Household power consumption Regression problem

Problem Statement

- **Regression problem**
  - Collect dataset from here https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption
  - Here the number of instances is very high, so take a random sample of 50k using the `sample()`.
  - Add all the three columns named `sub_metering_1`, `sub_metering_2` and `sub_metering_3` to get the total energy consumed.
  - Create a Regression model on the basis of attributes.
  - Create `Linear Regression`, `Ridge Regression`, `Lasso Regression`, `ElasticNet Regression`, `Support Vector Regression`.
- **Steps to be followed**
  - Data ingestion.
  - EDA (end to end).
  - Preprocessing of the data.
  - Use pickle to store the scaling of the data for later use.
  - Store the final processed data inside MongoDB.
  - Again load the data from MongoDB.
  - Model building.
  - Use `GridSearchCV` for hyper parameter tuning.
  - Evaluation.
    - `R2` and `adjusted R2` for regression model.

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.

1. Data Ingestion:

1.1 Import modules and data to create dataframe

```
# Importing the required libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import pymongo

sns.set()
%matplotlib inline
warnings.filterwarnings('ignore')

try:
    read_file = pd.read_csv('dataset/household_power_consumption.txt', 
delimiter = ';')
    read_file.to_csv(r"dataset\power_consumption.csv", index=None)
except Exception as err:
    print("Error is: ", err)
else:
    print("File format converted successfully.")
```

```
File format converted successfully.
```

1.2 Creating dataframe with random 50000 observations

```
data = pd.read_csv('dataset/power_consumption.csv')
data.shape
```

```
(2075259, 9)
```

Note

- Here at the present the number of rows is very high, let's take a sample of 50000 observations

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

```
df.head()
```

```
              Date      Time Global_active_power Global_reactive_power  
\
511536   6/12/2007  23:00:00               3.118                 0.108   

1990034  28/9/2010  16:38:00                   ?                     ?   

550140    2/1/2008  18:24:00               0.628                 0.000   

1369908  25/7/2009  01:12:00               0.300                 0.248   
```

```
604155    9/2/2008  06:39:00               0.224                 0.000


         Voltage Global_intensity Sub_metering_1 Sub_metering_2  \
511536   243.110           12.800          0.000          0.000
1990034        ?                ?              ?              ?
550140   239.030            2.600          0.000          0.000
1369908  243.610            1.600          0.000          0.000
604155   243.710            0.800          0.000          0.000

         Sub_metering_3
511536             18.0
1990034             NaN
550140              0.0
1369908             1.0
604155              0.0
```

```
df.shape
```

```
(50000, 9)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 511536 to 1110791
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         49388 non-null  float64
dtypes: float64(1), object(8)
memory usage: 3.8+ MB
```

Observations:

- Now there are 50000 rows and 9 columns (features) in the dataset.
- All the columns except `Sub_metering_3` is of object type, even though they have float values.

2. Data Cleaning:

```
# Name of the columns

df.columns
```

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

2.1 Dropping unnecesarry columns

```
df.drop(columns=['Date', 'Time'], axis=1, inplace=True)
df.head()
```

```
        Global_active_power Global_reactive_power  Voltage
Global_intensity  \
511536                3.118                 0.108  243.110
12.800
1990034                   ?                     ?        ?
?
550140                0.628                 0.000  239.030
2.600
1369908               0.300                 0.248  243.610
1.600
604155                0.224                 0.000  243.710
0.800

        Sub_metering_1 Sub_metering_2  Sub_metering_3
511536           0.000          0.000            18.0
1990034              ?              ?             NaN
550140           0.000          0.000             0.0
1369908          0.000          0.000             1.0
604155           0.000          0.000             0.0
```

2.2 Converting data types and replacing special characters

```
for column in df.columns:
    print(f"The unique values in column {column}:")
    print(df[column].unique())
    print(f"\nThe number of unique values in {column} is:
{len(df[column].unique())}")
    print("----------------------------------\n")
```

```
The unique values in column Global_active_power:
['1.012' '0.248' '0.308' ... '4.282' '4.316' '5.012']

The number of unique values in Global_active_power is: 3250
----------------------------------

The unique values in column Global_reactive_power:
['0.190' '0.104' '0.000' '0.164' '0.316' '0.232' '0.046' 0.092 '0.118'
 '0.048' '0.296' '0.292' '0.090' '0.234' '0.324' '0.156' '0.208'
'0.314'
 '0.132' '0.374' '0.180' '0.236' '0.138' '0.082' '0.084' '0.214'
'0.198'
```

 '0.254' '0.204' '0.226' '0.116' '0.144' '0.148' '0.228' '0.134'
'0.094'
 '0.400' '0.058' '0.278' '?' '0.068' '0.394' '0.076' '0.072' '0.224'
 '0.274' '0.106' '0.262' '0.092' '0.096' '0.074' '0.078' '0.070'
'0.240'
 '0.114' '0.080' '0.368' '0.126' '0.124' '0.060' '0.510' '0.416'
'0.130'
 '0.056' '0.194' '0.098' '0.268' '0.064' '0.454' '0.258' 0.0 '0.052'
 '0.356' '0.220' '0.128' '0.146' '0.062' '0.110' '0.122' '0.230'
'0.332'
 '0.160' '0.222' '0.050' '0.242' '0.108' '0.174' '0.102' '0.298'
'0.362'
 '0.066' '0.172' '0.334' '0.320' '0.284' 0.15 '0.088' '0.176' 0.172
 '0.270' '0.154' 0.284 '0.100' '0.168' '0.200' '0.206' '0.322' '0.340'
 '0.210' '0.238' '0.248' '0.366' '0.336' '0.348' '0.054' '0.184'
'0.412'
 '0.276' '0.432' '0.178' '0.260' '0.196' '0.192' 0.128 '0.212' '0.452'
 '0.436' '0.218' '0.308' '0.272' '0.386' '0.244' '0.112' '0.484'
'0.382'
 '0.300' 0.242 0.158 '0.376' '0.246' '0.120' '0.286' '0.152' '0.350'
 '0.250' 0.08 '0.344' '0.216' '0.346' '0.256' '0.086' '0.354' '0.188'
 0.064 0.074 '0.266' '0.306' 0.268 '0.202' '0.162' '0.358' '0.142'
'0.136'
 '0.408' '0.280' '0.538' '0.166' '0.496' 0.114 '0.264' '0.426' '0.372'
 '0.318' '0.150' 0.062 '0.420' '0.330' '0.312' 0.124 '0.478' '0.392'
 '0.378' '0.186' '0.546' '0.370' '0.490' 0.084 0.154 '0.600' '0.338'
 '0.282' '0.380' '0.472' '0.252' 0.06 '0.446' '0.364' '0.908' 0.078
0.13
 '0.480' '0.182' '0.518' '0.170' '0.140' '0.310' 0.066 '0.462' 0.132
 '0.552' '0.290' 0.116 '0.288' '0.936' '0.390' 0.054 '0.404' '0.516'
 '0.158' '0.428' '0.304' '0.294' '0.402' 0.1 0.258 0.046 '0.342'
'0.328'
 0.096 '0.388' '0.572' 0.21 '0.476' '0.562' '0.622' '0.574' '0.556'
0.384
 '0.514' '0.326' '0.360' 0.148 '0.414' 0.098 '0.842' '0.548' 0.294
'0.302'
 '0.384' '0.540' '0.440' '0.458' '0.470' 0.058 '0.352' '0.444' '0.456'
 '0.398' '0.700' 0.218 '0.450' 0.214 '0.576' '0.488' '0.508' '0.560'
 '0.502' 0.082 0.09 0.146 '0.512' '0.582' '0.748' '0.422' '0.430'
'0.442'
 '0.424' '0.474' 0.376 '0.770' '0.448' 0.204 '0.564' 0.472 '0.396'
0.178
 '0.590' 0.056 '0.406' '0.554' 0.684 0.2 0.496 '0.672' 0.152 0.164
'0.410'
 0.298 0.244 '0.434' 0.094 0.136 '0.530' '0.668' '0.418' 0.184 0.144
0.206
 0.05 0.22 '0.640' '0.466' 0.226 '0.684' '0.498' '0.438' 0.346 '0.570'
 '0.522' 0.448 '0.708' 0.126 0.138 0.134 0.104 0.068 '0.720' 0.196
0.166
 0.086 0.108 '0.792' 0.416 '0.670' 0.208 0.122 0.23 '0.604' '0.624'

```
 '0.506' 0.052 '0.536' 0.088 '0.652' 0.266 '0.494' 0.102 '0.768' 0.118
 0.072 '0.648' '0.460' 0.26 '0.646' 0.278 '0.756' '0.666' 0.222 0.552
 '0.802' 0.246 0.112 '0.520' '0.558' 0.07 '0.786' 0.17 0.402 0.168
'0.532'
 0.048 0.272 0.12 '0.780' '0.586' '0.762' 0.322 0.182 0.224 0.276
0.076
 0.234 0.262 '0.642' '0.486' '0.772' '0.660' '0.644' '0.524' 0.25
'0.592'
 '0.482' '0.504' '0.528' '0.500' '0.588' 0.666 '0.464' 0.544 0.14
'0.578'
 0.216 '0.610' '1.036' '0.594' '0.534' 0.464 0.312 0.198 0.19 0.194
0.176
 '0.568' '0.794' '0.492' 0.65 0.286 0.326 0.458 0.188 '0.664' 0.318
 '0.598' 0.238 0.342 0.162 0.202 '0.694' 0.516 0.348 '0.818' '0.550'
 '0.698' 0.292 0.636 0.106 '0.732' 0.374 0.18 0.368 0.16 0.212 '0.612'
 '0.628' 0.314 0.186 0.236 0.332 0.504 '0.544' 0.192 '0.468' '0.608'
0.142
 '0.774' '0.596' '0.742' '0.686' 0.11 0.254 0.394 0.388 '0.618'
'0.584'
 0.252 '0.602' '0.734' 0.256 0.522 0.3 0.232 '0.580' '0.626' '0.678'
0.24
 '0.778' 0.324 0.39 '0.728' 0.38 '0.632' 0.482 0.27 '0.922' '0.616'
 '0.566' 0.288 '0.542' 0.296 0.34 '0.526' 0.174 '0.654' 0.358 0.564
0.33
 0.338 '0.730' '0.706' '0.630' '0.606' 0.248 '0.636' '0.656' 0.614
'0.836'
 0.37 '0.696' 0.304 0.274 0.398 0.484 0.35 '0.674' '0.614' '0.838'
0.476
 '0.998' 0.406 0.282 '0.790' 0.156 '0.902' '0.784' 0.308 '0.718']

The number of unique values in Global_reactive_power is: 526
----------------------------------

The unique values in column Voltage:
['240.290' '241.160' '244.180' ... 235.36 243.14 '251.750']

The number of unique values in Voltage is: 2840
----------------------------------

The unique values in column Global_intensity:
['6.000' '1.000' '1.200' '5.400' '3.200' '2.000' '9.800' '6.400' 1.4
 '5.800' '12.000' '7.800' '11.400' '0.800' '8.000' '4.000' '11.000'
 '10.800' '4.400' '8.400' '6.800' '2.200' '0.600' '20.000' '1.600'
'6.600'
 '11.200' '5.200' '1.400' '4.200' '?' '3.000' '4.600' '5.600' '1.800'
 '5.000' '17.400' '6.200' '9.400' '19.600' '9.200' '16.200' '3.800'
 '21.000' '10.000' '2.400' '7.000' '7.600' '16.800' 14.0 '15.200'
'7.400'
 37.2 '8.200' '3.600' '7.200' '15.800' '11.600' '22.600' '12.600'
'0.200'
```

```
 '12.400' 3.4 '2.600' 5.6 2.0 '28.200' '10.400' '9.000' '17.600'
'16.600'
 3.0 '3.400' '2.800' '10.200' '18.000' '8.600' '14.600' '19.200'
'9.600'
 9.4 '18.400' '20.800' 2.8 '0.400' '8.800' 10.4 '14.800' '16.400'
'14.000'
 6.2 '15.600' '10.600' '12.800' 7.2 '19.800' '20.400' '14.400'
'19.400'
 '23.000' 6.8 '18.800' 1.6 '21.600' '23.400' '13.000' '27.400' '4.800'
4.0
 1.8 '18.200' 8.8 '28.800' '24.800' '11.800' '13.200' '15.400'
'16.000'
 5.8 '12.200' 5.0 2.2 '24.400' '21.800' 4.2 '13.400' '13.800' 1.2 7.0
 '27.600' 10.0 7.4 '20.200' '13.600' 1.0 '17.800' 10.2 '19.000'
'25.200'
 '17.200' '34.200' '24.200' 2.6 13.2 '28.400' '17.000' 11.8 11.4
'31.400'
 '14.200' '15.000' 15.2 10.6 8.4 '33.200' '24.600' 6.6 '27.200' 6.0
 '24.000' '26.800' 9.2 15.4 '22.400' '34.400' 3.2 12.0 '20.600' 4.4
 '18.600' '21.200' 7.6 '29.800' '23.200' 5.4 '22.200' 11.0 '26.200'
15.0
 7.8 '25.400' '25.800' 4.8 5.2 '23.800' 2.4 '23.600' 15.6 '31.600'
 '29.600' 10.8 '21.400' '26.400' 11.6 13.0 9.6 3.6 '25.600' 6.4
'26.000'
 '22.800' 12.8 4.6 '28.000' 43.0 '34.000' '22.000' 8.2 '25.000' 16.0
9.8
 13.4 9.0 12.6 '32.600' '32.800' 8.0 '34.600' '29.000' 21.2 16.6
'26.600'
 14.4 '33.000' 3.8 14.6 24.0 8.6 '37.600' '30.800' 25.2 11.2 '33.800'
21.4
 14.2 '33.600' 12.2 '27.000' 17.8 '28.600' 17.4 27.4 '43.400' '30.400'
 '32.400' 15.8 '29.400' '29.200' '30.000' 19.6 12.4 23.4 26.4 24.8
 '34.800' '27.800']

The number of unique values in Global_intensity is: 256
----------------------------------

The unique values in column Sub_metering_1:
['11.000' '0.000' 0.0 '1.000' '9.000' '37.000' '2.000' '?' '4.000'
 '38.000' 17.0 '36.000' '24.000' '39.000' '35.000' '41.000' 23.0
'23.000'
 6.0 '40.000' '45.000' '28.000' '30.000' '13.000' '5.000' '47.000'
'3.000'
 '25.000' '43.000' '33.000' '77.000' '16.000' 1.0 '14.000' 37.0
'12.000'
 26.0 '10.000' '15.000' '42.000' '22.000' '8.000' '29.000' '55.000'
 '7.000' '31.000' 21.0 '32.000' '6.000' '19.000' '74.000' '27.000'
41.0
 '49.000' '20.000' '26.000' '56.000' 2.0 13.0 '17.000' '73.000'
'18.000'
```

```
 38.0 '44.000' '72.000' '21.000' 43.0 '34.000' '76.000' 24.0 36.0
'48.000'
 '78.000' 25.0 '75.000' 35.0 9.0 47.0 '51.000' 15.0 '70.000' 52.0
'53.000'
 22.0 '71.000' '67.000' 48.0 '65.000' 32.0 14.0 '46.000' '54.000' 18.0
 70.0]

The number of unique values in Sub_metering_1 is: 94
----------------------------------

The unique values in column Sub_metering_2:
['0.000' '2.000' '1.000' 1.0 '4.000' '39.000' '23.000' '35.000' '?'
 '36.000' 0.0 '37.000' 68.0 '6.000' '29.000' '15.000' '17.000' 2.0
'5.000'
 '21.000' '40.000' '31.000' '28.000' '32.000' '22.000' '38.000'
'3.000'
 '20.000' '18.000' '30.000' '11.000' '24.000' '53.000' '76.000'
'43.000'
 '34.000' '16.000' '55.000' '27.000' '63.000' '50.000' 17.0 '26.000'
 '25.000' '7.000' '73.000' '71.000' '10.000' '19.000' '52.000'
'75.000'
 '33.000' '42.000' '68.000' '9.000' '54.000' '14.000' '72.000'
'41.000'
 '45.000' '49.000' '66.000' 3.0 '69.000' '12.000' 5.0 '74.000' 36.0
 '8.000' 67.0 39.0 35.0 '47.000' '58.000' 4.0 24.0 '13.000' '61.000'
 '77.000' '70.000' 71.0 74.0 '62.000' '64.000' 49.0 34.0 '67.000' 38.0
 28.0 40.0 '44.000']

The number of unique values in Sub_metering_2 is: 91
----------------------------------

The unique values in column Sub_metering_3:
[ 1.  0. 18. 19.  7. 23. 17. nan 13.  9. 28. 12. 16.  6. 29. 10. 21.
20.
 11.  3.  4. 25.  2.  8. 22. 26. 30.  5. 27. 15. 31. 14. 24.]

The number of unique values in Sub_metering_3 is: 33
----------------------------------
```

Observations:

- We have special character ? in columns `Sub_metering_1`, `Sub_metering_2`, `Global_intensity`.
- Also the columns `Global_active_power` and `Voltage` have more than 1000 unique values. So we need to check for special characters in them as well.
- We have nan in `Sub_metering_3` as well.

```python
# To find special characters in these 2 columns

df.loc[df['Global_active_power'] == "?", :]
```

```
        Global_active_power Global_reactive_power Voltage
Global_intensity  \
1931603                   ?                     ?       ?
?
1618807                   ?                     ?       ?
?
1933147                   ?                     ?       ?
?
1932297                   ?                     ?       ?
?
1936817                   ?                     ?       ?
?
...                     ...                   ...     ...
...
1989633                   ?                     ?       ?
?
1619426                   ?                     ?       ?
?
1311597                   ?                     ?       ?
?
1930475                   ?                     ?       ?
?
1310964                   ?                     ?       ?
?

        Sub_metering_1 Sub_metering_2  Sub_metering_3
1931603              ?              ?             NaN
1618807              ?              ?             NaN
1933147              ?              ?             NaN
1932297              ?              ?             NaN
1936817              ?              ?             NaN
...                ...            ...             ...
1989633              ?              ?             NaN
1619426              ?              ?             NaN
1311597              ?              ?             NaN
1930475              ?              ?             NaN
1310964              ?              ?             NaN

[617 rows x 7 columns]
```

```python
df.loc[df['Voltage'] == "?", :]
```

```
        Global_active_power Global_reactive_power Voltage
Global_intensity  \
1931603                   ?                     ?       ?
?
```

```
1618807                    ?                      ?        ?
?
1933147                    ?                      ?        ?
?
1932297                    ?                      ?        ?
?
1936817                    ?                      ?        ?
?
...                      ...                    ...      ...
...
1989633                    ?                      ?        ?
?
1619426                    ?                      ?        ?
?
1311597                    ?                      ?        ?
?
1930475                    ?                      ?        ?
?
1310964                    ?                      ?        ?
?

        Sub_metering_1 Sub_metering_2  Sub_metering_3
1931603              ?              ?             NaN
1618807              ?              ?             NaN
1933147              ?              ?             NaN
1932297              ?              ?             NaN
1936817              ?              ?             NaN
...                ...            ...             ...
1989633              ?              ?             NaN
1619426              ?              ?             NaN
1311597              ?              ?             NaN
1930475              ?              ?             NaN
1310964              ?              ?             NaN

[617 rows x 7 columns]
```

- **So yes there are 617 rows where the ? is present in the dataset**
- **Also it looks like the sign appears in all the columns at the same time**
- **As the percentage of these rows is 1% of the total dataset so we can drop them**

```
# Dropping the rows

df.drop(df.loc[df['Voltage'] == "?", :].index, inplace=True)
df.shape
```

```
(49383, 7)
```

```
# Again checking

df.loc[df['Voltage'] == "?", :]
```

```
Empty DataFrame
Columns: [Global_active_power, Global_reactive_power, Voltage,
Global_intensity, Sub_metering_1, Sub_metering_2, Sub_metering_3]
Index: []
```

```
df.loc[df['Global_active_power'] == "?", :]
```

```
Empty DataFrame
Columns: [Global_active_power, Global_reactive_power, Voltage,
Global_intensity, Sub_metering_1, Sub_metering_2, Sub_metering_3]
Index: []
```

```
# Now again checking for nan values

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

```
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
dtype: int64
```

```
# Converting the data types

df = df.astype({'Global_active_power':float,
'Global_reactive_power':float, 'Voltage':float,
'Global_intensity':float,
                'Sub_metering_1':float, 'Sub_metering_2':float})
```

```
# checking the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49383 entries, 1928275 to 1332279
Data columns (total 7 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Global_active_power    49383 non-null  float64
 1   Global_reactive_power  49383 non-null  float64
 2   Voltage                49383 non-null  float64
 3   Global_intensity       49383 non-null  float64
 4   Sub_metering_1         49383 non-null  float64
 5   Sub_metering_2         49383 non-null  float64
 6   Sub_metering_3         49383 non-null  float64
dtypes: float64(7)
memory usage: 3.0 MB
```

2.3 Let's check for duplicate values

```
df[df.duplicated()]

         Global_active_power  Global_reactive_power  Voltage  \
950270                 0.360                  0.078   241.18
950265                 0.360                  0.078   241.18
1047327                0.284                  0.048   241.40
950034                 0.360                  0.078   241.18
345658                 0.102                  0.000   235.86
...                      ...                    ...      ...
845145                 0.146                  0.000   238.42
950068                 0.360                  0.078   241.18
1914248                0.158                  0.000   242.90
1008734                0.228                  0.000   244.39
872385                 0.082                  0.000   242.34

         Global_intensity  Sub_metering_1  Sub_metering_2
Sub_metering_3
950270                1.6             0.0             0.0
1.0
950265                1.6             0.0             0.0
1.0
1047327               1.2             0.0             0.0
0.0
950034                1.6             0.0             0.0
1.0
345658                0.4             0.0             0.0
0.0
...                   ...             ...             ...
...
845145                0.6             0.0             0.0
1.0
950068                1.6             0.0             0.0
1.0
1914248               0.6             0.0             0.0
0.0
1008734               1.0             0.0             0.0
0.0
872385                0.2             0.0             0.0
1.0

[194 rows x 7 columns]

# Dropping the duplicated values as well

df.drop_duplicates(inplace=True)

# Creating a new column for 'total energy consumed'
# Then removing the columns 'Sub_metering_1', 'Sub_metering_2' and
'Sub_metering_3'
```

```
df["Total_energy_consumed"] = df['Sub_metering_1'] +
df['Sub_metering_2'] + df['Sub_metering_3']
df.drop(columns=['Sub_metering_1', 'Sub_metering_2',
'Sub_metering_3'], axis=1, inplace=True)
df.head()
```

```
         Global_active_power  Global_reactive_power  Voltage  \
1928275                1.012                  0.190   240.29
1892037                0.248                  0.104   241.16
533121                 0.308                  0.000   244.18
725441                 1.314                  0.000   240.09
1078558                0.754                  0.164   243.22

         Global_intensity  Total_energy_consumed
1928275               6.0                   12.0
1892037               1.0                    2.0
533121                1.2                    0.0
725441                5.4                   18.0
1078558               3.2                    0.0
```

```
df.shape
```

```
(49189, 5)
```

3. Exploratory data analysis

3.1 Basic Profile of the data

```
# Checking the details of the dataframe

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49189 entries, 1928275 to 1332279
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Global_active_power    49189 non-null  float64
 1   Global_reactive_power  49189 non-null  float64
 2   Voltage                49189 non-null  float64
 3   Global_intensity       49189 non-null  float64
 4   Total_energy_consumed  49189 non-null  float64
dtypes: float64(5)
memory usage: 2.3 MB
```

Differentiating numerical and categorical columns

```
numerical_features = [feature for feature in df.columns if
df[feature].dtypes != 'O']
categorical_features = [feature for feature in df.columns if
df[feature].dtypes == 'O']
```

```
print(f"The number of Numerical features are: 
{len(numerical_features)}, and the column names are:\
n{numerical_features}")
print(f"\nThe number of Categorical features are: 
{len(categorical_features)}, and the column names are:\
n{categorical_features}")
```

```
The number of Numerical features are: 5, and the column names are:
['Global_active_power', 'Global_reactive_power', 'Voltage', 
'Global_intensity', 'Total_energy_consumed']

The number of Categorical features are: 0, and the column names are:
[]
```

Observations:

- Now we have 49189 rows with no null and duplicate values and all the 5 columns have numerical (float) data type.

3.2 Statistical Analysis of the data

```
df.describe().T
```

```
                         count        mean        std      min  
25%  \
Global_active_power    49189.0    1.093096   1.056170    0.078  
0.308   
Global_reactive_power  49189.0    0.124045   0.112738    0.000  
0.048   
Voltage                49189.0  240.839414   3.246788  225.720  
238.980   
Global_intensity       49189.0    4.634906   4.438016    0.200  
1.400   
Total_energy_consumed  49189.0    8.937466  12.955325    0.000  
0.000   

                           50%      75%      max  
Global_active_power      0.606    1.528    9.938  
Global_reactive_power    0.100    0.194    1.036  
Voltage                241.030  242.900  253.100  
Global_intensity         2.600    6.400   43.400  
Total_energy_consumed    1.000   18.000  126.000  
```

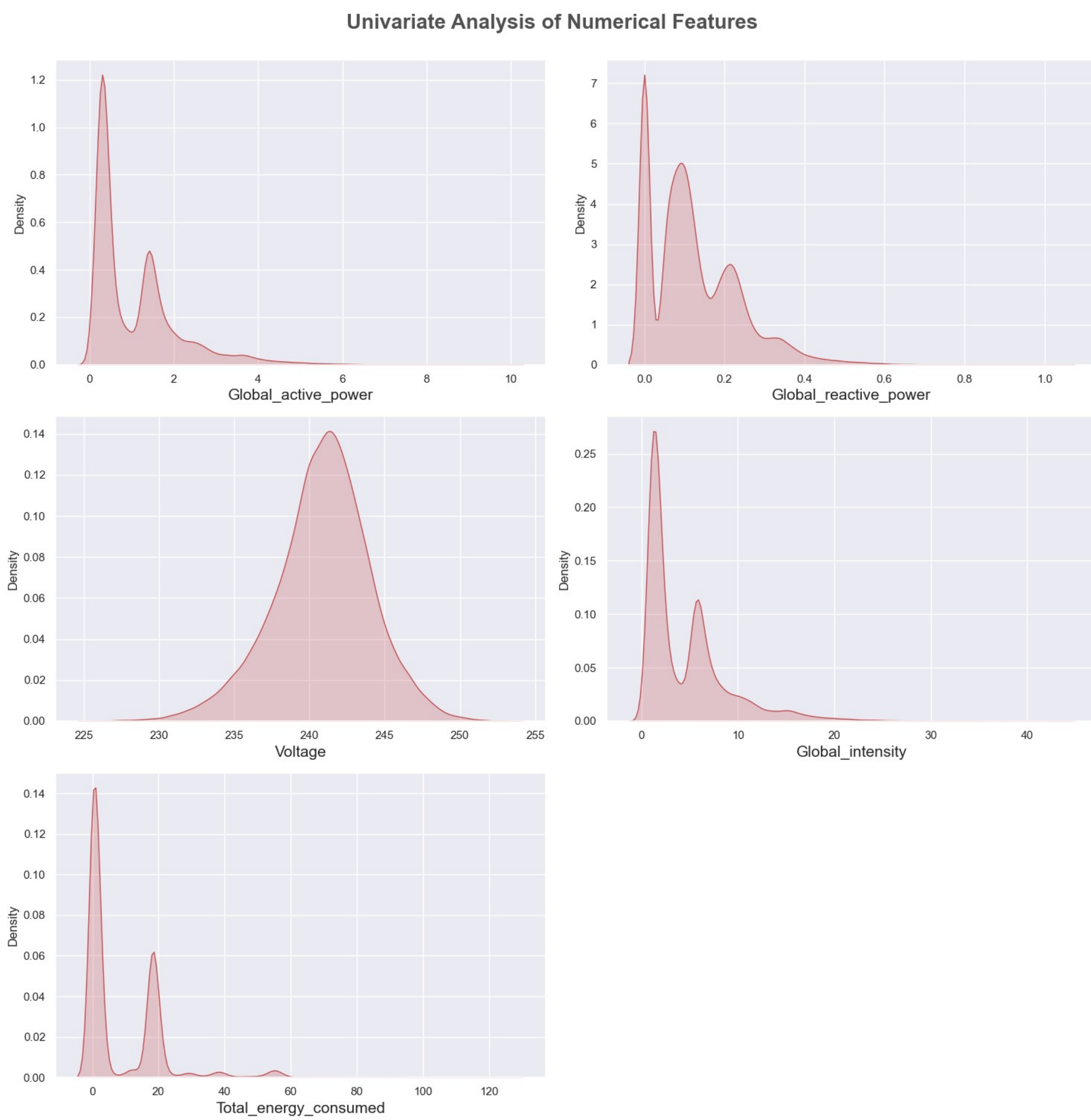
Observations:

- There are possible Outliers in columns `Global_active_power`, `Global_intensity`, `Total_energy_consumed`.

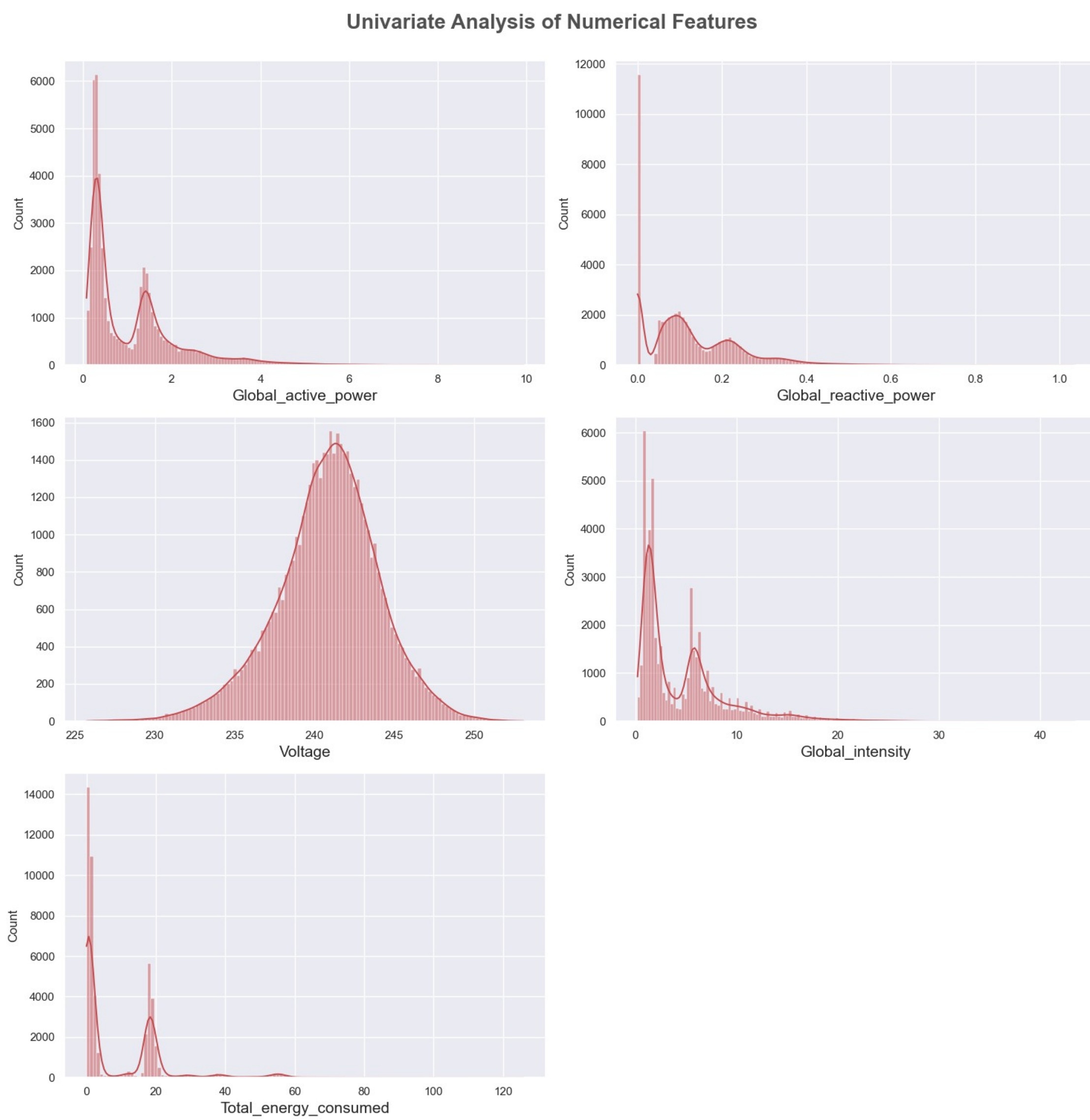
3.3 Graphical Analysis of the data

3.3.1 Univariate Analysis

```
# For numerical features
# Kernal Density plots

plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)

for i in range(0, len(numerical_features)):
    plt.subplot(3, 2, i+1)
    sns.kdeplot(x=df[numerical_features[i]], shade=True, color='r')
    plt.xlabel(numerical_features[i], fontsize=15)
    plt.tight_layout()
```



```python
# Histograms

plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)

for i in range(0, len(numerical_features)):
    plt.subplot(3, 2, i+1)
    sns.histplot(x=df[numerical_features[i]], kde=True, color='r')
    plt.xlabel(numerical_features[i], fontsize=15)
    plt.tight_layout()
```



Observations:

- Only `Voltage` has normal distribution.
- All other columns are right skewed and they may have outliers.

- Too many values near to 0 in `Global_active_power`, `Global_reactive_power`, `Global_intensity` and `Total_energy_consumed` columns.

3.3.2 Multivariate Analysis

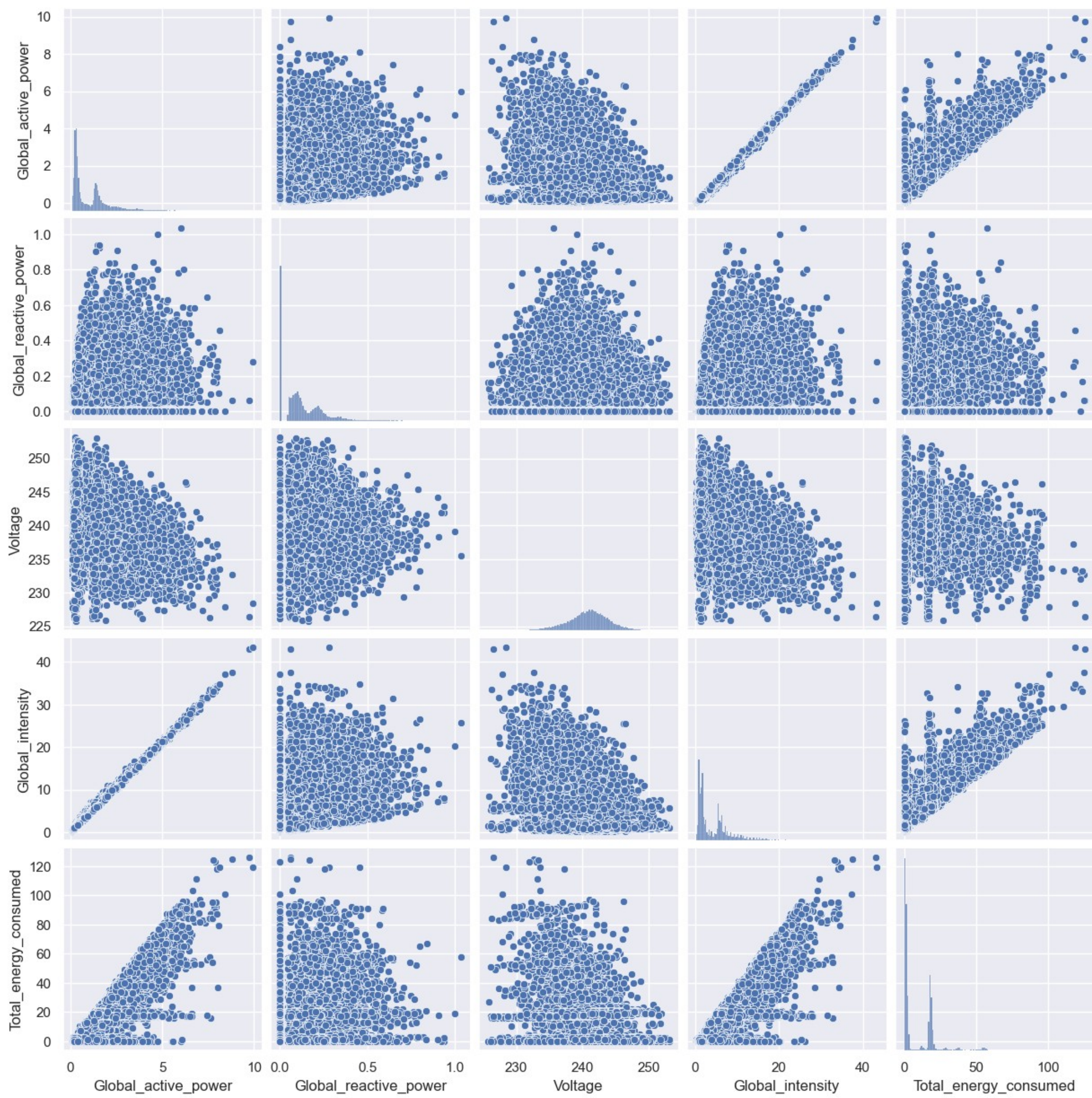
Checking Multicollinearity in the numerical features

```
df[list(df[numerical_features].columns)].corr()
```

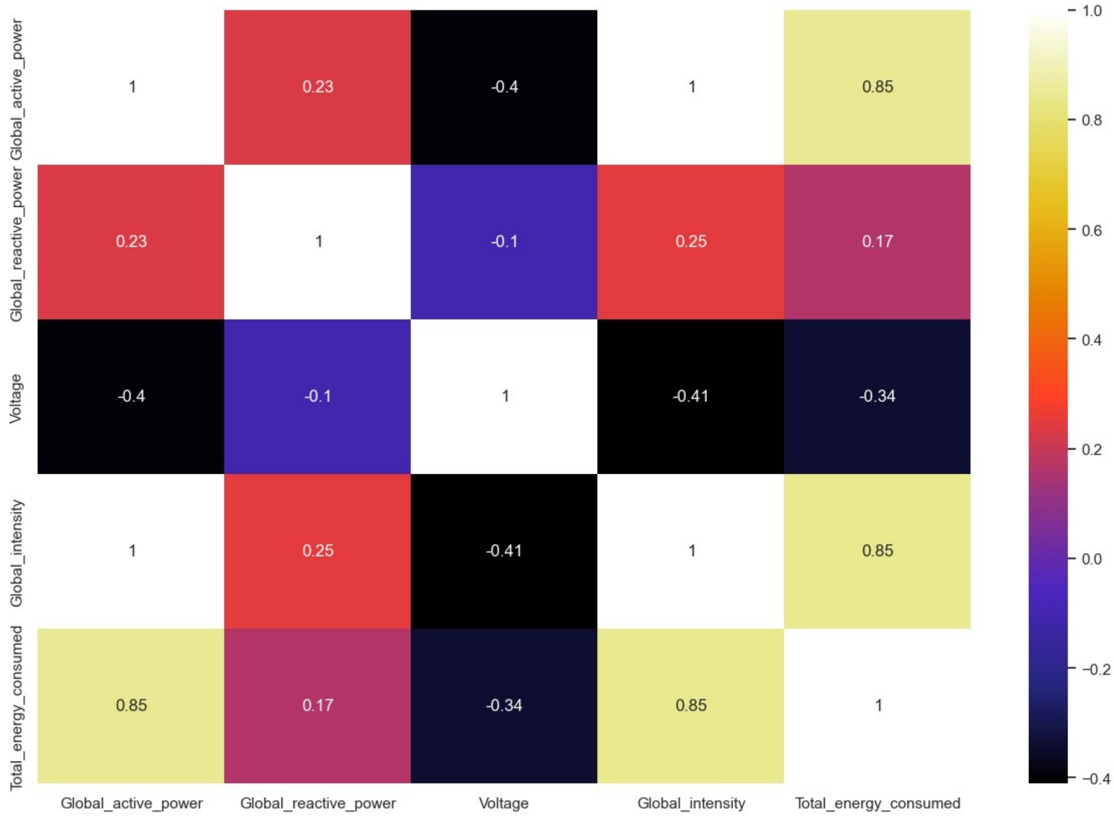
```
                       Global_active_power  Global_reactive_power
Voltage  \
Global_active_power               1.000000               0.231006 -
0.397982
Global_reactive_power             0.231006               1.000000 -
0.103893
Voltage                          -0.397982              -0.103893
1.000000
Global_intensity                  0.998883               0.249991 -
0.409457
Total_energy_consumed             0.848864               0.166474 -
0.342788

                       Global_intensity  Total_energy_consumed
Global_active_power            0.998883               0.848864
Global_reactive_power          0.249991               0.166474
Voltage                       -0.409457              -0.342788
Global_intensity               1.000000               0.845797
Total_energy_consumed          0.845797               1.000000
```

Graphical representation

```
sns.pairplot(df[numerical_features])
plt.show()
```




```
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(df[numerical_features].corr(), cmap='CMRmap', annot=True)
plt.show()
```



Observations:

- `Global_intensity` and `Global_active_power` is completely correlated.
- `Total_energy_consumed` is also highly correlated with `Global_intensity` and `Global_active_power`.

4. Data Pre-Processing

4.1 Number of unique values in each column

```
df.nunique()
```

```
Global_active_power      2659
Global_reactive_power     352
Voltage                  2097
Global_intensity          170
Total_energy_consumed     107
dtype: int64
```

4.2 Outlier handling

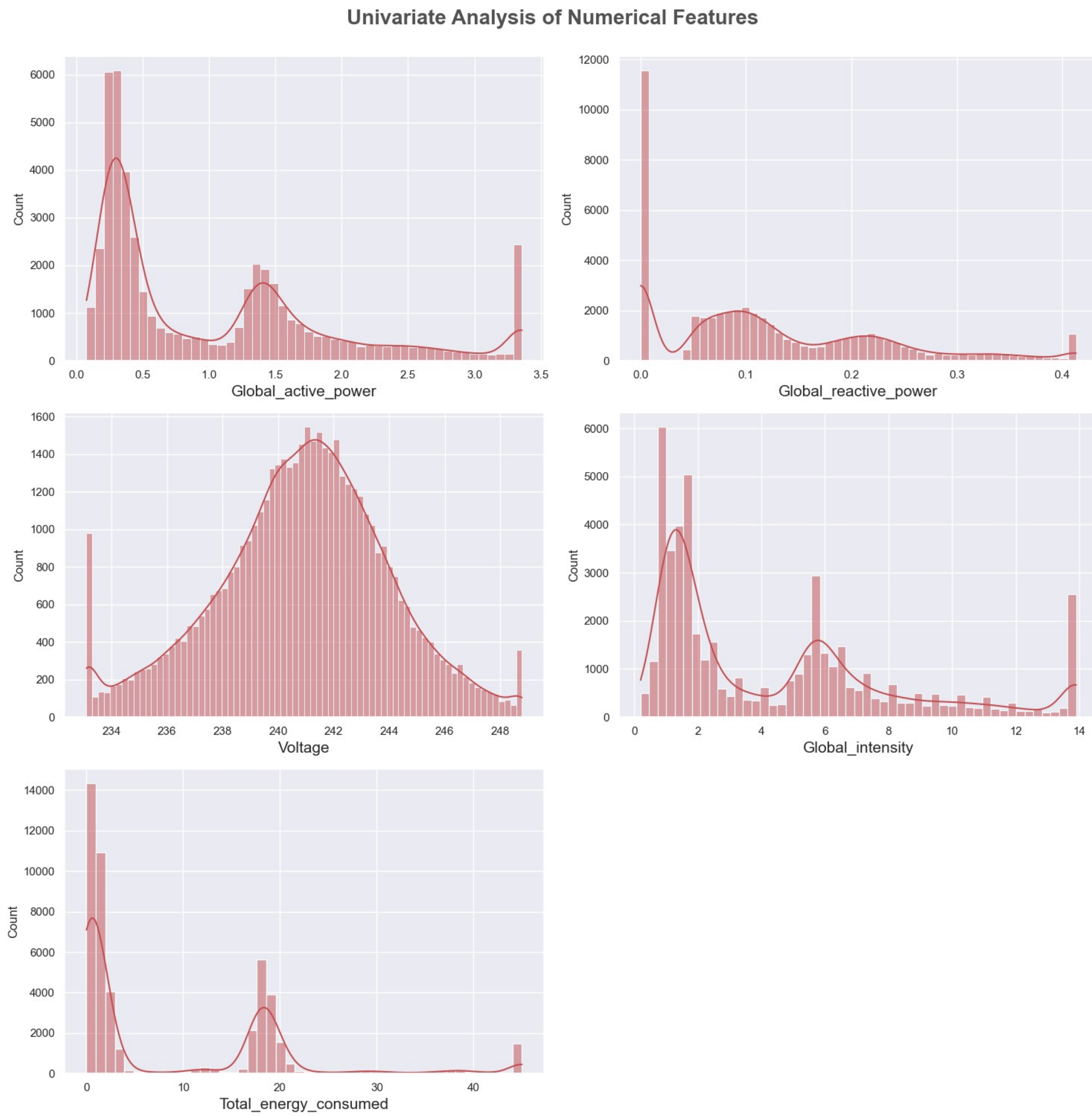
```
# Creating a function to detect outliers

def detect_outliers(col):
    percentile25 = df[col].quantile(0.25)
    percentile75 = df[col].quantile(0.75)
    print('\n ####', col , '####')
```

```python
    print("25percentile: ",percentile25)
    print("75percentile: ",percentile75)
    iqr = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    print("Upper limit: ",upper_limit)
    print("Lower limit: ",lower_limit)
    df.loc[(df[col]>upper_limit), col]= upper_limit
    df.loc[(df[col]<lower_limit), col]= lower_limit
    return df

# Now applying the function on all the columns as all are of
continupus type

for col in numerical_features:
         detect_outliers(col)
```

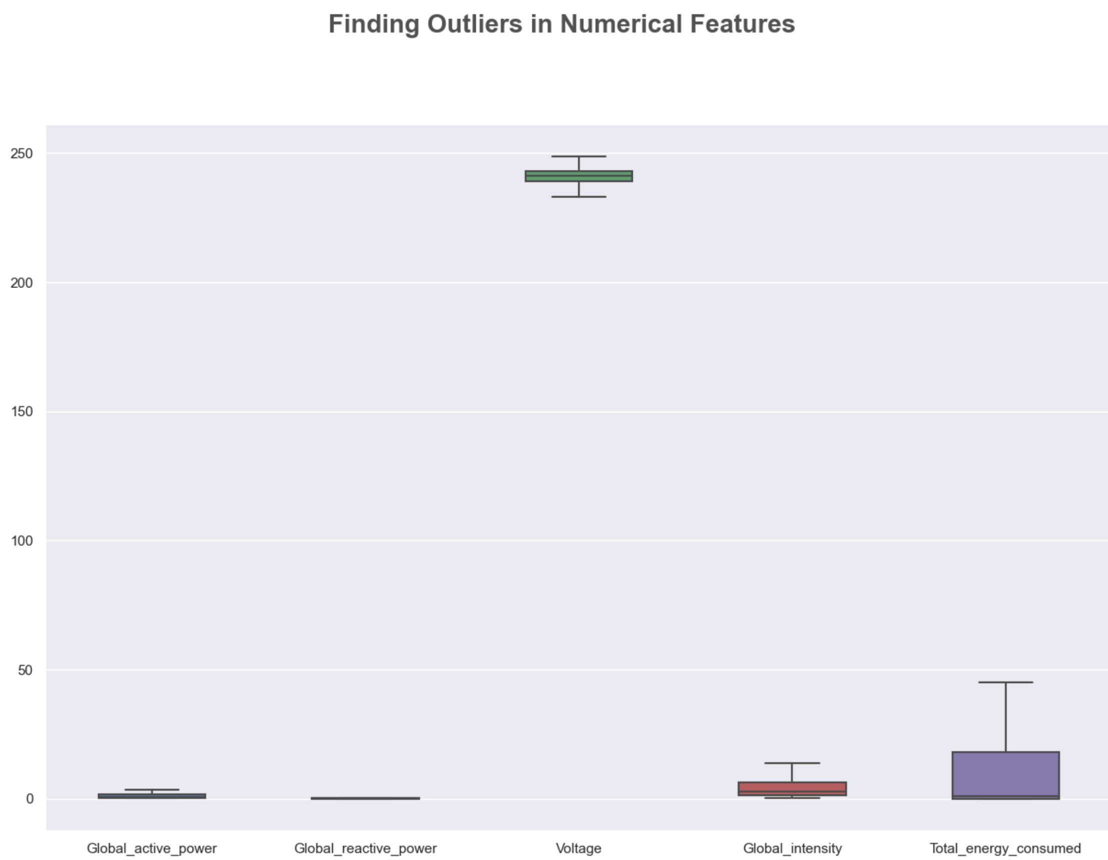
```
 #### Global_active_power ####
25percentile:  0.308
75percentile:  1.528
Upper limit:  3.358
Lower limit:  -1.522

 #### Global_reactive_power ####
25percentile:  0.048
75percentile:  0.194
Upper limit:  0.41300000000000003
Lower limit:  -0.17100000000000004

 #### Voltage ####
25percentile:  238.98
75percentile:  242.9
Upper limit:  248.78000000000003
Lower limit:  233.09999999999997

 #### Global_intensity ####
25percentile:  1.4
75percentile:  6.4
Upper limit:  13.9
Lower limit:  -6.1

 #### Total_energy_consumed ####
25percentile:  0.0
75percentile:  18.0
Upper limit:  45.0
Lower limit:  -27.0
```

```python
# Again checking
```

```
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)

for i in range(0, len(numerical_features)):
    plt.subplot(3, 2, i+1)
    sns.histplot(x=df[numerical_features[i]], kde=True, color='r')
    plt.xlabel(numerical_features[i], fontsize=15)
    plt.tight_layout()
```




```
fig, ax = plt.subplots(figsize=(15,10))
plt.suptitle('Finding Outliers in Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)
sns.boxplot(data=df[numerical_features], width= 0.5, ax=ax,
fliersize=3)
plt.show()
```



Observations:

- Now we can see that the outliers are gone.

Let's save the clean data to a folder and then to mongodb for later use

```
try:
    df.to_csv("dataset/power_consumption_cleaned.csv", index=None)
except Exception as err:
    print("Error is: ", err)
else:
    print("Clean csv file created successfully.")
```

```
Clean csv file created successfully.
```

```
# converting to json file

df2 = pd.read_csv('dataset/power_consumption_cleaned.csv')

try:
    df2.to_json('dataset/power_consumption_cleaned.json')
except Exception as err:
    print("Error is: ", err)
else:
    print("Json file created successfully.")
```

```
Json file created successfully.
```

MongoDB part

```
# Checking the file

df_json = pd.read_json('dataset/power_consumption_cleaned.json')
df_json.head()
```

```
   Global_active_power  Global_reactive_power  Voltage
Global_intensity  \
0                1.012                  0.190   240.29
6.0
1                0.248                  0.104   241.16
1.0
2                0.308                  0.000   244.18
1.2
3                1.314                  0.000   240.09
5.4
4                0.754                  0.164   243.22
3.2

   Total_energy_consumed
0                     12
1                      2
2                      0
3                     18
4                      0
```

```
# connecting with the server
try:
    client =
pymongo.MongoClient("mongodb+srv://ineuron:Project1@cluster0.rp4qzrr.m
ongodb.net/?retryWrites=true&w=majority")
except Exception as e:
    print(e)
else:
    print("Connection to MongoDB server is successful.")
```

```
Connection to MongoDB server is successful.
```

```
# creating database and collection

db = client["household_power_consumption"]
coll = db['power_consumption']

try:
    import json
except ImportError:
    import simplejson as json
```

```
# Inserting the data into the collection

try:
    with open('dataset/power_consumption_cleaned.json') as file:
        file_data = json.load(file)

        coll.insert_many([file_data])
except Exception as e:
    print(e)
else:
    print("Data inserted successfully.")
```

```
Data inserted successfully.
```

Loading the data from MongoDB

```
# Now to read the data

# importing the library to take care of the objectid created by
mongodb
import bson.json_util as json_util

results = coll.find()
try:
    for result in results:
        data = json_util.dumps(result)
        clean_df = pd.read_json(data, orient='index')
except Exception as e:
    print(e)
else:
    clean_df

clean_df
```

```
                                              $oid        0        1
2  \
_id                       63628dbb4240b6a33e820c11      NaN      NaN
NaN
Global_active_power                            NaN    1.012    0.248
0.308
Global_reactive_power                          NaN    0.190    0.104
0.000
Voltage                                        NaN  240.290  241.160
244.180
Global_intensity                               NaN    6.000    1.000
1.200
Total_energy_consumed                          NaN   12.000    2.000
0.000

                             3        4       5        6        7
8  \
```

```
_id                        NaN      NaN     NaN      NaN      NaN
NaN
Global_active_power      1.314    0.754    0.22    0.378    2.264
1.520
Global_reactive_power    0.000    0.164    0.00    0.316    0.232
0.046
Voltage                240.090  243.220  240.16  237.630  238.660
239.320
Global_intensity         5.400    3.200    1.00    2.000    9.800
6.400
Total_energy_consumed   18.000    0.000    0.00    1.000   20.000
18.000

                       ...    49179    49180   49181    49182    49183
\
_id                    ...      NaN      NaN     NaN      NaN      NaN

Global_active_power    ...    2.256    0.414    0.30    0.188    1.422

Global_reactive_power  ...    0.104    0.096    0.11    0.050    0.058

Voltage                ...  239.810  240.420  243.35  240.080  237.520

Global_intensity       ...    9.800    1.800    1.40    0.800    6.000

Total_energy_consumed  ...   30.000    0.000    1.00    1.000   17.000


                         49184    49185    49186    49187    49188
_id                        NaN      NaN      NaN      NaN      NaN
Global_active_power      1.260    0.196    1.494    0.376    1.404
Global_reactive_power    0.408    0.128    0.144    0.000    0.144
Voltage                243.280  237.060  241.470  244.820  238.840
Global_intensity         5.400    1.000    6.200    1.600    5.800
Total_energy_consumed   14.000    0.000   19.000    0.000    1.000

[6 rows x 49190 columns]
```

```
clean_df = clean_df.transpose()
clean_df.head()
```

```
                           _id Global_active_power
Global_reactive_power  \
$oid  63628dbb4240b6a33e820c11                 NaN
NaN
0                          NaN               1.012
0.19
1                          NaN               0.248
0.104
2                          NaN               0.308
```

```
0.0
3                          NaN               1.314
0.0

     Voltage Global_intensity Total_energy_consumed
$oid     NaN              NaN                   NaN
0     240.29              6.0                  12.0
1     241.16              1.0                   2.0
2     244.18              1.2                   0.0
3     240.09              5.4                  18.0
```

Removing the column _id and row oid as we donot need them

```
# Removing the '_id' column

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

```
      Global_active_power Global_reactive_power Voltage
Global_intensity  \
$oid                  NaN                   NaN     NaN
NaN
0                   1.012                  0.19  240.29
6.0
1                   0.248                 0.104  241.16
1.0
2                   0.308                   0.0  244.18
1.2
3                   1.314                   0.0  240.09
5.4
...                   ...                   ...     ...                .
..
49184                1.26                 0.408  243.28
5.4
49185               0.196                 0.128  237.06
1.0
49186               1.494                 0.144  241.47
6.2
49187               0.376                   0.0  244.82
1.6
49188               1.404                 0.144  238.84
5.8

      Total_energy_consumed
$oid                    NaN
0                      12.0
1                       2.0
2                       0.0
3                      18.0
...                     ...
```

```
49184                  14.0
49185                   0.0
49186                  19.0
49187                   0.0
49188                   1.0

[49190 rows x 5 columns]
```

```python
# Again transposing so we can get the '$oid' as a column

clean_df = clean_df.transpose()
clean_df.head()
```

```
                      $oid       0       1       2       3       4
5  \
Global_active_power    NaN   1.012   0.248   0.308   1.314   0.754
0.22
Global_reactive_power  NaN    0.19   0.104     0.0     0.0   0.164
0.0
Voltage                NaN  240.29  241.16  244.18  240.09  243.22
240.16
Global_intensity       NaN     6.0     1.0     1.2     5.4     3.2
1.0
Total_energy_consumed  NaN    12.0     2.0     0.0    18.0     0.0
0.0

                            6       7       8  ...   49179   49180
49181  \
Global_active_power     0.378   2.264    1.52  ...   2.256   0.414
0.3
Global_reactive_power   0.316   0.232   0.046  ...   0.104   0.096
0.11
Voltage                237.63  238.66  239.32  ...  239.81  240.42
243.35
Global_intensity          2.0     9.8     6.4  ...     9.8     1.8
1.4
Total_energy_consumed     1.0    20.0    18.0  ...    30.0     0.0
1.0

                        49182   49183   49184   49185   49186   49187
49188
Global_active_power     0.188   1.422    1.26   0.196   1.494   0.376
1.404
Global_reactive_power    0.05   0.058   0.408   0.128   0.144     0.0
0.144
Voltage                240.08  237.52  243.28  237.06  241.47  244.82
238.84
Global_intensity          0.8     6.0     5.4     1.0     6.2     1.6
5.8
Total_energy_consumed     1.0    17.0    14.0     0.0    19.0     0.0
```

```
1.0

[5 rows x 49190 columns]
```

```python
# Removing the '$oid' column

clean_df.drop(['$oid'], axis=1, inplace=True)
clean_df
```

```
                              0        1        2        3        4        5  
6  \
Global_active_power       1.012    0.248    0.308    1.314    0.754     0.22  
0.378
Global_reactive_power      0.19    0.104      0.0      0.0    0.164      0.0  
0.316
Voltage                  240.29   241.16   244.18   240.09   243.22   240.16  
237.63
Global_intensity            6.0      1.0      1.2      5.4      3.2      1.0  
2.0
Total_energy_consumed      12.0      2.0      0.0     18.0      0.0      0.0  
1.0

                              7        8        9  ...   49179   49180  
49181  \
Global_active_power       2.264     1.52    0.342  ...   2.256   0.414  
0.3
Global_reactive_power     0.232    0.046    0.092  ...   0.104   0.096  
0.11
Voltage                  238.66   239.32   244.61  ...  239.81  240.42  
243.35
Global_intensity            9.8      6.4      1.4  ...     9.8     1.8  
1.4
Total_energy_consumed      20.0     18.0      1.0  ...    30.0     0.0  
1.0

                          49182    49183    49184    49185    49186    49187  
49188
Global_active_power       0.188    1.422     1.26    0.196    1.494    0.376  
1.404
Global_reactive_power      0.05    0.058    0.408    0.128    0.144      0.0  
0.144
Voltage                  240.08   237.52   243.28   237.06   241.47   244.82  
238.84
Global_intensity            0.8      6.0      5.4      1.0      6.2      1.6  
5.8
Total_energy_consumed       1.0     17.0     14.0      0.0     19.0      0.0  
1.0

[5 rows x 49189 columns]
```

```python
# getting the actual dataframe

clean_df = clean_df.transpose()
clean_df.head()
```

```
  Global_active_power Global_reactive_power Voltage 
Global_intensity  \
0               1.012                  0.19  240.29             6.0   

1               0.248                 0.104  241.16             1.0   

2               0.308                   0.0  244.18             1.2   

3               1.314                   0.0  240.09             5.4   

4               0.754                 0.164  243.22             3.2   


  Total_energy_consumed  
0                  12.0  
1                   2.0  
2                   0.0  
3                  18.0  
4                   0.0  
```

```python
final_df = clean_df.copy()
final_df
```

```
      Global_active_power Global_reactive_power Voltage 
Global_intensity  \
0                   1.012                  0.19  240.29              
6.0   
1                   0.248                 0.104  241.16              
1.0   
2                   0.308                   0.0  244.18              
1.2   
3                   1.314                   0.0  240.09              
5.4   
4                   0.754                 0.164  243.22              
3.2   
...                   ...                   ...     ...              .
..   
49184                1.26                 0.408  243.28              
5.4   
49185               0.196                 0.128  237.06              
1.0   
49186               1.494                 0.144  241.47              
6.2   
49187               0.376                   0.0  244.82              
1.6   
```

```
49188                1.404                 0.144  238.84
5.8

      Total_energy_consumed
0                      12.0
1                       2.0
2                       0.0
3                      18.0
4                       0.0
...                     ...
49184                  14.0
49185                   0.0
49186                  19.0
49187                   0.0
49188                   1.0

[49189 rows x 5 columns]
```

4.3 Creating independent and dependent variables

Split X and y

- **Split Dataframe to X and y**
- **Here we set a variable X i.e, independent columns, and a variable y i.e, dependent column as the `Total_energy_consumed` column**.

```
X = final_df.drop("Total_energy_consumed", axis=1)
y = final_df["Total_energy_consumed"]
```

```
# Checking the independent and dependent variables

X.head()
```

```
  Global_active_power Global_reactive_power Voltage Global_intensity
0               1.012                  0.19  240.29              6.0
1               0.248                 0.104  241.16              1.0
2               0.308                   0.0  244.18              1.2
3               1.314                   0.0  240.09              5.4
4               0.754                 0.164  243.22              3.2
```

```
y.head()
```

```
0    12.0
1     2.0
2     0.0
3    18.0
4     0.0
Name: Total_energy_consumed, dtype: object
```

```
# Doing Test Train split

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)
```

Let's see the datasets

```
X_train.head()
```

```
      Global_active_power Global_reactive_power Voltage
Global_intensity
8331                3.358                  0.08  238.96
13.9
12276               0.184                   0.0  242.04
0.8
18649                0.33                 0.096   239.5
1.4
4127                  0.2                 0.078  242.01
1.0
20274               1.246                 0.068  241.46
5.2
```

```
y_train.head()
```

```
8331     45.0
12276     0.0
18649     2.0
4127      0.0
20274     1.0
Name: Total_energy_consumed, dtype: object
```

```
X_test.head()
```

```
      Global_active_power Global_reactive_power Voltage
Global_intensity
38057               0.308                 0.084   243.6
1.2
43453                1.78                 0.132  236.75
7.4
27271               2.044                   0.0  243.94
8.4
45630               0.452                 0.074  241.73
2.0
28595               0.226                 0.142  242.24
1.0
```

```
y_test.head()
```

```
38057     2.0
43453    18.0
27271    20.0
45630     0.0
28595     0.0
Name: Total_energy_consumed, dtype: object
```

Let's check the shapes of each datasets

```
X_train.shape
```

```
(32956, 4)
```

```
X_test.shape
```

```
(16233, 4)
```

Observations:

- So now we have 32956 rows for training and 16233 for test datasets.

4.4 Standardizing or feature scaling the dataset

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler
```

```
StandardScaler()
```

```
# calculate the mean and std dev
# Here we are fitting only the training data without transforming

scale = scaler.fit(X_train)
scale
```

```
StandardScaler()
```

```
# Printing the mean

print(scale.mean_)
```

```
[1.05394908e+00 1.22367004e-01 2.40849102e+02 4.45267326e+00]
```

Saving the `scale` to use it later to transform the data and predict the values

```
# To save a Standard scaler object
import pickle

with open('scaled.pkl', 'wb') as f:
    pickle.dump(scale, f)
```

```
# Loading the scaled object to transform the data

with open('scaled.pkl', 'rb') as f:
    scaled = pickle.load(f)
```

```
# Now transforming the train and test dataset

X_train_tf = scaled.transform(X_train)
X_test_tf = scaled.transform(X_test)
```

```
# Checking the transformed data

X_train_tf

array([[ 2.51411749, -0.39834171, -0.60179369,  2.48867721],
       [-0.9492647 , -1.15051518,  0.3793731 , -0.96221132],
       [-0.78995349, -0.24790702, -0.42977094, -0.80415535],
       ...,
       [-0.30983749,  0.25981008,  2.52647185, -0.38267279],
       [ 0.24884294, -1.15051518, -0.71010431,  0.24955107],
       [-0.80741225, -0.04105931,  0.02258517, -0.80415535]])

X_test_tf

array([[-0.81395928, -0.36073304,  0.8763277 , -0.85684068],
       [ 0.79224695,  0.09057105, -1.30581272,  0.77640428],
       [ 1.08031654, -1.15051518,  0.98463832,  1.03983088],
       ...,
       [-0.9492647 , -1.15051518,  0.15000943, -0.96221132],
       [-0.85105916, -1.15051518,  0.93048301, -0.85684068],
       [-0.74412423,  0.86154885, -0.2577482 , -0.69878471]])
```

5. Model Building

5.1 Import required packages for model training

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso,
ElasticNet
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
```

5.2 Create a Function to evaluate all the models

```
def evaluate_model(true, predicted, X_test_tf):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    adj_r2 = 1 - (1 - r2_square)*(len(true)-1)/(len(true) -
X_test_tf.shape[1] - 1)
    return mae, rmse, r2_square, adj_r2

models = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(),
    "Ridge": Ridge(),
    "Elastic": ElasticNet(),
    "svr": SVR()
}

model_list = []
r2_list =[]
```

```
adj_r2_list = []

for i in range(len(list(models))):
    model = list(models.values())[i]

    # Train model
    model.fit(X_train_tf, y_train)

    # Make predictions
    y_train_pred = model.predict(X_train_tf)
    y_test_pred = model.predict(X_test_tf)

    # Evaluate Train and Test dataset
    model_train_mae , model_train_rmse, model_train_r2,
model_train_adjusted_r2 = evaluate_model(y_train, y_train_pred,
X_test_tf)

    model_test_mae , model_test_rmse, model_test_r2,
model_test_adjusted_r2 = evaluate_model(y_test, y_test_pred,
X_test_tf)


    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error:
{:.4f}".format(model_train_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))
    print("- Adjusted R2 Score:
{:.4f}".format(model_train_adjusted_r2))

    print('----------------------------------')

    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".format(model_test_r2))
    print("- Adjusted R2 Score:
{:.4f}".format(model_test_adjusted_r2))
    r2_list.append(model_test_r2)
    adj_r2_list.append(model_test_adjusted_r2)

    print('='*50)
    print('\n')

Linear Regression
Model performance for Training set
- Root Mean Squared Error: 6.2380
```

```
- Mean Absolute Error: 4.1482
- R2 Score: 0.6999
- Adjusted R2 Score: 0.6999
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.1659
- Mean Absolute Error: 4.0543
- R2 Score: 0.7061
- Adjusted R2 Score: 0.7060
===================================================


Lasso
Model performance for Training set
- Root Mean Squared Error: 6.3860
- Mean Absolute Error: 4.4077
- R2 Score: 0.6855
- Adjusted R2 Score: 0.6855
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.3039
- Mean Absolute Error: 4.3113
- R2 Score: 0.6928
- Adjusted R2 Score: 0.6927
===================================================


Ridge
Model performance for Training set
- Root Mean Squared Error: 6.2381
- Mean Absolute Error: 4.1472
- R2 Score: 0.6999
- Adjusted R2 Score: 0.6999
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.1656
- Mean Absolute Error: 4.0530
- R2 Score: 0.7061
- Adjusted R2 Score: 0.7060
===================================================


Elastic
Model performance for Training set
- Root Mean Squared Error: 6.7325
- Mean Absolute Error: 4.9730
- R2 Score: 0.6504
- Adjusted R2 Score: 0.6504
----------------------------------
Model performance for Test set
```

```
- Root Mean Squared Error: 6.6634
- Mean Absolute Error: 4.8884
- R2 Score: 0.6567
- Adjusted R2 Score: 0.6567
==================================================


svr
Model performance for Training set
- Root Mean Squared Error: 6.1315
- Mean Absolute Error: 3.0420
- R2 Score: 0.7101
- Adjusted R2 Score: 0.7100
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.1046
- Mean Absolute Error: 2.9965
- R2 Score: 0.7119
- Adjusted R2 Score: 0.7118
==================================================
```

5.3 Results of all Models

```
adj_r2_list
```

```
[0.7059998203542576,
 0.6926925692846377,
 0.7060327002780958,
 0.6566502290168923,
 0.7118220744716512]
```

```
pd.DataFrame(list(zip(model_list, r2_list, adj_r2_list)),
             columns=['Model Name', 'R2_Score', 'Adjusted 
R2_Score']).sort_values(by=["R2_Score"], ascending=False)
```

	Model Name	R2_Score	Adjusted R2_Score
4	svr	0.711893	0.711822
2	Ridge	0.706105	0.706033
0	Linear Regression	0.706072	0.706000
1	Lasso	0.692768	0.692693
3	Elastic	0.656735	0.656650

Observations:

- We can see the best Adjusted R2 value is of the SVR model but the SVR, `Ridge`, `Linear Regression` models are also very close.
- So now we can use SVR for Hyper Parameter Tuning to find it's best values.

5.4 Hyper Parameter Tuning (using GridSearchCV)

```
# importing the library

from sklearn.model_selection import GridSearchCV
```

```
# Creating the svr model

svr = SVR()
svr
```

```
SVR()
```

```
# training the model

svr.fit(X_train_tf, y_train)
```

```
SVR()
```

```
params = {'kernel':('linear', 'rbf')}

grid = GridSearchCV(estimator=svr, param_grid=params, cv=3, verbose=2,
n_jobs=-1)
grid.fit(X_train_tf, y_train)
print(grid.best_params_)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
{'kernel': 'rbf'}
```

5.5 Training the model with best Parameters

```
best_model = SVR(kernel='rbf')
best_model
```

```
SVR()
```

```
# training the model

best_model.fit(X_train_tf, y_train)
```

```
SVR()
```

5.6 Saving the optimized model for later usage

```
# saving the model

with open('model.pkl', 'wb') as f:
    pickle.dump(best_model, f)
```

Testing the model with new data to get prediction

```
# Inserting data from outside

Global_active_power = float(input("Enter The value: "))
Global_reactive_power = float(input("Enter The value: "))
```

```
Voltage = float(input("Enter The value: "))
Global_intensity = float(input("Enter The value: "))

test_set = {'Global_active_power': Global_active_power,
'Global_reactive_power': Global_reactive_power,
            'Voltage':Voltage, 'Global_intensity':Global_intensity}
print("\nThe entered values are:\n")
print(test_set)
```

```
Enter The value: 0.300
Enter The value: 0.248
Enter The value: 243.610
Enter The value: 1.600

The entered values are:

{'Global_active_power': 0.3, 'Global_reactive_power': 0.248,
'Voltage': 243.61, 'Global_intensity': 1.6}
```

```
# creating a dataframe of the entered data

test_set = pd.DataFrame(test_set, index=[1])
test_set
```

```
   Global_active_power  Global_reactive_power  Voltage
Global_intensity
1                  0.3                  0.248   243.61
1.6
```

```
test_set.shape
```

```
(1, 4)
```

```
# Loading the scaled object to transform the entered data

with open('scaled.pkl', 'rb') as f:
    scaled = pickle.load(f)
```

```
# Now transforming the enetered data

test_set_tf = scaled.transform(test_set)
test_set_tf
```

```
array([[-0.82268867,  1.18122257,  0.87951331, -0.75147003]])
```

```
# loading the model

with open('model.pkl', 'rb') as f:
    new_model = pickle.load(f)
```

```
# predicting the output
```

```
test_set_pred = new_model.predict(test_set_tf)
print("So total energy consumed by the entered data will be:
{:.2f}".format(float(test_set_pred)))
```

```
So total energy consumed by the entered data will be: 1.54
```