Dataset collection

Source - https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption

Import Data and Required Packages

In [122…
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [123…
```
entire_data = pd.read_csv(r'E:\household_power_consumption\household_power_consumption.t
```

In [124…
```
## checking shape

entire_data.shape
```

Out[124]:
```
(2075259, 9)
```

In [125…
```
entire_data.head()
```

Out[125]:

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

In [126…
```
## selecting random 30000 sample data

data = entire_data.sample(30000)
```

In [127…
```
data.shape
```

Out[127]:
```
(30000, 9)
```

In [128…
```
data.head()
```

Out[128]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_meterin
1875250	10/7/2010	23:34:00	0.338	0.220	242.380	1.600	0.
1477562	7/10/2009	19:26:00	4.064	0.350	233.240	17.800	0.
996807	7/11/2008	22:51:00	0.392	0.074	238.760	1.600	0.
1028423	29/11/2008	21:47:00	2.866	0.000	240.490	12.000	1.
749072	19/5/2008	21:56:00	1.170	0.322	241.420	5.000	1.

Dataset 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 [129...

```
## checking data type

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30000 entries, 1875250 to 1019532
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   30000 non-null  object 
 1   Time                   30000 non-null  object 
 2   Global_active_power    30000 non-null  object 
 3   Global_reactive_power  30000 non-null  object 
 4   Voltage                30000 non-null  object 
 5   Global_intensity       30000 non-null  object 
 6   Sub_metering_1         30000 non-null  object 
 7   Sub_metering_2         30000 non-null  object 
 8   Sub_metering_3         29638 non-null  float64
dtypes: float64(1), object(8)
memory usage: 2.3+ MB
```

importing date time

In [130...

```
import datetime as dt
```

seperating date, month and year

In [131...

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

In [132...

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

In [133...

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

In [134...

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

seprating hours minutes and seconds

In [135...

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

In [136...

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

replacing special characters from data

In [137...
```
data.replace('?',np.nan, inplace=True)
```

In [138...
```
data.replace(",", np.nan, inplace=True)
```

In [139...
```
data.replace(" ", np.nan, inplace=True)
```

In [140...
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30000 entries, 1875250 to 1019532
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   30000 non-null  datetime64[ns]
 1   Time                   30000 non-null  object
 2   Global_active_power    29638 non-null  object
 3   Global_reactive_power  29638 non-null  object
 4   Voltage                29638 non-null  object
 5   Global_intensity       29638 non-null  object
 6   Sub_metering_1         29638 non-null  object
 7   Sub_metering_2         29638 non-null  object
 8   Sub_metering_3         29638 non-null  float64
 9   date                   30000 non-null  int64
 10  month                  30000 non-null  int64
 11  year                   30000 non-null  int64
 12  hour                   30000 non-null  int64
 13  Minutes                30000 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(5), object(7)
memory usage: 3.4+ MB
```

converting the data type of dataset

In [141...
```
data['Global_active_power'] = data['Global_active_power'].astype(float)
```

In [142...
```
data['Global_reactive_power'] = data['Global_reactive_power'].astype(float)
```

In [143...
```
data['Voltage'] = data['Voltage'].astype(float)
```

In [144...
```
data['Global_intensity'] = data['Global_intensity'].astype(float)
```

In [145...
```
data['Sub_metering_1'] = data['Sub_metering_1'].astype(float)
```

In [146...
```
data['Sub_metering_2'] = data['Sub_metering_2'].astype(float)
```

In [147...
```
data['Sub_metering_3'] = data['Sub_metering_3'].astype(float)
```

replacing null values with mean

In [148...
```
data['Global_active_power'] = data['Global_active_power'].fillna(data['Global_active_pow
```

In [149...
```
data['Global_reactive_power'] = data['Global_reactive_power'].fillna(data['Global_reacti
```

In [150...
```
data['Voltage'] = data['Voltage'].fillna(data['Voltage'].mean())
```

In [151...
```
data['Global_intensity'] = data['Global_intensity'].fillna(data['Global_intensity'].mean
```

In [152…
```
data['Sub_metering_1'] = data['Sub_metering_1'].fillna(data['Sub_metering_1'].mean())
```

In [153…
```
data['Sub_metering_2'] = data['Sub_metering_2'].fillna(data['Sub_metering_2'].mean())
```

In [154…
```
data['Sub_metering_3'] = data['Sub_metering_3'].fillna(data['Sub_metering_3'].mean())
```

In [155…
```
data.isnull().sum()
```

Out[155]:
```
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
date                     0
month                    0
year                     0
hour                     0
Minutes                  0
dtype: int64
```

creating total metering column

In [156…
```
data['Total_metering'] = data['Sub_metering_1'] + data['Sub_metering_2'] + data['Sub_met
```

In [157…
```
data.describe()
```

Out[157]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.00000
mean	1.079779	0.122780	240.855735	4.576888	1.079729	1.25420
std	1.041191	0.111851	3.227113	4.377274	6.000329	5.56829
min	0.076000	0.000000	225.120000	0.200000	0.000000	0.00000
25%	0.310000	0.048000	239.020000	1.400000	0.000000	0.00000
50%	0.616000	0.100000	240.980000	2.600000	0.000000	0.00000
75%	1.506000	0.190000	242.890000	6.200000	0.000000	1.00000
max	8.670000	1.390000	253.510000	37.200000	79.000000	75.00000

observations

1. Global active power :- has observed mean power of 1.09879 with standard deviation of 1.047. Minimum power obderved 0.07 and maximum power observed 9.24 And and range 25th to 75th percentile is [0.31 to 1.52].

2. Global reactive power :- has observed mean power of 0.12 with standard deviation of 0.11. Minimum power observed 0 and maximum power observed 1.07 And and range 25th to 75th percentile is [0.05 to 1.07].

3. Voltage :- has observed mean voltage of 240.82 with standard deviation of 3.21. Minimum voltage observed 225.02 and maximum voltage observed 242.86 And and range 25th to 75th percentile is

[239.01000 - 252.75000].

4. Global intensity :- has observed mean intensity of 4.6600 with standard deviation of 4.401490. Minimum intensity observed 0.200000 and Maximum intensity observed 40.600000 And and range 25th to 75th percentile is [1.400000 to 40.600000]

5. Total metering :- has observed mean metering of 8.946802 with standard deviation of 12.767918. Minimum metering observed is 0 and maximum metering observed 134.000000. And and range 25th to 75th percentile is [0 to 18.00000].

dropping date, time , submetering 1,2,3 columns

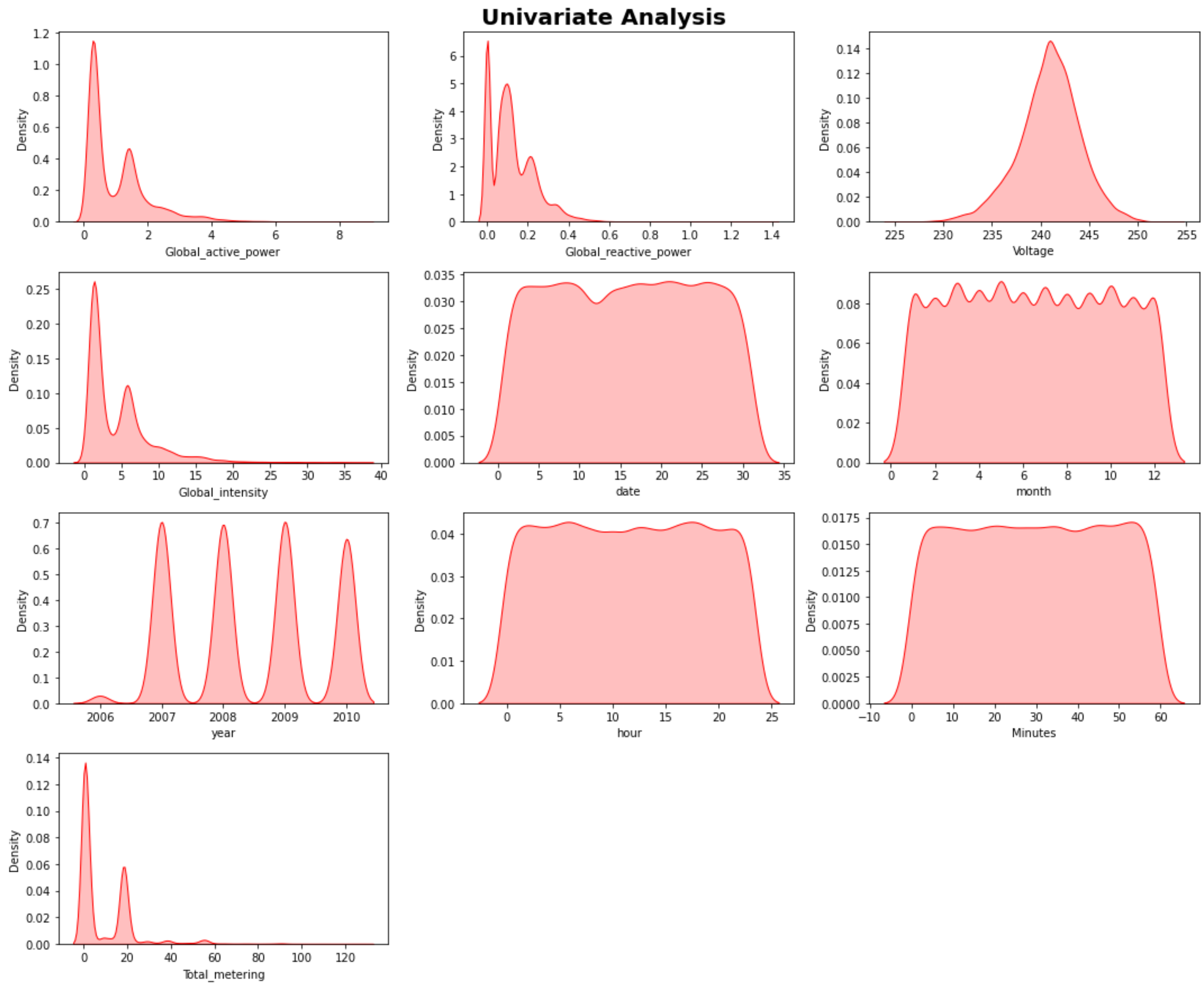
In [158…
```
new_data = data.drop(columns=['Date', 'Time',
       'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3'])
```

In [159…
```
new_data.head()
```

Out[159]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	date	month	year	hour	Minut
1875250	0.338	0.220	242.38	1.6	7	10	2010	23	3
1477562	4.064	0.350	233.24	17.8	10	7	2009	19	2
996807	0.392	0.074	238.76	1.6	11	7	2008	22	5
1028423	2.866	0.000	240.49	12.0	29	11	2008	21	4
749072	1.170	0.322	241.42	5.0	19	5	2008	21	5

In [160…
```
plt.figure(figsize=(15,15))
plt.suptitle("Univariate Analysis", fontsize=20, fontweight= 'bold')

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()
```



Observation

1. Global_active_power

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

1. Global reactive power

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

1. Voltage

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

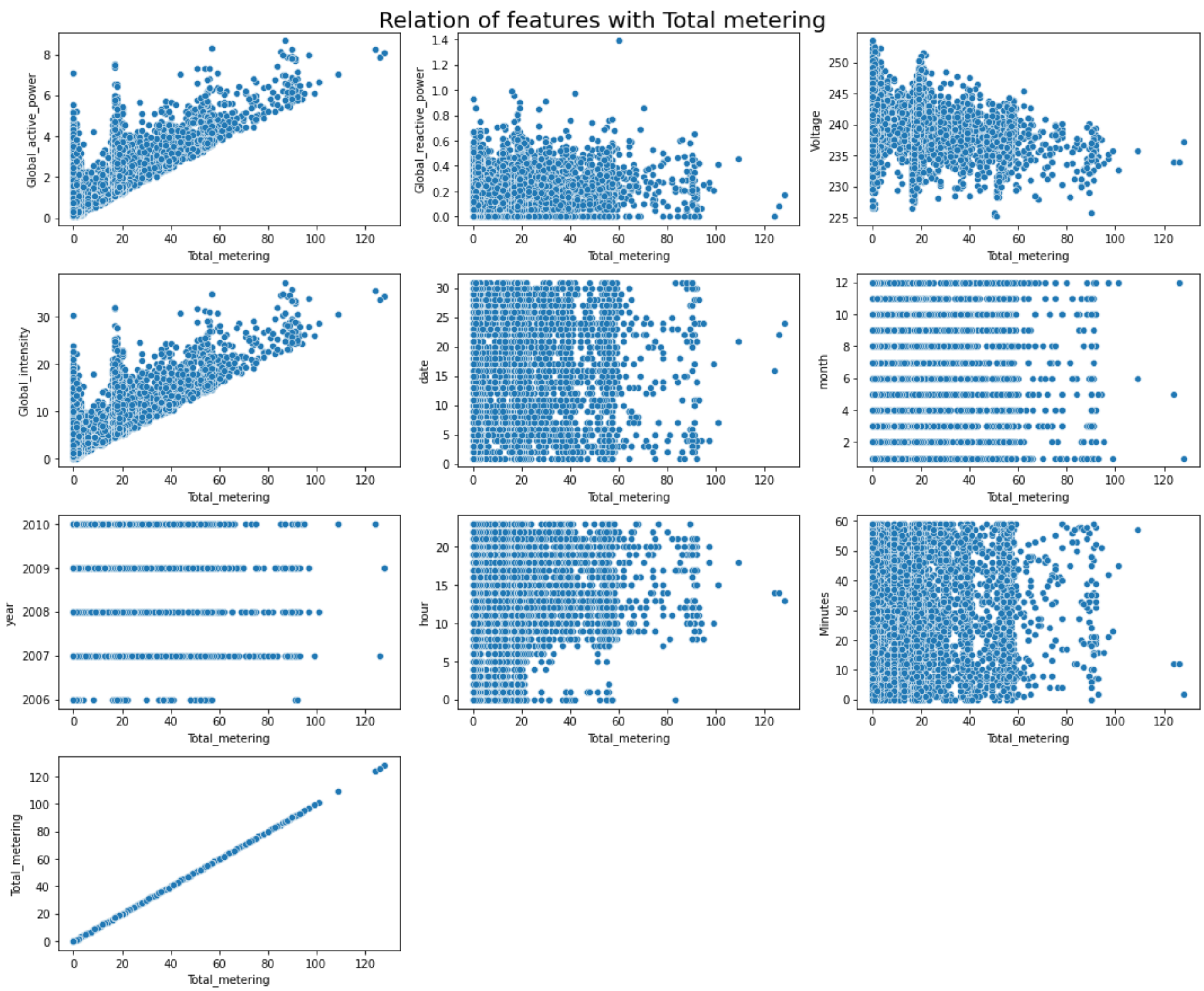
1. Global intensity

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

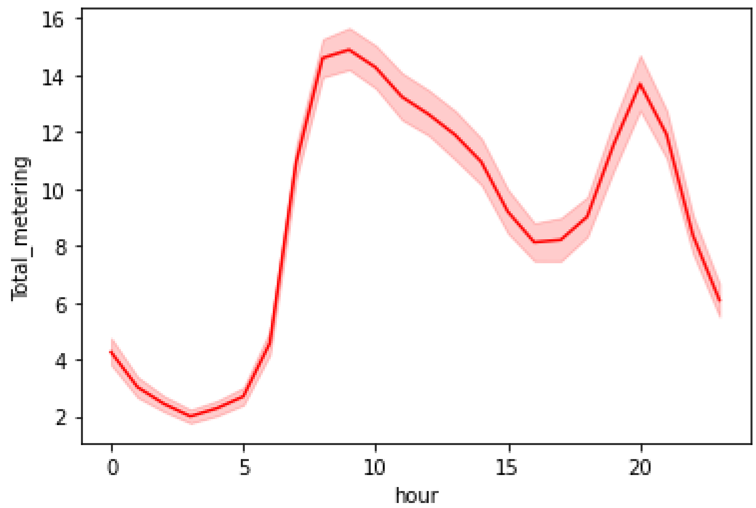
1. Total metering

- Metering is distributed between 0 to 60
- most of the distribution is between 0 to 20.

In [161…
```
plt.figure(figsize=(15,15))
plt.suptitle('Relation of features with Total metering', fontsize=20)

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()
```



In [162…
```
sns.lineplot(x='hour', y='Total_metering', data=new_data, color = 'red');
```

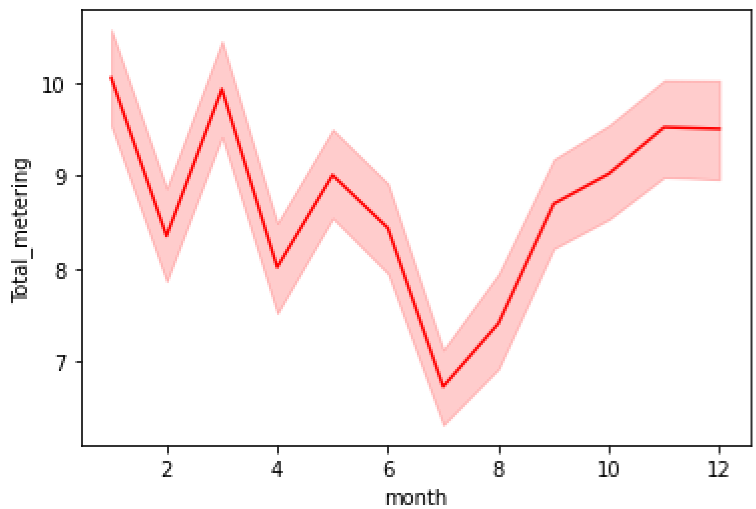


observation

- Total metering id maximum between 7 to 10 hour.

In [163…

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

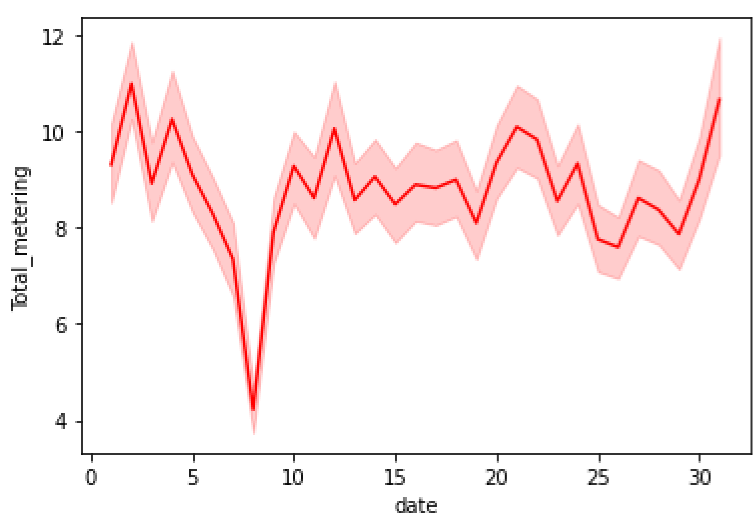


observation

- in july there is least power consumption

In [164…

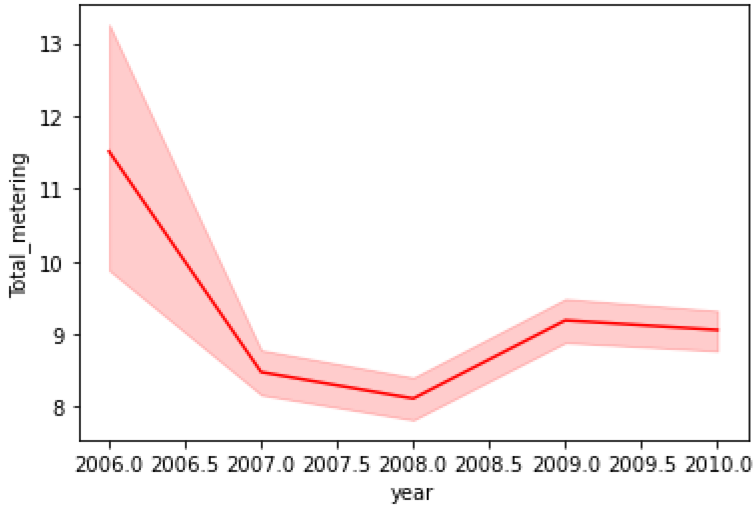
```
sns.lineplot(x='date', y='Total_metering', data=new_data, color = 'red');
```



observation

- power is very less consumed between dates 6 and 7

In [165…
```
sns.lineplot(x='year', y='Total_metering', data=new_data, color = 'red');
```



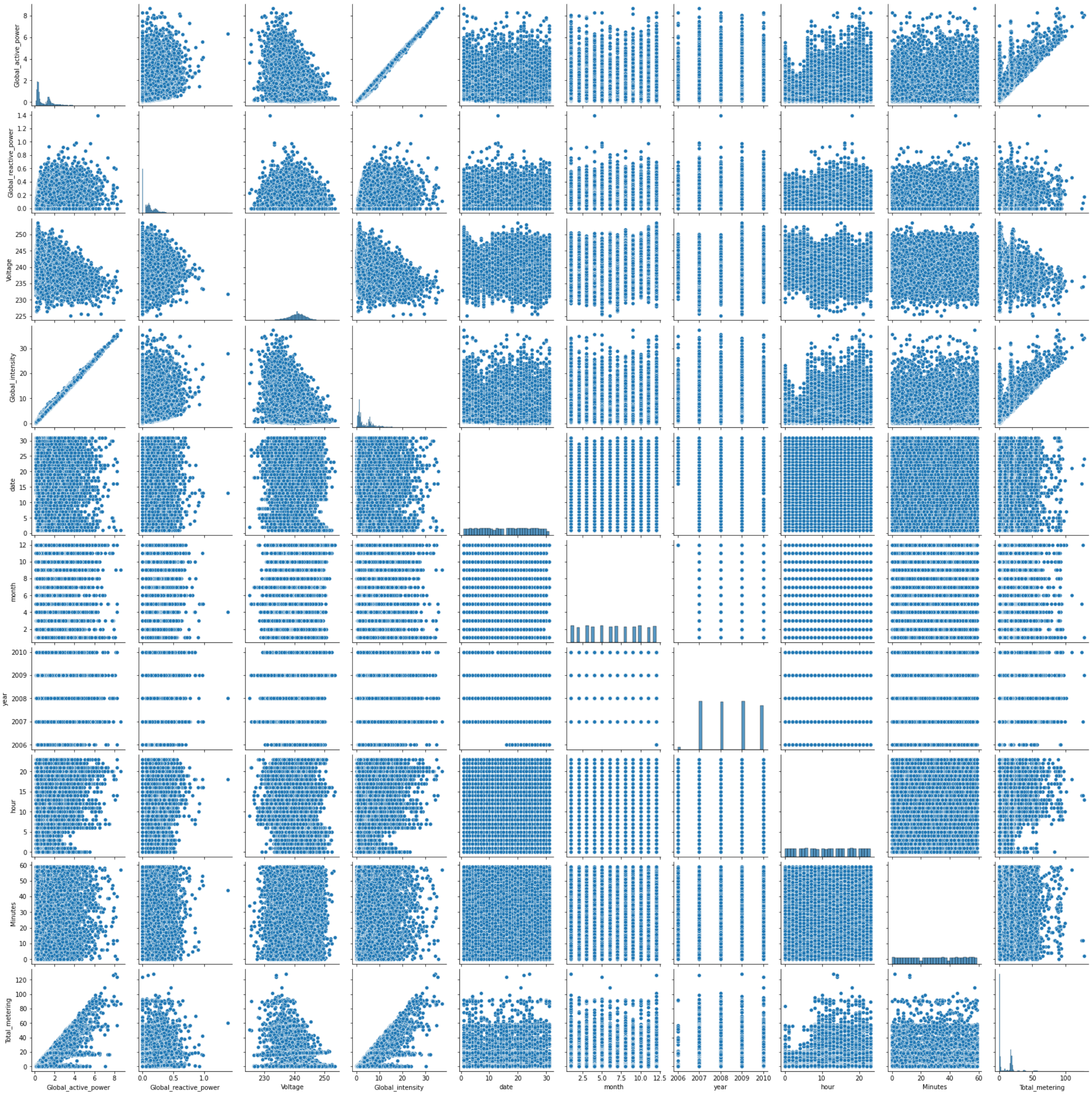
observation

- Power consumption decreased from 2006

Checking with pairplot

In [166…
```
sns.pairplot(new_data)
```

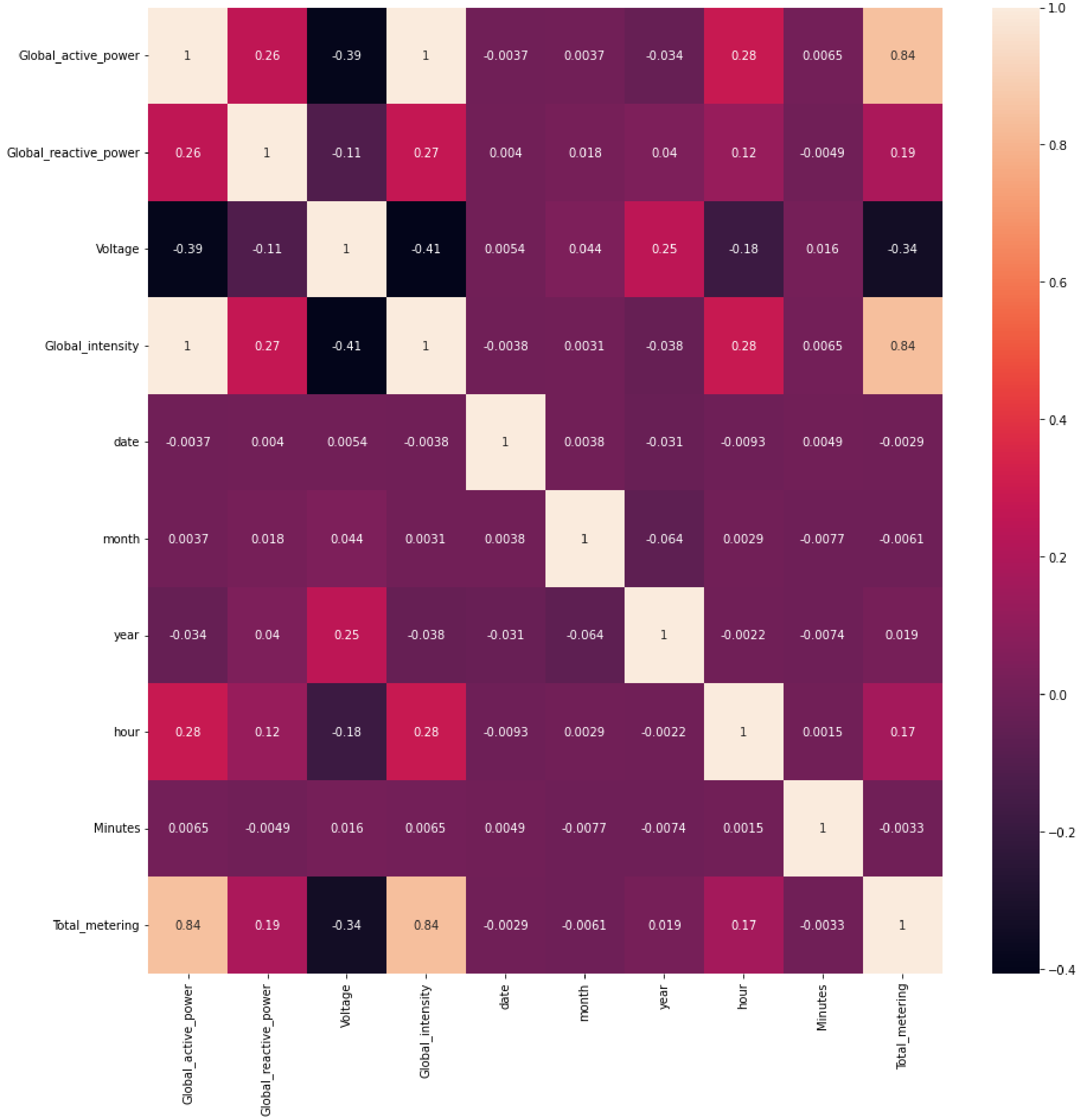
Out[166]:
```
<seaborn.axisgrid.PairGrid at 0x20846341fa0>
```



Checking correlation

In [167…

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



observation

- Global active power and Global intensity are highly correlated

Variance

In [168…

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

In [169…

```python
vif_data = pd.DataFrame()
vif_data['VIF'] = [variance_inflation_factor(new_data.values,i) for i in range(len(new_d
vif_data['features'] = new_data.columns
vif_data
```

Out[169]:

	VIF	features
0	1263.307134	Global_active_power

1	2.933370	Global_reactive_power
2	7524.087773	Voltage
3	1280.863208	Global_intensity
4	4.201976	date
5	4.537380	month
6	7608.625121	year
7	4.183764	hour
8	3.897966	Minutes
9	5.252017	Total_metering

In [170…

```
## dropping Global_Active_power and year feature from data due to multicollinearity

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

In [171…

```
new_data.head()
```

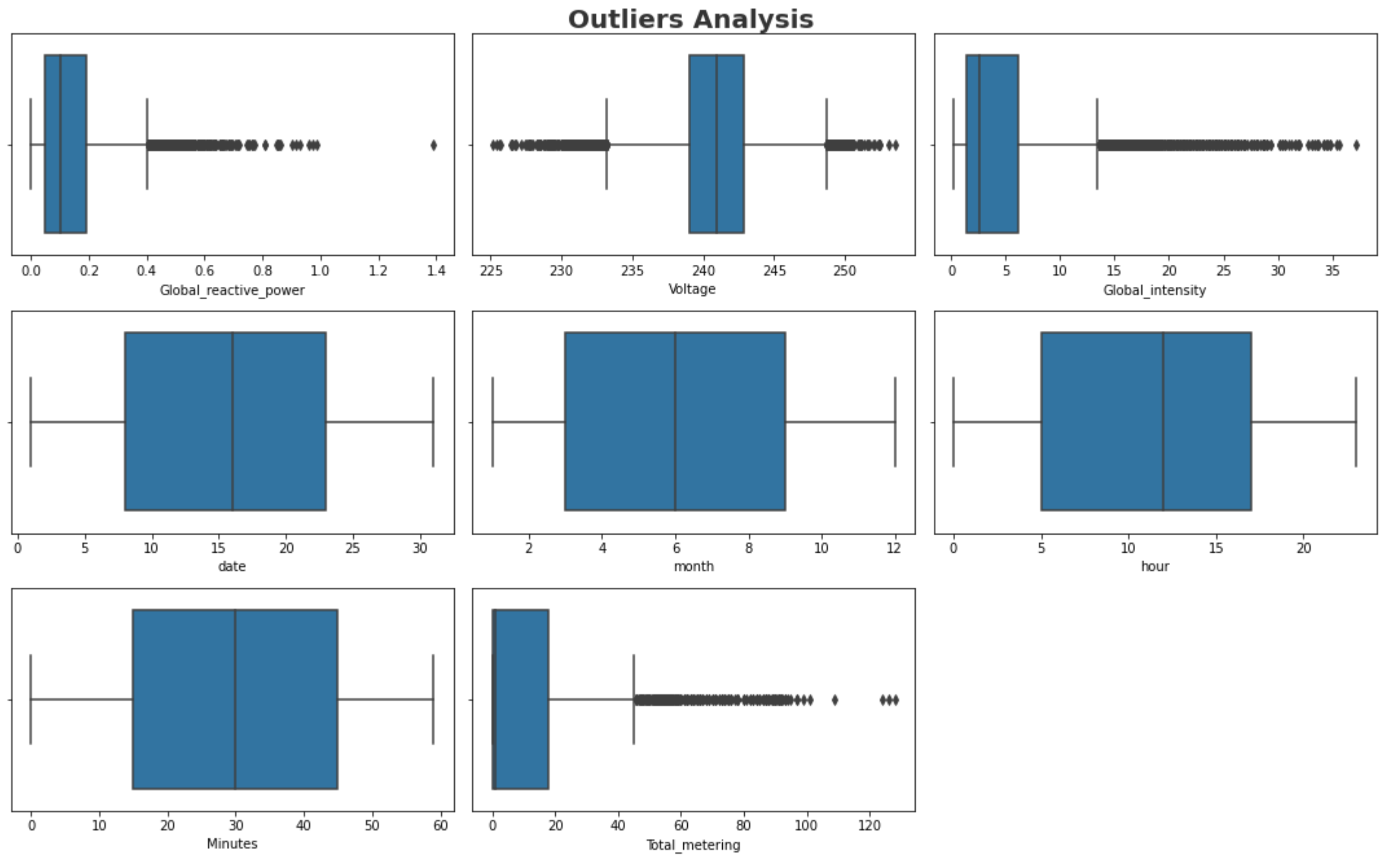
Out[171]:

	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes	Total_metering
1875250	0.220	242.38	1.6	7	10	23	34	2.0
1477562	0.350	233.24	17.8	10	7	19	26	27.0
996807	0.074	238.76	1.6	11	7	22	51	1.0
1028423	0.000	240.49	12.0	29	11	21	47	19.0
749072	0.322	241.42	5.0	19	5	21	56	2.0

Checking for outliers

In [175…

```
plt.figure(figsize=(15,15))
plt.suptitle("Outliers 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()
```



importing winsorizer

In [178…

```
from feature_engine.outliers.winsorizer import Winsorizer
```

In [179…

```
winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries
                        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_
new_data
```

Out[179]:

	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes	Total_metering
1875250	0.220	242.38	1.6	7	10	23	34	2.0
1477562	0.350	233.24	17.8	10	7	19	26	27.0
996807	0.074	238.76	1.6	11	7	22	51	1.0
1028423	0.000	240.49	12.0	29	11	21	47	19.0
749072	0.322	241.42	5.0	19	5	21	56	2.0
...	...	...	...	...	...	...	...	...
1451421	0.296	239.47	6.4	19	9	15	45	19.0
118428	0.000	241.06	1.8	3	8	23	12	0.0
1226108	0.210	239.41	6.4	16	4	4	32	19.0
125203	0.116	241.53	1.4	13	3	16	7	0.0
1019532	0.190	235.19	18.4	23	11	17	36	17.0

30000 rows × 8 columns

Outlier handling for Voltage feature

In [180…

```
winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries
                        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']])
```

Outlier handling for Global_intensity feature

In [181…

```
winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries
                        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']])
```

Outlier handling for Total metering feature

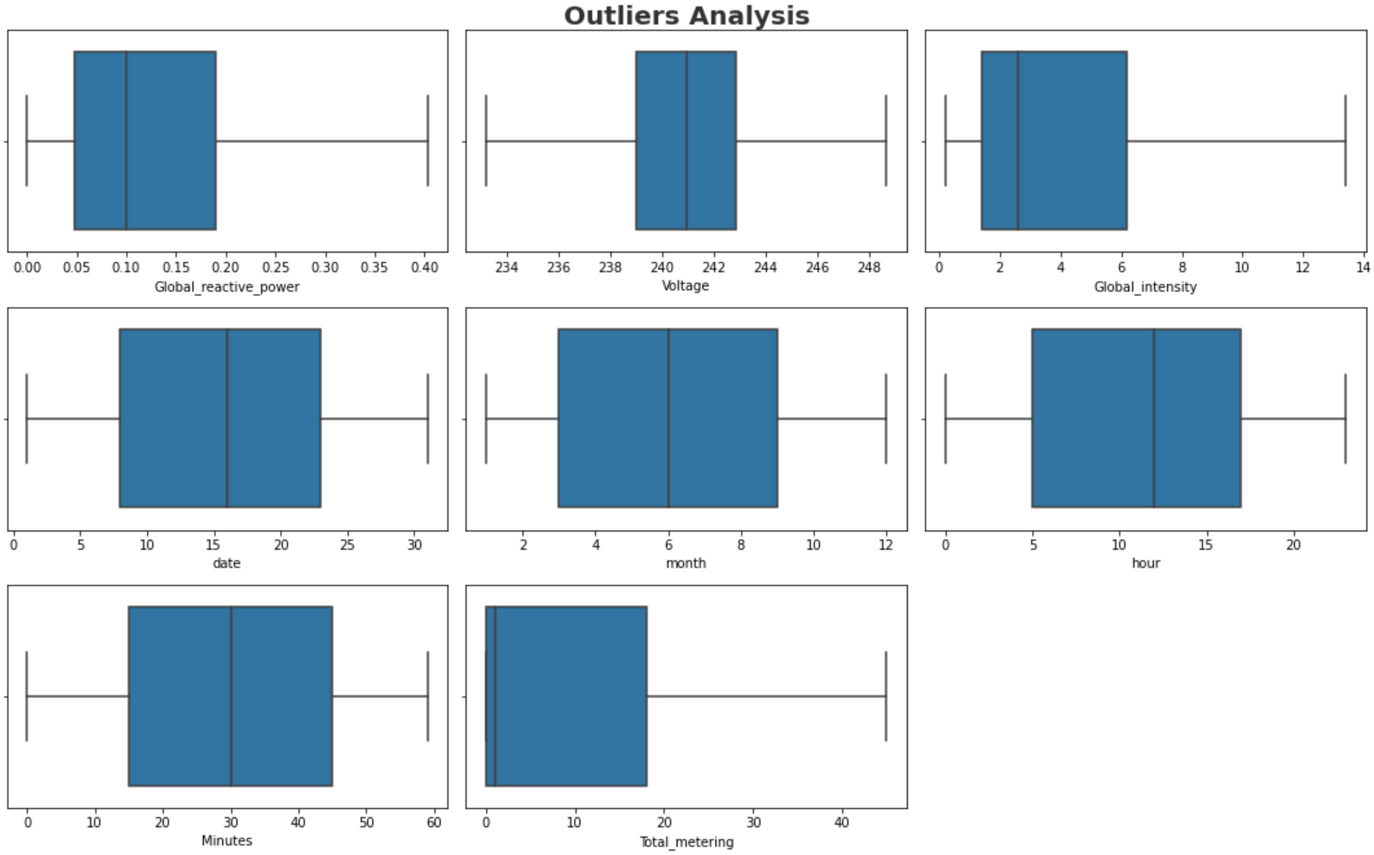
In [182…

```
winsorizer = Winsorizer(capping_method = 'iqr', # choose skewed for IQR rule boundaries
                        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']])
```

Checking for outliers after outlier treatment

In [183…

```
plt.figure(figsize=(15,15))
plt.suptitle("Outliers 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()
```



save cleaned data

In [184...

```
new_data.to_csv("cleaned_power_consumption_data.csv")
```

uploading the data in Mongodb

In [185...

```
import pymongo
```

In [186...

```
client = pymongo.MongoClient("mongodb+srv://Sawan_7588:Sawan321@cluster.e9dj7rv.mongodb.
database = client['power_consumption']
collection = database['power_consumption_data']
data_dict = new_data.to_dict("rocords")
collection.insert_many(data_dict)
```

Out[186]:

```
<pymongo.results.InsertManyResult at 0x20862ba45e0>
```

loading the data from mongodb

In [187...

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

In [188...

```
data_db.head()
```

Out[188]:

	_id	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes	Tot
0	636b43552765269750ca75da	0.220	242.38	1.6	7	10	23	34	
1	636b43552765269750ca75db	0.350	233.24	13.4	10	7	19	26	
2	636b43552765269750ca75dc	0.074	238.76	1.6	11	7	22	51	
3	636b43552765269750ca75dd	0.000	240.49	12.0	29	11	21	47	
4	636b43552765269750ca75de	0.322	241.42	5.0	19	5	21	56	

In [189…
```
#### dropping id column

data_db.drop(columns=['_id'],axis=1,inplace=True)
```

In [190…
```
data_db.head()
```

Out[190]:

	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes	Total_metering
0	0.220	242.38	1.6	7	10	23	34	2.0
1	0.350	233.24	13.4	10	7	19	26	27.0
2	0.074	238.76	1.6	11	7	22	51	1.0
3	0.000	240.49	12.0	29	11	21	47	19.0
4	0.322	241.42	5.0	19	5	21	56	2.0

splitting data into independent and dependent features

In [191…
```
X = data_db.drop('Total_metering', axis=1)
```

In [192…
```
X.head()
```

Out[192]:

	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes
0	0.220	242.38	1.6	7	10	23	34
1	0.350	233.24	13.4	10	7	19	26
2	0.074	238.76	1.6	11	7	22	51
3	0.000	240.49	12.0	29	11	21	47
4	0.322	241.42	5.0	19	5	21	56

In [193…
```
y = data_db['Total_metering']
```

In [194…
```
y.head()
```

Out[194]:
```
0     2.0
1    27.0
2     1.0
3    19.0
4     2.0
Name: Total_metering, dtype: float64
```

splitting train and test data

In [195…
```
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=1
```

In [196…
```
X_train.head()
```

Out[196]:

	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes
13707	0.202	233.88	1.4	25	7	11	1
10403	0.082	242.42	1.6	17	3	2	26
6673	0.180	239.83	1.4	8	1	2	46

28904	0.000	245.96	1.4	1	2	23	30
2987	0.000	243.47	0.8	29	10	4	44

In [197…

```
y_train.head()
```

Out[197]:

```
13707    0.0
10403    0.0
6673     2.0
28904    1.0
2987     0.0
Name: Total_metering, dtype: float64
```

In [198…

```
X_test.head()
```

Out[198]:

	Global_reactive_power	Voltage	Global_intensity	date	month	hour	Minutes
20412	0.000	239.16	8.2	22	3	18	3
1296	0.403	240.33	3.6	31	8	12	58
3906	0.264	239.39	11.6	31	7	22	30
20454	0.154	242.03	1.0	8	11	14	14
5200	0.234	244.98	1.8	4	1	16	3

In [199…

```
y_test.head()
```

Out[199]:

```
20412    18.0
1296      2.0
3906     22.0
20454     1.0
5200      2.0
Name: Total_metering, dtype: float64
```

Standardizing data

In [200…

```
from sklearn.preprocessing import StandardScaler
```

In [201…

```
scaler = StandardScaler()
```

Using fit_transform to standardise Train data

In [202…

```
X_train = scaler.fit_transform(X_train)
X_train
```

Out[202]:

```
array([[ 0.77792657, -2.23953047, -0.80501412, ...,  0.1513342 ,
        -0.07053633, -1.64368897],
       [-0.36383572,  0.48885226, -0.7504291 , ..., -1.0076241 ,
        -1.36798459, -0.20452951],
       [ 0.56860348, -0.33860808, -0.80501412, ..., -1.58710325,
        -1.36798459,  0.94679807],
       ...,
       [-0.49704132, -0.12135981,  0.34127123, ...,  1.02055293,
         0.50610733, -1.47098984],
       [-1.14403995,  0.40259191, -0.64125907, ...,  1.02055293,
        -0.93550184, -0.54992778],
       [ 0.02417392, -0.01090359,  0.06203824, ...,  0.73081335,
        -1.07966275, -0.72262691]])
```

using only transform to avoid data leakage

In [204…
```
X_test = scaler.transform(X_test)
X_test
```
Out[204]:
```
array([[-1.14403995, -0.55266153,  1.05087645, ..., -1.0076241 ,
         0.93859008, -1.52855621],
       [ 2.6903784 , -0.1788667 , -0.20457894, ...,  0.44107378,
         0.07362458,  1.63759461],
       [ 1.36783708, -0.47918049,  1.97882173, ...,  0.1513342 ,
         1.51523375,  0.02573601],
       ...,
       [-0.15451264,  1.271585  , -0.7504291 , ..., -0.42814495,
         1.51523375,  1.46489548],
       [-0.53510007, -0.10538567,  0.66878133, ..., -1.58710325,
        -1.22382367, -1.18315794],
       [ 1.00627903,  0.29077295, -0.80501412, ...,  0.73081335,
        -0.79134092, -1.35585708]])
```

Linear Regression

In [206…
```
from sklearn.linear_model import LinearRegression
```

In [207…
```
# creating linear regression model

linear_reg = LinearRegression()
linear_reg
```
Out[207]:
```
LinearRegression()
```

In [208…
```
# Passing training data(X and y) to the model

linear_reg.fit(X_train, y_train)
```
Out[208]:
```
LinearRegression()
```

In [209…
```
# Printing co-efficients and intercept of best fit hyperplane

print("1. Coefficients of independent features is {}" .format(linear_reg.coef_))
print("2. Intercept of best fit hyper plane is {}".format(linear_reg.intercept_))
```
```
1. Coefficients of independent features is [-0.36871248 -0.20198901  9.39415512  0.01974
707 -0.09055011 -0.90089742
 -0.14200249]
2. Intercept of best fit hyper plane is 8.321786760240233
```

Using model to get predictions of test data

In [210…
```
linear_reg_pred = linear_reg.predict(X_test)
linear_reg_pred
```
Out[210]:
```
array([18.20405343,  5.13941554, 25.15529958, ..., -0.45254123,
       16.20874176,  1.18510445])
```

Validating model using assumptions of Linear regression

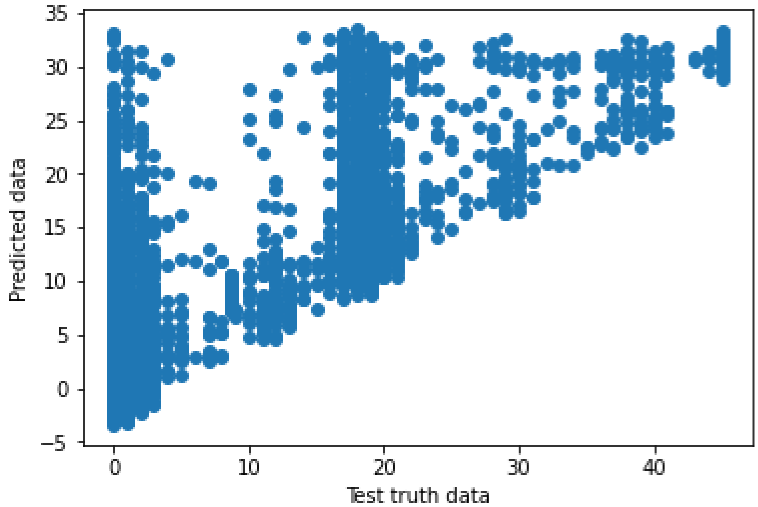
Linear relationship

- Test truth data and Predicted data should follow linear relationship.
- This is an indication of a good model.

In [211…
```
plt.scatter(x=y_test, y= linear_reg_pred)
```

```
plt.xlabel("Test truth data")
plt.ylabel("Predicted data")
```

Out[211]: Text(0, 0.5, 'Predicted data')



Residual distribution

- Residuals should follow normal distribution.

In [212…

```
residual_linear_reg = y_test - linear_reg_pred
residual_linear_reg.head()
```

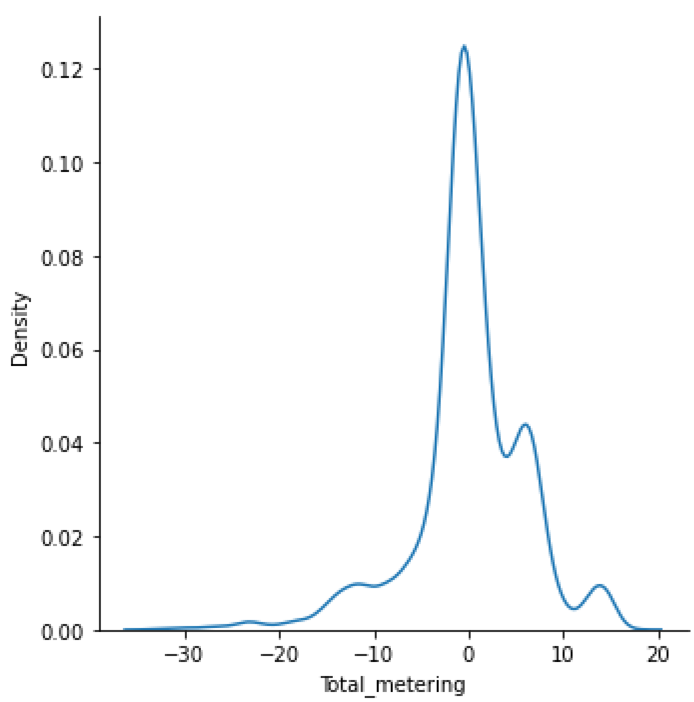
Out[212]:

```
20412   -0.204053
1296    -3.139416
3906    -3.155300
20454    1.793139
5200     1.129463
Name: Total_metering, dtype: float64
```

In [213…

```
sns.displot(x=residual_linear_reg, kind='kde')
```

Out[213]: <seaborn.axisgrid.FacetGrid at 0x2086d923c10>

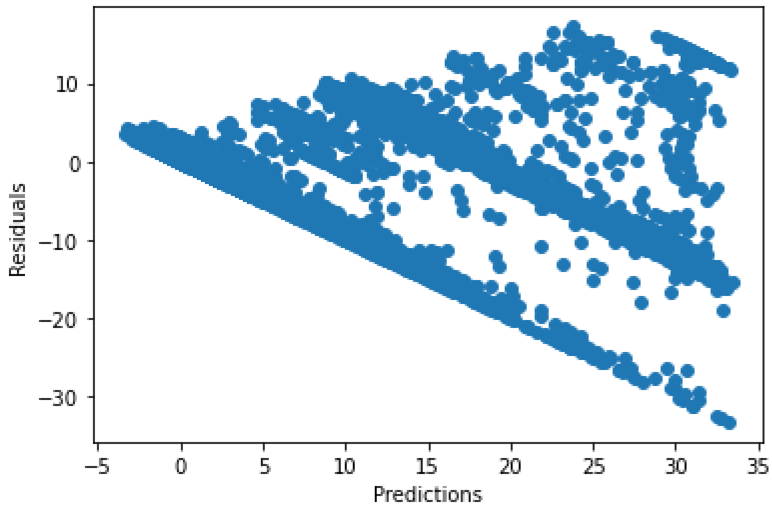


Uniform distribution

- Residuals vs Predictions should follow a uniform distribution.

In [214…
```
plt.scatter(x=linear_reg_pred, y=residual_linear_reg)
plt.xlabel('Predictions')
plt.ylabel('Residuals')
```

Out[214]:
```
Text(0, 0.5, 'Residuals')
```



Performance Matrix

Cost function values

In [215…
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

MSE, MAE and RMSE

In [216…
```
print("Mean squared Error is {}".format(round(mean_squared_error(y_test,linear_reg_pred)
print("Mean absolute Error is {}".format(round(mean_absolute_error(y_test,linear_reg_pre
print("Root mean squared Error is {}".format(round(np.sqrt(mean_squared_error(y_test,lin
```

```
Mean squared Error is 38.08
Mean absolute Error is 4.16
Root mean squared Error is 6.17
```

R Square and Adjusted R Square values

In [217…
```
from sklearn.metrics import r2_score
```

In [219…
```
linear_reg_r2_score = r2_score(y_test, linear_reg_pred)
print("Linear Regression model has {} % accuracy".format(round(linear_reg_r2_score*100,

linear_reg_adj_r2_score=1-((1-linear_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.s
print("Adjusted R square accuracy is {} percent".format(round(linear_reg_adj_r2_score*10
```

```
Linear Regression model has 69.202 % accuracy
Adjusted R square accuracy is 69.17 percent
```

Ridge Regression

In [220…
```
from sklearn.linear_model import Ridge
```

In [221…
```
# Creating Ridge regression model

ridge_reg = Ridge()
ridge_reg
```

Out[221]:
```
Ridge()
```

In [222…
```
# Passing training data(X and y) to the model

ridge_reg.fit(X_train,y_train)
```

Out[222]:
```
Ridge()
```

In [223…
```
# 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 [-0.36858751 -0.20216072  9.39358895  0.01974724 -
0.09053794 -0.90073391
 -0.14199239]
Intercept of best fit hyperplane is 8.321786760240233
```

Using model to get predictions of test data

In [224…
```
ridge_reg_pred = ridge_reg.predict(X_test)
ridge_reg_pred
```

Out[224]:
```
array([18.20353626,  5.13993254, 25.1546826 , ..., -0.45209658,
       16.20808272,  1.185502  ])
```

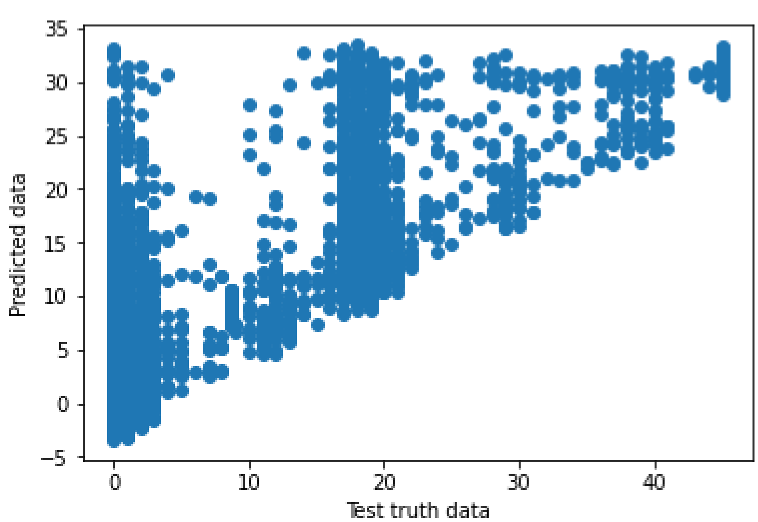
Validating model using assumptions of Ridge regression

Linear relationship

- Test truth data and Predicted data should follow linear relationship.

In [225…
```
plt.scatter(x=y_test,y=ridge_reg_pred)
plt.xlabel("Test truth data")
plt.ylabel("Predicted data")
```

Out[225]:
```
Text(0, 0.5, 'Predicted data')
```



Residual distribution

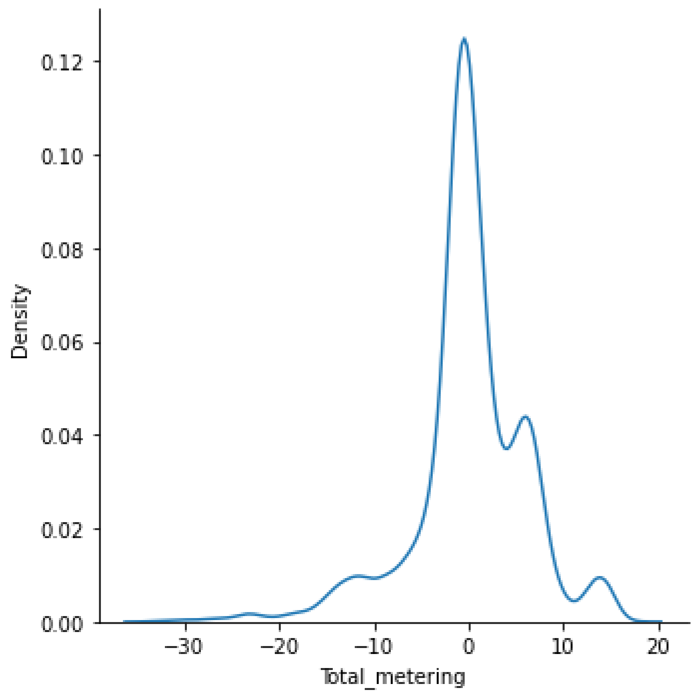
- Residuals should follow normal distribution.

In [226…
```
residual_ridge_reg = y_test - ridge_reg_pred
residual_ridge_reg.head()
```

Out[226]:
```
20412   -0.203536
1296    -3.139933
3906    -3.154683
20454    1.792578
5200     1.129087
Name: Total_metering, dtype: float64
```

In [227…
```
sns.displot(x=residual_ridge_reg, kind='kde')
```

Out[227]:
```
<seaborn.axisgrid.FacetGrid at 0x20860f96400>
```

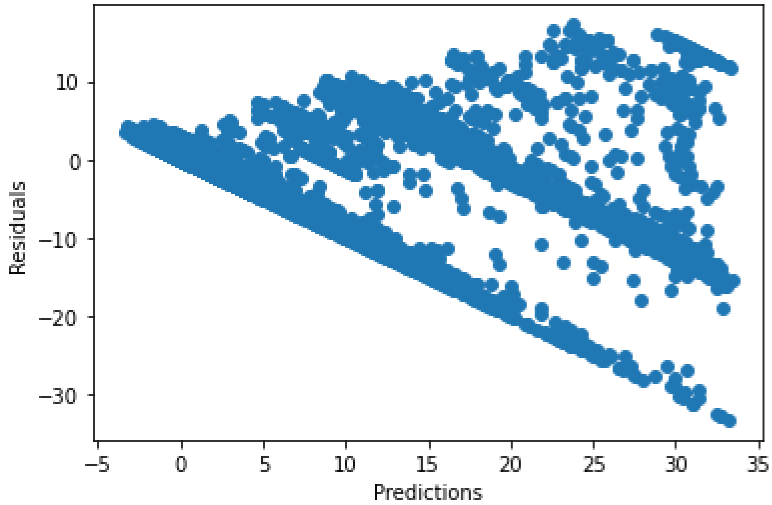


Uniform distribution

- Residuals vs Predictions should follow a uniform distribution.

In [228…
```
plt.scatter(x=ridge_reg_pred, y=residual_ridge_reg)
plt.xlabel('Predictions')
plt.ylabel('Residuals')
```

Out[228]:
```
Text(0, 0.5, 'Residuals')
```



Performance Matrix Ridge regression

Cost function values

MSE, MAE and RMSE

In [229…

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

```
Mean squared error is 38.08
Mean absolute error is 4.16
Root Mean squared error is 6.17
```

R Square and Adjusted R Square values

In [230…

```
ridge_reg_r2_score=r2_score(y_test, ridge_reg_pred)
print("Our Ridge regression model has {} % accuracy".format(round(ridge_reg_r2_score*100

ridge_reg_adj_r2_score=1-((1-ridge_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.sha
print("Adjusted R square accuracy is {} percent".format(round(ridge_reg_adj_r2_score*100
```

```
Our Ridge regression model has 69.202 % accuracy
Adjusted R square accuracy is 69.17 percent
```

Lasso Regression

In [231…

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

In [232…

```
# creating Lasso regression model

lasso_reg = Lasso()
lasso_reg
```

Out[232]:

```
Lasso()
```

In [234…

```
 # Passing training data(X and y) to the model

lasso_reg.fit(X_train, y_train)
```

Out[234]:

```
Lasso()
```

In [235…

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

```
print("1. Co-efficients of independent features is {}".format(lasso_reg.coef_))
print("2. Intercept of best fit hyper plane is {}".format(lasso_reg.intercept_))
```

```
1. Co-efficients of independent features is [-0.          -0.          8.10926397  0.
     -0.          -0.
 -0.         ]
2. Intercept of best fit hyper plane is 8.321786760240231
```

Using model to get predictions of test data

In [236…
```
lasso_reg_pred = lasso_reg.predict(X_test)
lasso_reg_pred
```

Out[236]:
```
array([16.84362129,  6.66280217, 24.36857455, ...,  2.23635907,
       13.74511112,  1.79371476])
```

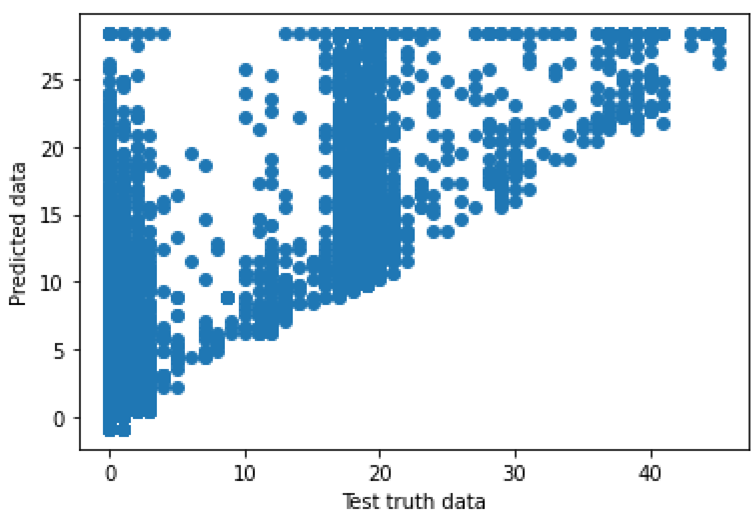
Validating model using assumptions of Lasso regression

Linear relationship

- Test truth data and Predicted data should follow linear relationship

In [237…
```
plt.scatter(x=y_test,y=lasso_reg_pred)
plt.xlabel("Test truth data")
plt.ylabel("Predicted data")
```

Out[237]:
```
Text(0, 0.5, 'Predicted data')
```



Residual distribution

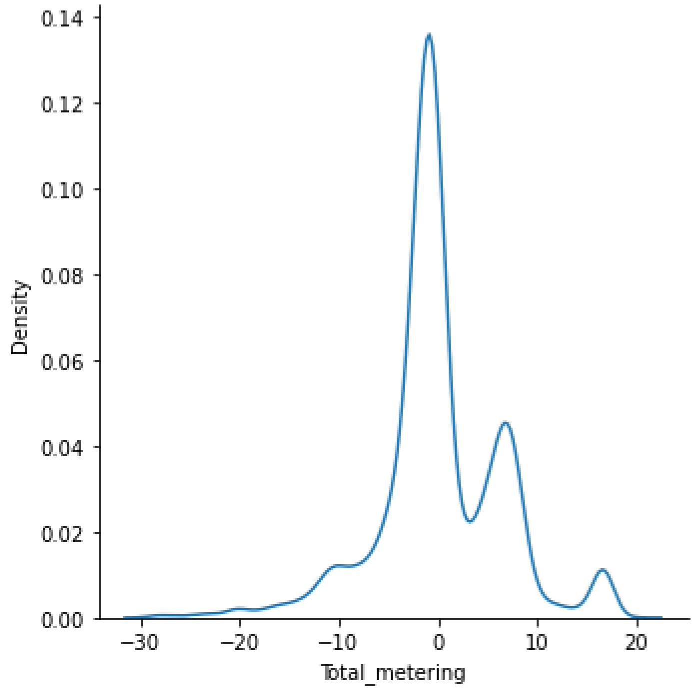
- Residuals should follow normal distribution.

In [238…
```
residual_lasso_reg=y_test-lasso_reg_pred
residual_lasso_reg.head()
```

Out[238]:
```
20412    1.156379
1296    -4.662802
3906    -2.368575
20454    0.091574
5200    -0.679003
Name: Total_metering, dtype: float64
```

In [239…
```
sns.displot(x=residual_lasso_reg, kind='kde')
```

Out[239]: <seaborn.axisgrid.FacetGrid at 0x20847a72a90>



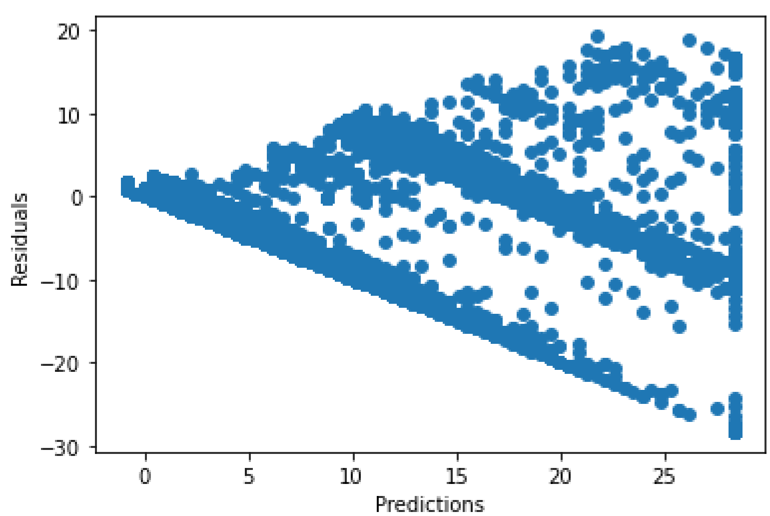
Uniform distribution

- Residuals vs Predictions should follow a uniform distribution.

In [240…

```
plt.scatter(x=lasso_reg_pred, y=residual_lasso_reg)
plt.xlabel('Predictions')
plt.ylabel('Residuals')
```

Out[240]: Text(0, 0.5, 'Residuals')



Performance Matrix of Lasso regression

Performance Matrix

In [241…

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

```
Mean squared error is 39.78
Mean absolute error is 4.35
```

```
Root Mean squared error is 6.31
```

R Square and Adjusted R Square values

In [242…

```
lasso_reg_r2_score=r2_score(y_test, lasso_reg_pred)
print("Lasso regression model has {} % accuracy".format(round(lasso_reg_r2_score*100,3)))

lasso_reg_adj_r2_score=1-((1-lasso_reg_r2_score)*(len(y_test)-1)/(len(y_test)-X_test.sha
print("Adjusted R square accuracy is {} percent".format(round(lasso_reg_adj_r2_score*100
```

```
Lasso regression model has 67.823 % accuracy
Adjusted R square accuracy is 67.79 percent
```

Elastic-Net Regression

In [243…

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

In [244…

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

Out[244]:

```
ElasticNet()
```

In [245…

```
# Passing training data(X and y) to the model
elastic_reg.fit(X_train, y_train)
```

Out[245]:

```
ElasticNet()
```

In [246…

```
# Printing co-efficients and intercept of best fit hyperplane
print("1. Co-efficients of independent features is {}".format(elastic_reg.coef_))
print("2. Intercept of best fit hyper plane is {}".format(elastic_reg.intercept_))
```

```
1. Co-efficients of independent features is [ 0.         -0.67641894  5.56186879  0.
     -0.          0.        
 -0.        ]
2. Intercept of best fit hyper plane is 8.321786760240233
```

Using model to get predictions of test data

In [247…

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

Out[247]:

```
array([14.5404544 ,  7.30493439, 19.65186035, ...,  3.28787438,
       12.11274564,  3.64771953])
```

Validating model using assumptions of Elastic-Net regression

Linear relationship

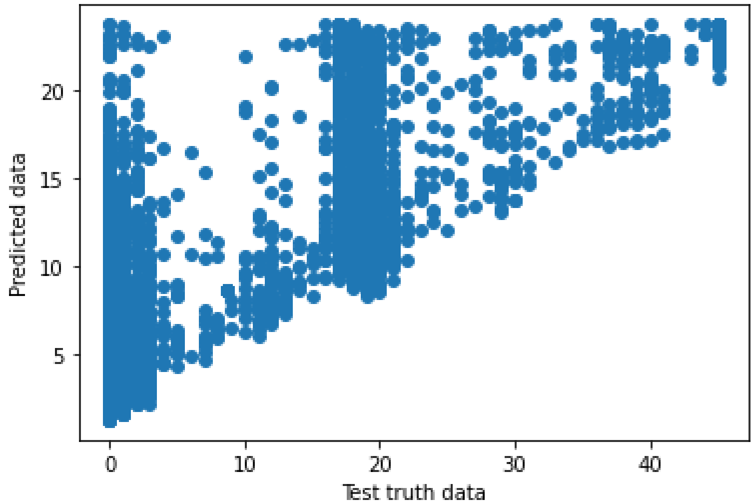
- Test truth data and Predicted data should follow linear relationship.

In [248…

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

Out[248]:

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



Residual distribution

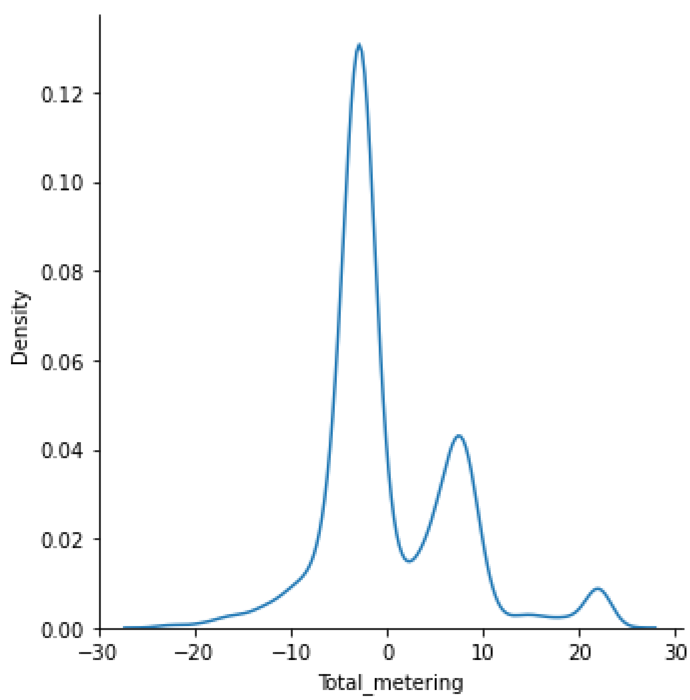
- Residuals should follow normal distribution.

In [249…
```
residual_elastic_reg=y_test-elastic_reg_pred
residual_elastic_reg.head()
```

Out[249]:
```
20412    3.459546
1296    -5.304934
3906     2.348140
20454   -1.990826
5200    -1.567698
Name: Total_metering, dtype: float64
```

In [250…
```
sns.displot(x=residual_elastic_reg, kind='kde')
```

Out[250]: <seaborn.axisgrid.FacetGrid at 0x20848a90a30>



Uniform distribution

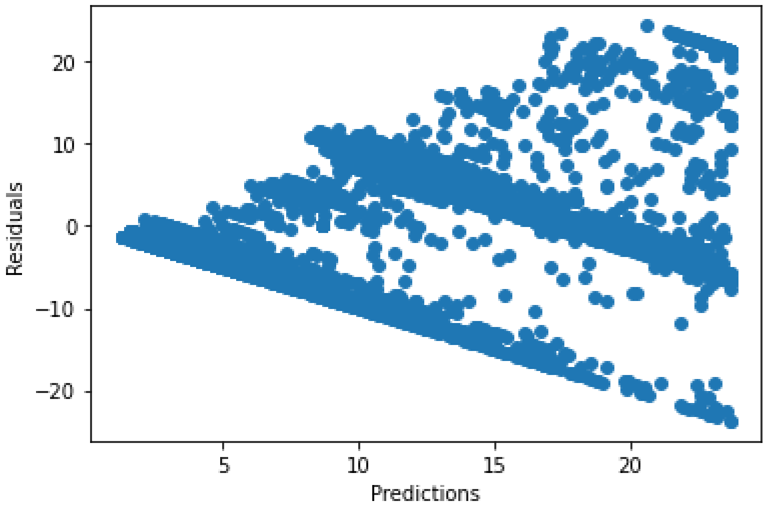
- Residuals vs Predictions should follow a uniform distribution.

In [251…
```
plt.scatter(x=elastic_reg_pred, y=residual_elastic_reg)
```

```
plt.xlabel('Predictions')
plt.ylabel('Residuals')
```

Out[251]:

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



Performance Matrix Elastic Net regression

Cost function values

MSE, MAE and RMSE

In [252…

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

```
Mean squared error is '50.11'
Mean absolute error is '5.4'
Root Mean squared error is '7.08'
```

R Square and Adjusted R Square values

In [254…

```
elastic_reg_r2_score=r2_score(y_test, elastic_reg_pred)
print("Our Elastic-Net regression model has {} % accuracy".format(round(elastic_reg_r2_s

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

```
Our Elastic-Net regression model has 59.473 % accuracy
Adjusted R square accuracy is 59.43 percent
```

SVR

In [255…

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

In [256…

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

Out[256]:

```
SVR()
```

In [257…

```
# Passing training data(X and y) to the model
svr.fit(X_train, y_train)
```

Out[257]: SVR()

Using model to get predictions of test data

In [259…
```
svr_pred=svr.predict(X_test)
svr_pred
```

Out[259]:
```
array([21.11026948,  7.03640666, 25.56832251, ...,  0.46556868,
       14.31089306,  0.52465158])
```

Performance Matrix SVR

Cost function values

MSE, MAE and RMSE

In [260…
```
print("Mean squared error is '{}'".format(round(mean_squared_error(y_test, svr_pred),2)))
print("Mean absolute error is '{}'".format(round(mean_absolute_error(y_test, svr_pred),2
print("Root Mean squared error is '{}'".format(round(np.sqrt(mean_squared_error(y_test,
```

```
Mean squared error is '33.7'
Mean absolute error is '3.36'
Root Mean squared error is '5.81'
```

R Square and Adjusted R Square values

In [261…
```
svr_r2_score=r2_score(y_test, svr_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 72.745 % accuracy
Adjusted R square accuracy is 72.72 percent
```

Apply hyperparameter tuning

In [262…
```
params = { 'kernel' : ['linear','poly','sigmoid','rbf']
}
```

In [263…
```
from sklearn.model_selection import GridSearchCV
```

In [264…
```
grid = GridSearchCV(estimator = svr, param_grid = params,cv=10, n_jobs= -1 )
```

In [265…
```
grid.fit(X_train,y_train)
```

Out[265]:
```
GridSearchCV(cv=10, estimator=SVR(), n_jobs=-1,
             param_grid={'kernel': ['linear', 'poly', 'sigmoid', 'rbf']})
```

In [266…
```
grid.best_score_
```

Out[266]: 0.7295816330213211

In [267…
```
new_svr = grid.best_params_
```

In [268…
```
new_svr
```

Out[268]: {'kernel': 'rbf'}

In [271…
```
results = {'models':['Linear', 'Ridge', 'Lasso', 'ElasticNet', 'SVR'],
           'R-Squared':[linear_reg_r2_score,ridge_reg_r2_score,lasso_reg_r2_score,elasti
           'Adjusted-R-Square':[linear_reg_adj_r2_score,ridge_reg_adj_r2_score,lasso_reg
```

In [272…
```
results = pd.DataFrame(results)
```

In [273…
```
results
```

Out[273]:

	models	R-Squared	Adjusted-R-Square
0	Linear	0.692019	0.691731
1	Ridge	0.692019	0.691732
2	Lasso	0.678234	0.677934
3	ElasticNet	0.594727	0.594349
4	SVR	0.727450	0.727196

In [ ]: