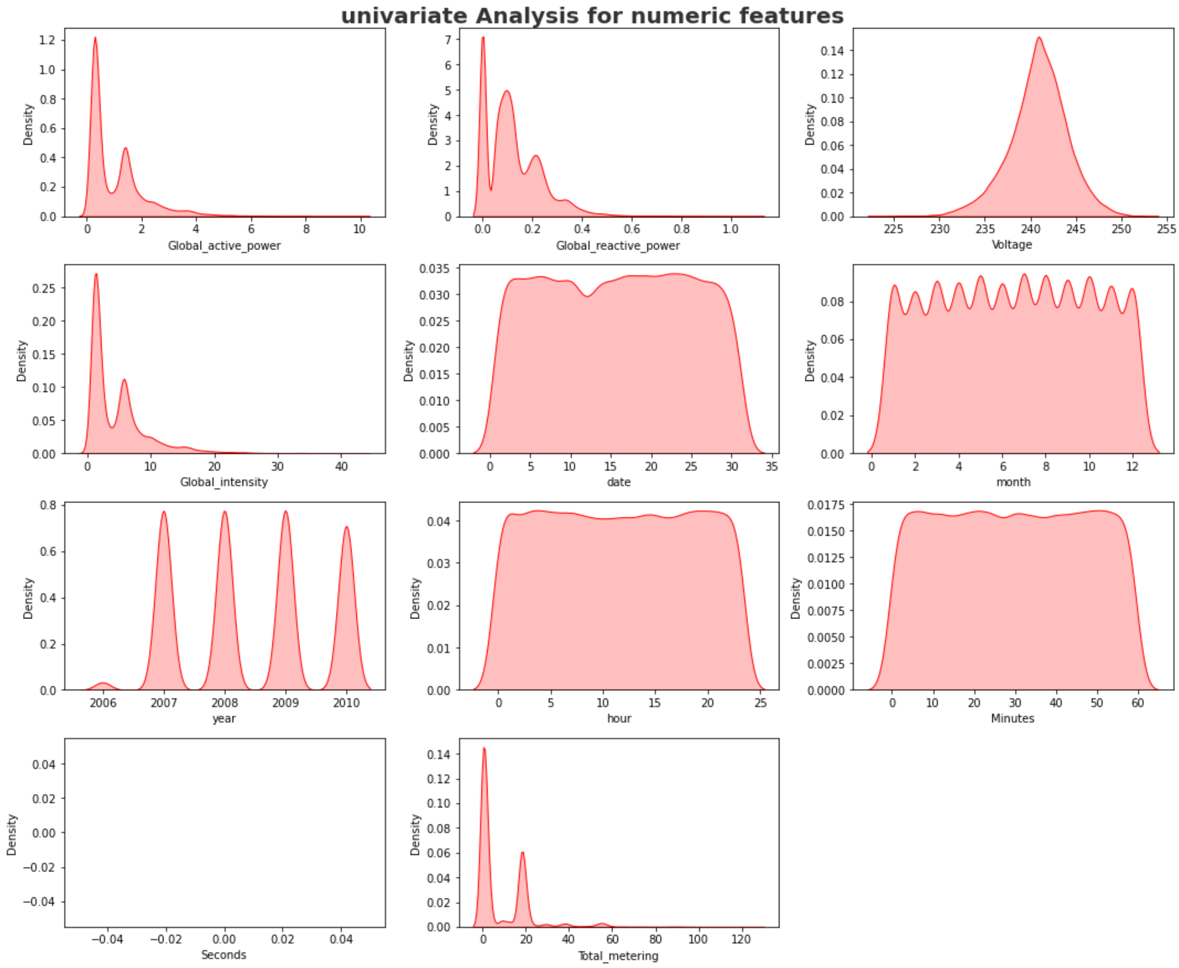
Out[58]:

```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'date', 'month', 'year', 'hour', 'Minutes',
       'Seconds', 'Total_metering'],
      dtype='object')
```

In [59]:

```
plt.figure(figsize =(15,15))
plt.suptitle('univariate Analysis for numeric features',fontsize = 20, fontweight='bold', a

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




```
# since seconds doesnot have any information we can drop it
# Voltage is showing normal distribution
```

In [60]:

```
# dropping Seconds columns as it doesnot show any variation
new_data.drop(columns=['Seconds'], axis =1, inplace = True)
```

In [61]:

```
new_data.columns
```

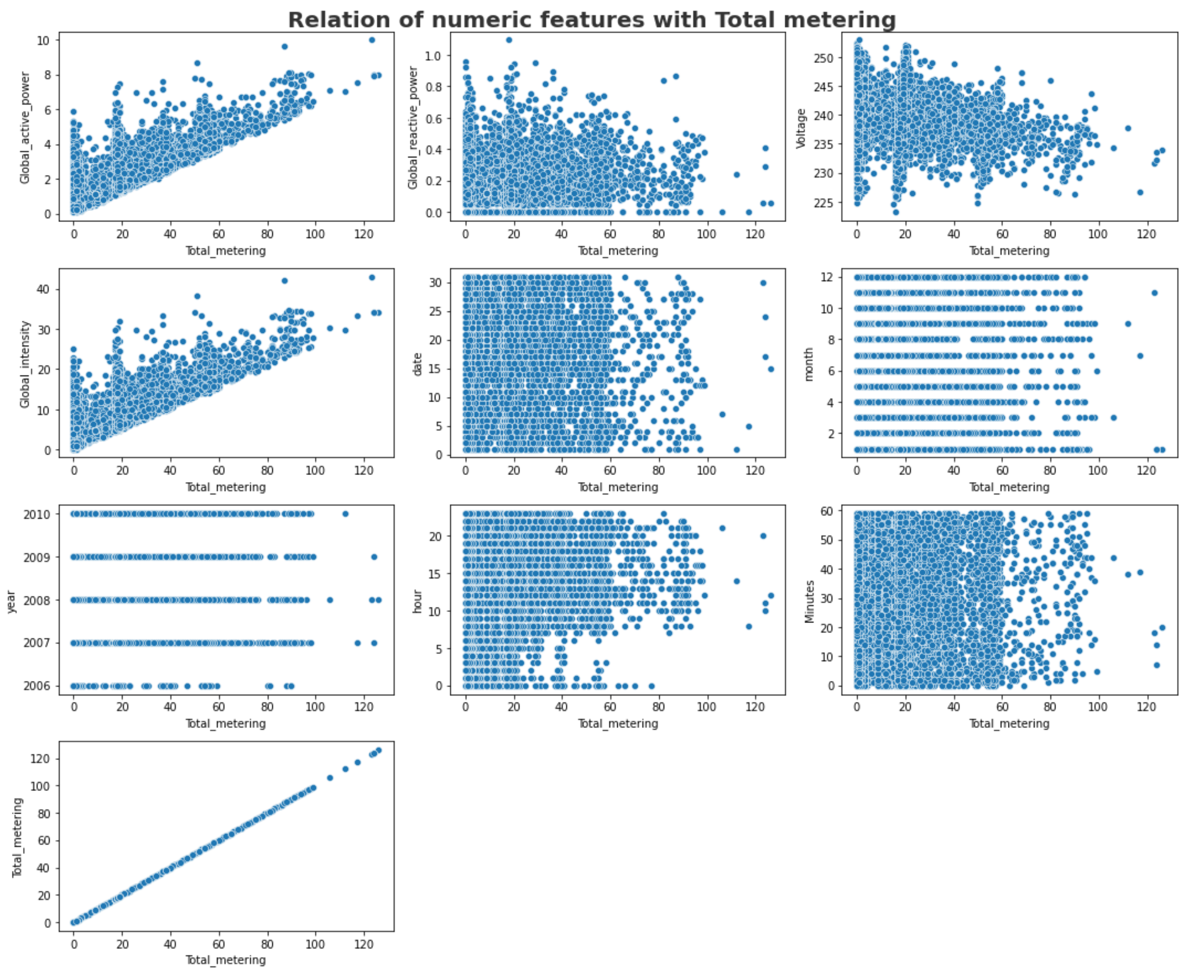
Out[61]:

```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'date', 'month', 'year', 'hour', 'Minutes',
       'Total_metering'],
      dtype='object')
```

In [62]:

```
plt.figure(figsize =(15,15))
plt.suptitle('Relation of numeric features with Total metering',fontsize = 20, fontweight='

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



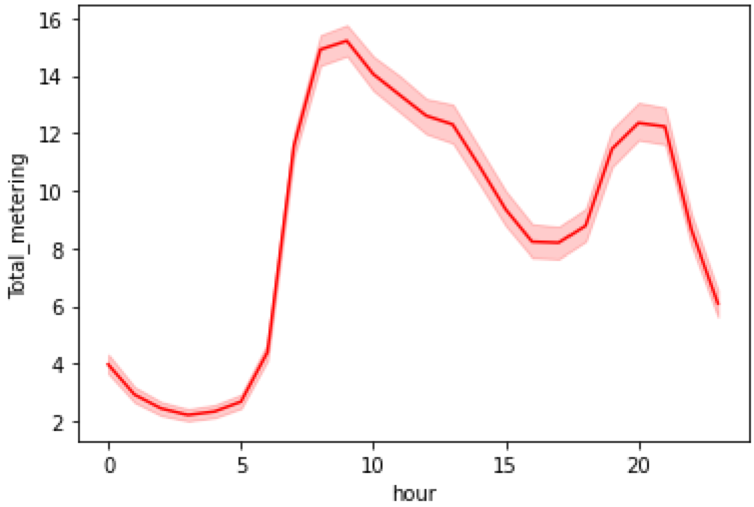

```
# Global Intensity and Global active power is highly related with Total metering
```

In [63]:

```
sns.lineplot(x="hour", y="Total_metering",
             data=new_data, color = 'red')
```

Out[63]:

<AxesSubplot:xlabel='hour', ylabel='Total_metering'>



In [64]:

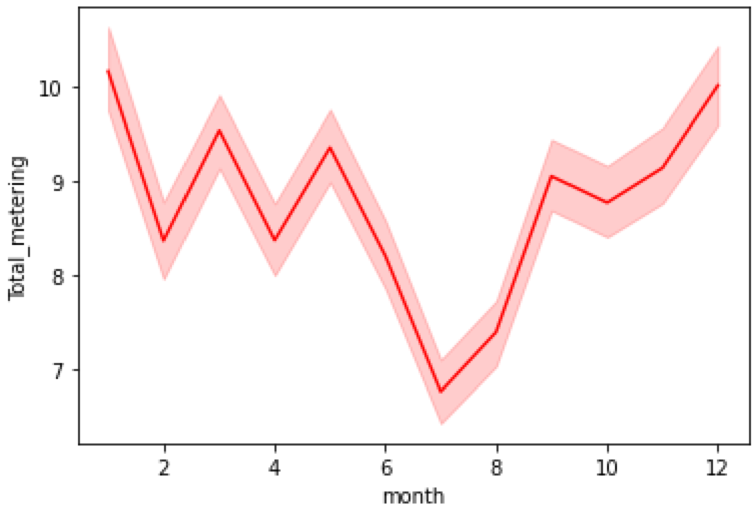
```
# Peak power consumption is between 9 am to 10 am
```

In [65]:

```
sns.lineplot(x="month", y="Total_metering",data=new_data, color='red')
```

Out[65]:

<AxesSubplot:xlabel='month', ylabel='Total_metering'>



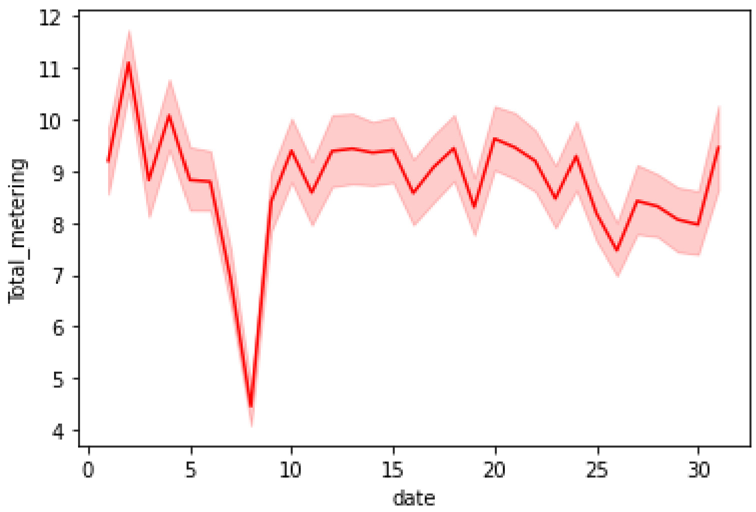

```
# In July there is least power consumption
```

In [66]:

```
sns.lineplot(x="date", y="Total_metering",data=new_data, color='red')
```

Out[66]:

<AxesSubplot:xlabel='date', ylabel='Total_metering'>

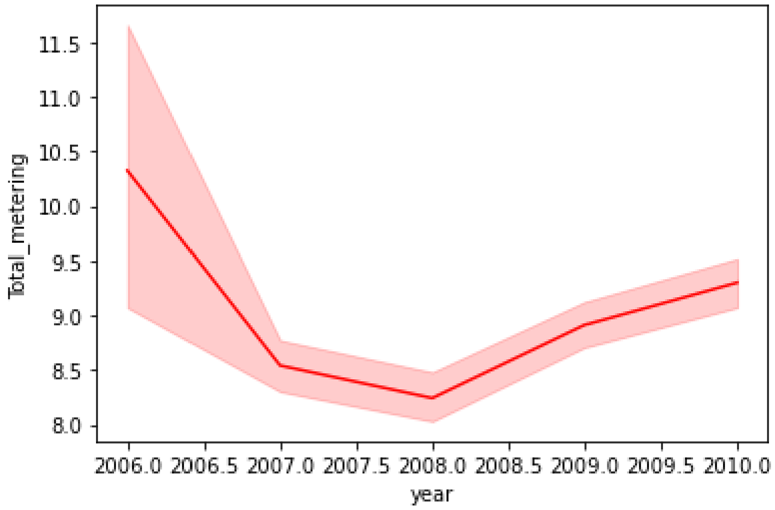


In [67]:

```
sns.lineplot(x="year", y="Total_metering",data=new_data, color='red')
```

Out[67]:

<AxesSubplot:xlabel='year', ylabel='Total_metering'>



In [68]:

```
# Power consumption has decreased from 2006.
```

In [69]:

```
# Checking correlation between features
new_data.corr()
```

Out[69]:

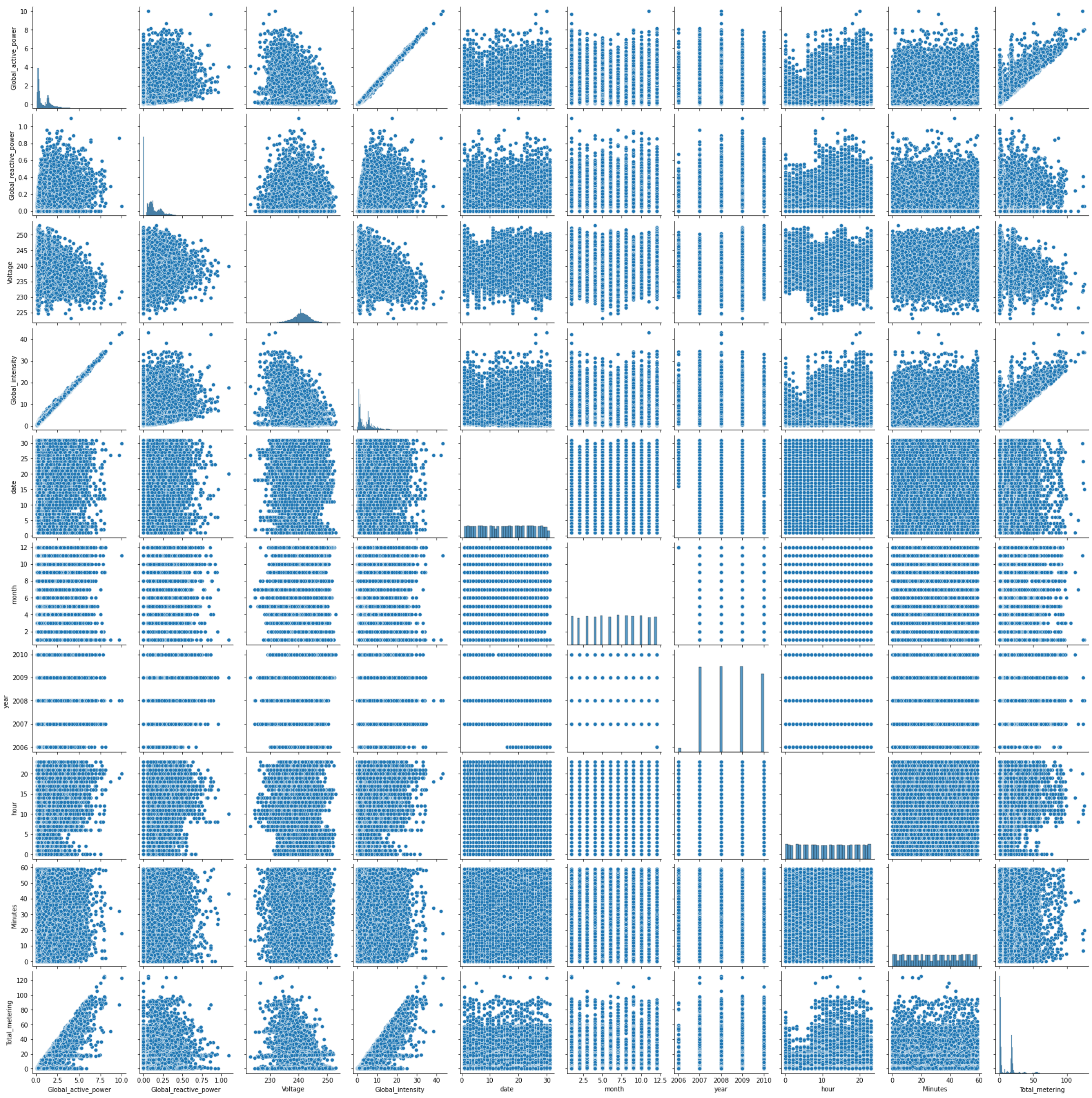
	Global_active_power	Global_reactive_power	Voltage	Global_intensity
Global_active_power	1.000000	0.247526	-0.405717	0.998912
Global_reactive_power	0.247526	1.000000	-0.117495	0.266897
Voltage	-0.405717	-0.117495	1.000000	-0.417265
Global_intensity	0.998912	0.266897	-0.417265	1.000000
date	-0.013610	0.010332	-0.001232	-0.013326
month	0.002231	0.015479	0.037846	0.001421
year	-0.034249	0.041110	0.255353	-0.038451
hour	0.281613	0.124570	-0.179564	0.282101
Minutes	0.000932	-0.002103	0.004345	0.000769
Total_metering	0.843304	0.181717	-0.350024	0.840450

In [70]:

```
sns.pairplot(new_data)
```

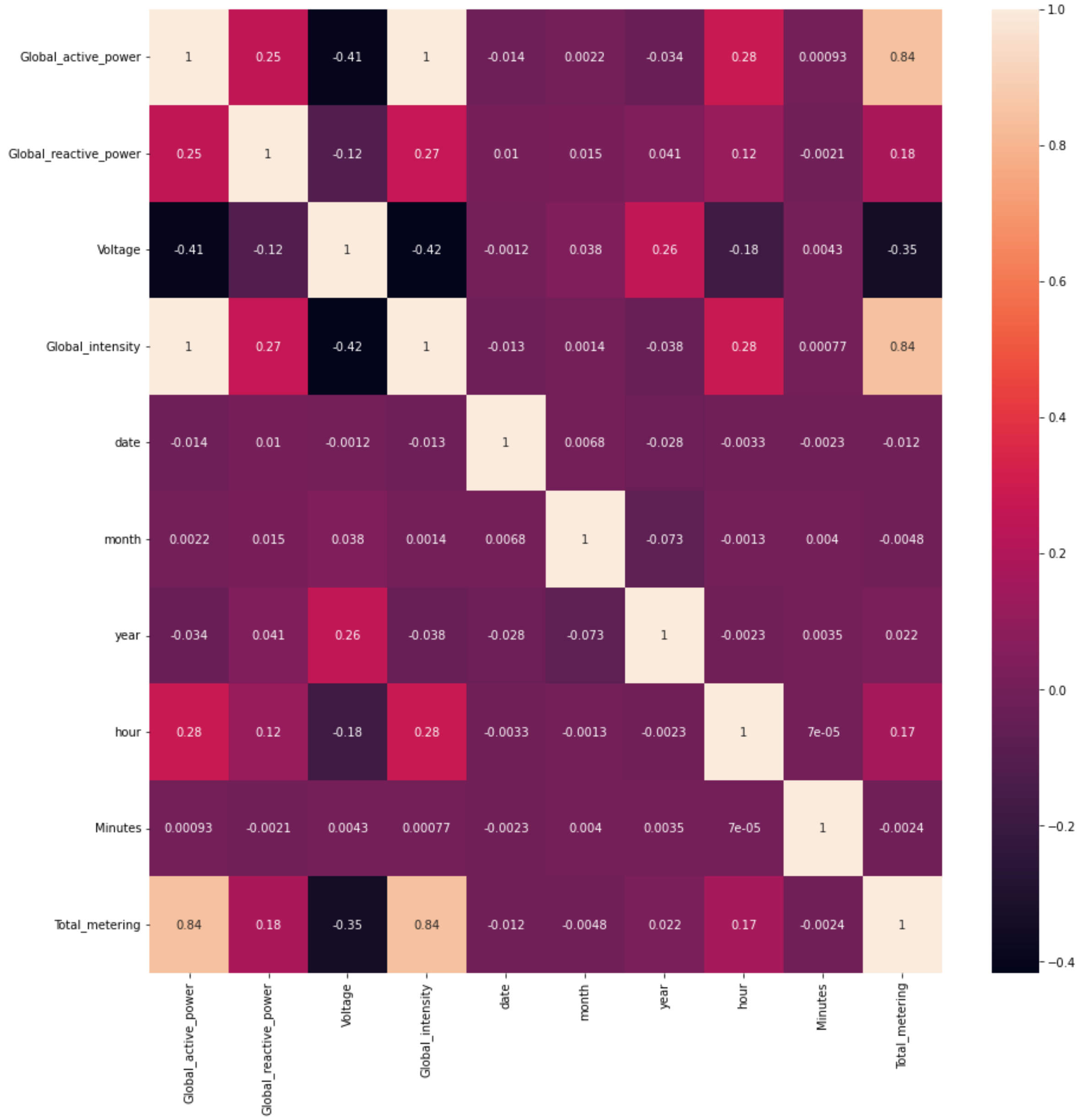
Out[70]:

```
<seaborn.axisgrid.PairGrid at 0x2334a631490>
```



In [71]:

```
# Checking the correlation between the features
plt.figure(figsize=(15,15))
sns.heatmap(data=new_data.corr(), annot=True)
plt.show()
```




```
# Global Intensity and global_active_power are highly correlated
```

In [72]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [73]:

```
vif_data = pd.DataFrame()
```

In [74]:

```
vif_data["VIF"] = [variance_inflation_factor(new_data.values, i)
                          for i in range(len(new_data.columns))]
```

In [75]:

```
vif_data['features'] = new_data.columns
```

In [76]:

```
vif_data
```

Out[76]:

	VIF	features
0	1293.380415	Global_active_power
1	2.968642	Global_reactive_power
2	7539.560937	Voltage
3	1313.775187	Global_intensity
4	4.203920	date
5	4.616676	month
6	7637.321115	year
7	4.160524	hour
8	3.872897	Minutes
9	5.254709	Total_metering

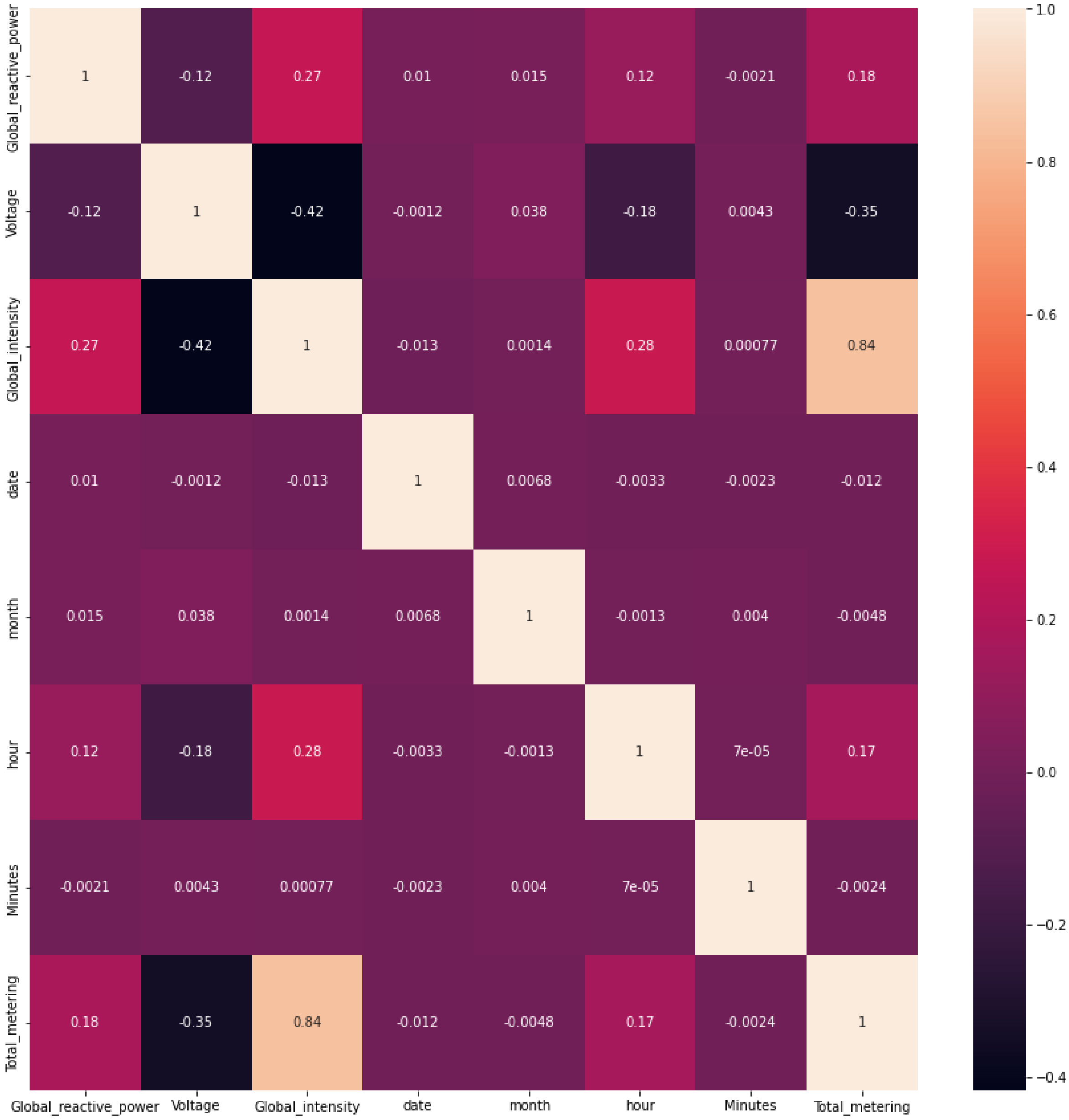
In [77]:

```
# Droping Global active power & year due to multicollinearity

new_data.drop(columns=['Global_active_power','year'], axis=1, inplace= True)
```

In [78]:

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



In [79]:

```
new_data.columns
```

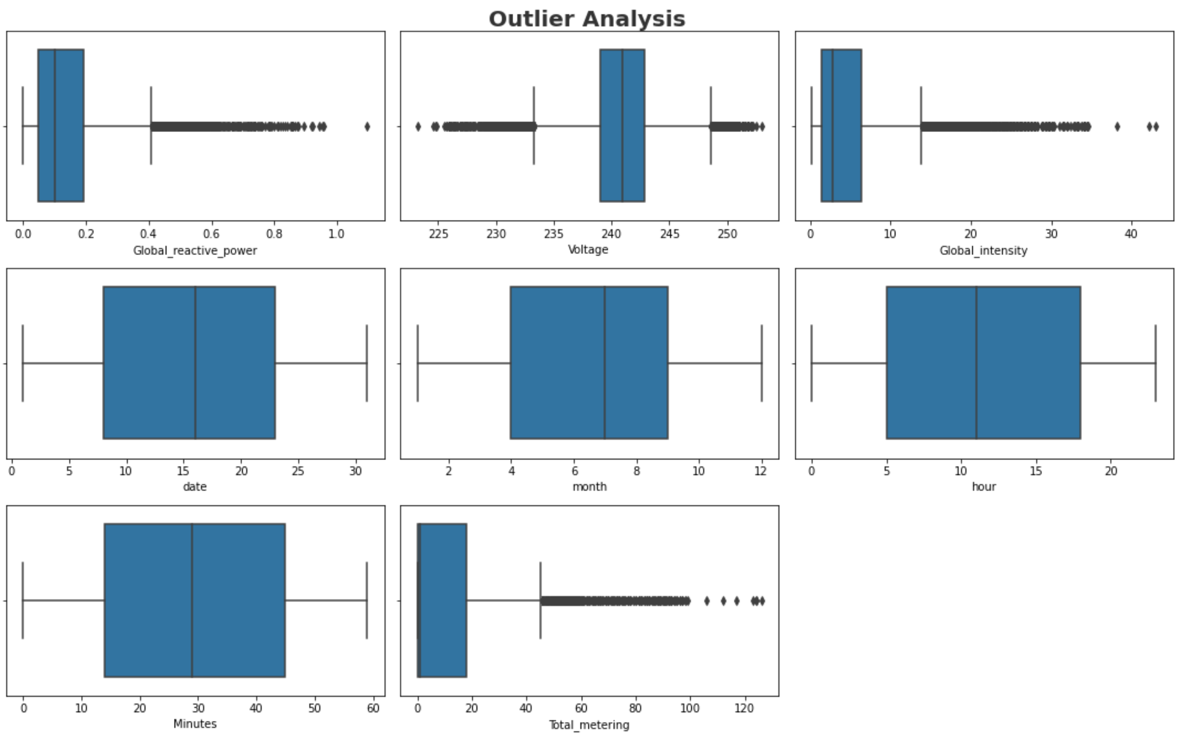
Out[79]:

```
Index(['Global_reactive_power', 'Voltage', 'Global_intensity', 'date', 'mont
h',
       'hour', 'Minutes', 'Total_metering'],
      dtype='object')
```

In [80]:

```
# Checking for outliers
plt.figure(figsize =(15,15))
plt.suptitle('Outlier Analysis',fontsize = 20, fontweight='bold', alpha=0.8 )

for i in range (0, len(new_data.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(new_data[new_data.columns[i]])
    plt.tight_layout()
```



In [81]:

```
# Treating outliers

from feature_engine.outliers.winsorizer import Winsorizer
```

In [82]:

```
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
new_data['Global_reactive_power'] = winsorizer.fit_transform(new_data[['Global_reactive_pow
```

In [83]:

```
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
new_data['Voltage'] = winsorizer.fit_transform(new_data[['Voltage']])
```

In [84]:

```
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
new_data['Global_intensity'] = winsorizer.fit_transform(new_data[['Global_intensity']])
```

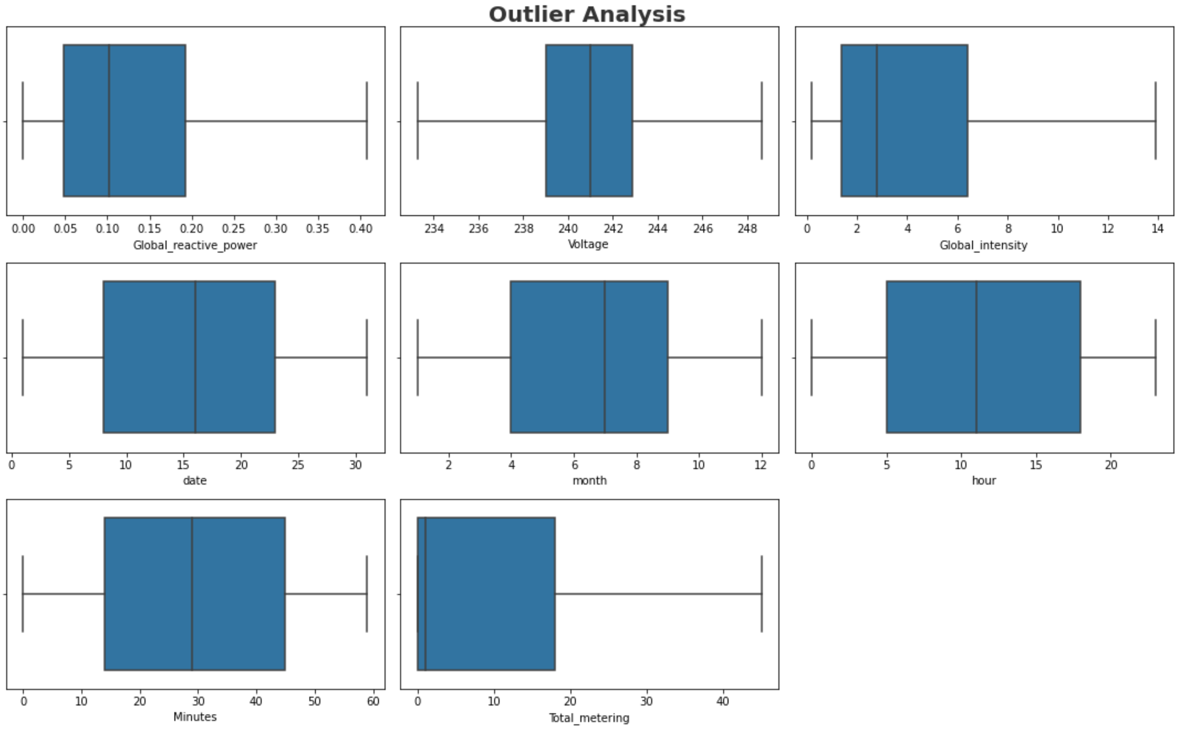
In [85]:

```
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
new_data['Total_metering'] = winsorizer.fit_transform(new_data[['Total_metering']])
```

In [86]:

```python
# Checking for outliers after outlier treatment
plt.figure(figsize =(15,15))
plt.suptitle('Outlier Analysis',fontsize = 20, fontweight='bold', alpha=0.8 )

for i in range (0, len(new_data.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(new_data[new_data.columns[i]])
    plt.tight_layout()
```



In [87]:

```python
# new_data.to_csv("power_consumption_cleaned.csv")
```

In [88]:

```
pip install pymongo
```

```
Requirement already satisfied: pymongo in c:\users\sahil\anaconda3\lib\site-
packages (4.3.2)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in c:\users\sahil\an
aconda3\lib\site-packages (from pymongo) (2.2.1)
Note: you may need to restart the kernel to use updated packages.
```

In [89]:

```python
# Uploading the data in MongoDB database
import pymongo
```

In [90]:

```python
client = pymongo.MongoClient("mongodb+srv://sahil5723:NEWlife123@cluster0.1bbad.mongodb.net
```

In [91]:

```
# database = client['power_consumption']
# collection = database['household_power_data']
```

In [92]:

```
# data_dict = new_data.to_dict("records")
```

In [93]:

```
# collection.insert_many(data_dict)
```

In [94]:

```
# Loading the data from MongoDB

db = client.power_consumption
collection = db.household_power_data
data_db = pd.DataFrame(list(collection.find()))
```

In [95]:

```
data_db.drop(columns=['_id'], inplace=True)
```

In [96]:

```
data_db
```

Out[96]:

	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes	Total_met
0	0.072	238.99	5.2	13	8	12	33	
1	0.198	240.90	2.8	28	5	19	45	
2	0.082	240.55	1.4	19	5	13	50	
3	0.286	235.68	10.2	11	6	15	23	
4	0.076	241.70	2.6	30	5	16	22	
...	...	...	...	...	...	...	...	
49995	0.268	240.16	2.6	22	4	1	3	
49996	0.364	244.96	2.4	23	3	14	53	
49997	0.000	244.79	1.2	2	2	0	41	
49998	0.052	241.25	3.6	11	11	11	54	
49999	0.000	239.50	10.8	14	4	7	32	

50000 rows × 8 columns

In [97]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [98]:

```
scale = StandardScaler()
```

In [99]:

```
x = data_db.iloc[:,:-1]
```

In [100]:

```
y = data_db['Total_metering']
```

In [101]:

```
x.columns
```

Out[101]:

```
Index(['Global_reactive_power', 'Voltage', 'Global_intensity', 'date', 'mont
h',
       'hour', 'Minutes'],
      dtype='object')
```

In [102]:

```
# Splitting the data into train and test

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.25, random_state=42)
```

In [103]:

```
scale.fit(x_train)
```

Out[103]:

```
StandardScaler()
```

In [104]:

```
import pickle
```

In [105]:

```
# Saving the standard Scaler model

pickle_out = open("scale.pkl","wb")
pickle.dump(scale,pickle_out)
pickle_out.close()
```

In [106]:

```
# Loading the standard scaler model

pickle_in = open('scale.pkl','rb')
scaler = pickle.load(pickle_in)
```

In [107]:

```
x_train_tf = scaler.transform(x_train)
```

In [108]:

```
x_test_tf = scaler.transform(x_test)
```

In [109]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.svm import SVR
```

In [110]:

```
linear = LinearRegression()
```

In [111]:

```
linear.fit(x_train_tf, y_train)
```

Out[111]:

```
LinearRegression()
```

In [112]:

```
# Predicting using linear regression model

linear_pred_test = linear.predict(x_test_tf)
```

In [113]:

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

In [114]:

```
mae_linear = mean_absolute_error(y_test, linear_pred_test)
```

In [115]:

```
# Mean Absolute Error after applying linear regression
mae_linear
```

Out[115]:

```
4.186119972092438
```

In [116]:

```
rmse_linear = np.sqrt(mean_squared_error(y_test, linear_pred_test))
```

In [117]:

```
# RMSE obtained after Linear regression

rmse_linear
```

Out[117]:

6.229779759245607

In [118]:

```
linear_r2_score = r2_score(y_test, linear_pred_test)
```

In [119]:

```
# R-Squared
linear_r2_score
```

Out[119]:

0.6817080461979973

In [120]:

```
# adjusted R-squared

adjusted_r2_linear = 1 - ((1-linear_r2_score)*(len(y_test)-1))/(len(y_test)-(x_test.shape[1
```

In [121]:

```
adjusted_r2_linear
```

Out[121]:

0.6815296885549766

In [122]:

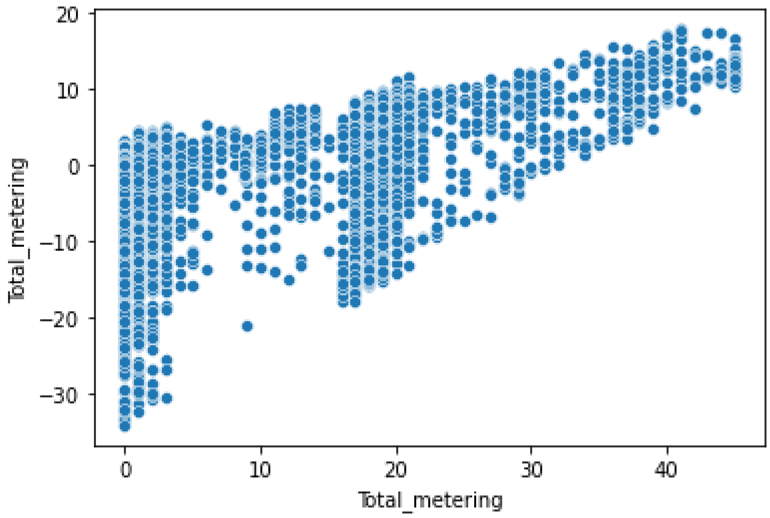
```
linear_residuals = y_test - linear_pred_test
```

In [123]:

```
sns.scatterplot(y_test, linear_residuals)
```

Out[123]:

<AxesSubplot:xlabel='Total_metering', ylabel='Total_metering'>



Applying Lasso Regression

In [124]:

```
lasso = Lasso()
```

In [125]:

```
# Fitting the lasso regression

lasso.fit(x_train_tf, y_train)
```

Out[125]:

Lasso()

In [126]:

```
# Predicting using Lasso regression

lasso_test_pred = lasso.predict(x_test_tf)
```

In [127]:

```
mae_lasso = mean_absolute_error(y_test, lasso_test_pred)
```

In [128]:

```
# Mean Absolute error

mae_lasso
```

Out[128]:

```
4.360652031923295
```

In [129]:

```
rmse_lasso = np.sqrt(mean_squared_error(y_test, lasso_test_pred))
```

In [130]:

```
# Root Mean squared

rmse_lasso
```

Out[130]:

```
6.337821658596121
```

In [131]:

```
lasso_r2_score = r2_score(y_test, lasso_test_pred)
```

In [132]:

```
# R-Squared

lasso_r2_score
```

Out[132]:

```
0.6705721574804308
```

In [133]:

```
len(y_test)
```

Out[133]:

```
12500
```

In [134]:

```
adjusted_r2_lasso = 1- ((1-lasso_r2_score) * (len(y_test)-1))/(len(y_test) - (x_test.shape[
```

In [135]:

```
# Adjusted R-Squared

adjusted_r2_lasso
```

Out[135]:

```
0.6704403230629025
```

Applying Ridge Regression

In [136]:

```
ridge = Ridge()
```

In [137]:

```
# Fitting Ridge Regression

ridge.fit(x_train_tf, y_train)
```

Out[137]:

```
Ridge()
```

In [138]:

```
# Prediction using ridge regression

ridge_test_pred = ridge.predict(x_test_tf)
```

In [139]:

```
ridge_mae = mean_absolute_error(y_test, ridge_test_pred)
```

In [140]:

```
# Mean Absolute Error

ridge_mae
```

Out[140]:

```
4.186126145627163
```

In [141]:

```
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_test_pred))
```

In [142]:

```
# Root Mean squared Error

ridge_rmse
```

Out[142]:

```
6.229769380682283
```

In [143]:

```
ridge_r2_score = r2_score(y_test, ridge_test_pred)
```

In [144]:

```
# R-squared

ridge_r2_score
```

Out[144]:

```
0.6817091067203829
```

In [145]:

```
adjusted_r2_score_ridge = 1 - ((1-ridge_r2_score)*(len(y_test)-1))/ (len(y_test)-(x_test.sh
```

In [146]:

```
# Adjusted r - squared
adjusted_r2_score_ridge
```

Out[146]:

```
0.6815817292218718
```

Applying ElasticNet

In [147]:

```
elastic = ElasticNet()
```

In [148]:

```
# applying Elastic Net Regression

elastic.fit(x_train_tf, y_train)
```

Out[148]:

```
ElasticNet()
```

In [149]:

```
# prediction Using ElasticNet Regression
elastic_test_pred = elastic.predict(x_test_tf)
```

In [150]:

```
elastic_mae = mean_absolute_error(y_test, elastic_test_pred)
```

In [151]:

```
# Mean Absolute Error
elastic_mae
```

Out[151]:

```
5.433588211519576
```

In [152]:

```
elastic_rmse = np.sqrt(mean_squared_error(y_test, elastic_test_pred))
```

In [153]:

```
# Root Mean Squared error
elastic_rmse
```

Out[153]:

```
7.044465331296436
```

In [154]:

```
elastic_r2_score = r2_score(y_test, elastic_test_pred)
```

In [155]:

```
# R-Squared
elastic_r2_score
```

Out[155]:

```
0.5930169279853222
```

In [156]:

```
elastic_adjusted_r2_score = 1 - ((1-elastic_r2_score)* (len(y_test)-1))/(len(y_test)- (x_te
```

In [157]:

```
# Adjusted R-Squared
elastic_adjusted_r2_score
```

Out[157]:

```
0.592854056578241
```

Applying Support Vector Regressor

In [158]:

```
svr = SVR()
```

In [159]:

```
# Applying Support Vector Regressor

svr.fit(x_train_tf, y_train)
```

Out[159]:

```
SVR()
```

In [160]:

```
svr_test_pred = svr.predict(x_test_tf)
```

In [161]:

```
svr_mae = mean_absolute_error(y_test, svr_test_pred)
```

In [162]:

```
# Mean Squared Error
svr_mae
```

Out[162]:

```
3.2435580976498413
```

In [163]:

```
svr_rmse = np.sqrt(mean_squared_error(y_test, svr_test_pred))
```

In [164]:

```
# Root Mean Sqaured Error
svr_rmse
```

Out[164]:

```
5.557421363030737
```

In [165]:

```
# Accuarcy using SVR
svr_r2_score = r2_score(y_test, svr_test_pred)
```

In [166]:

```
svr_r2_score
```

Out[166]:

```
0.746704819911103
```

In [167]:

```
adjusted_r2_score_svr =  1- ((1-svr_r2_score) * (len(y_test)-1))/(len(y_test) - (x_test.sha
```

In [182]:

```
# Adjusted R-Squared
adjusted_r2_score_svr
```

Out[182]:

```
0.746603453183038
```

```
# Apply hyperparameter tuning
```

In [173]:

```
params = { 'kernel' : ['linear','poly','sigmoid','rbf']
         }
```

In [174]:

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

In [175]:

```
grid = GridSearchCV(estimator = svr, param_grid = params,cv=10, n_jobs= -1 )
```

In [176]:

```
grid.fit(x_train_tf, y_train)
```

Out[176]:

```
GridSearchCV(cv=10, estimator=SVR(), n_jobs=-1,
             param_grid={'kernel': ['linear', 'poly', 'sigmoid', 'rbf']})
```

In [177]:

```
grid.best_score_
```

Out[177]:

```
0.7437546829653663
```

In [179]:

```
new_svr = grid.best_params_
```

In [181]:

```
new_svr
```

Out[181]:

```
{'kernel': 'rbf'}
```

In [184]:

```
results = {'models':['Linear', 'Ridge', 'Lasso', 'ElasticNet', 'SVR'],
'R-Squared':[linear_r2_score, ridge_r2_score, lasso_r2_score, elastic_r2_score, svr_r2_scor
'Adjusted_R_squared':[adjusted_r2_linear, adjusted_r2_score_ridge, adjusted_r2_lasso, elast
```

In [185]:

```
results = pd.DataFrame(results)
```

In [186]:

```
results
```

Out[186]:

	models	R-Squared	Adjusted_R_squared
0	Linear	0.681708	0.681530
1	Ridge	0.681709	0.681582
2	Lasso	0.670572	0.670440
3	ElasticNet	0.593017	0.592854
4	SVR	0.746705	0.746603

In [ ]: