DTR_Household_Power_Consumption_Regression_Problem

November 18, 2022

**

Regression Problem (Decsision Trees Regressor and Extra Trees Regressor and Hyperparameter Tuning)

* To predict daily power consumption**

Datatable : To read large dataset

[1]:
```
# !pip install datatable
```

Import required libraries

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
# import datatable as dt
%matplotlib inline
```

Complete dataset is available on my GitHub * GitHub Link: https://github.com/subhashdixit/Regression_Model_Tasks/tree/main/Household_Power_Consumption_Regres

[3]:
```
# # Datatable python library
# datatable_df = dt.fread("/content/drive/MyDrive/FSDS_Job_Gurantee/Tasks/
↪Regression_Problems/Household_Power_Consumption_Regression_Problem/
↪household_power_consumption.txt")
```

Convert Datatable into Pandas Dataframe

[4]:
```
# df = datatable_df.to_pandas()
```

[5]:
```
# df.head()
```

[6]:
```
# df.columns
```

[7]:
```
# df.shape
```

1 Sampling

- **Take 100000 samples out of 2075259**

[8]:
```
# df=df.sample(100000).reset_index().drop('index',axis=1)
```

[9]:
```
# df.head()
```

Store sample taken into csv for faster operation in future and also to avoid sampling every time. If we do sampling evry time then our results will be impacted

[10]:
```
# from google.colab import files
# df.to_csv('household_power_consumption_100000_samples.csv')
# files.download('household_power_consumption_100000_samples.csv')
```

Read Data From GitHub

Sample dataset is also available on my GitHub * GitHub Link: https://raw.githubusercontent.com/subhashdixit/Regression_Model_Tasks/main/Household_Power_Consumpt

[11]:
```
url = 'https://raw.githubusercontent.com/subhashdixit/Regression_Model_Tasks/
↪main/Household_Power_Consumption_Regression_Problem/
↪household_power_consumption_100000_samples.csv'
df = pd.read_csv(url)
```

[12]:
```
df.head()
```

[12]:
```
   Unnamed: 0       Date      Time Global_active_power Global_reactive_power  \
0           0  23/9/2007  20:53:00               4.548                 0.048
1           1  16/6/2008  05:44:00               0.332                 0.266
2           2  20/7/2010  11:26:00               0.456                 0.300
3           3  21/5/2008  07:58:00               1.402                 0.110
4           4  16/6/2007  13:07:00               1.662                 0.114

   Voltage Global_intensity Sub_metering_1 Sub_metering_2  Sub_metering_3
0  233.920           19.400         36.000          0.000            17.0
1  240.490            1.800          0.000          1.000             1.0
2  241.510            2.400          0.000          0.000             0.0
3  238.880            5.800          0.000          0.000            18.0
4  240.190            7.000          0.000          1.000            17.0
```

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) * **We have taken 100000 samples only to predict power consumption**

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.

[13]:
```
df.columns
```

[13]:
```
Index(['Unnamed: 0', 'Date', 'Time', 'Global_active_power',
       'Global_reactive_power', 'Voltage', 'Global_intensity',
       'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3'],
      dtype='object')
```

Drop "Unnamed: 0" column because it is of no use

[14]:
```
df.drop(['Unnamed: 0'], axis = 1, inplace = True)
```

2 EDA

Informaation about the dataset

[15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Date                   100000 non-null  object
 1   Time                   100000 non-null  object
 2   Global_active_power    100000 non-null  object
 3   Global_reactive_power  100000 non-null  object
 4   Voltage                100000 non-null  object
 5   Global_intensity       100000 non-null  object
 6   Sub_metering_1         100000 non-null  object
 7   Sub_metering_2         100000 non-null  object
 8   Sub_metering_3         98733 non-null   float64
dtypes: float64(1), object(8)
memory usage: 6.9+ MB
```

We will do our analysis on the basis of Daily Data and ignore time column

[16]:
```
df['Date'] = pd.to_datetime(df['Date'])
```

[17]:
```
df.drop(['Time'], axis = 1, inplace  = True)
```

[18]:
```
df.head()
```

[18]:
```
        Date Global_active_power Global_reactive_power  Voltage  \
0 2007-09-23               4.548                 0.048  233.920
1 2008-06-16               0.332                 0.266  240.490
2 2010-07-20               0.456                 0.300  241.510
3 2008-05-21               1.402                 0.110  238.880
4 2007-06-16               1.662                 0.114  240.190

  Global_intensity Sub_metering_1 Sub_metering_2  Sub_metering_3
0           19.400         36.000          0.000            17.0
1            1.800          0.000          1.000             1.0
2            2.400          0.000          0.000             0.0
3            5.800          0.000          0.000            18.0
4            7.000          0.000          1.000            17.0
```

[19]:
```
df.isnull().sum()
```

[19]:
```
Date                      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         1267
dtype: int64
```

[20]:
```
df.duplicated().sum()
```

[20]:
```
1326
```

Drop duplicates data

[21]:
```
df.drop_duplicates(inplace = True)
```

[22]:
```
df.columns
```

[22]:
```
Index(['Date', 'Global_active_power', 'Global_reactive_power', 'Voltage',
       'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3'],
      dtype='object')
```

[23]:
```
df['Sub_metering_1'].unique()
```

[23]:
```
array(['36.000', '0.000', '2.000', '?', '3.000', '1.000', '37.000',
       '12.000', '13.000', '7.000', '33.000', '38.000', '39.000', '0.0',
       '40.000', '19.000', '31.000', '14.000', '29.000', '42.000',
       '35.000', '8.000', '5.000', '27.000', '18.000', '10.000', '16.000',
       '1.0', '9.000', '6.000', '21.000', '25.000', '43.000', '34.000',
       '52.000', '22.000', '11.000', '49.000', '41.000', '51.000',
       '24.000', '32.000', '44.000', '23.000', '47.000', '30.000',
       '15.000', '76.000', '53.000', '26.000', '20.000', '37.0', '28.000',
       '17.000', '2.0', '4.000', '45.000', '79.000', '38.0', '48.000',
       '78.000', '73.000', '75.000', '12.0', '74.000', '77.000', '46.000',
       '70.000', '54.000', '71.000', '67.000', '50.000', '60.000',
       '72.000', '9.0', '55.000'], dtype=object)
```

[24]:
```
df.replace('?', np.nan, inplace=True)
```

[25]:
```
df.isnull().sum()
```

[25]:
```
Date                    0
Global_active_power    29
Global_reactive_power  29
```

```
Voltage                  29
Global_intensity         29
Sub_metering_1           29
Sub_metering_2           29
Sub_metering_3           29
dtype: int64
```

[26]:
```
df.dropna(how = 'any', inplace = True)
```

[27]:
```
# df.fillna(df.median().round(1), inplace=True)
```

[28]:
```
df.isnull().sum()
```

[28]:
```
Date                     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
```

Convert data to float datatype because all values are in decimal

[29]:
```
convert_data = {'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'}
df = df.astype(convert_data)
```

Take date wise data only

[30]:
```
df= df.groupby('Date').sum()
```

[31]:
```
df.reset_index(inplace = True)
```

[32]:
```
df['year']=df['Date'].dt.year
df['month']=df['Date'].dt.month
```

[33]:
```
df.groupby('year').sum()
```

[33]:
```
      Global_active_power  Global_reactive_power     Voltage  \
year
2006             2089.234                145.794   266246.20
2007            27953.588               2934.368  6026508.66
```

```
2008           27070.042                2976.726  6106483.96
2009           26946.336                3304.390  6046970.02
2010           23376.046                2867.062  5311922.51

      Global_intensity  Sub_metering_1  Sub_metering_2  Sub_metering_3  month
year
2006            8834.6          1319.0          2443.0          7970.0    192
2007          119247.4         32306.0         41834.0        144122.0   2378
2008          114927.2         27497.0         32388.0        152028.0   2384
2009          113853.8         27616.0         27921.0        171174.0   2370
2010           98698.0         21101.0         24204.0        158850.0   1981
```

[34]:
```python
df.groupby('month').sum()
```

[34]:
```
       Global_active_power  Global_reactive_power     Voltage  \
month
1                11008.308                947.904  2006703.97
2                 8152.446                821.946  1828442.49
3                 9745.836                988.560  2018621.08
4                 8474.336               1001.432  1956011.97
5                 8809.694               1078.724  2002497.07
6                 7884.870               1171.386  1959560.14
7                 6921.870               1138.596  2034521.85
8                 6927.570               1128.382  1983726.10
9                 8492.082               1049.824  1947568.65
10                9680.734               1017.446  2097908.71
11               10113.624                915.628  1928346.74
12               11223.876                968.512  1994222.58

       Global_intensity  Sub_metering_1  Sub_metering_2  Sub_metering_3  \
month
1               46386.6         11270.0         10824.0         61548.0
2               34349.2          6770.0          9704.0         46932.0
3               41139.8         10014.0         13379.0         58749.0
4               35922.0          8530.0         10991.0         50916.0
5               37616.8         10187.0         11647.0         55326.0
6               33819.6          9297.0         10440.0         49966.0
7               29844.6          6263.0          9510.0         41610.0
8               29753.4          7099.0          9424.0         44356.0
9               36132.8         10411.0         10226.0         51863.0
10              40851.6          9198.0         10639.0         55897.0
11              42647.4         10149.0         10803.0         55449.0
12              47097.2         10651.0         11203.0         61532.0

         year
month
1      245034
```

```
2        224950
3        247044
4        237003
5        247044
6        234992
7        247044
8        239004
9        234990
10       247044
11       230970
12       240950
```

Drop year and month column. We have created these two just to perform basic analyis

[35]:
```python
df.shape
```

[35]:
```
(1432, 10)
```

[36]:
```python
df.duplicated().sum()
```

[36]:
```
0
```

Remove year- 2006 because it may create problem while analysis

[37]:
```python
df = df[df['Date']>'2006-12-31']
```

[38]:
```python
df.shape
```

[38]:
```
(1416, 10)
```

[39]:
```python
df.isnull().sum()
```

[39]:
```
Date                     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
year                     0
month                    0
dtype: int64
```

- Global active power is the real power consumption i.e. the power consumed by electrical appliances other than the sub metered appliances.
- Active energy consumed every minute (in watt hour) $= \frac{Global\_active\_power*1000}{60} - Sub\_metering\_1 - Sub\_metering\_2 - Sub\_metering\_3)$

- Active energy consumed every minute (in watt hour) 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

Calculation of target variable - "power_consumption"

[40]:
```
a = (df['Global_active_power']*1000/60)
b = df['Sub_metering_1'] + df['Sub_metering_2'] + df['Sub_metering_3']
df['power_consumption'] = a - b
df.head()
```

[40]:
```
          Date  Global_active_power  Global_reactive_power   Voltage  \
16 2007-01-01              136.092                  8.790  17063.52
17 2007-01-02               96.230                  7.922  20438.87
18 2007-01-03               25.416                  4.938  17871.77
19 2007-01-04               97.752                  8.124  14654.69
20 2007-01-05               72.016                  7.874  15283.45

    Global_intensity  Sub_metering_1  Sub_metering_2  Sub_metering_3  year  \
16             563.4             0.0            17.0           209.0  2007
17             403.2             8.0            14.0           681.0  2007
18             105.0             0.0            15.0            89.0  2007
19             410.4            88.0           130.0           795.0  2007
20             316.2           338.0            28.0           300.0  2007

    month  power_consumption
16      1        2042.200000
17      1         900.833333
18      1         319.600000
19      1         616.200000
20      1         534.266667
```

Sum all the values of sub meters into one features i.e., "Sub_metering"

[41]:
```
df['Sub_metering']=df['Sub_metering_1']+df['Sub_metering_2']+df['Sub_metering_3']
```

[42]:
```
df = df.drop(['Sub_metering_1','Sub_metering_2','Sub_metering_3'],axis=1)
```

[43]:
```
df.head()
```

[43]:
```
          Date  Global_active_power  Global_reactive_power   Voltage  \
16 2007-01-01              136.092                  8.790  17063.52
17 2007-01-02               96.230                  7.922  20438.87
18 2007-01-03               25.416                  4.938  17871.77
19 2007-01-04               97.752                  8.124  14654.69
20 2007-01-05               72.016                  7.874  15283.45

    Global_intensity  year  month  power_consumption  Sub_metering
```

```
16          563.4  2007     1     2042.200000       226.0
17          403.2  2007     1      900.833333       703.0
18          105.0  2007     1      319.600000       104.0
19          410.4  2007     1      616.200000      1013.0
20          316.2  2007     1      534.266667       666.0
```

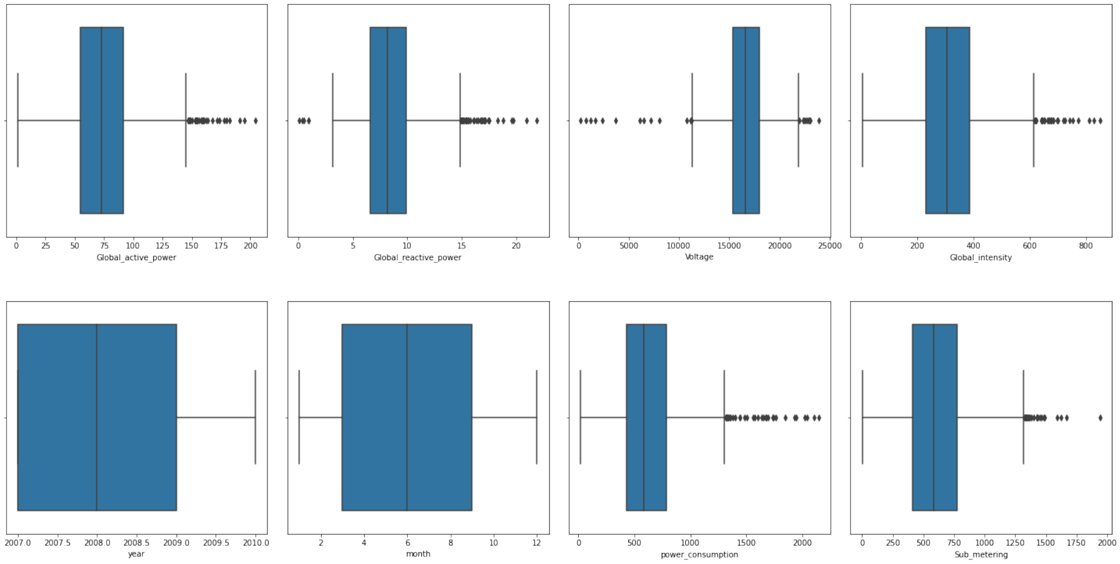
[44]:
```
df.isnull().sum()
```

[44]:
```
Date                     0
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
year                     0
month                    0
power_consumption        0
Sub_metering             0
dtype: int64
```

3 Graphical Analysis

3.1 Outliers

[45]:
```
fig, ax = plt.subplots(ncols=4, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  if col!='Date':
    sns.boxplot(x = col, data = df, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



[46]:
```
def find_boundaries(df, variable, distance):
    IQR = df[variable].quantile(0.75) - df[variable].quantile(0.25)
    lower_boundary = df[variable].quantile(0.25) - (IQR*distance)
    upper_boundary = df[variable].quantile(0.75) + (IQR*distance)
    return upper_boundary, lower_boundary
```

[47]:
```
outliers_columns = ['Global_active_power',␣
↪'Global_reactive_power','Voltage','Global_intensity','power_consumption','Sub_metering']
for i in outliers_columns:
    upper_boundary, lower_boundary = find_boundaries(df,i, 1.5)
    outliers = np.where(df[i] > upper_boundary, True, np.where(df[i] <␣
↪lower_boundary, True, False))
    outliers_df = df.loc[outliers, i]
    df_trimed= df.loc[~outliers, i]
    df[i] = df_trimed
```

[48]:
```
df.isnull().sum()
```

[48]:
```
Date                      0
Global_active_power      33
Global_reactive_power    42
Voltage                  23
Global_intensity         32
year                      0
month                     0
power_consumption        34
Sub_metering             24
dtype: int64
```

[49]:
```
df.fillna(df.median().round(1), inplace=True)
```

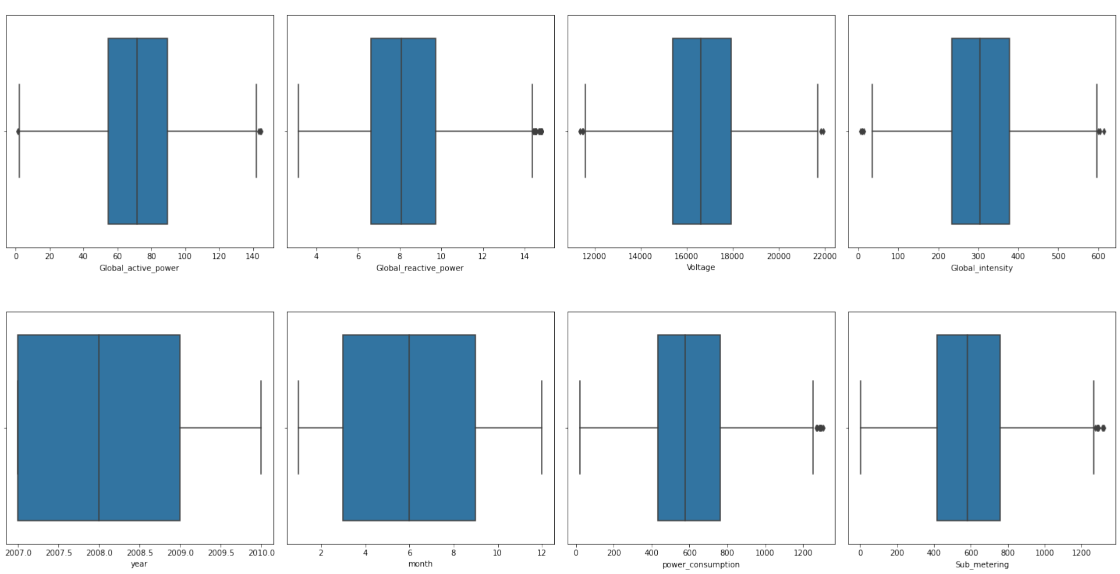
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning:
DataFrame.mean and DataFrame.median with numeric_only=None will include
datetime64 and datetime64tz columns in a future version.
  """Entry point for launching an IPython kernel.
```

[50]:
```
df.dropna(inplace = True)
```

[51]:
```
df.isnull().sum()
```

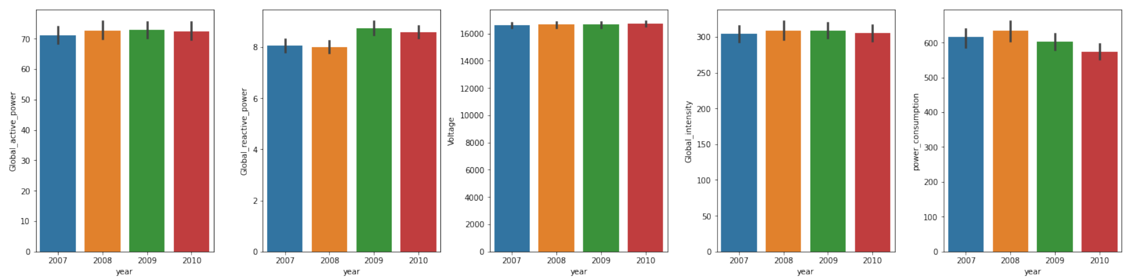
[51]:
```
Date                     0
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
year                     0
month                    0
power_consumption        0
Sub_metering             0
dtype: int64
```

[52]:
```
fig, ax = plt.subplots(ncols=4, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  if col!='Date':
    sns.boxplot(x = col, data = df, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

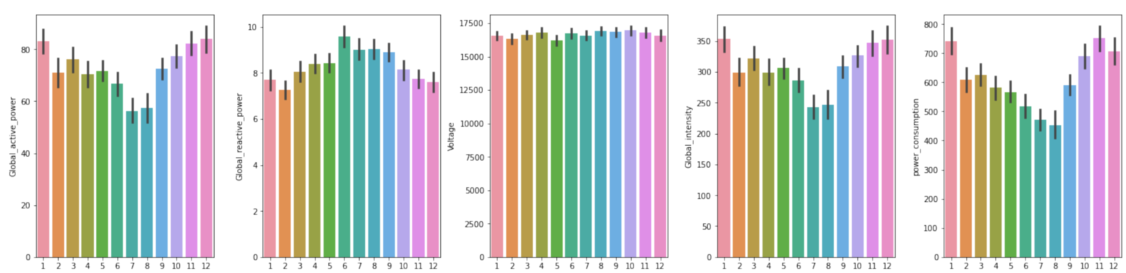


3.2 Barplot

[53]:
```
fig, ax = plt.subplots(ncols = 5, nrows = 1, figsize=(20,5))
index = 0
ax = ax.flatten()
for col, value in df.items():
  if col not in ['Date', 'year', 'month']:
    sns.barplot(y = df[col], x = df['year'], data = df, ax=ax[index] )
    index += 1
  if index == 5:
    break
plt.tight_layout(pad=1, w_pad=1, h_pad=10.0)
```



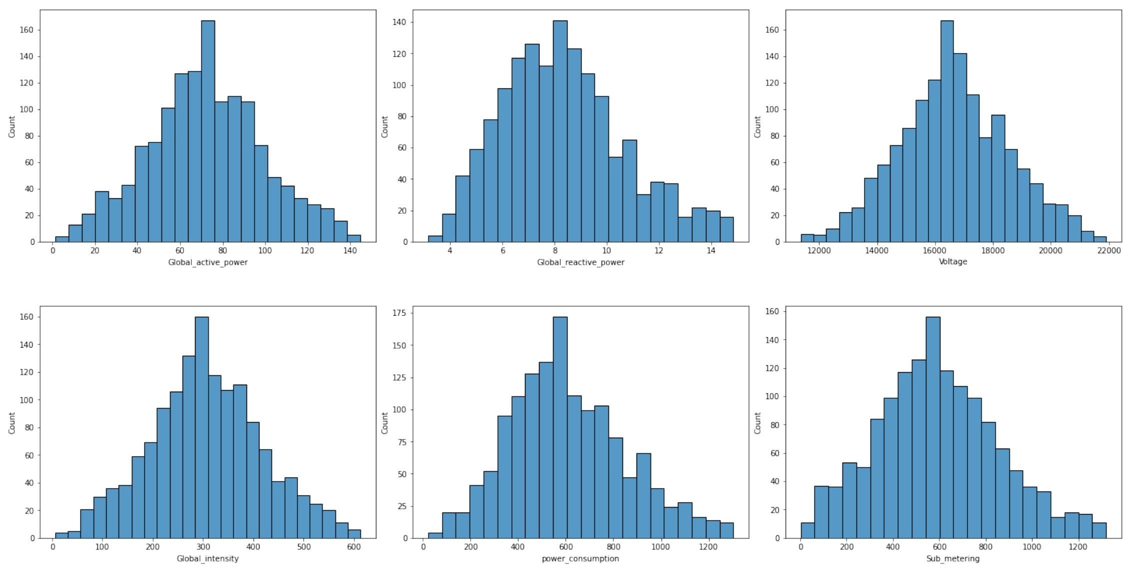
[54]:
```
fig, ax = plt.subplots(ncols = 5, nrows = 1, figsize=(20,5))
index = 0
ax = ax.flatten()
for col, value in df.items():
  if col not in ['Date', 'year', 'month']:
    sns.barplot(y = df[col], x = df['month'], data = df, ax=ax[index] )
    index += 1
  if index == 5:
    break
plt.tight_layout(pad=1, w_pad=1, h_pad=10.0)
```



Observation * Power consumption in November and January are on higher side * Voltage is almost equal in every month

3.3 Histplot

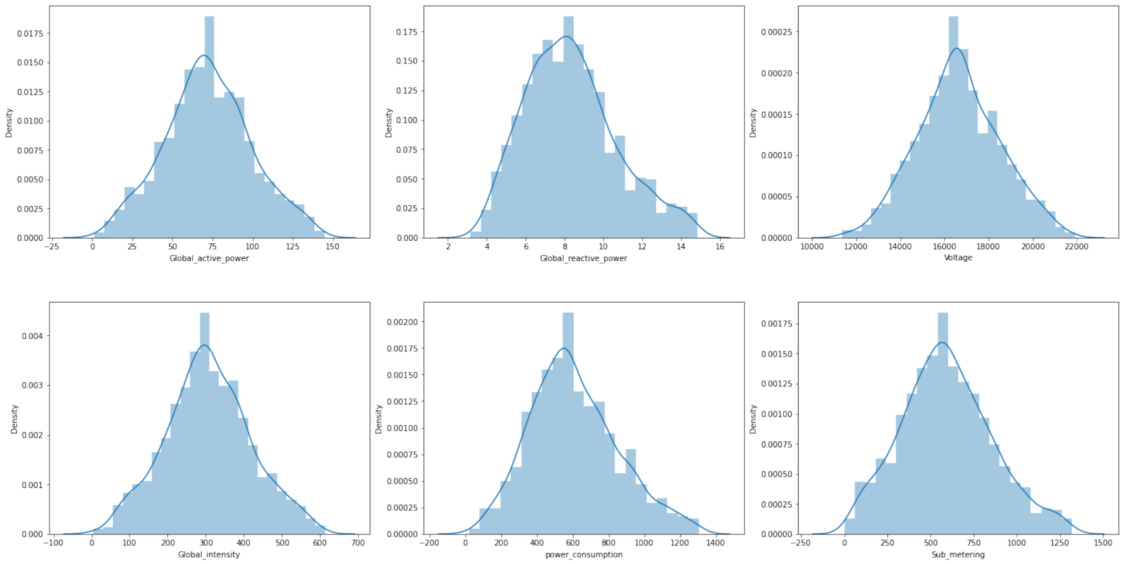
[55]:
```
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
    if col not in ['Date', 'year', 'month']:
      sns.histplot(value, ax=ax[index])
      index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



3.4 Distplot

[56]:
```
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  if col not in ['Date', 'year', 'month']:
    sns.distplot(value, ax=ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



4 Statistical Analysis

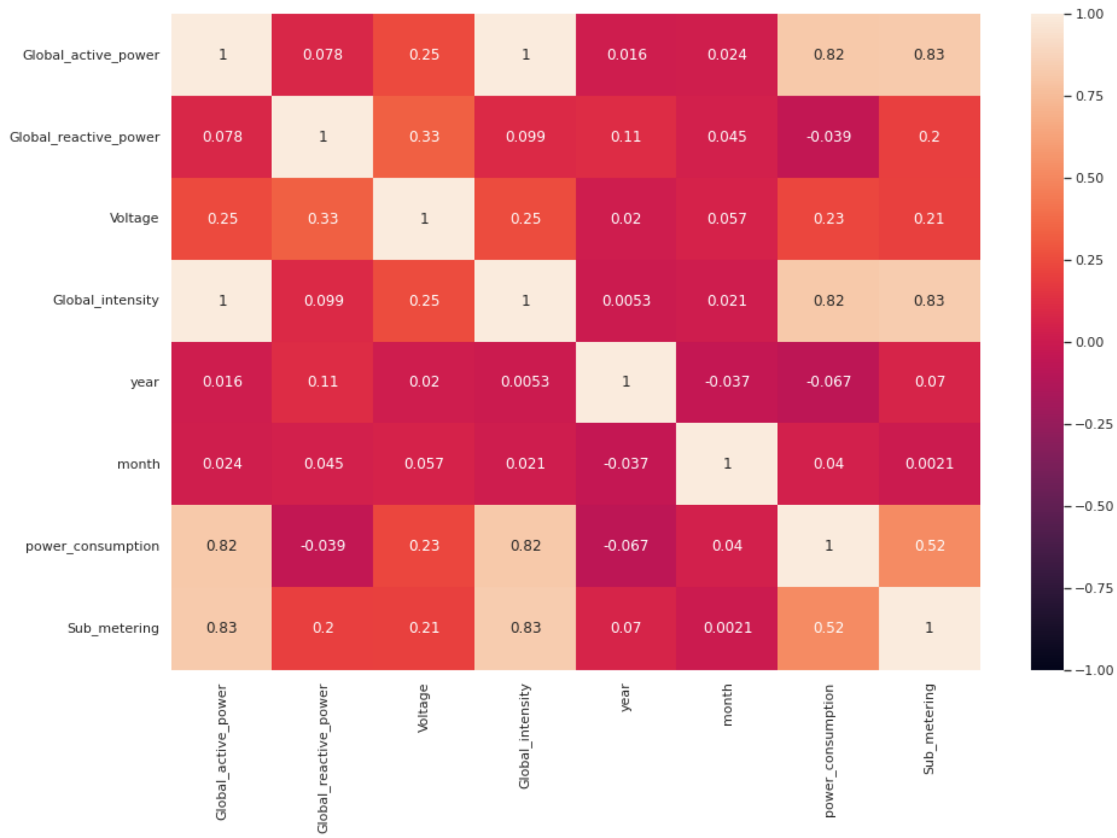
[57]:
```
df.corr()
```

[57]:
```
                       Global_active_power  Global_reactive_power   Voltage  \
Global_active_power               1.000000               0.078498  0.248528
Global_reactive_power             0.078498               1.000000  0.329612
Voltage                           0.248528               0.329612  1.000000
Global_intensity                  0.996324               0.099367  0.253729
year                              0.015729               0.113831  0.020252
month                             0.024475               0.044594  0.056966
power_consumption                 0.819335              -0.039454  0.227614
Sub_metering                      0.827409               0.196993  0.208694

                       Global_intensity      year     month  \
Global_active_power            0.996324  0.015729  0.024475
Global_reactive_power          0.099367  0.113831  0.044594
Voltage                        0.253729  0.020252  0.056966
Global_intensity               1.000000  0.005334  0.021361
year                           0.005334  1.000000 -0.036917
month                          0.021361 -0.036917  1.000000
power_consumption              0.818675 -0.067469  0.039837
Sub_metering                   0.832383  0.069651  0.002115

                       power_consumption  Sub_metering
Global_active_power             0.819335      0.827409
```

```
Global_reactive_power          -0.039454      0.196993
Voltage                         0.227614      0.208694
Global_intensity                0.818675      0.832383
year                           -0.067469      0.069651
month                           0.039837      0.002115
power_consumption               1.000000      0.520581
Sub_metering                    0.520581      1.000000
```

[58]:
```python
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(data=df.corr(), annot=True,  vmin=-1, vmax=1)
```

[58]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc9c7e72810>



Observation * Global_active_power, Global_intensity and sub_metering are higly correlated

[59]:
```python
df.describe().T
```

[59]:
```
                        count          mean          std           min  \
Global_active_power    1416.0     72.277684    27.051160      1.372000
Global_reactive_power  1416.0      8.335674     2.362348      3.160000
Voltage                1416.0  16657.260410  1895.779688  11361.000000
```

```
Global_intensity       1416.0     306.822458   112.187710     5.800000
year                   1416.0    2008.455508     1.104183  2007.000000
month                  1416.0       6.435734     3.416252     1.000000
power_consumption      1416.0     607.389736   245.578489    20.866667
Sub_metering           1416.0     593.437853   262.875053     2.000000

                              25%      50%        75%          max
Global_active_power       54.5390     71.6    89.7280   144.784000
Global_reactive_power      6.6215      8.1     9.7285    14.832000
Voltage                15396.3875  16618.0 17948.9475 21911.440000
Global_intensity         233.1500    304.0   379.1000   613.600000
year                    2007.0000   2008.0  2009.0000  2010.000000
month                      3.0000      6.0     9.0000    12.000000
power_consumption        432.9750    579.6   761.0750  1304.166667
Sub_metering             415.0000    579.5   757.2500  1320.000000
```

Observation * Maximum power consumption in a day is 2146 w/h * Average consumption is 631 w/h * Minimum cosmption is 21 w/h

5 Segregating Independent and Dependent Features

[60]:
```
X = df.iloc[ : , [1,2,3,4,6,8]]
y = df.iloc[ : , -2]
```

[61]:
```
X.shape
```

[61]:
```
(1416, 6)
```

[62]:
```
y.shape
```

[62]:
```
(1416,)
```

[63]:
```
X.head()
```

[63]:
```
    Global_active_power  Global_reactive_power   Voltage  Global_intensity  \
16              136.092                  8.790  17063.52             563.4
17               96.230                  7.922  20438.87             403.2
18               25.416                  4.938  17871.77             105.0
19               97.752                  8.124  14654.69             410.4
20               72.016                  7.874  15283.45             316.2

    month  Sub_metering
16      1         226.0
17      1         703.0
18      1         104.0
```

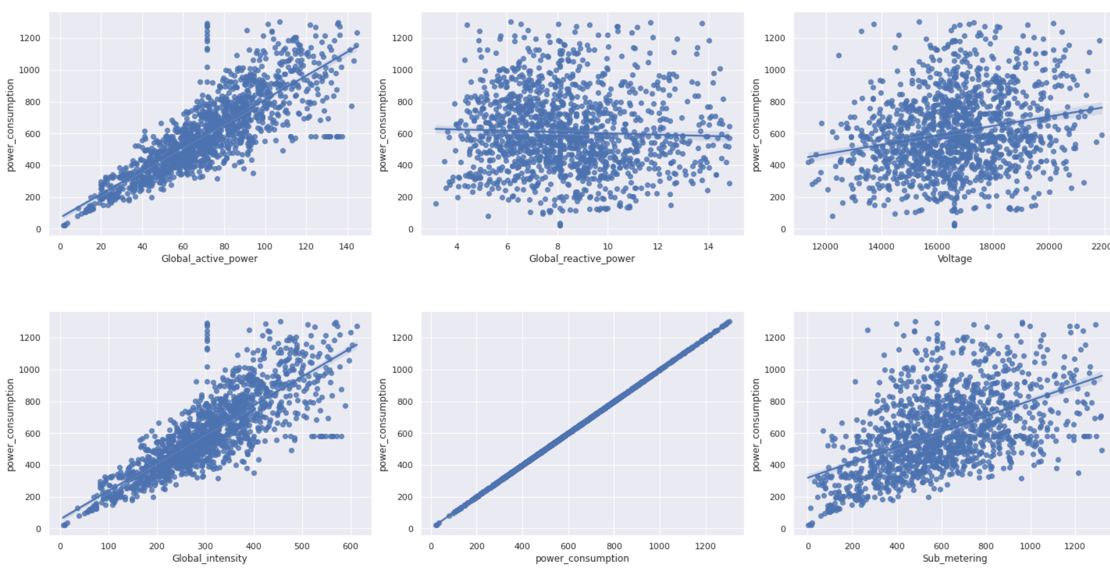
```
19       1          1013.0
20       1           666.0
```

[64]:
```
y.head()
```

[64]:
```
16    579.600000
17    900.833333
18    319.600000
19    616.200000
20    534.266667
Name: power_consumption, dtype: float64
```

Regplot

[65]:
```
fig, ax = plt.subplots(ncols=3, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  if col not in ['Date', 'year', 'month']:
    sns.regplot(x = df[col],y = df["power_consumption"], data = df , ax =␣
 ↪ax[index])
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



6 Train Test Split

[66]:
```
from sklearn.model_selection import train_test_split
```

[67]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=7,test_size=0.
↪33)
```

7 Scaling

- **Not required for Decision Trees**

[68]:
```
# from sklearn.preprocessing import StandardScaler
```

[69]:
```
# scaler=StandardScaler()
```

[70]:
```
# X_train = scaler.fit_transform(X_train)
```

[71]:
```
# X_test = scaler.transform(X_test)
```

[72]:
```
# len(X_train)
```

[73]:
```
# mnm
```

8 Save Preprocess Model Data Using Pickle

[74]:
```
# preprocess_model = [X_train,y_train,X_test,y_test]
```

[75]:
```
# import pickle
```

[76]:
```
# pickle.dump(preprocess_model, open('preprocess_model.pkl','wb'))
```

[77]:
```
# preprocess_model = pickle.load(open('preprocess_model.pkl','rb'))
```

Note * We have successfully stored our scaled data into pickel file so we can use it further in other file by just importing it

9 Save Data into MongoDb

[78]:
```
# y_train.T
```

```
[79]: # database_df = pd.DataFrame([X_train.T[0],X_train.T[1],X_train.T[2],X_train.
       ↪T[3], X_train.T[4], X_train.T[5],y_train]).T
```

```
[80]: # database_df.columns=['Global_active_power', 'Global_reactive_power',␣
       ↪'Voltage', 'Global_intensity',  'month', 'Sub_metering', 'power_consumption']
```

```
[81]: # database_df.head()
```

```
[82]: # l=[]
      # for i ,row in database_df.iterrows():
      #   l.append(dict(row))
```

```
[83]: # import pymongo
      # from pymongo import MongoClient
```

```
[84]: # client = pymongo.MongoClient("mongodb+srv://subhashdixit17:Anushka27@cluster0.
       ↪elq8eyt.mongodb.net/?retryWrites=true&w=majority")
```

```
[85]: # db=client['Household_Power_Preprocessed_Data']
      # collections = db['Training__Independent_and_Dependent_Dataset']
      # collections.insert_many(l)
```

10 Load Preprocessed data using Pickle

```
[86]: # preprocess_model = pickle.load(open('preprocess_model.pkl','rb'))
```

```
[87]: # X_train = preprocess_model[0]
      # y_train = preprocess_model[1]
      # X_test = preprocess_model[2]
      # y_test = preprocess_model[3]
```

```
[88]: # X_train =pd.DataFrame(X_train)
      # X_test =pd.DataFrame(X_test)
      # X_train.columns=['Global_active_power', 'Global_reactive_power', 'Voltage',␣
       ↪'Global_intensity', 'month', 'Sub_metering']
      # X_test.columns=['Global_active_power', 'Global_reactive_power', 'Voltage',␣
       ↪'Global_intensity',  'month','Sub_metering']
```

11 VIF Check

- **To check multicollinearity**

[89]:
```
# X_train2 = X_train.copy()
# X_train= pd.DataFrame(X_train)
```

[90]:
```
# X_train
```

[91]:
```
# from statsmodels.stats.outliers_influence import variance_inflation_factor
# vif = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
↪shape[1])]
# print(X_train.columns)
# print(vif)
```

[92]:
```
# while (max(vif) > 5):
#     indx = vif.index(max(vif)) #Get the index of variable with highest VIF
#     print(indx)
#     X_train.drop(X_train.columns[indx],axis = 1, inplace = True)
#     vif = [variance_inflation_factor(X_train.values, i) for i in␣
↪range(X_train.shape[1])]
# vif = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
↪shape[1])]
# print(X_train.columns)
# print(vif)
```

[93]:
```
# X_test = pd.DataFrame(X_test)
# X_test = X_test[X_train.columns]
```

12 Model Creation

[94]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import ExtraTreeRegressor
from sklearn.model_selection import GridSearchCV
```

[95]:
```
parameters = {
            "splitter":["best","random"],
            "max_depth" : [1,3,5,7,9,11,12],
           "min_samples_leaf":[1,2,3,4,5,6,7,8,9,10],
           "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4,0.5],
           "max_features":["auto","log2","sqrt",None],
           "max_leaf_nodes":[None,10,20,30,40,50,60,70,80,90]
            }
## We will train that models
models = {
 1: DecisionTreeRegressor(random_state=0),
```

```
 2: ExtraTreeRegressor(random_state=0),
 3: GridSearchCV(DecisionTreeRegressor(random_state=42), param_grid=parameters,␣
↪verbose=1, cv=3),
 4: GridSearchCV(ExtraTreeRegressor(random_state=42), param_grid=parameters,␣
↪verbose=1, cv=3)
}
```

[96]:
```
map_keys = list(models.keys())
```

[97]:
```
# Get model name using id from linear_model_collection
def get_model_building_technique_name(num):
 if num == 1:
  return 'DecisionTreeRegressor()'
 if num == 2:
  return 'ExtraTreeRegressor()'
 if num == 3:
  return "GridSearchCV()_DTR"
 if num == 4:
  return "GridSearchCV()_ETR"
 return ''
```

[98]:
```
results = [];
for key_index in range(len(map_keys)):
  key = map_keys[key_index]
  model = models[key]
  print(key_index)
  model.fit(X_train, y_train)

  '''Test Accuracy'''
  y_pred = model.predict(pd.DataFrame(X_test))

  R_Squared_Test = r2_score(y_test,y_pred)
  Adjusted_R_Squared_Test = (1 - (1-R_Squared_Test)*(len(y_test)-1)/
↪(len(y_test)-X_test.shape[1]-1))


  '''Train Accuracy'''
  y_pred_train = model.predict(X_train)

  R_Squared_Train = r2_score(y_train,y_pred_train)
  Adjusted_R_Squared_Train = (1 - (1-R_Squared_Train)*(len(y_train)-1)/
↪(len(y_train)-X_test.shape[1]-1))

  results.append({
      'Model Name' : get_model_building_technique_name(key),
      'Trained Model' : model,
      'R_Squared_Test' : R_Squared_Test,
```

```
            'Adjusted_R_Squared_Test' : Adjusted_R_Squared_Test,
            'R_Squared_Train' : R_Squared_Train,
            'Adjusted_R_Squared_Train' : Adjusted_R_Squared_Train
            })
```

```
0
1
2
Fitting 3 folds for each of 28000 candidates, totalling 84000 fits
3
Fitting 3 folds for each of 28000 candidates, totalling 84000 fits
```

12.1 Train and Test Accuracy

[99]:
```
result_df = pd.DataFrame(results)
result_df
```

[99]:
```
                Model Name                                    Trained Model  \
0  DecisionTreeRegressor()              DecisionTreeRegressor(random_state=0)
1     ExtraTreeRegressor()                 ExtraTreeRegressor(random_state=0)
2        GridSearchCV()_DTR  GridSearchCV(cv=3, estimator=DecisionTreeRegre…
3        GridSearchCV()_ETR  GridSearchCV(cv=3, estimator=ExtraTreeRegresso…

   R_Squared_Test  Adjusted_R_Squared_Test  R_Squared_Train  \
0        0.879728                 0.878163          1.00000
1        0.858121                 0.856275          1.00000
2        0.712480                 0.708737          0.72561
3        0.712480                 0.708737          0.72561

   Adjusted_R_Squared_Train
0                   1.00000
1                   1.00000
2                   0.72386
3                   0.72386
```

12.2 Checking Best Model

[100]:
```
Best_Model_Name = 'GridSearchCV()_DTR' # Because Train and Test accuarcy are in␣
 ↪range. It means our model is not overfitted.
```

12.3 Save Best Model

[102]:
```
import pickle
Best_Trained_model = Best_Model_Name
with open('DTR_Household_Power_Consumption_Regression_Problem.sav', 'wb') as␣
 ↪best_model_pickle:
 pickle.dump(Best_Trained_model, best_model_pickle)
```

**

The End

**