Household power consumption Regression problem

- **Have to delete the dataset folder as the folder is too big to upload in github.**

Problem Statement:

- *Using `bagging regressor`, `extra tree regressor`, `voting regressor` and `random forest regressor` on the dataset to find the best 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 collection

1.1 Import modules and create the 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 as 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  \
574143    19/1/2008  10:27:00               1.376
0.080
1728774   31/3/2010  06:18:00               1.384
0.096
1522856   8/11/2009  06:20:00               0.224
0.000
1218182   10/4/2009  16:26:00               0.370
0.128
20796    31/12/2006  04:00:00               0.216
0.000

         Voltage Global_intensity Sub_metering_1 Sub_metering_2  \
574143   239.340            5.600          0.000          1.000
1728774  242.140            5.600          0.000          1.000
1522856  244.440            0.800          0.000          0.000
1218182  244.310            1.600          0.000          0.000
20796    244.750            1.000          0.000          0.000

         Sub_metering_3
574143             17.0
1728774            18.0
1522856             1.0
1218182             1.0
20796               0.0
```

```
df.shape
```

```
(50000, 9)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 574143 to 700930
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         49392 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 some of them 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 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 Date:
['19/1/2008' '31/3/2010' '8/11/2009' ... '22/7/2007' '30/12/2008'
 '24/2/2007']

The number of unique values in Date is: 1442
---------------------------------

The unique values in column Time:
```

```
['10:27:00' '06:18:00' '06:20:00' ... '14:19:00' '06:42:00'
'10:41:00']

The number of unique values in Time is: 1440
---------------------------------

The unique values in column Global_active_power:
['1.376' '1.384' '0.224' ... 0.794 '4.056' '4.226']

The number of unique values in Global_active_power is: 3230
---------------------------------

The unique values in column Global_reactive_power:
['0.080' '0.096' '0.000' '0.128' '0.242' '0.132' '0.048' '0.102'
'0.348'
 '0.174' '0.072' '0.266' '0.124' '0.220' '0.426' '0.206' '0.092'
'0.196'
 '0.248' '0.118' '0.070' '0.120' '0.240' '?' '0.218' '0.180' '0.074'
 '0.164' '0.224' '0.062' '0.336' '0.346' 0.048 '0.234' 0.0 '0.106'
'0.082'
 '0.144' '0.150' '0.316' '0.056' '0.354' '0.332' '0.094' '0.104'
'0.320'
 '0.308' '0.270' '0.396' '0.296' '0.046' '0.226' '0.178' '0.188'
'0.084'
 '0.136' '0.176' '0.288' '0.166' '0.260' '0.214' '0.076' '0.100'
'0.088'
 '0.108' '0.078' '0.204' '0.086' '0.114' '0.198' '0.098' '0.262'
'0.222'
 '0.112' '0.250' '0.378' '0.122' '0.238' '0.626' 0.054 '0.394' '0.130'
 '0.328' '0.212' '0.246' '0.064' 0.074 '0.398' '0.216' '0.068' '0.140'
 '0.162' '0.050' '0.184' '0.282' '0.244' '0.172' '0.152' '0.148'
'0.052'
 '0.060' '0.304' '0.210' '0.254' '0.182' '0.156' '0.192' '0.146'
'0.252'
 '0.058' '0.090' '0.340' '0.306' 0.248 '0.126' '0.390' '0.352' '0.280'
 '0.168' '0.622' '0.428' '0.066' '0.054' '0.170' '0.554' '0.386'
'0.110'
 0.164 '0.202' '0.228' '0.258' '0.278' '0.330' '0.116' '0.276' '0.154'
 '0.466' '0.142' '0.324' '0.452' 0.218 '0.230' '0.134' '0.388' '0.190'
 '0.368' 0.142 '0.310' '0.284' '0.430' '0.492' '0.138' 0.08 '0.194'
 '0.186' '0.358' '0.444' '0.232' '0.548' '0.318' '0.274' '0.264'
'0.438'
 '0.618' '0.200' '0.342' '0.392' '0.294' '0.300' '0.454' '0.682' 0.086
 '0.326' '0.412' '0.528' '0.208' '0.604' '0.350' '0.370' '0.344'
'0.476'
 0.192 '0.384' '0.652' '0.314' 0.11 '0.298' 0.06 0.346 '0.312' '0.514'
 '0.630' '0.364' 0.184 '0.424' '0.616' '0.534' '0.380' 0.066 '0.292'
 '0.800' '0.410' '0.684' '0.272' '0.236' 0.134 '0.480' '0.356' '0.338'
 '0.500' 0.072 '0.160' '0.334' 0.18 '0.606' '0.256' '0.360' '0.404'
 '0.290' '0.372' '0.516' 0.12 '0.322' '0.472' '0.482' '0.474' '0.366'
```

```
 '0.158' '0.362' '0.510' '0.628' 0.262 '0.432' '0.414' 0.172 '0.478'
 '0.552' 0.076 '0.598' '0.778' 0.064 0.166 0.122 '0.268' '0.524'
'0.584'
 '0.408' '0.402' '0.286' '0.468' '0.436' 0.068 '0.302' '0.520' '0.496'
 '0.540' '0.416' '0.382' 0.33 0.3 0.082 0.108 0.322 '0.418' 0.052
0.138
 0.056 '0.600' 0.1 0.062 0.182 0.078 '0.502' 0.096 0.382 '0.462'
'0.508'
 0.32 0.208 '0.448' '0.570' '0.464' 0.058 0.232 0.146 0.102 0.278
'0.450'
 0.168 '0.736' 0.176 0.046 '0.442' 0.088 '0.526' 0.276 '0.440' 0.292
 '0.806' '0.374' 0.394 '0.688' '0.406' '0.506' '0.488' '0.376' 0.106
 '0.400' 0.216 0.084 0.65 0.206 0.05 '0.568' 0.48 0.356 '0.608' 0.228
 '0.578' '0.434' 0.092 0.392 0.212 '0.592' 0.144 0.214 0.186 '0.572'
0.09
 '0.420' 0.26 '0.580' '0.982' '0.644' '0.530' 0.126 0.07 '0.624'
'0.798'
 0.46 '0.562' '0.656' '0.560' 0.52 '0.690' '0.498' '0.470' 0.238 0.152
 0.132 0.328 '0.654' 0.286 0.284 '0.422' 0.35 0.272 0.244 '0.532'
0.118
 '0.460' '0.586' 0.376 '0.614' '0.494' '0.490' 0.196 '0.808' '0.822'
 '0.718' 0.136 '0.646' 0.296 0.098 0.104 '0.512' 0.124 0.112 0.21
0.306
 '0.802' 0.242 0.162 '0.840' 0.4 '0.484' 0.13 '0.780' '0.590' 0.14
'0.556'
 0.188 0.226 0.39 '0.752' '0.794' '0.456' '0.546' '0.724' 0.546
'0.784'
 0.178 '0.446' 0.434 0.342 '0.676' '0.522' '0.542' 0.308 0.386 0.2
0.174
 0.258 0.116 '0.550' 0.158 '0.716' '0.854' '0.612' 0.094 '0.574' 0.202
 '0.558' '0.658' '0.804' 0.148 '0.668' '0.504' '0.732' 0.54 '0.680'
 '0.792' '0.582' 0.304 '0.458' '0.730' 0.224 0.456 0.738 '0.486' 0.352
 '0.544' 0.198 0.586 0.236 '0.518' '0.712' '0.678' 0.246 '0.734'
'0.672'
 '0.726' 0.554 '0.632' 0.27 0.156 '0.620' 0.266 0.274 0.23 0.234 0.312
 0.412 '0.694' 0.316 0.398 0.358 '0.538' '0.698' 0.128 0.19 0.222 0.15
 0.22 0.298 '0.662' 0.114 '0.642' 0.41 0.504 0.59 0.154 0.566 '0.836'
 0.254 0.16 '0.754' '0.634' 0.406 '0.566' '0.944' '0.748' 0.302 0.204
 0.622 '0.650' 0.402 0.578 '0.596' '0.640' '0.782' '0.872' 0.34 0.252
 0.256 0.384 0.362 0.516 '0.602' '0.706' '0.818' '0.876' 0.726 '0.770'
 '0.576' '0.810' '0.660' 0.454 '0.912' '0.856' '0.588' '0.768' '0.536'
 '1.014' '0.670' 0.396 0.194 0.368 '0.972' '0.756' '0.696' 0.25
'0.710']

The number of unique values in Global_reactive_power is: 533
---------------------------------

The unique values in column Voltage:
['239.340' '242.140' '244.440' ... '226.740' 245.04 246.23]
```

```
The number of unique values in Voltage is: 2834
---------------------------------

The unique values in column Global_intensity:
['5.600' '0.800' '1.600' '1.000' '5.200' '1.400' '2.400' '6.000'
'1.800'
 '24.200' '11.200' '6.800' '6.400' '2.000' '15.200' '3.600' '1.200'
 '10.000' '2.200' '9.400' '0.600' '?' '6.600' '17.600' '2.800'
'14.200'
 1.4 6.6 '6.200' '4.600' '12.000' '5.000' '4.200' '4.000' '7.800'
'8.200'
 '16.000' '2.600' '7.000' '5.800' '5.400' 5.6 '12.400' '10.600'
'8.800'
 '7.600' '7.400' '8.400' '20.200' '14.000' 5.8 '17.000' '11.000' 4.4
 '16.600' '4.800' '20.400' '3.000' '11.400' '8.600' '11.800' '21.400'
 '3.800' 2.0 '8.000' '0.400' '9.600' '10.200' '12.800' '14.400'
'7.200'
 '16.800' 7.4 '13.200' '14.600' '13.400' '16.400' '10.800' '3.400' 1.0
 '12.600' '9.200' 1.6 '3.200' '15.600' '9.800' '0.200' '13.000'
'19.600'
 '15.000' '16.200' '4.400' '13.800' '10.400' '9.000' '22.800' '13.600'
 '12.200' '15.400' '18.000' '23.000' '26.200' 6.2 21.8 5.4 '15.800'
 '11.600' 9.4 '24.800' '19.400' '19.800' 7.0 8.0 '23.200' 5.2 '22.400'
 '14.800' 11.4 '18.400' '22.600' '23.400' 14.6 '21.000' '17.800'
'24.600'
 1.8 '22.000' '26.000' '19.200' '18.600' 4.6 '33.000' '25.800'
'31.800'
 '26.600' '27.000' 7.2 '18.800' '21.600' 4.0 3.6 '17.400' 10.0
'28.800'
 4.8 2.8 '21.200' 11.2 4.2 3.0 9.0 '21.800' '24.000' '19.000' 2.4 2.6
 '20.600' '27.400' 9.2 15.0 '22.200' 11.8 6.0 2.2 '17.200' '26.400'
 '20.000' '18.200' 12.8 '25.000' '32.000' 3.4 9.8 1.2 25.4 6.8 11.0
 '28.400' '27.200' 9.6 12.0 '26.800' '24.400' 6.4 '20.800' 18.4
'25.200'
 10.6 '23.800' 8.4 '27.600' 7.6 7.8 10.4 '36.400' 10.2 3.8 '32.400'
14.8
 11.6 21.2 3.2 5.0 '28.000' 12.2 '36.000' '33.200' 19.4 '29.600' 16.8
 '33.600' 8.2 22.8 '23.600' 8.8 '28.200' '31.400' '25.400' '27.800'
15.4
 '30.200' '31.600' 10.8 '29.200' 16.6 12.6 8.6 '25.600' 21.4 '30.000'
12.4
 20.0 19.2 '29.800' 20.4 16.0 '29.000' 15.6 '33.400' '31.000' 23.8
18.6
 15.2 13.0 15.8 '35.600' 17.8 14.4 '30.400' 18.2 14.2 '32.600'
'32.200'
 '35.200' 25.0 '28.600' '30.800' 13.8 26.0 13.2 '34.800' 14.0 '34.000'
 '34.200' 21.0 24.2]

The number of unique values in Global_intensity is: 266
---------------------------------
```

```
The unique values in column Sub_metering_1:
['0.000' '38.000' '?' 0.0 '40.000' '1.000' '19.000' '39.000' '2.000'
 '13.000' '37.000' '11.000' 1.0 '36.000' '3.000' '31.000' '12.000'
 '30.000' '4.000' '35.000' '73.000' '9.000' '20.000' '33.000' '8.000'
 '27.000' '32.000' '26.000' '82.000' '17.000' '25.000' '42.000'
'45.000'
 '29.000' 23.0 '22.000' '44.000' '7.000' '18.000' 21.0 '14.000'
'24.000'
 '5.000' 31.0 '34.000' '15.000' '58.000' 8.0 '72.000' 37.0 33.0 2.0
 '10.000' 11.0 '75.000' '43.000' 16.0 '48.000' '6.000' '28.000'
'46.000'
 '53.000' 53.0 '76.000' '16.000' 38.0 '23.000' '41.000' 13.0 '55.000'
 '51.000' 46.0 '21.000' 35.0 '79.000' '47.000' 12.0 36.0 15.0 14.0
 '56.000' 19.0 '60.000' '50.000' '67.000' '64.000' 50.0 26.0 '78.000'
48.0]

The number of unique values in Sub_metering_1 is: 90
----------------------------------

The unique values in column Sub_metering_2:
['1.000' '0.000' '2.000' '5.000' '39.000' '?' 0.0 '36.000' '3.000' 2.0
 '38.000' 1.0 '26.000' '25.000' '37.000' '4.000' '40.000' '74.000'
'8.000'
 '19.000' '31.000' 35.0 '21.000' '27.000' '24.000' '18.000' '35.000'
 '10.000' 36.0 '73.000' '30.000' '64.000' '32.000' '17.000' '70.000'
 '23.000' '65.000' '28.000' '42.000' '33.000' '9.000' '59.000'
'75.000'
 '6.000' '34.000' '12.000' '41.000' 4.0 '22.000' '16.000' '7.000'
'67.000'
 '20.000' 73.0 '29.000' '43.000' '13.000' '72.000' '66.000' '63.000'
3.0
 '15.000' '14.000' '69.000' '58.000' 40.0 '71.000' 39.0 '49.000'
'53.000'
 '11.000' '68.000' '56.000' '47.000' 20.0 38.0 '76.000' 24.0 '57.000'
27.0
 26.0 28.0 13.0 '44.000' '52.000' 5.0 6.0 '60.000' 15.0 '62.000' 71.0]

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

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

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.

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

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

```
              Date      Time Global_active_power Global_reactive_power
\
1985194  25/9/2010  07:58:00                   ?                     ?

1936781  22/8/2010  17:05:00                   ?                     ?

191464   28/4/2007  16:28:00                   ?                     ?

1310491  13/6/2009  18:55:00                   ?                     ?

1712811  20/3/2010  04:15:00                   ?                     ?

...            ...       ...                 ...                   ...

1312215  14/6/2009  23:39:00                   ?                     ?

192585   29/4/2007  11:09:00                   ?                     ?

1617706  13/1/2010  03:10:00                   ?                     ?

1617058  12/1/2010  16:22:00                   ?                     ?

190851   28/4/2007  06:15:00                   ?                     ?


        Voltage Global_intensity Sub_metering_1 Sub_metering_2
Sub_metering_3
1985194       ?                ?              ?              ?
NaN
1936781       ?                ?              ?              ?
NaN
191464        ?                ?              ?              ?
NaN
1310491       ?                ?              ?              ?
NaN
1712811       ?                ?              ?              ?
NaN
```

```
...         ...               ...             ...             ...
...
1312215       ?                 ?               ?               ?
NaN
192585        ?                 ?               ?               ?
NaN
1617706       ?                 ?               ?               ?
NaN
1617058       ?                 ?               ?               ?
NaN
190851        ?                 ?               ?               ?
NaN

[608 rows x 9 columns]
```

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

```
              Date      Time Global_active_power Global_reactive_power
\
1985194  25/9/2010  07:58:00                  ?                     ?

1936781  22/8/2010  17:05:00                  ?                     ?

191464   28/4/2007  16:28:00                  ?                     ?

1310491  13/6/2009  18:55:00                  ?                     ?

1712811  20/3/2010  04:15:00                  ?                     ?

...            ...       ...                ...                   ...

1312215  14/6/2009  23:39:00                  ?                     ?

192585   29/4/2007  11:09:00                  ?                     ?

1617706  13/1/2010  03:10:00                  ?                     ?

1617058  12/1/2010  16:22:00                  ?                     ?

190851   28/4/2007  06:15:00                  ?                     ?


        Voltage Global_intensity Sub_metering_1 Sub_metering_2
Sub_metering_3
1985194       ?                 ?               ?               ?
NaN
1936781       ?                 ?               ?               ?
NaN
191464        ?                 ?               ?               ?
```

```
NaN
1310491       ?                ?              ?              ?
NaN
1712811       ?                ?              ?              ?
NaN
...         ...              ...            ...            ...
...
1312215       ?                ?              ?              ?
NaN
192585        ?                ?              ?              ?
NaN
1617706       ?                ?              ?              ?
NaN
1617058       ?                ?              ?              ?
NaN
190851        ?                ?              ?              ?
NaN

[608 rows x 9 columns]
```

- **So yes there are 608 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 around 1% of the total dataset so we can drop them**

```
# Dropping the rows

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

```
(49392, 9)
```

```
# Now again checking for nan values

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

```
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
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: 49392 entries, 574143 to 700930
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   49392 non-null  object 
 1   Time                   49392 non-null  object 
 2   Global_active_power    49392 non-null  float64
 3   Global_reactive_power  49392 non-null  float64
 4   Voltage                49392 non-null  float64
 5   Global_intensity       49392 non-null  float64
 6   Sub_metering_1         49392 non-null  float64
 7   Sub_metering_2         49392 non-null  float64
 8   Sub_metering_3         49392 non-null  float64
dtypes: float64(7), object(2)
memory usage: 3.8+ MB
```

2.2 Checking Duplicates and null values

```
df[df.duplicated()].sum().sum()
```

```
0.0
```

```
# Now again checking for nan values

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

```
0
```

Observations:

- So there are no duplicate and na values in the dataset.

2.3 Creating new column and dropping unnecessary columns

```
# 1st converting the 'Date' column to datetime format
df["Date"] = pd.to_datetime(df["Date"], dayfirst=True)
df.head()
```

```
              Date      Time  Global_active_power  
Global_reactive_power  \
574143  2008-01-19  10:27:00                1.376  
0.080
1728774 2010-03-31  06:18:00                1.384  
0.096
1522856 2009-11-08  06:20:00                0.224  
0.000
```

```
1218182 2009-04-10  16:26:00                0.370
0.128
20796   2006-12-31  04:00:00                0.216
0.000

         Voltage  Global_intensity  Sub_metering_1  Sub_metering_2  \
574143    239.34               5.6             0.0             1.0
1728774   242.14               5.6             0.0             1.0
1522856   244.44               0.8             0.0             0.0
1218182   244.31               1.6             0.0             0.0
20796     244.75               1.0             0.0             0.0

         Sub_metering_3
574143             17.0
1728774            18.0
1522856             1.0
1218182             1.0
20796               0.0
```

```
# Checking the types

df.dtypes
```

```
Date                     datetime64[ns]
Time                             object
Global_active_power             float64
Global_reactive_power           float64
Voltage                         float64
Global_intensity                float64
Sub_metering_1                  float64
Sub_metering_2                  float64
Sub_metering_3                  float64
dtype: object
```

```
# Creating a column 'month' from the dataset
# Also creating a new column for 'total energy consumed'

# 1st renaming the 'Date' column as 'month'
df.rename(columns={"Date":"month"}, inplace=True)

# Now extracting only month in that column 'month'
df['month'] = df['month'].dt.month

# Now creating the new column 'total energy consumed'
df["Total_energy_consumed"] = df['Sub_metering_1'] +
df['Sub_metering_2'] + df['Sub_metering_3']

df.head()
```

```
         month      Time  Global_active_power  Global_reactive_power
Voltage  \
574143       1  10:27:00                1.376                  0.080
239.34
1728774      3  06:18:00                1.384                  0.096
242.14
1522856     11  06:20:00                0.224                  0.000
244.44
1218182      4  16:26:00                0.370                  0.128
244.31
20796       12  04:00:00                0.216                  0.000
244.75

         Global_intensity  Sub_metering_1  Sub_metering_2
Sub_metering_3  \
574143                5.6             0.0             1.0
17.0
1728774               5.6             0.0             1.0
18.0
1522856               0.8             0.0             0.0
1.0
1218182               1.6             0.0             0.0
1.0
20796                 1.0             0.0             0.0
0.0

         Total_energy_consumed
574143                    18.0
1728774                   19.0
1522856                    1.0
1218182                    1.0
20796                      0.0
```

```
# Now removing the columns 'Time' 'Sub_metering_1', 'Sub_metering_2'
and 'Sub_metering_3'

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

```
         month  Global_active_power  Global_reactive_power  Voltage  \
574143       1                1.376                  0.080   239.34
1728774      3                1.384                  0.096   242.14
1522856     11                0.224                  0.000   244.44
1218182      4                0.370                  0.128   244.31
20796       12                0.216                  0.000   244.75

         Global_intensity  Total_energy_consumed
574143                5.6                   18.0
1728774               5.6                   19.0
```

```
1522856               0.8                    1.0
1218182               1.6                    1.0
20796                 1.0                    0.0
```

```
# Checking the dtypes

df.dtypes
```

```
month                     int64
Global_active_power     float64
Global_reactive_power   float64
Voltage                 float64
Global_intensity        float64
Total_energy_consumed   float64
dtype: object
```

```
df.shape
```

```
(49392, 6)
```

Observations

- So now we have all the columns having numerical values only.

3. EDA

3.1 Statistical Analysis of the data

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

```
                         count        mean        std      min
25%  \
month                  49392.0    6.424097   3.431237    1.000
3.000
Global_active_power    49392.0    1.096532   1.053539    0.078
0.310
Global_reactive_power  49392.0    0.123894   0.113127    0.000
0.048
Voltage                49392.0  240.853660   3.233231  224.850
239.000
Global_intensity       49392.0    4.648635   4.426401    0.200
1.400
Total_energy_consumed  49392.0    8.902656  12.817829    0.000
0.000

                          50%      75%      max
month                   6.000    9.000   12.000
Global_active_power     0.612    1.540    8.524
Global_reactive_power   0.100    0.194    1.014
Voltage               241.020  242.900  253.400
Global_intensity        2.600    6.400   36.400
Total_energy_consumed   1.000   18.000  129.000
```

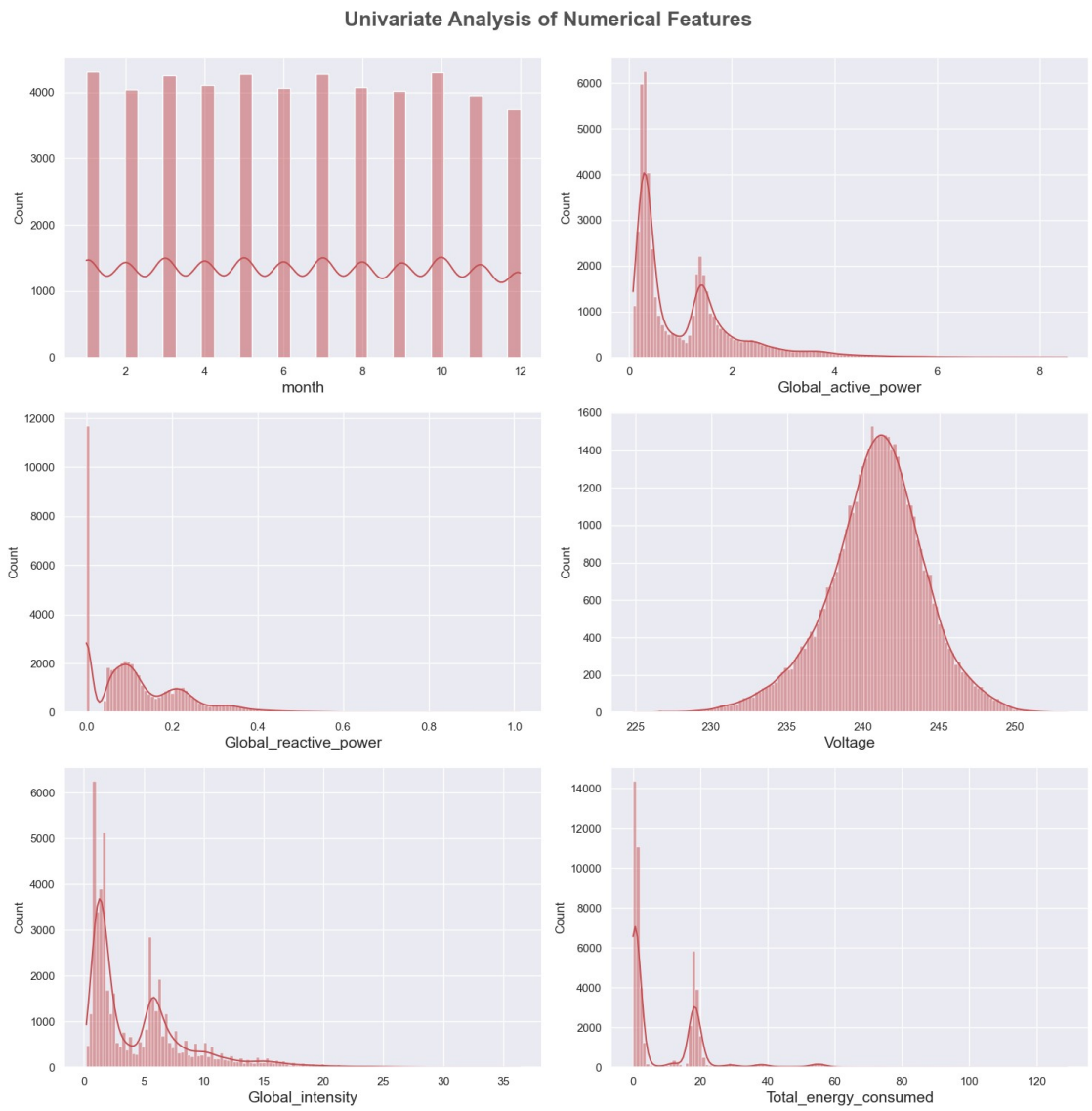
Observations:

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

3.2 Graphical representation of the data

3.2.1 Univariate Analysis

```
# For numerical features

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


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()
```

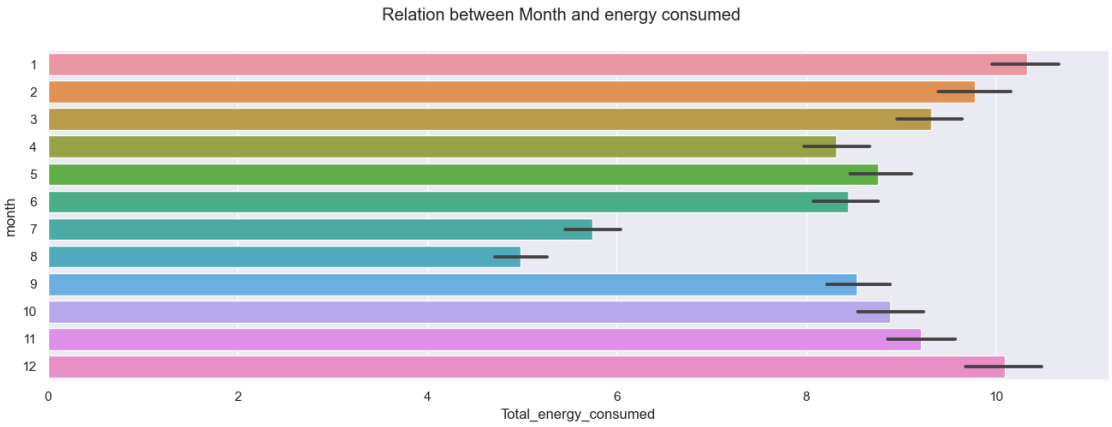


Observations:

- Only `Voltage` has normal distribution.
- The `month` column shows all months have almost equal amount of data.
- 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.

3.2.2 Bivariate Analysis

```
plt.figure(figsize=(16, 5))
plt.suptitle("Relation between Month and energy consumed")
sns.barplot(x='Total_energy_consumed', y='month', data=df, orient='h')
plt.show()
```



Observations:

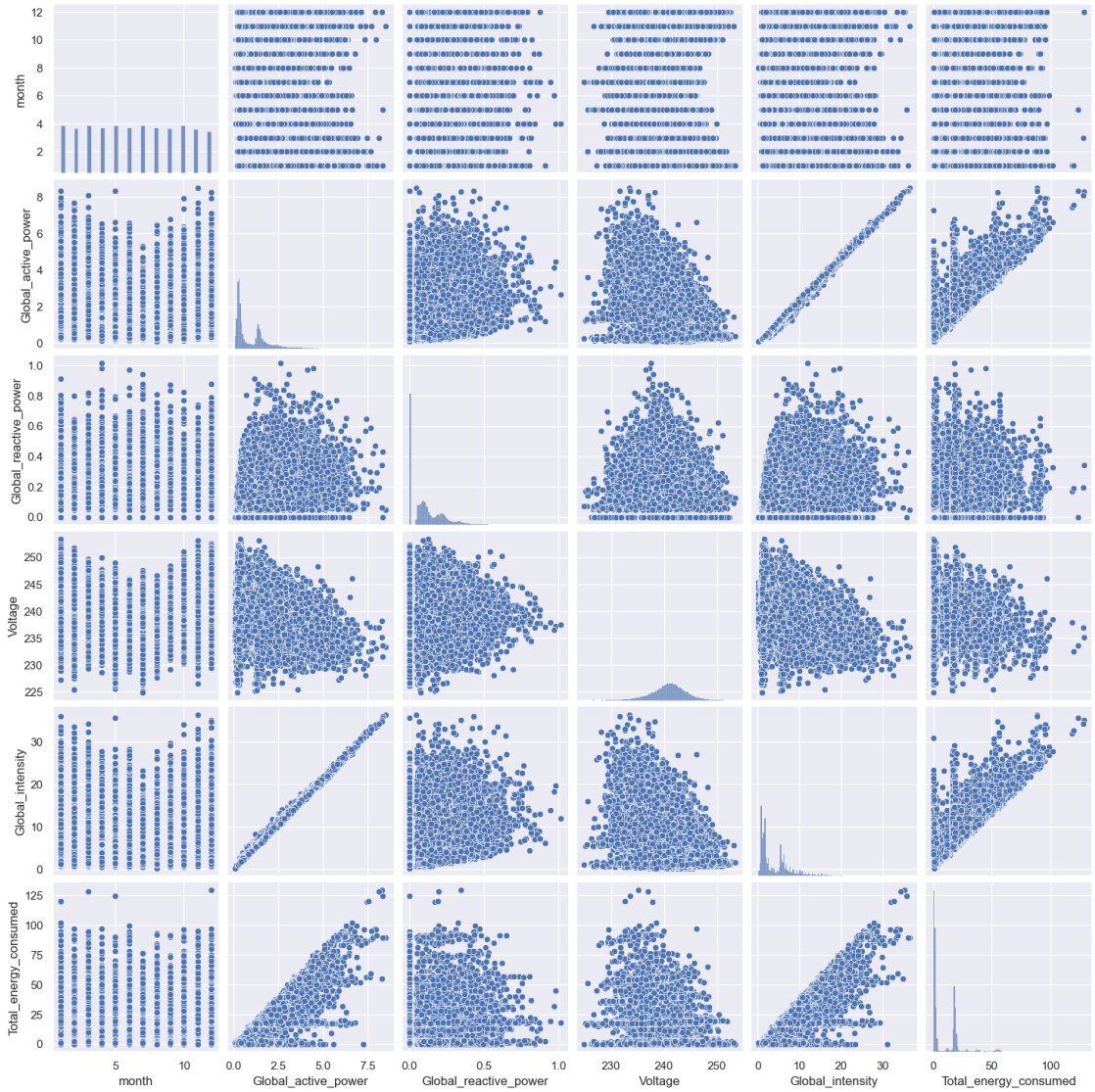
- It seems that during winter more power is consumed.

3.2.3 Multivariate Analysis

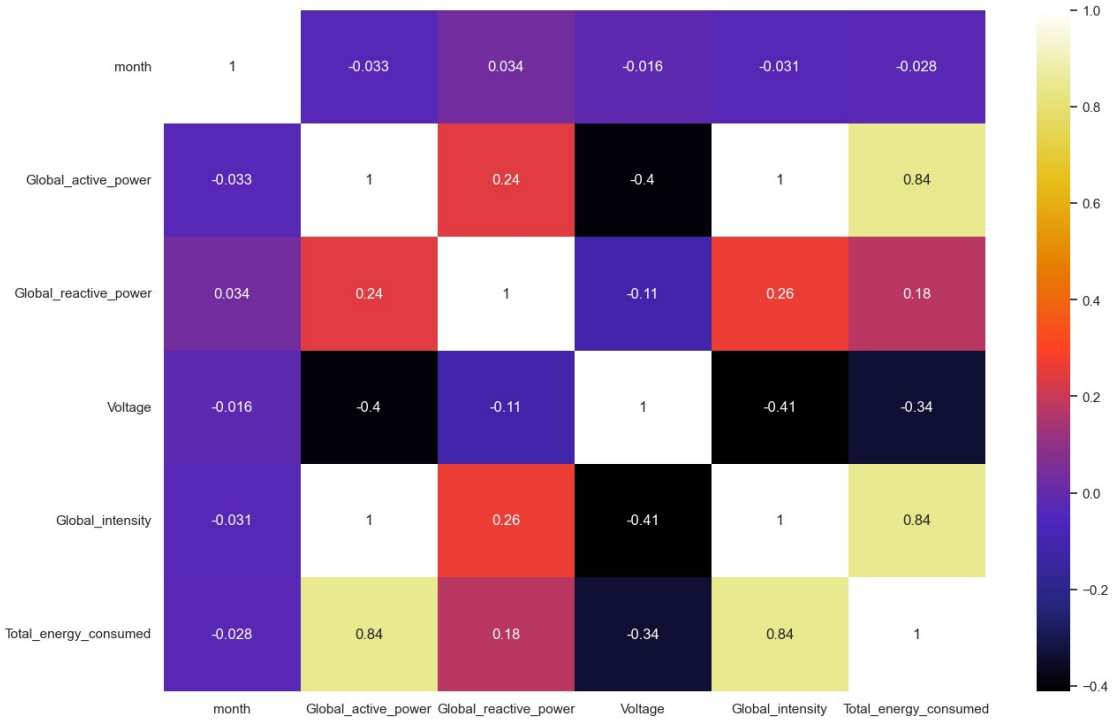
```
df[list(df.columns)].corr()
```

```
                         month  Global_active_power
Global_reactive_power  \
month                 1.000000            -0.032733
0.034030
Global_active_power   -0.032733             1.000000
0.244281
Global_reactive_power  0.034030             0.244281
1.000000
Voltage               -0.016036            -0.398967              -
0.108698
Global_intensity      -0.031239             0.998856
0.263357
Total_energy_consumed -0.028051             0.844302
0.177883

                          Voltage  Global_intensity
Total_energy_consumed
month                 -0.016036         -0.031239              -
0.028051
Global_active_power   -0.398967          0.998856
0.844302
Global_reactive_power -0.108698          0.263357
0.177883
Voltage                1.000000         -0.410652              -
0.336719
Global_intensity      -0.410652          1.000000
0.840950
Total_energy_consumed -0.336719          0.840950
1.000000
```

```
sns.pairplot(df)
plt.show()
```




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



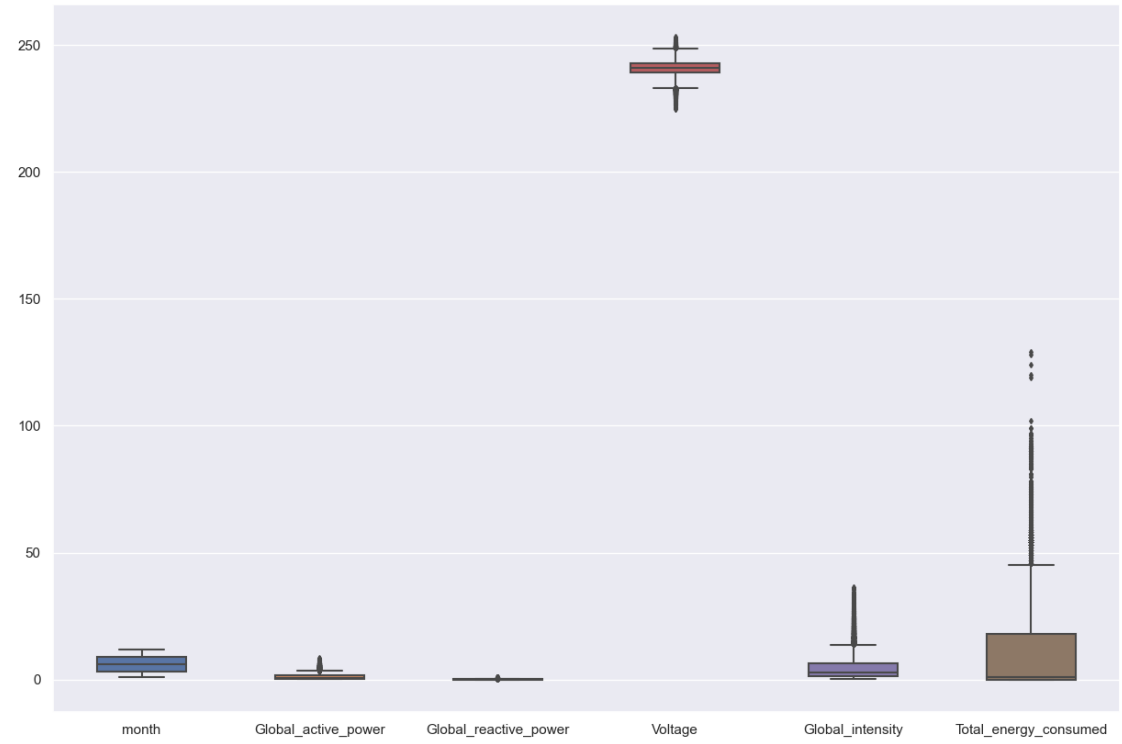
Observations:

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

4. Data Pre-Processing

4.1 Outlier handling

```
# Detecting outliers

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



Observations:

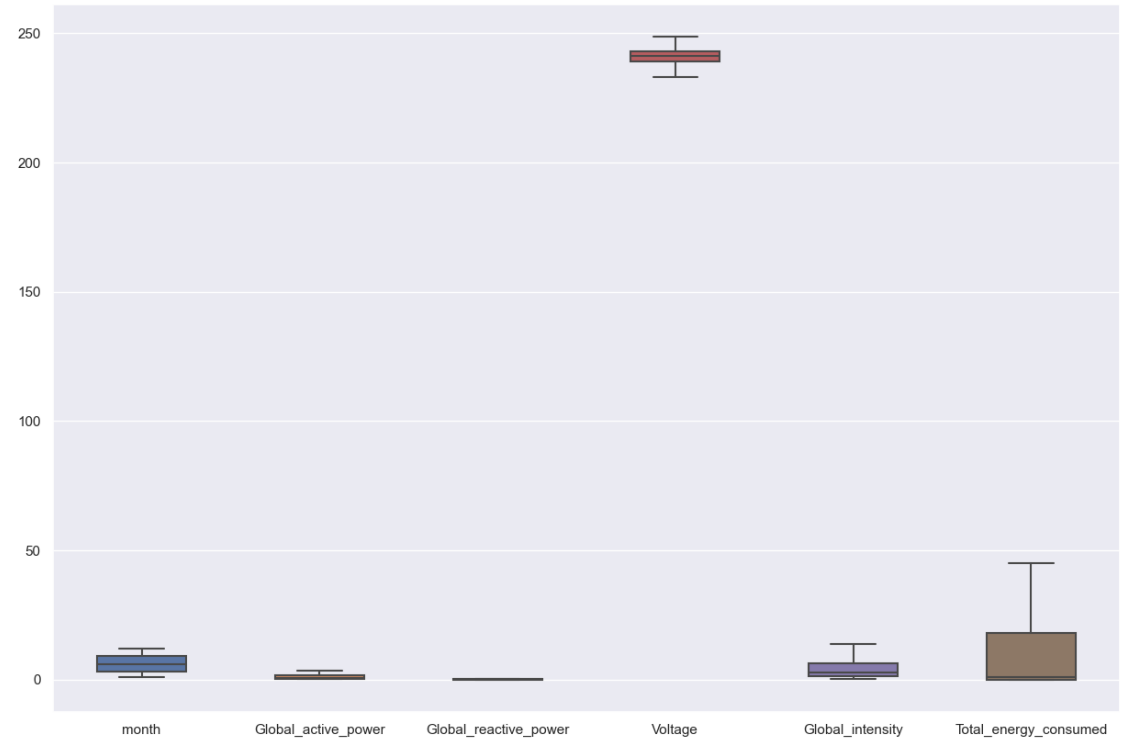
- There is outlier in all columns except `month`.

```
# 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 , '####')
    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
```

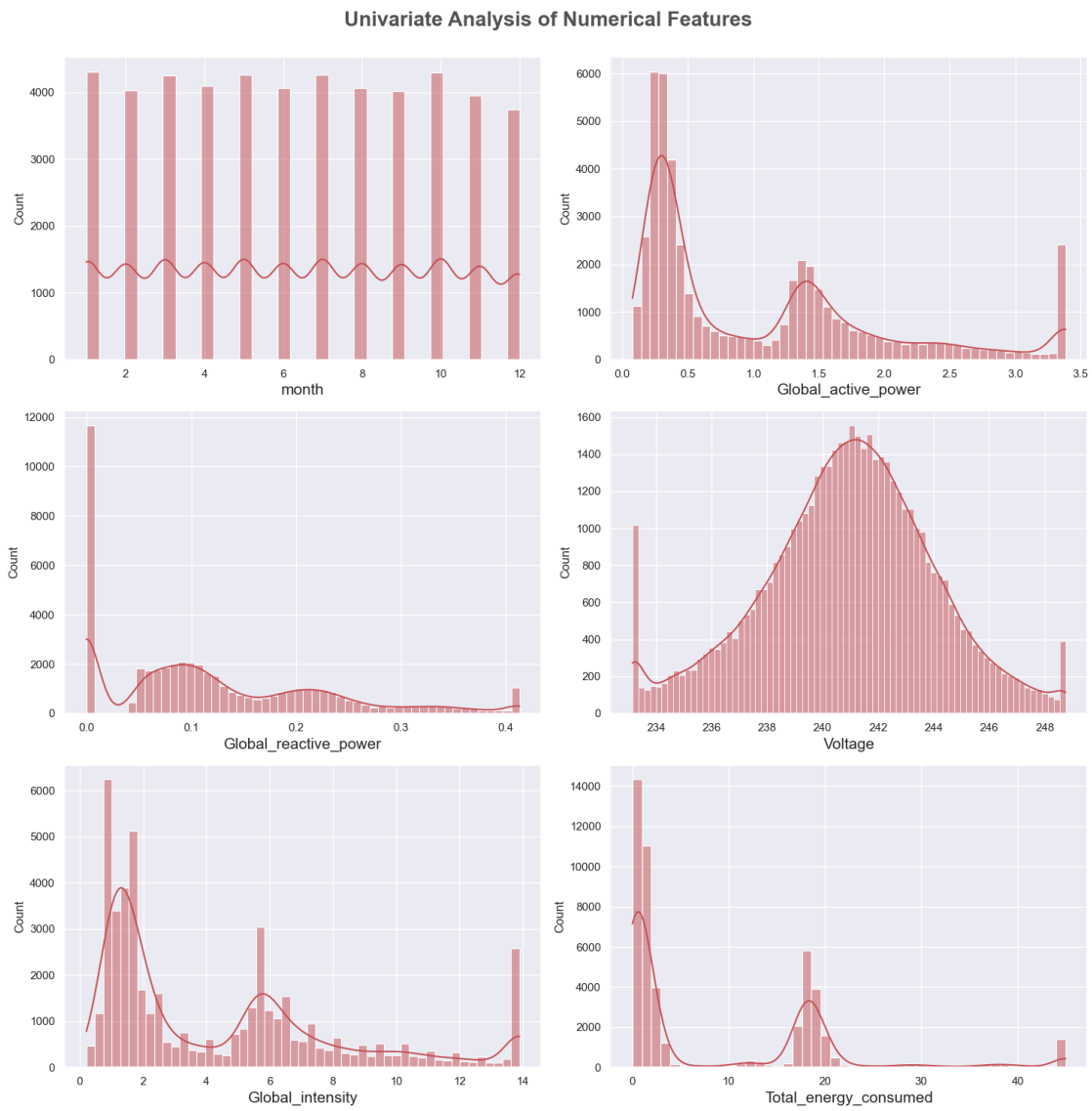
```
for col in df.columns:
         detect_outliers(col)
```

```
 #### month ####
25percentile:  3.0
75percentile:  9.0
Upper limit:  18.0
Lower limit:  -6.0

 #### Global_active_power ####
25percentile:  0.31
75percentile:  1.54
Upper limit:  3.385
Lower limit:  -1.535

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

 #### Voltage ####
25percentile:  239.0
75percentile:  242.9
Upper limit:  248.75
Lower limit:  233.14999999999998

 #### 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
```

```
# Again checking for outliers

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




```
# Let's see the distribution again

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


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

- Now the data is clean.

Let's save the clean data to mongodb for later use

```
power_dict = df.to_dict('records')

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

```
        database = client['ml_algo']
        collection = database['power_consumption_data']
        collection.insert_many(power_dict)
    except Exception as e:
        print("Error is: ",e)
    else:
        print("\nRecord inserted successfully.")
finally:
    print("\nRecored uploaded to mongoDB.")
```

```
Connection to MongoDB server is successful.

Record inserted successfully.

Recored uploaded to mongoDB.
```