EDA, FE and Regression Model (Household Power ConsumptionDataset)

Rajesh Sekar

Linkedin:https://lnkd.in/gu7QnXPV (https://lnkd.in/gu7QnXPV)

Github link :https://github.com/Rajeshsekar1504/ML_Projects (https://github.com/Rajeshsekar1504/ML_Projects)

EDA and FE

1. Data Profiling
2. Stastical analysis
3. Graphical Analysis
4. Data Cleaning
5. Data Scaling

Linear Regression Model

1.Linear Regression Model

2.Performance metrics for above model

Ridge Regression Model

1. Ridge Regression Model
2. Performance metrics for above model

lasso Regression Model

1. Lasso Regression Model
2. Performance metrics for above model

Elastic-Net Regression Model

1. Elastic-Net Regression Model
2. Performance metrics for above model

Support Vector Regressor Model

1. Support Vector Regressor Model

2. Performance metrics for above model

Dataset:https://archive.ics.uci.edu/ml/machine-learning-databases/00235/ (https://archive.ics.uci.edu/ml/machine-learning-databases/00235/)

Importing required libraries

In [3]:

```
import pandas as pd
import numpy as np

### Visualisation libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

### For Q-Q Plot

import scipy.stats as stats

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

In [4]:

```
## Importing original dataset

df = pd.read_csv("household_power_consumption.txt",sep=';')
df.head(5)
```

Out[4]:

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

In [5]:

```
## <font color=red>Dataset Information</font>
```

Data Set Information:

This archive contains 2075259 measurements gathered in a house located in Sceaux (7km of Paris, France) between December 2006 and November 2010 (47 months). Notes:

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

Attribute Information:

1. date: Date in format dd/mm/yyyy
2. time: time in format hh:mm:ss
3. global_active_power: household global minute-averaged active power (in kilowatt)
4. global_reactive_power: household global minute-averaged reactive power (in kilowatt)
5. voltage: minute-averaged voltage (in volt)
6. global_intensity: household global minute-averaged current intensity (in ampere)
7. sub_metering_1: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered).
8. sub_metering_2: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light.
9. sub_metering_3: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

In [6]:

```
## Getting shape of the original dataset
df.shape
```

Out[6]:

```
(2075259, 9)
```

In [7]:

```
## Checking data types of features in original dataset
df.dtypes
```

Out[7]:

```
Date                      object
Time                      object
Global_active_power       object
Global_reactive_power     object
Voltage                   object
Global_intensity          object
Sub_metering_1            object
Sub_metering_2            object
Sub_metering_3           float64
dtype: object
```

Taking 50k Records as a sample from dataset

In [8]:

```
## Taking 50,000 samples from original dataset without replacement
## resetting the index of records and droppnig index

data_sample=df.sample(n=50000,replace=False)
data_sample=data_sample.reset_index()
data_sample.drop('index',axis=1, inplace=True)
data_sample.head()
```

Out[8]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	S
0	21/8/2009	19:21:00	1.412	0.198	241.020	5.800	
1	20/9/2008	04:13:00	0.954	0.120	241.720	4.000	
2	16/3/2007	08:58:00	1.406	0.000	238.910	5.800	
3	13/3/2007	05:50:00	0.318	0.088	239.240	1.400	
4	25/7/2009	09:33:00	0.500	0.206	242.520	2.200	

In [9]:

```
## Checking shape of the sample dataset
data_sample.shape
```

Out[9]:

```
(50000, 9)
```

In [11]:

```
### Saving sampled dataset to csv file
data_sample.to_csv('sampled_data.csv',index=False)
```

In [9]:

```
## checking unique values in each feature to form data cleaning strategy if necessary

for feature in [feature for feature in data_sample.columns if feature not in ['Date','Time'
    print("feature {} has these {} unique values\n".format(feature,data_sample[feature].uni
```

```
feature Sub_metering_2 has these ['0.000' '1.000' '2.000' '35.000' 2.0 '3
9.000' 0.0 '3.000' '37.000' '?'
 5.0 '21.000' 1.0 '11.000' '71.000' '36.000' '75.000' '27.000' '28.000'
 '5.000' '73.000' '25.000' '26.000' '29.000' '38.000' '18.000' '4.000'
 '8.000' '24.000' '32.000' '23.000' '33.000' '49.000' '13.000' '40.000'
 '31.000' '6.000' '41.000' '72.000' '19.000' '60.000' '20.000' '22.000'
 37.0 '16.000' '67.000' '66.000' '10.000' '30.000' '34.000' '15.000'
 '74.000' '14.000' 69.0 '9.000' 68.0 '7.000' '70.000' 73.0 '58.000'
 '57.000' '12.000' '46.000' '64.000' '69.000' '51.000' '62.000' '17.000'
 23.0 18.0 '63.000' 30.0 '42.000' 35.0 36.0 '76.000' '61.000' '43.000'
 '56.000' 28.0 '44.000' '47.000' 75.0 '65.000' '55.000' 21.0 '48.000' 31.
0
 '77.000' 27.0 8.0 6.0 '45.000' 3.0 38.0] unique values

feature Sub_metering_3 has these [ 1.  0. 18. 17. 19. 16. 12. 11. nan 13.
7. 24. 21. 23.  3. 10. 29. 20.
  8.  9.  6. 28.  2. 27.  5. 30. 22. 15. 31. 25.  4. 14. 26.] unique valu
es
```

In [10]:

```
## Checking no of records in each feature that have value as?

for feature in [feature for feature in data_sample.columns if feature not in ['Date','Time'
    print("The feature {} has {} ? in it".format(feature,data_sample[data_sample[feature]==
```

```
The feature Global_active_power has (640, 9) ? in it
The feature Global_reactive_power has (640, 9) ? in it
The feature Voltage has (640, 9) ? in it
The feature Global_intensity has (640, 9) ? in it
The feature Sub_metering_1 has (640, 9) ? in it
The feature Sub_metering_2 has (640, 9) ? in it
The feature Sub_metering_3 has (0, 9) ? in it
```

In [12]:

```
## replacing ? values with nan values
data_sample.replace('?',np.nan, inplace=True)
```

In [13]:

```
### Checking no of records in each feature that have value as ? after replacing them

for feature in [feature for feature in data_sample.columns if feature not in ['Date','Time'
    print("The feature {} has {} ? in it".format(feature,data_sample[data_sample[feature]==
```

```
The feature Global_active_power has (0, 9) ? in it
The feature Global_reactive_power has (0, 9) ? in it
The feature Voltage has (0, 9) ? in it
The feature Global_intensity has (0, 9) ? in it
The feature Sub_metering_1 has (0, 9) ? in it
The feature Sub_metering_2 has (0, 9) ? in it
The feature Sub_metering_3 has (0, 9) ? in it
```

In [14]:

```
## dropping nan values
data_sample.dropna(inplace=True)
```

In [15]:

```
### Checking data type, shape and null values
data_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49411 entries, 0 to 49999
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   49411 non-null  object 
 1   Time                   49411 non-null  object 
 2   Global_active_power    49411 non-null  object 
 3   Global_reactive_power  49411 non-null  object 
 4   Voltage                49411 non-null  object 
 5   Global_intensity       49411 non-null  object 
 6   Sub_metering_1         49411 non-null  object 
 7   Sub_metering_2         49411 non-null  object 
 8   Sub_metering_3         49411 non-null  float64
dtypes: float64(1), object(8)
memory usage: 3.8+ MB
```

In [16]:

```
### checking values in sub_merging_1 feature
# observation: all values are integers but in float data type so can be converted to int da

data_sample.Sub_metering_1.unique()
```

Out[16]:

```
array(['0.000', '1.000', 0.0, '2.000', '38.000', '37.000', '36.000',
       '7.000', '39.000', '20.000', '42.000', '11.000', '17.000',
       '21.000', '14.000', '16.000', '35.000', '33.000', '26.000',
       '10.000', '27.000', '25.000', 1.0, '15.000', '40.000', '4.000',
       '13.000', '30.000', 27.0, 2.0, '29.000', '12.000', '46.000',
       '3.000', '19.000', '18.000', '48.000', '44.000', '6.000', 35.0,
       '31.000', '22.000', '9.000', '23.000', '71.000', 12.0, '28.000',
       39.0, '53.000', 15.0, '5.000', '32.000', 14.0, '50.000', '47.000',
       '73.000', '34.000', '41.000', '60.000', '45.000', 9.0, 38.0, 36.0,
       '43.000', '8.000', '49.000', '75.000', 13.0, '24.000', 42.0,
       '56.000', 19.0, '79.000', 37.0, 21.0, '72.000', '51.000', 32.0,
       30.0, 33.0, '52.000', 45.0], dtype=object)
```

In [17]:

```
### checking values in Sub_metering_2 feature
# observation:all values are integers but in float datatype so can be converted to int data
data_sample.Sub_metering_2.unique()
```

Out[17]:

```
array(['1.000', '0.000', '69.000', 41.0, '2.000', 0.0, '12.000', '3.000',
       '6.000', '32.000', '38.000', '17.000', '25.000', '35.000',
       '28.000', '30.000', '36.000', 1.0, '4.000', 2.0, '10.000', 42.0,
       '57.000', '23.000', '22.000', '39.000', '66.000', '49.000',
       '37.000', '34.000', '72.000', '5.000', '16.000', '43.000',
       '19.000', '13.000', '75.000', '73.000', '44.000', '60.000',
       '29.000', 5.0, '8.000', '18.000', '40.000', '33.000', '24.000',
       '7.000', '26.000', '21.000', '62.000', '27.000', '9.000', '74.000',
       '67.000', 38.0, '31.000', '20.000', '45.000', '46.000', '63.000',
       '14.000', '70.000', '71.000', '11.000', '54.000', '55.000', 40.0,
       28.0, '64.000', '68.000', 8.0, 27.0, '15.000', '41.000', 31.0,
       '76.000', '42.000', 68.0, '50.000', 36.0, 35.0, 4.0, 18.0, 3.0,
       '56.000', 39.0, 7.0, '65.000', '61.000', 9.0, 73.0, '47.000', 34.0,
       '58.000'], dtype=object)
```

In [18]:

```
### checking values in Sub_metering_3 feature
# observation:all values are integers but in float datatype so can be converted to int data
data_sample.Sub_metering_3.unique()
```

Out[18]:

```
array([18.,  0., 17.,  1., 13., 19., 11., 12., 16., 29.,  6.,  5., 24.,
        3., 20.,  9.,  2., 27.,  7.,  4., 30., 28., 14., 10.,  8., 21.,
       25., 23., 31., 26., 22., 15.])
```

In [19]:

```
## Converting to str datatype so replace function can be used
data_sample['Sub_metering_3']=data_sample['Sub_metering_3'].astype(str)
```

In [20]:

```
### stripping  and zero from below mentioned features so it can be converted to integer

data_sample['Sub_metering_1']=data_sample['Sub_metering_1'].str.split(".", expand=True)[0]
data_sample['Sub_metering_2']=data_sample['Sub_metering_2'].str.split(".", expand=True)[0]
data_sample['Sub_metering_3']=data_sample['Sub_metering_3'].str.split(".", expand=True)[0]
```

In [21]:

```
## Checking integer values

data_sample.Sub_metering_1.unique()
```

Out[21]:

```
array(['0', '1', nan, '2', '38', '37', '36', '7', '39', '20', '42', '11',
       '17', '21', '14', '16', '35', '33', '26', '10', '27', '25', '15',
       '40', '4', '13', '30', '29', '12', '46', '3', '19', '18', '48',
       '44', '6', '31', '22', '9', '23', '71', '28', '53', '5', '32',
       '50', '47', '73', '34', '41', '60', '45', '43', '8', '49', '75',
       '24', '56', '79', '72', '51', '52'], dtype=object)
```

In [22]:

```
## Checking integer values

data_sample.Sub_metering_2.unique()
```

Out[22]:

```
array(['1', '0', '69', nan, '2', '12', '3', '6', '32', '38', '17', '25',
       '35', '28', '30', '36', '4', '10', '57', '23', '22', '39', '66',
       '49', '37', '34', '72', '5', '16', '43', '19', '13', '75', '73',
       '44', '60', '29', '8', '18', '40', '33', '24', '7', '26', '21',
       '62', '27', '9', '74', '67', '31', '20', '45', '46', '63', '14',
       '70', '71', '11', '54', '55', '64', '68', '15', '41', '76', '42',
       '50', '56', '65', '61', '47', '58'], dtype=object)
```

In [23]:

```
## Checking integer values

data_sample.Sub_metering_3.unique()
```

Out[23]:

```
array(['18', '0', '17', '1', '13', '19', '11', '12', '16', '29', '6', '5',
       '24', '3', '20', '9', '2', '27', '7', '4', '30', '28', '14', '10',
       '8', '21', '25', '23', '31', '26', '22', '15'], dtype=object)
```

In [24]:

```
### checking null values

data_sample.isnull().sum()
```

Out[24]:

```
Date                      0
Time                      0
Global_active_power       0
Global_reactive_power     0
Voltage                   0
Global_intensity          0
Sub_metering_1          973
Sub_metering_2          973
Sub_metering_3            0
dtype: int64
```

In [25]:

```
## dropping null values
data_sample.dropna(inplace=True)
```

In [26]:

```
## Checking data type and null values
data_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48438 entries, 0 to 49999
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   Date                   48438 non-null  object
 1   Time                   48438 non-null  object
 2   Global_active_power    48438 non-null  object
 3   Global_reactive_power  48438 non-null  object
 4   Voltage                48438 non-null  object
 5   Global_intensity       48438 non-null  object
 6   Sub_metering_1         48438 non-null  object
 7   Sub_metering_2         48438 non-null  object
 8   Sub_metering_3         48438 non-null  object
dtypes: object(9)
memory usage: 3.7+ MB
```

In [27]:

```
### creating dict of datatype conversion
datatypes_convert = {'Global_active_power':'float64', 'Global_reactive_power':'float64',
'Voltage':'float64', 'Global_intensity':'float64', 'Sub_metering_1':'int64', 'Sub_metering_
'Sub_metering_3':'int64'}
```

In [28]:

```
### changing data type of numerical features to float or int
data_sample=data_sample.astype(datatypes_convert)
data_sample.dtypes
```

Out[28]:

```
Date                      object
Time                      object
Global_active_power      float64
Global_reactive_power    float64
Voltage                  float64
Global_intensity         float64
Sub_metering_1             int64
Sub_metering_2             int64
Sub_metering_3             int64
dtype: object
```

In [29]:

```
### converting datatype of Date feature to datetime

data_sample['Date']=pd.to_datetime(data_sample['Date'],format="%d/%m/%Y")
```

In [30]:

```
### separating day and month, creating new feature for day and month

data_sample['day']=data_sample['Date'].dt.day
data_sample['month']=data_sample['Date'].dt.month
```

In [31]:

```
### dropping time this is not important and Date is already used in day and month
data_sample.drop('Time',axis=1, inplace=True)
data_sample.drop('Date',axis=1, inplace=True)
```

In [32]:

```
## Checking values and features after data cleaning

data_sample.head()
```

Out[32]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_
0	1.412	0.198	241.02	5.8	0	
1	0.954	0.120	241.72	4.0	0	
2	1.406	0.000	238.91	5.8	1	
3	0.318	0.088	239.24	1.4	0	
4	0.500	0.206	242.52	2.2	0	

In [33]:

```
### Checking null values
data_sample.isnull().sum()
```

Out[33]:

```
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
day                      0
month                    0
dtype: int64
```

In [34]:

```
### Checking datatypes

data_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48438 entries, 0 to 49999
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Global_active_power    48438 non-null  float64
 1   Global_reactive_power  48438 non-null  float64
 2   Voltage                48438 non-null  float64
 3   Global_intensity       48438 non-null  float64
 4   Sub_metering_1         48438 non-null  int64  
 5   Sub_metering_2         48438 non-null  int64  
 6   Sub_metering_3         48438 non-null  int64  
 7   day                    48438 non-null  int64  
 8   month                  48438 non-null  int64  
dtypes: float64(4), int64(5)
memory usage: 3.7 MB
```

Creating Dependent feature

In [35]:

```
## Creating target feature as Total_power_use which is sum of Sub_metering_1,2 and 3

data_sample['Total_power_use']=data_sample['Sub_metering_1']+data_sample['Sub_metering_2']+
data_sample.head()
```

Out[35]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_
0	1.412	0.198	241.02	5.8	0	
1	0.954	0.120	241.72	4.0	0	
2	1.406	0.000	238.91	5.8	1	
3	0.318	0.088	239.24	1.4	0	
4	0.500	0.206	242.52	2.2	0	

In [36]:

```
### Saving cleaned dataset to csv

data_sample.to_csv('household_power_consumption_cleaned.csv',index=False)
```

Uploading Data to MongoDB

In [37]:

```
### Uploading dataset to MongoDB
```

In [38]:

```
### Creating connection with Mongodb
import pymongo
client = pymongo.MongoClient("mongodb+srv://rajesh1504:rajesh1504@atlascluster.zxudjtu.mong
```

In [39]:

```
### creating database and collection in MongoDB
db =client['Power_consumption']
collection=db['Household_power_data']
```

In [40]:

```
### Converting dataframe to dictionary so it can be uploaded to MongoDB
data_sample.reset_index(inplace=True)
data_dict = data_sample.to_dict("records")
```

In [41]:

```
## Insert collection to MongoDB
collection.insert_many(data_dict)
```

Out[41]:

```
<pymongo.results.InsertManyResult at 0x26fed689250>
```

Retrieving Data to MongoDB

In [42]:

```
### Locating our collection and data in MongoDb using find() method
data_from_mongodb=collection.find()
```

In [43]:

```
### converting data from MongoDb to Dataframe in pandas
data_mongodb=pd.DataFrame(data_from_mongodb)
```

In [44]:

```
## First 5 records in dataset
data_mongodb.head()
```

Out[44]:

	_id	index	Global_active_power	Global_reactive_power	Voltage	Globa
0	636cf77c65f633b9db536ad3	0	0.454	0.168	243.45	
1	636cf77c65f633b9db536ad4	1	0.140	0.000	242.60	
2	636cf77c65f633b9db536ad5	2	0.476	0.214	241.14	
3	636cf77c65f633b9db536ad6	3	3.634	0.076	232.45	
4	636cf77c65f633b9db536ad7	4	2.192	0.070	239.12	

In [45]:

```
### Dropping _id and index feature from imported dataset Mongodb
data_mongodb.drop(['_id','index'], axis=1, inplace=True)
data_mongodb.head()
```

Out[45]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_
0	0.454	0.168	243.45	2.0	0	
1	0.140	0.000	242.60	0.6	0	
2	0.476	0.214	241.14	2.2	0	
3	3.634	0.076	232.45	15.6	0	
4	2.192	0.070	239.12	9.2	0	

In [46]:

```
### Saving dataset imported from MongoDB to csv file

data_mongodb.to_csv('data_from_mongodb_power_consumption.csv')
```

Analysing Dataset

In [47]:

```
### Getting difference in min and max values of features
data_mongodb.max()-data_mongodb.min()
```

Out[47]:

```
Global_active_power       10.272
Global_reactive_power      1.192
Voltage                   30.090
Global_intensity          44.400
Sub_metering_1            80.000
Sub_metering_2            77.000
Sub_metering_3            31.000
day                       30.000
month                     11.000
Total_power_use          129.000
dtype: float64
```

Numerical Features

In [48]:

```
### Getting list of numerical features
numerical_features = data_mongodb.columns
print(numerical_features)
```

```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3', 'day', 'month', 'Total_power_use'],
      dtype='object')
```

In [49]:

```
### getting count of unique values in each numerical feature
for feature in numerical_features:
    print("Feature {} has {} no of unique values".format(feature,data_mongodb[feature].nuni
```

```
Feature Global_active_power has 3154 no of unique values
Feature Global_reactive_power has 409 no of unique values
Feature Voltage has 2344 no of unique values
Feature Global_intensity has 187 no of unique values
Feature Sub_metering_1 has 72 no of unique values
Feature Sub_metering_2 has 78 no of unique values
Feature Sub_metering_3 has 32 no of unique values
Feature day has 31 no of unique values
Feature month has 12 no of unique values
Feature Total_power_use has 114 no of unique values
```

In [50]:

```
## <font color=red>Discrete Numerical Features</font>
```

In [51]:

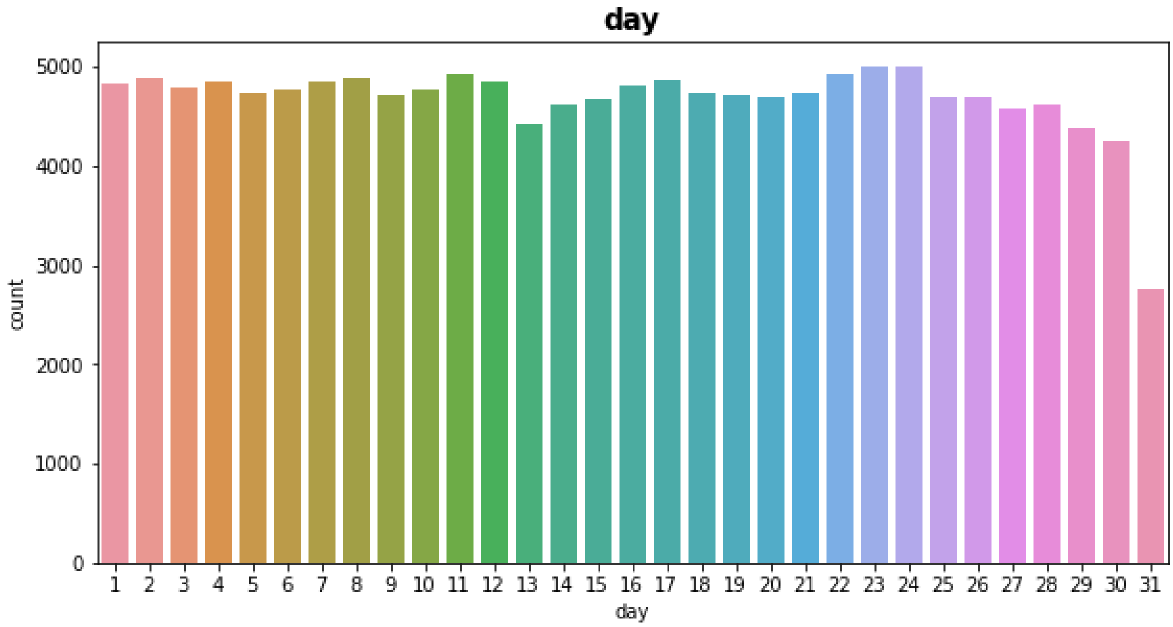
```
### Getting list of Discrete features
discrete_features=[feature for feature in numerical_features if data_mongodb[feature].nuniq
discrete_features
```

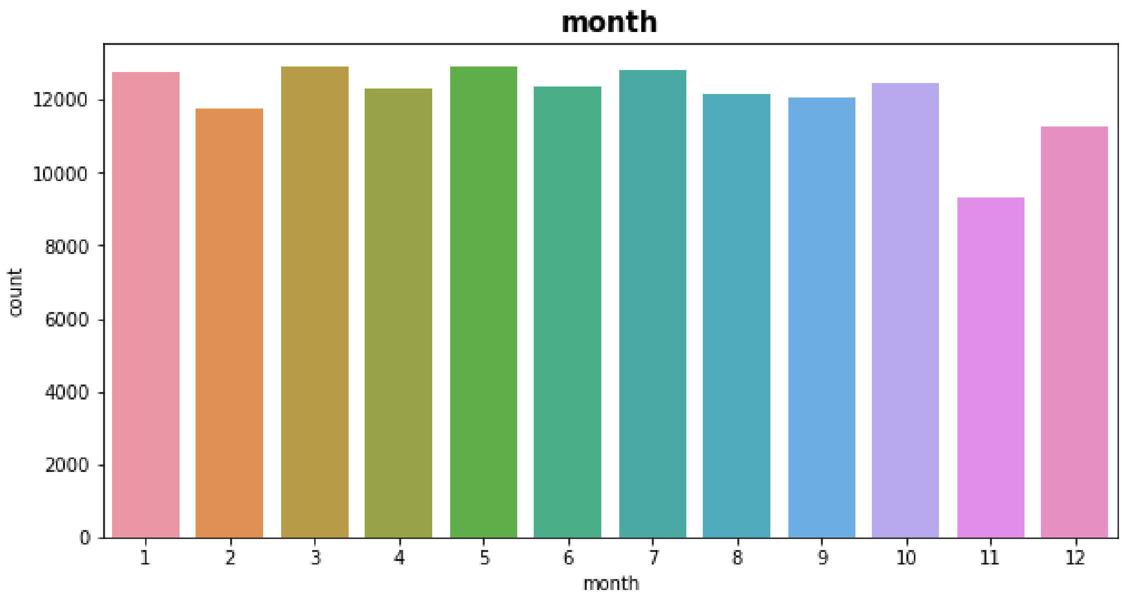
Out[51]:

```
['day', 'month']
```

In [52]:

```
for feature in discrete_features:
    plt.figure(figsize=(10,5))
    sns.countplot(data=data_mongodb, x=feature)
    plt.title(feature, fontsize=15, weight='bold')
    plt.show();
```





Continuous Numerical Features

In [53]:

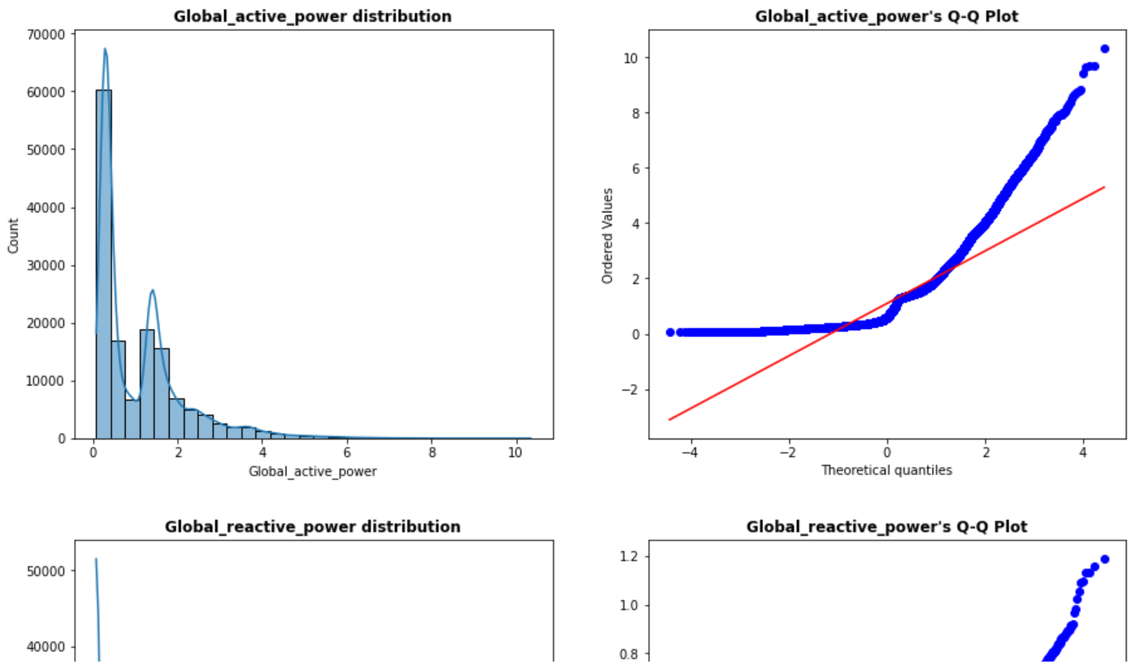
```
### Getting List of continuous features
continuous_features=[feature for feature in numerical_features if feature not in discrete_f
print(continuous_features)
```

```
['Global_active_power', 'Global_reactive_power', 'Voltage', 'Global_intensit
y', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3', 'Total_power_use']
```

Distribution of Continuous Numerical Features

In [54]:

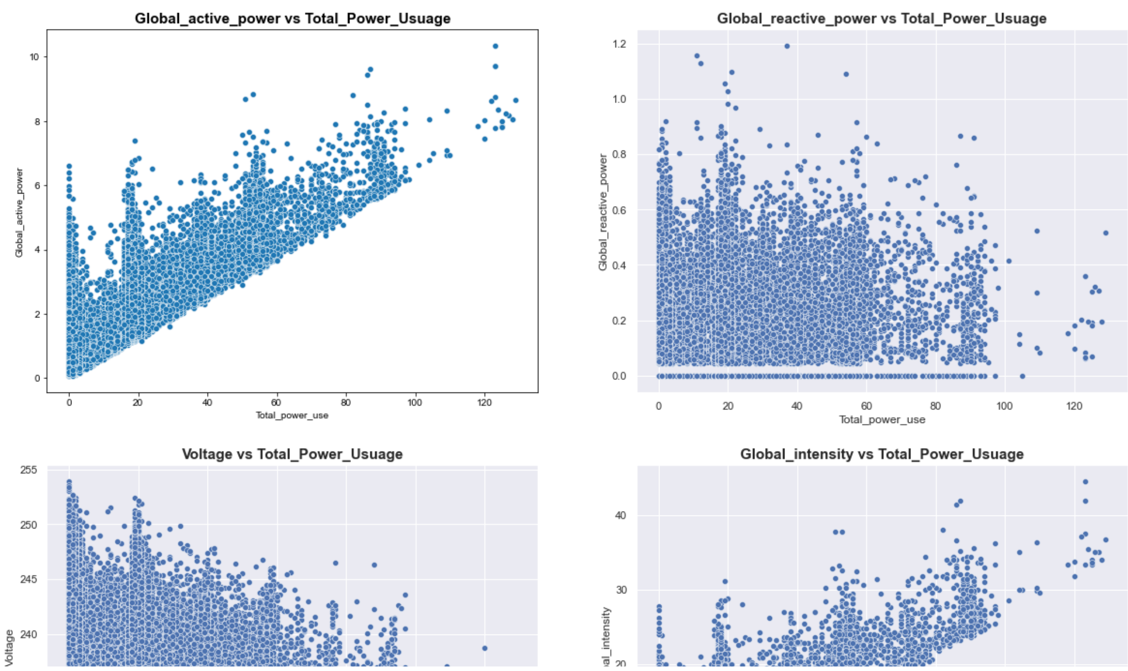
```
### checking distribution of continuous numerical features

for i in continuous_features:
    plt.figure(figsize=(15,6))
    plt.subplot(121)
    sns.histplot(data=data_mongodb, x=i, kde=True, bins=30)
    plt.title("{} distribution".format(i),fontweight='bold')
    plt.subplot(122)
    stats.probplot(data_mongodb[i], dist='norm', plot=plt)
    plt.title("{}'s Q-Q Plot".format(i),fontweight='bold')
    plt.show();
```



Comparing Numerical features with Dependent feature

In [55]:

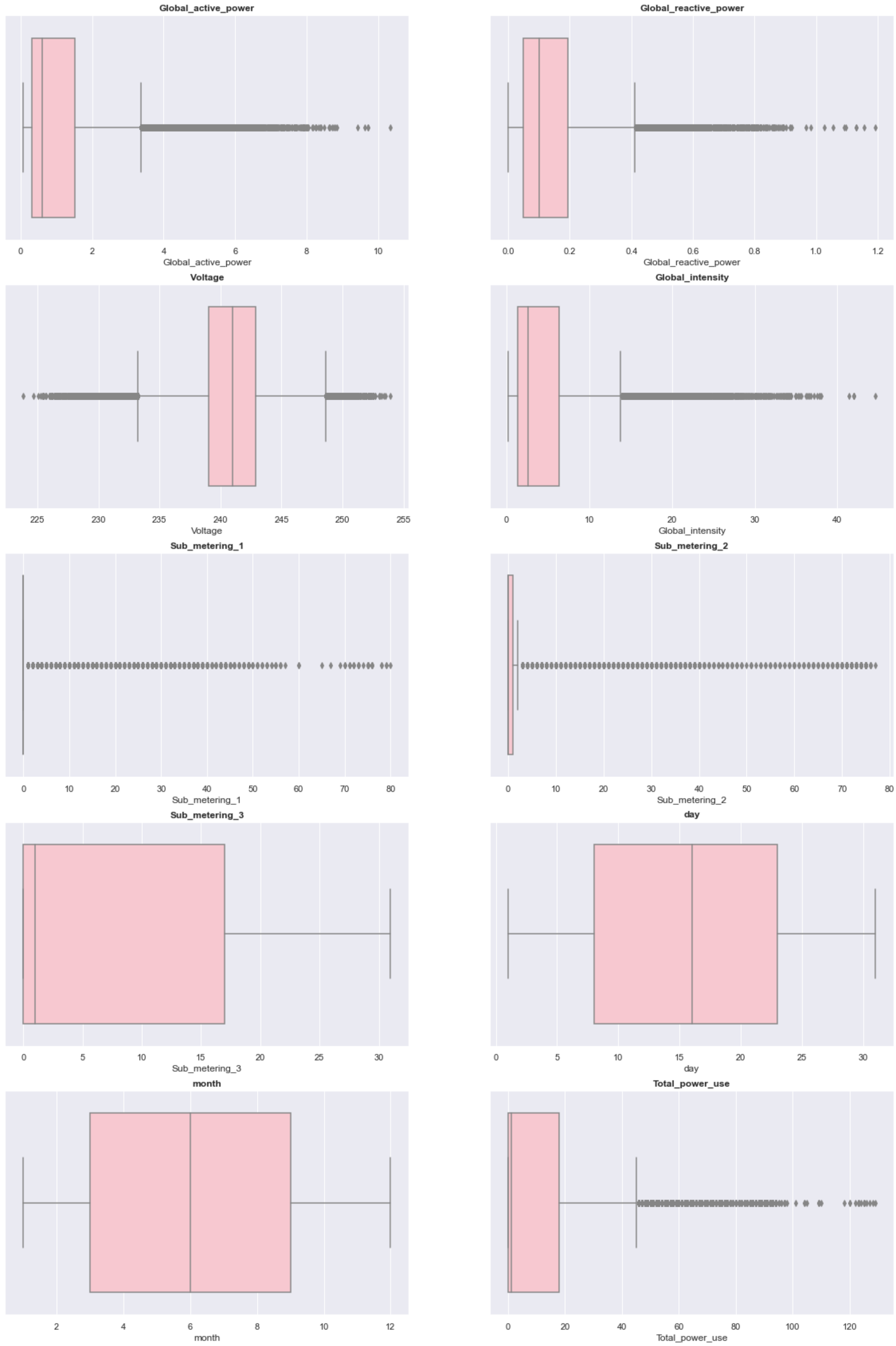
```
plt.figure(figsize=(20,40))
for i in enumerate([feature for feature in numerical_features if feature not in ['Total_pow
    plt.subplot(5, 2, i[0]+1)
    sns.set(rc={'figure.figsize':(10,8)})
    sns.scatterplot(data=data_mongodb, y=i[1],x='Total_power_use')
    plt.title("{} vs Total_Power_Usuage".format(i[1]),fontsize=15, fontweight='bold')
```



Checking outliers

In [56]:

```
### Checking outliers in numerical features

plt.figure(figsize=(20,30))
for i in enumerate(numerical_features):
    plt.subplot(5,2,i[0]+1)
    sns.set(rc={'figure.figsize':(10,6)})
    sns.boxplot(data=data_mongodb, x=i[1], color='pink')
    plt.title("{}".format(i[1]),fontweight='bold')
```



Correleation and heatmap

In [57]:

```
corr=round(data_mongodb.corr(),2)
corr
```

Out[57]:

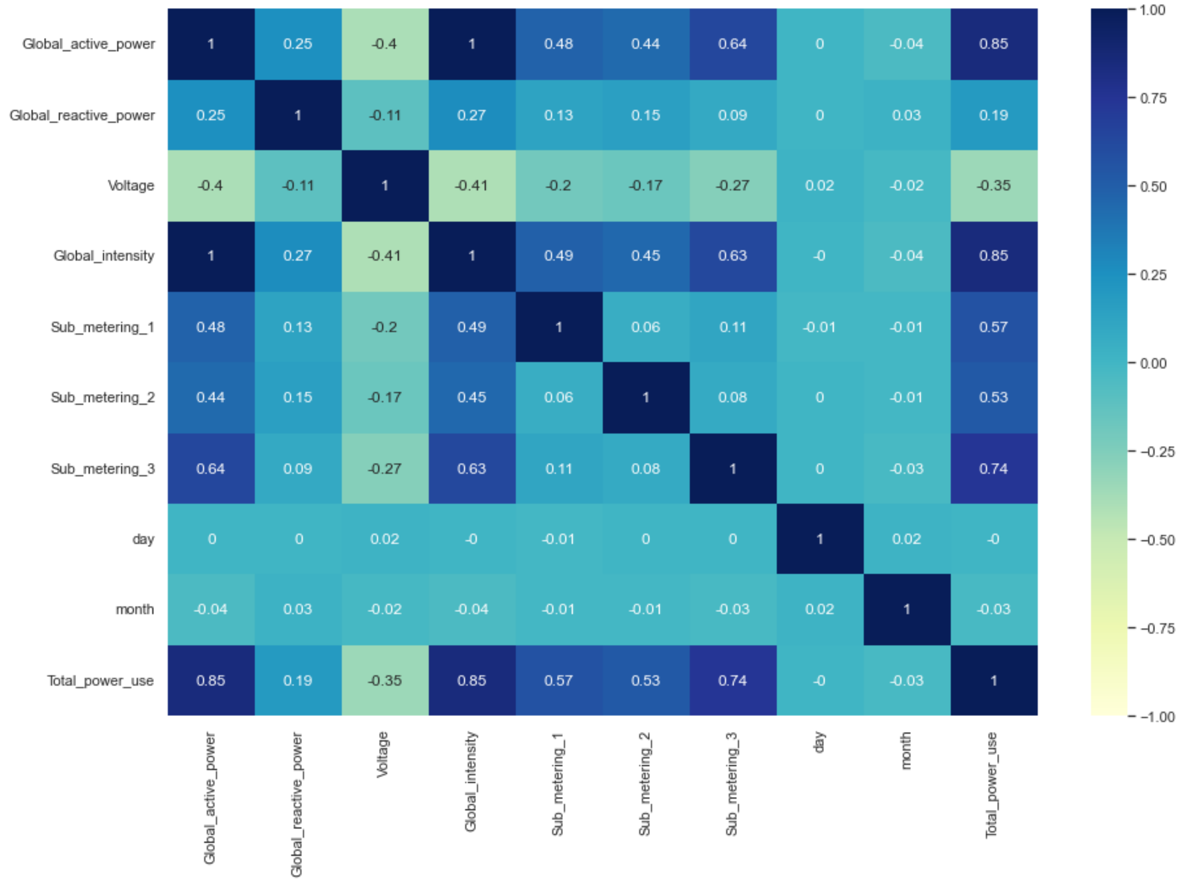
	Global_active_power	Global_reactive_power	Voltage	Global_intensity	S
Global_active_power	1.00	0.25	-0.40	1.00	
Global_reactive_power	0.25	1.00	-0.11	0.27	
Voltage	-0.40	-0.11	1.00	-0.41	
Global_intensity	1.00	0.27	-0.41	1.00	
Sub_metering_1	0.48	0.13	-0.20	0.49	
Sub_metering_2	0.44	0.15	-0.17	0.45	
Sub_metering_3	0.64	0.09	-0.27	0.63	
day	0.00	0.00	0.02	-0.00	
month	-0.04	0.03	-0.02	-0.04	
Total_power_use	0.85	0.19	-0.35	0.85	

In [58]:

```
### Plotting heatmap for visualising the correlation between features
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(data=corr, annot=True, vmin=-1, vmax=1, cmap="YlGnBu")
```

Out[58]:

<AxesSubplot:>

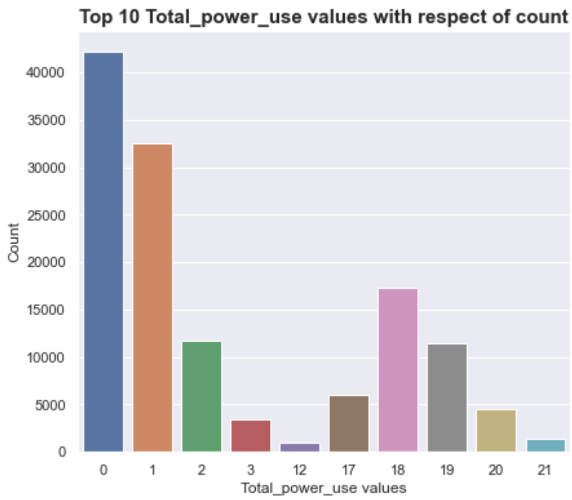


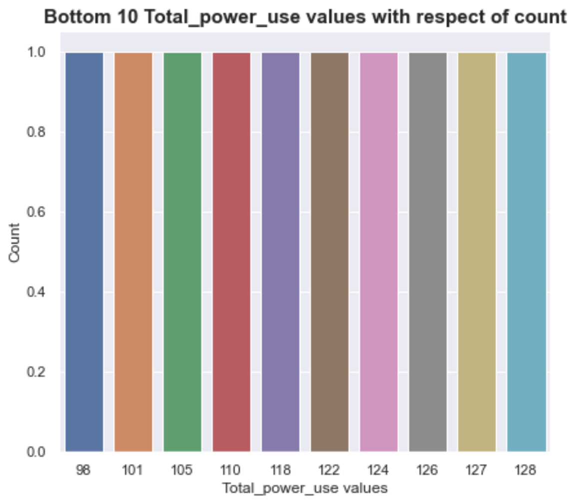
	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	day	month	Total_power_use
Global_active_power	1	0.25	-0.4	1	0.48	0.44	0.64	0	-0.04	0.85
Global_reactive_power	0.25	1	-0.11	0.27	0.13	0.15	0.09	0	0.03	0.19
Voltage	-0.4	-0.11	1	-0.41	-0.2	-0.17	-0.27	0.02	-0.02	-0.35
Global_intensity	1	0.27	-0.41	1	0.49	0.45	0.63	-0	-0.04	0.85
Sub_metering_1	0.48	0.13	-0.2	0.49	1	0.06	0.11	-0.01	-0.01	0.57
Sub_metering_2	0.44	0.15	-0.17	0.45	0.06	1	0.08	0	-0.01	0.53
Sub_metering_3	0.64	0.09	-0.27	0.63	0.11	0.08	1	0	-0.03	0.74
day	0	0	0.02	-0	-0.01	0	0	1	0.02	-0
month	-0.04	0.03	-0.02	-0.04	-0.01	-0.01	-0.03	0.02	1	-0.03
Total_power_use	0.85	0.19	-0.35	0.85	0.57	0.53	0.74	-0	-0.03	1

Top and Bottom 10 Total Power Use values wrt count

In [59]:

```
for feature in ['Total_power_use']:
    plt.figure(figsize=(15,6))
    plt.subplot(121)
    sns.barplot(y=data_mongodb[feature].value_counts()[:10], x=data_mongodb[feature].value_
    plt.ylabel('Count')
    plt.xlabel('{} values'.format(feature))
    plt.title("Top 10 {} values with respect of count".format(feature),fontsize=15, fontwei
    plt.subplot(122)
    sns.barplot(y=data_mongodb[feature].value_counts()[-10:], x=data_mongodb[feature].value
    plt.ylabel('Count')
    plt.xlabel('{} values'.format(feature))
    plt.title("Bottom 10 {} values with respect of count".format(feature),fontsize=15, font
    plt.show();
```

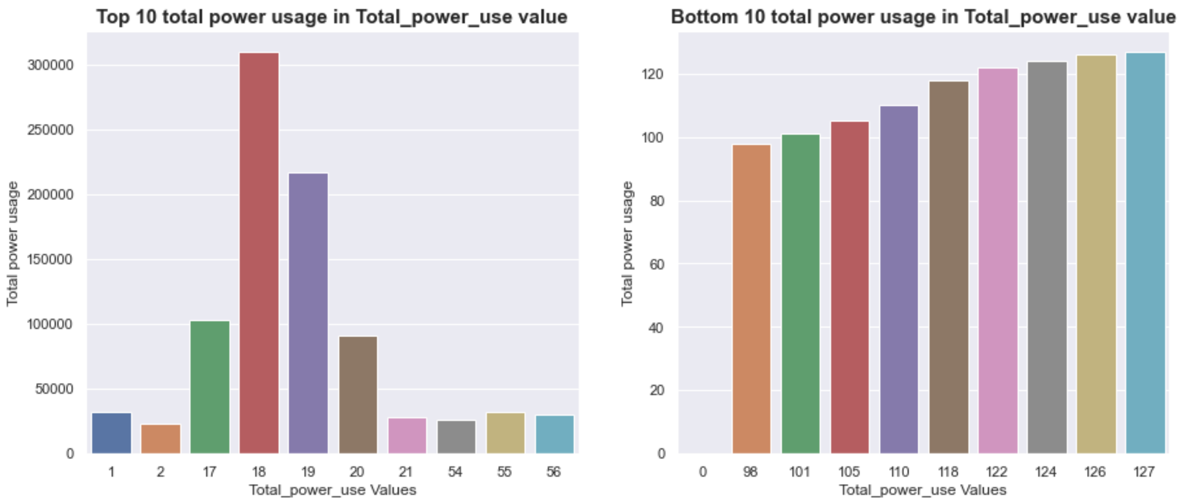




Top and Bottom 10 Total Power Use values wrt sum

In [60]:

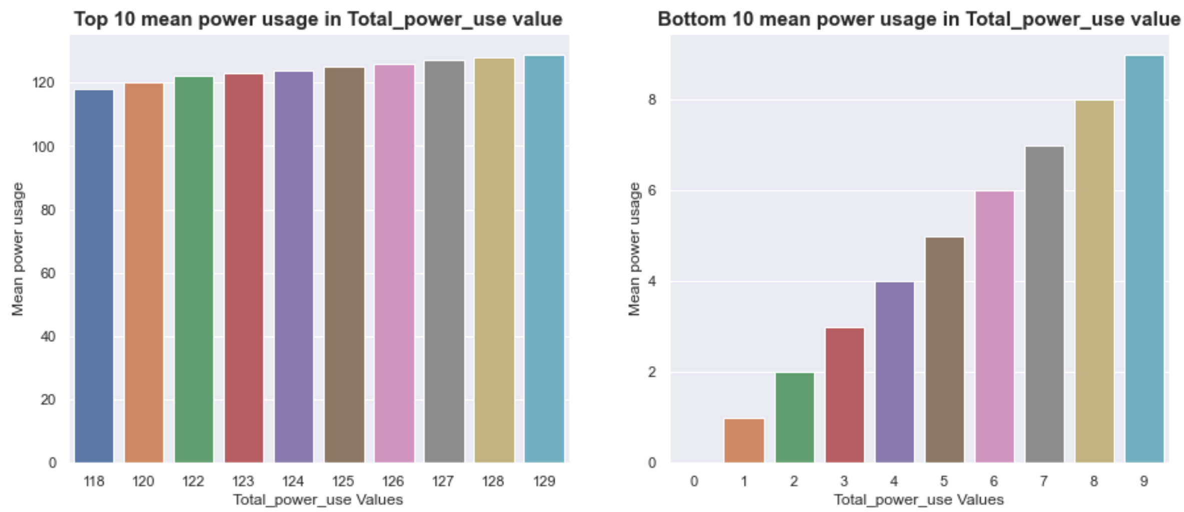
```
for feature in ['Total_power_use']:
    plt.figure(figsize=(15,6))
    plt.subplot(121)
    sns.barplot(y=data_mongodb.groupby(feature)[feature].sum().sort_values(ascending=False)
           x=data_mongodb.groupby(feature)[feature].sum().sort_values(ascending=False).inde
    plt.title("Top 10 total power usage in {} value".format(feature), fontsize=15, fontweig
    plt.ylabel("Total power usage")
    plt.xlabel('{} Values'.format(feature))
    plt.subplot(122)
    sns.barplot(y=data_mongodb.groupby(feature)[feature].sum().sort_values(ascending=False)
           x=data_mongodb.groupby(feature)[feature].sum().sort_values(ascending=False).inde
    plt.title("Bottom 10 total power usage in {} value".format(feature), fontsize=15, fontw
    plt.ylabel("Total power usage")
    plt.xlabel('{} Values'.format(feature))
    plt.show();
```



Top and Bottom 10 Total Power Use values wrt Mean

In [61]:

```
for feature in ['Total_power_use']:
    plt.figure(figsize=(15,6))
    plt.subplot(121)
    sns.barplot(y=data_mongodb.groupby(feature)[feature].mean().sort_values(ascending=False
           x=data_mongodb.groupby(feature)[feature].mean().sort_values(ascending=False).ind
    plt.title("Top 10 mean power usage in {} value".format(feature), fontsize=15, fontweigh
    plt.ylabel("Mean power usage")
    plt.xlabel('{} Values'.format(feature))
    plt.subplot(122)
    sns.barplot(y=data_mongodb.groupby(feature)[feature].mean().sort_values(ascending=False
            x=data_mongodb.groupby(feature)[feature].mean().sort_values(ascending=False).in
    plt.title("Bottom 10 mean power usage in {} value".format(feature), fontsize=15, fontwe
    plt.ylabel("Mean power usage")
    plt.xlabel('{} Values'.format(feature))
    plt.show();
```



Importing required libraries

In [62]:

```
import pandas as pd
import numpy as np

## MongoDB library

import pymongo

### Machine Learning libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso,Ridge, ElasticNet
from sklearn.svm import SVR
from sklearn.metrics import r2_score

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

Retrieving data from MongoDB

In [63]:

```
### Retrieving data from Mongodb
### Creating connection with Mongodb

client = pymongo.MongoClient("mongodb+srv://rajesh1504:rajesh1504@atlascluster.zxudjtu.mong
```

In [64]:

```
### creating database and collection in MongoDB

db =client['Power_consumption']
collection=db['Household_power_data']
```

In [65]:

```
### Locating our collection from MongoDB to DataFrame in pandas

data_from_mongodb=collection.find()
```

In [66]:

```
### converting data from MongoDb to DataFrame in pandas

data_mongodb = pd.DataFrame(data_from_mongodb)
```

In [67]:

```
### first 5 records in dataset
data_mongodb.head()
```

Out[67]:

	_id	index	Global_active_power	Global_reactive_power	Voltage	Globa
0	636cf77c65f633b9db536ad3	0	0.454	0.168	243.45	
1	636cf77c65f633b9db536ad4	1	0.140	0.000	242.60	
2	636cf77c65f633b9db536ad5	2	0.476	0.214	241.14	
3	636cf77c65f633b9db536ad6	3	3.634	0.076	232.45	
4	636cf77c65f633b9db536ad7	4	2.192	0.070	239.12	

In [ ]:

```
### Dropping _id and index feature from dataset imported from MongoDB
```

In [68]:

```
data_mongodb.drop(['_id','index'],axis=1,inplace=True)
data_mongodb.head()
```

Out[68]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_
0	0.454	0.168	243.45	2.0	0	
1	0.140	0.000	242.60	0.6	0	
2	0.476	0.214	241.14	2.2	0	
3	3.634	0.076	232.45	15.6	0	
4	2.192	0.070	239.12	9.2	0	

Model Evaluation

Seperating Independent and Dependent features

In [69]:

```
X=data_mongodb.iloc[:,:-1]
y=data_mongodb.iloc[:,-1]
X.head()
```

Out[69]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_
0	0.454	0.168	243.45	2.0	0	
1	0.140	0.000	242.60	0.6	0	
2	0.476	0.214	241.14	2.2	0	
3	3.634	0.076	232.45	15.6	0	
4	2.192	0.070	239.12	9.2	0	

In [70]:

```
y.head()
```

Out[70]:

```
0     1
1     1
2     2
3    16
4    18
Name: Total_power_use, dtype: int64
```

Train Test Split

In [71]:

```
## random state train test split will be same with all people using random_state=19

X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.25, random_state=19)
X_train.head()
```

Out[71]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
74985	0.150	0.000	233.83	0.6	0
99887	2.334	0.046	235.96	9.8	37
14533	0.250	0.046	248.99	1.2	0
59613	1.484	0.130	245.86	6.0	0
135007	1.548	0.186	240.49	6.4	0

In [72]:

```
y_train.head()
```

Out[72]:

```
74985      1
99887     37
14533      0
59613     19
135007    20
Name: Total_power_use, dtype: int64
```

In [73]:

```
X_test.head()
```

Out[73]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
17512	0.382	0.182	243.86	1.6	0
102878	1.394	0.122	244.78	5.6	1
74519	0.422	0.106	236.57	2.0	0
96052	1.426	0.214	241.67	6.0	0
103481	0.268	0.000	246.34	1.0	0

In [74]:

```
y_test.head()
```

Out[74]:

```
17512      0
102878     1
74519      3
96052     21
103481     1
Name: Total_power_use, dtype: int64
```

In [75]:

```
## Both will have same type

X_train.shape,y_train.shape
```

Out[75]:

```
((108808, 9), (108808,))
```

In [77]:

```
### Both will have same shape
X_test.shape, y_test.shape
```

Out[77]:

```
((36270, 9), (36270,))
```

In [ ]:

```
### <font color =red>Feature Scaling</font>
```

In [78]:

```
scaler=StandardScaler()
scaler
```

Out[78]:

```
StandardScaler()
```

In [80]:

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

Out[80]:

```
array([[-0.88871351, -1.09698515, -2.19091696, ..., -0.64870297,
        -1.44228507, -0.10093041],
       [ 1.1732659 , -0.6892764 , -1.52503338, ..., -0.76703931,
         0.26346955,  0.78142073],
       [-0.79430053, -0.6892764 ,  2.54842347, ..., -0.76703931,
        -1.66971902, -1.27739861],
       ...,
       [-0.73765275, -0.17520885, -0.84664494, ..., -0.64870297,
         1.51435627,  1.07553778],
       [-0.95291433, -1.09698515,  0.5007533 , ..., -0.76703931,
         1.05948837,  0.48730368],
       [ 0.71630709, -0.24611472, -0.4808779 , ...,  1.24467837,
         1.74179022, -0.98328156]])
```

In [81]:

```
X_test=scaler.transform(X_test)
X_test
```

Out[81]:

```
array([[-0.6696754 ,  0.51612338,  0.94467569, ..., -0.76703931,
         1.17320534,  1.36965483],
       [ 0.28578392, -0.01567064,  1.23228737, ..., -0.76703931,
         0.83205442,  1.66377188],
       [-0.63191021, -0.15748238, -1.33433432, ..., -0.64870297,
         1.40063929, -0.10093041],
       ...,
       [ 1.51881739, -1.09698515, -0.88103329, ...,  1.3630147 ,
        -1.44228507, -0.68916451],
       [ 0.44250946, -1.09698515,  0.89153005, ...,  1.3630147 ,
        -1.3285681 , -0.98328156],
       [ 2.11173088, -1.09698515, -0.91854786, ...,  1.24467837,
         0.4909035 , -1.27739861]])
```

Linear Regression

In [83]:

```
linear_reg=LinearRegression()
linear_reg
```

Out[83]:

```
LinearRegression()
```

In [84]:

```
linear_reg.fit(X_train,y_train)
```

Out[84]:

```
LinearRegression()
```

In [85]:

```
linear_reg_pred=linear_reg.predict(X_test)
linear_reg_pred
```

Out[85]:

```
array([-8.8817842e-14,  1.0000000e+00,  3.0000000e+00, ...,
        1.8000000e+01,  1.9000000e+01,  1.7000000e+01])
```

In [87]:

```
r2score_linear_reg=r2_score(y_test, linear_reg_pred)
print("Our Linear Regeression model has {} % accuracy".format(round(r2score_linear_reg*100,
```

```
Our Linear Regeression model has 100.0 % accuracy
```

In [88]:

```
adjusted_r2_score_linear_reg=1-((1-r2score_linear_reg)*(len(y_test)-1)/(len(y_test)-X_test.
print("Adjusted R square accuracy is {} % ".format(round(adjusted_r2_score_linear_reg*100,3
```

```
Adjusted R square accuracy is 100.0 %
```

Ridge Regression

In [90]:

```
ridge=Ridge()
ridge
```

Out[90]:

```
Ridge()
```

In [91]:

```
ridge.fit(X_train,y_train)
```

Out[91]:

```
Ridge()
```

In [92]:

```
ridge_pred = ridge.predict(X_test)
ridge_pred
```

Out[92]:

```
array([4.61639507e-05, 1.00033852e+00, 2.99997346e+00, ...,
       1.80001859e+01, 1.89998458e+01, 1.70004211e+01])
```

In [93]:

```
r2score_ridge=r2_score(y_test, ridge_pred)
print("Our Ridge Regression model has {} % accuracy".format(round(r2score_ridge*100,3)))
```

```
Our Ridge Regression model has 100.0 % accuracy
```

In [94]:

```
adjusted_r2_score_ridge=1-((1-r2score_ridge)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1
print("Adjusted R square accuracy is {} % ".format(round(adjusted_r2_score_ridge*100,3)))
```

```
Adjusted R square accuracy is 100.0 %
```

In [ ]:

```
### <font color =red>Lasso Regression</font>
```

In [95]:

```
lasso=Lasso()
lasso
```

Out[95]:

```
Lasso()
```

In [96]:

```
lasso.fit(X_train,y_train)
```

Out[96]:

```
Lasso()
```

In [98]:

```
lasso_pred=lasso.predict(X_test)
lasso_pred
```

Out[98]:

```
array([ 0.90192096,  2.85739923,  3.30951158, ..., 18.32255819,
       17.76641035, 18.2361491 ])
```

In [99]:

```
r2score_lasso=r2_score(y_test, lasso_pred)
print("Our Lasso Regression model has {} % accuracy".format(round(r2score_lasso*100,5)))
```

```
Our Lasso Regression model has 98.18654 % accuracy
```

In [100]:

```
adjusted_r2_score_lasso=1-((1-r2score_lasso)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1
print("Adjusted R square accuracy is {} % ".format(round(adjusted_r2_score_lasso*100,5)))
```

```
Adjusted R square accuracy is 98.18609 %
```

Elastic-Net Regression

In [102]:

```
elastic=ElasticNet()
elastic
```

Out[102]:

```
ElasticNet()
```

In [103]:

```
elastic.fit(X_train,y_train)
```

Out[103]:

```
ElasticNet()
```

In [104]:

```
elastic_pred = elastic.predict(X_test)
elastic_pred
```

Out[104]:

```
array([ 1.81825258,  6.25605958,  3.71954066, ..., 19.84795293,
       15.23245466, 21.85103079])
```

In [105]:

```
r2score_elastic=r2_score(y_test, elastic_pred)
print("Our Elastic-Net Regression model has {} % accuracy".format(round(r2score_elastic*100
```

```
Our Elastic-Net Regression model has 90.13101 % accuracy
```

In [106]:

```
adjusted_r2_score_elastic=1-((1-r2score_elastic)*(len(y_test)-1)/(len(y_test)-X_test.shape[
print("Adjusted R square accuracy is {} % ".format(round(adjusted_r2_score_elastic*100,5)))
```

```
Adjusted R square accuracy is 90.12856 %
```

In [ ]:

```
### <font color =red>Support Vector Regressor</font>
```

In [108]:

```
svr=SVR()
svr
```

Out[108]:

```
SVR()
```

In [109]:

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

Out[109]:

```
SVR()
```

In [110]:

```
svr_pred=svr.predict(X_test)
svr_pred
```

Out[110]:

```
array([ 0.02065745,  0.87630968,  2.85794057, ..., 18.04899273,
       18.97069549, 16.97012888])
```

In [111]:

```
r2score_svr=r2_score(y_test, svr_pred)
print("Our Support Vector Regressor model has {} % accuracy".format(round(r2score_svr*100,3
```

```
Our Support Vector Regressor model has 98.414 % accuracy
```

In [112]:

```
adjusted_r2_score_svr=1-((1-r2score_svr)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))
print("Adjusted R square accuracy is {} % ".format(round(adjusted_r2_score_svr*100,3)))
```

```
Adjusted R square accuracy is 98.414 %
```

Model Comparision

In [129]:

```
,3),round(r2score_ridge*100,3),round(r2score_lasso*100,3), round(r2score_elastic*100,3) ))
```

```
Accuracy of all the models is as below:

Linear Regression: 100.0 %
Ridge Regression: 100.0 %
Lasso Regression: 98.187 %
Elastic-Net Regression: 90.131%
Support Vector Regressor: 98.414 %

Best Model is 'Linear Regression' and 'Ridge Regression'
```

Thank you!!!

In [ ]:

```

```