# Individual Household Electric Power Consumption

### Aim : To predict the individual household electricity consumption depending on the following attributes.

## 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:

- (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.
- 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:

- date: Date in format dd/mm/yyyy
- time: time in format hh:mm:ss
- global_active_power: household global minute-averaged active power (in kilowatt)
- global_reactive_power: household global minute-averaged reactive power (in kilowatt)
- voltage: minute-averaged voltage (in volt)
- global_intensity: household global minute-averaged current intensity (in ampere)
- 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).
- 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.
- 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.

### Dataset link: https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power

In [1]:
```python
# Importing required libs

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Reading dataset
df = pd.read_csv(r"""E:\Learning Files\Data Science\dataset\household_power_consum
```

In [3]:
```
# Getting a particular data sample for ease of calculation. Real data having over 2
df = df.sample(n=30000, random_state=200, ignore_index=True)
```

In [4]:
```
# Looking top 5 data row to get idea about the dataset
df.head()
```

Out[4]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Su |
|---|---|---|---|---|---|---|---|
| **0** | 27/7/2007 | 10:25:00 | 0.196 | 0.074 | 234.170 | 0.800 | |
| **1** | 28/8/2010 | 18:07:00 | 1.262 | 0.054 | 241.420 | 5.200 | |
| **2** | 30/9/2008 | 00:48:00 | 0.344 | 0.076 | 242.460 | 1.600 | |
| **3** | 12/10/2009 | 06:54:00 | 2.672 | 0.000 | 238.840 | 11.400 | |
| **4** | 10/1/2010 | 23:56:00 | 0.254 | 0.000 | 250.220 | 1.000 | |

In [5]:
```
# Looking below 5 data row
df.tail()
```

Out[5]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity |
|---|---|---|---|---|---|---|
| **29995** | 6/11/2009 | 09:32:00 | 1.746 | 0.052 | 240.800 | 7.200 |
| **29996** | 5/5/2009 | 16:39:00 | 0.404 | 0.134 | 243.810 | 1.800 |
| **29997** | 5/2/2010 | 09:04:00 | 1.794 | 0.168 | 238.480 | 7.400 |
| **29998** | 25/8/2009 | 19:14:00 | 2.416 | 0.586 | 237.710 | 10.400 |
| **29999** | 22/7/2009 | 04:15:00 | 0.150 | 0.000 | 243.490 | 0.600 |

# Exploratory Data Analysis (EDA)

In [6]:
```
# Total number of rows and columns present in the dataset
df.shape
```

Out[6]:
```
(30000, 9)
```

In [7]:
```
# Checking if there any null or empty value present
df.isnull().sum()
```

Out[7]:
```
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           370
dtype: int64
```

### 👉 Obeservation: Only 'Sub_metering_3' column contains null values

In [8]:
```
# Describing the datset
df.describe(include='all').T
```

Out[8]:

| | count | unique | top | freq | mean | std | min | 25% | 50% |
|---|---|---|---|---|---|---|---|---|---|
| **Date** | 30000 | 1442 | 24/6/2009 | 38 | NaN | NaN | NaN | NaN | NaN |
| **Time** | 30000 | 1440 | 19:05:00 | 36 | NaN | NaN | NaN | NaN | NaN |
| **Global_active_power** | 30000 | 2831 | ? | 370 | NaN | NaN | NaN | NaN | NaN |
| **Global_reactive_power** | 30000 | 484 | 0.000 | 6772 | NaN | NaN | NaN | NaN | NaN |
| **Voltage** | 30000 | 2444 | ? | 370 | NaN | NaN | NaN | NaN | NaN |
| **Global_intensity** | 30000 | 244 | 1.000 | 2478 | NaN | NaN | NaN | NaN | NaN |
| **Sub_metering_1** | 30000 | 77 | 0.000 | 26645 | NaN | NaN | NaN | NaN | NaN |
| **Sub_metering_2** | 30000 | 80 | 0.000 | 20363 | NaN | NaN | NaN | NaN | NaN |
| **Sub_metering_3** | 29630.0 | NaN | NaN | NaN | 6.395714 | 8.417606 | 0.0 | 0.0 | 1.0 |

## 🖌 Data Cleaning

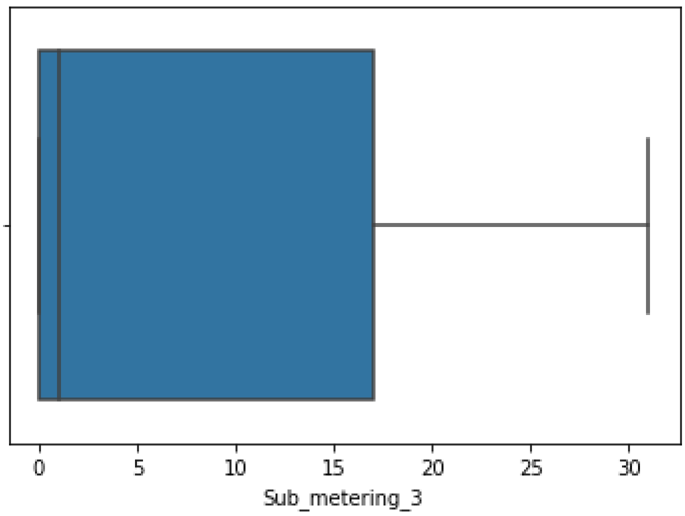
### As the 'Sub_metering_3' column contains null or empty values, we have to clean the data.

### Some process of filling null values

- Fill NaN values with mean(or median, if having outliers) or other significant value.
- Fill NaN values with Prev or after row or column value.(.ffill())
- Fill NaN with Linearly Interpolated Value with .interpolate().
- Fill NaN with Outlier or Zero.

In [9]:
```
# Checking if Sub_metering_3 having outliers or not using boxplot
sns.boxplot(df['Sub_metering_3'])
```

Out[9]: <AxesSubplot:xlabel='Sub_metering_3'>




### 👉 Observation: 'Sub_metering_3' column having no outliers

In [10]:
```
# We are filling Sub_metering_3 NaN values with mean of that column
df['Sub_metering_3'].fillna(value=df['Sub_metering_3'].mean(), inplace=True)
```

In [11]:
```
# After filling NaN value will mean, we are again checking if there any null value
df.isnull().sum()
```

Out[11]:
```
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
```

In [12]:
```
# There is some impurate value. We replace it with '0' so that it will not disturb
df.replace(['?', np.nan], '0', inplace=True)
```

### 👉 Observation: Now any columns having NaN values

## 👉 Feature Modification

In [13]:
```
# Checking datatypes according to the column
df.dtypes
```

Out[13]:
```
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
```

In [14]:
```
# Converting the Date to to_dateTime Format
df['Date'] = pd.to_datetime(df['Date'])
```

In [15]:
```
# Now some columns having numeric value but in object form, we have to first conver
targetted_df = df.iloc[:,2:8]
targetted_df
```

Out[15]:

| | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_metering_1 | Su |
|---|---|---|---|---|---|---|
| **0** | 0.196 | 0.074 | 234.170 | 0.800 | 0.000 | |
| **1** | 1.262 | 0.054 | 241.420 | 5.200 | 0.000 | |
| **2** | 0.344 | 0.076 | 242.460 | 1.600 | 0.000 | |
| **3** | 2.672 | 0.000 | 238.840 | 11.400 | 0.000 | |
| **4** | 0.254 | 0.000 | 250.220 | 1.000 | 0.000 | |
| **...** | ... | ... | ... | ... | ... | |
| **29995** | 1.746 | 0.052 | 240.800 | 7.200 | 0.000 | |
| **29996** | 0.404 | 0.134 | 243.810 | 1.800 | 0.000 | |
| **29997** | 1.794 | 0.168 | 238.480 | 7.400 | 0.000 | |
| **29998** | 2.416 | 0.586 | 237.710 | 10.400 | 0.000 | |
| **29999** | 0.150 | 0.000 | 243.490 | 0.600 | 0.000 | |

30000 rows × 6 columns

In [16]:

```
# Getting targetted columns
targetted_cols = targetted_df.columns
targetted_cols
```

Out[16]:

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

In [17]:

```
# Converting to numeric form of targetted column data
for col in targetted_cols:
    df[col] = pd.to_numeric(targetted_df[col])
```

In [18]:

```
df.dtypes
```

Out[18]:

```
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
```

In [19]:

```
# After feature modification
df.head()
```

Out[19]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_me |
|---|---|---|---|---|---|---|---|
| **0** | 2007-07-27 | 10:25:00 | 0.196 | 0.074 | 234.17 | 0.8 | |
| **1** | 2010-08-28 | 18:07:00 | 1.262 | 0.054 | 241.42 | 5.2 | |
| **2** | 2008-09-30 | 00:48:00 | 0.344 | 0.076 | 242.46 | 1.6 | |
| **3** | 2009-12-10 | 06:54:00 | 2.672 | 0.000 | 238.84 | 11.4 | |
| **4** | 2010-10-01 | 23:56:00 | 0.254 | 0.000 | 250.22 | 1.0 | |

## 👉 Getting targetted feature

In [20]:

```
df['Total_Consumption'] = df['Sub_metering_1']+df['Sub_metering_2']+df['Sub_meteri
```

In [21]:

```
df.head()
```

Out[21]:

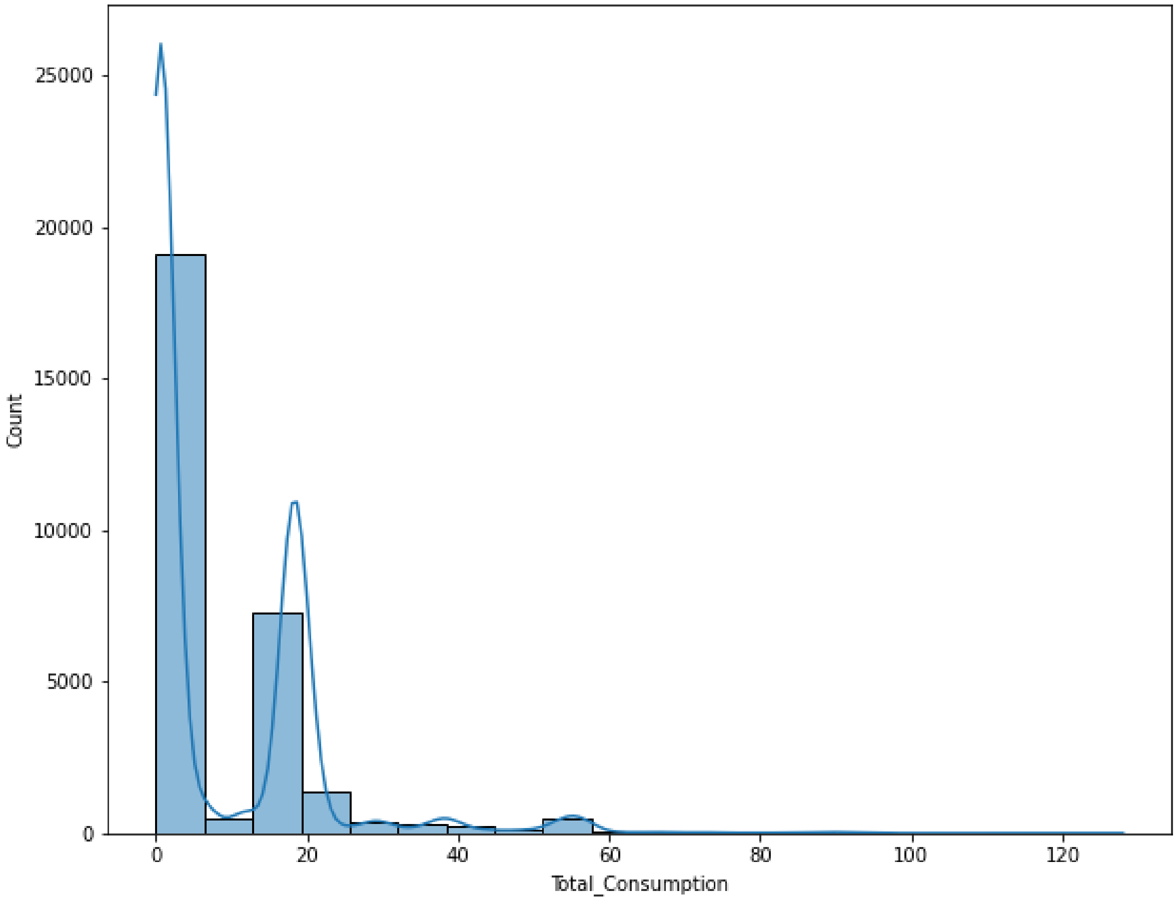
| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_me |
|---|---|---|---|---|---|---|---|
| **0** | 2007-07-27 | 10:25:00 | 0.196 | 0.074 | 234.17 | 0.8 | |
| **1** | 2010-08-28 | 18:07:00 | 1.262 | 0.054 | 241.42 | 5.2 | |
| **2** | 2008-09-30 | 00:48:00 | 0.344 | 0.076 | 242.46 | 1.6 | |
| **3** | 2009-12-10 | 06:54:00 | 2.672 | 0.000 | 238.84 | 11.4 | |
| **4** | 2010-10-01 | 23:56:00 | 0.254 | 0.000 | 250.22 | 1.0 | |

## 👉 Feature Visualization

In [22]:

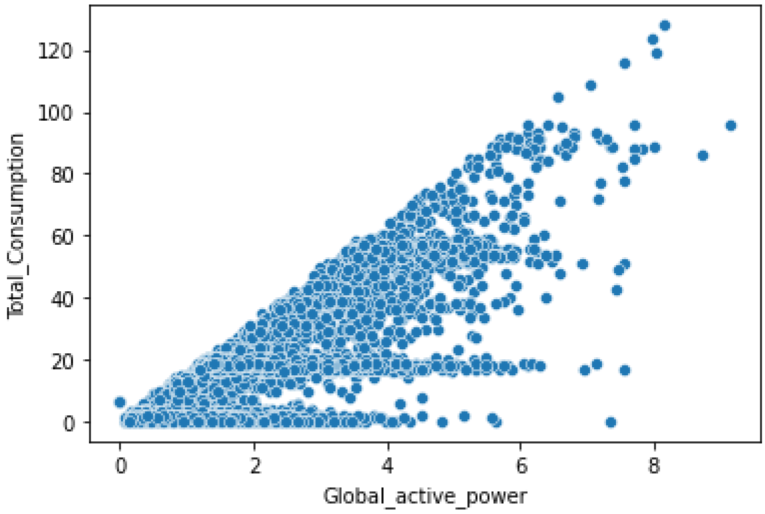
```
# Getting Total Consumption Histogram
plt.figure(figsize=(10,8))
sns.histplot(x='Total_Consumption',data= df,bins=20,kde=True)
```

Out[22]: <AxesSubplot:xlabel='Total_Consumption', ylabel='Count'>



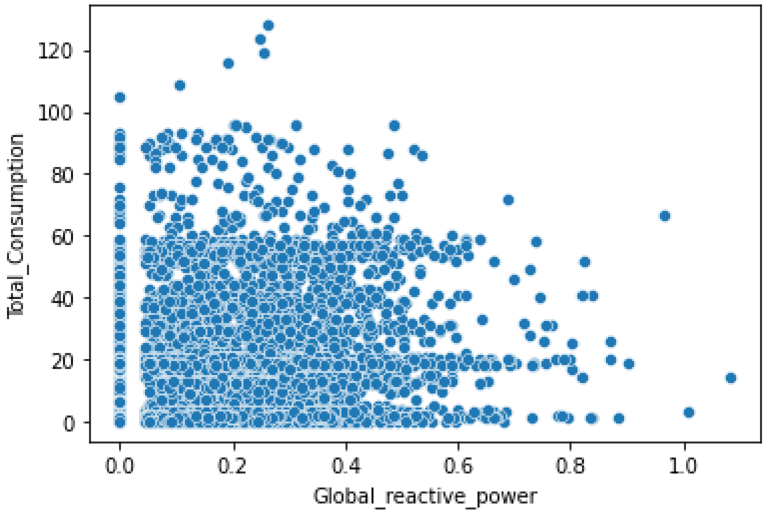

In [23]:
```
# Getting graphical relation between 'Global_active_power' and 'Total_Consumption'
sns.scatterplot(data=df, x='Global_active_power', y='Total_Consumption')
```

Out[23]: <AxesSubplot:xlabel='Global_active_power', ylabel='Total_Consumption'>



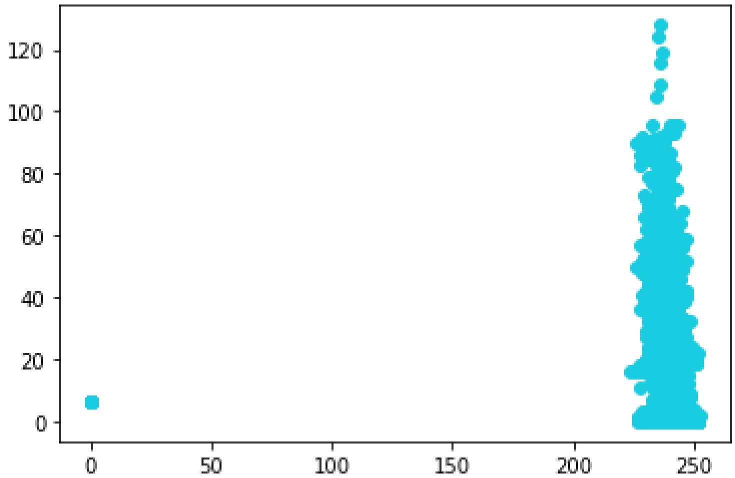

In [24]:
```
# Getting graphical relation between 'Global_reactive_power' and 'Total_Consumptio
sns.scatterplot(data=df, x='Global_reactive_power', y='Total_Consumption')
```

Out[24]: <AxesSubplot:xlabel='Global_reactive_power', ylabel='Total_Consumption'>



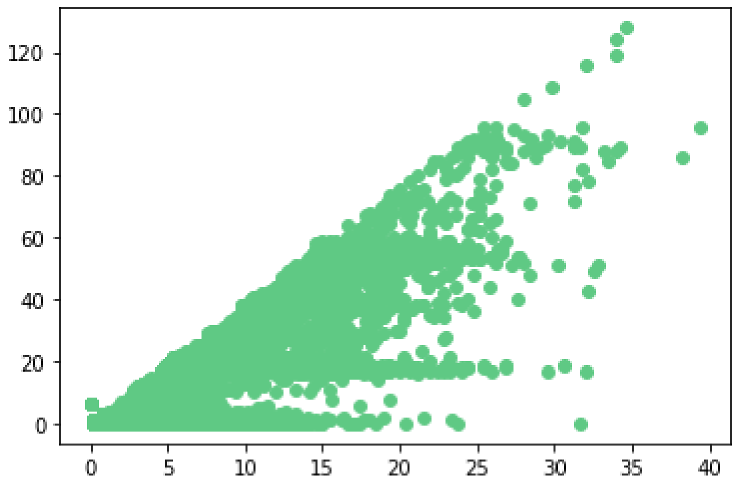

In [25]:
```python
# Getting graphical relation between 'Voltage' and 'Total_Consumption'
plt.scatter(data=df, x='Voltage', y='Total_Consumption',c='#19cce0')
```

Out[25]:
```
<matplotlib.collections.PathCollection at 0x1ce41e2f7f0>
```




In [26]:
```python
# Getting graphical relation between 'Global_intensity' and 'Total_Consumption'
plt.scatter(data=df, x='Global_intensity', y='Total_Consumption',c='#5fc984')
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x1ce41e6cfd0>
```




In [27]:
```python
# Get understandable time of the day from hour
def get_time_of_day(hour):
    if hour in range(6,12):
        return 'Morning'
```

```
    if hour in range(12,16):
        return 'After Noon'
    if hour in range(16,22):
        return 'Evening'
    if hour in range(22,25):
        return 'Night'
    return 'Late Night'
```

In [28]:
```
df['Time_of_day'] = pd.to_datetime(df['Time']).dt.hour.apply(get_time_of_day)
```

In [29]:
```
# Getting month from Date
df['month'] = df['Date'].dt.month_name()
```

In [30]:
```
# Getting year from Date
df['year'] = df['Date'].dt.year
```

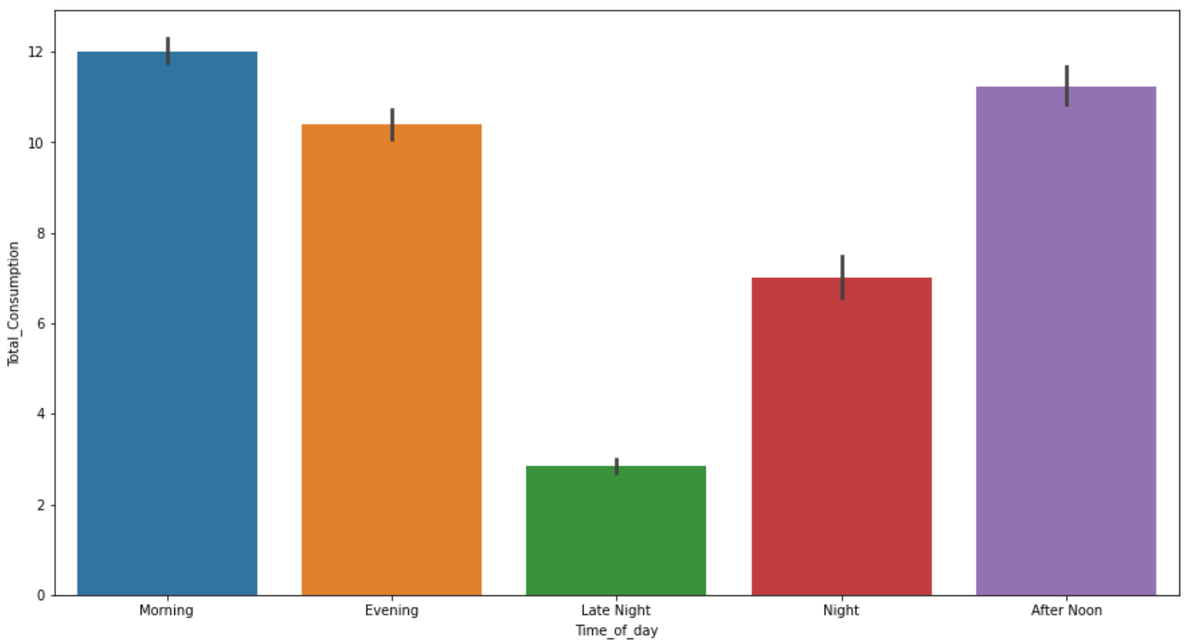
In [31]:
```
# After adding some columns data overview
df.head()
```

Out[31]:

| | Date | Time | Global_active_power | Global_reactive_power | Voltage | Global_intensity | Sub_me |
|---|---|---|---|---|---|---|---|
| **0** | 2007-07-27 | 10:25:00 | 0.196 | 0.074 | 234.17 | 0.8 | |
| **1** | 2010-08-28 | 18:07:00 | 1.262 | 0.054 | 241.42 | 5.2 | |
| **2** | 2008-09-30 | 00:48:00 | 0.344 | 0.076 | 242.46 | 1.6 | |
| **3** | 2009-12-10 | 06:54:00 | 2.672 | 0.000 | 238.84 | 11.4 | |
| **4** | 2010-10-01 | 23:56:00 | 0.254 | 0.000 | 250.22 | 1.0 | |

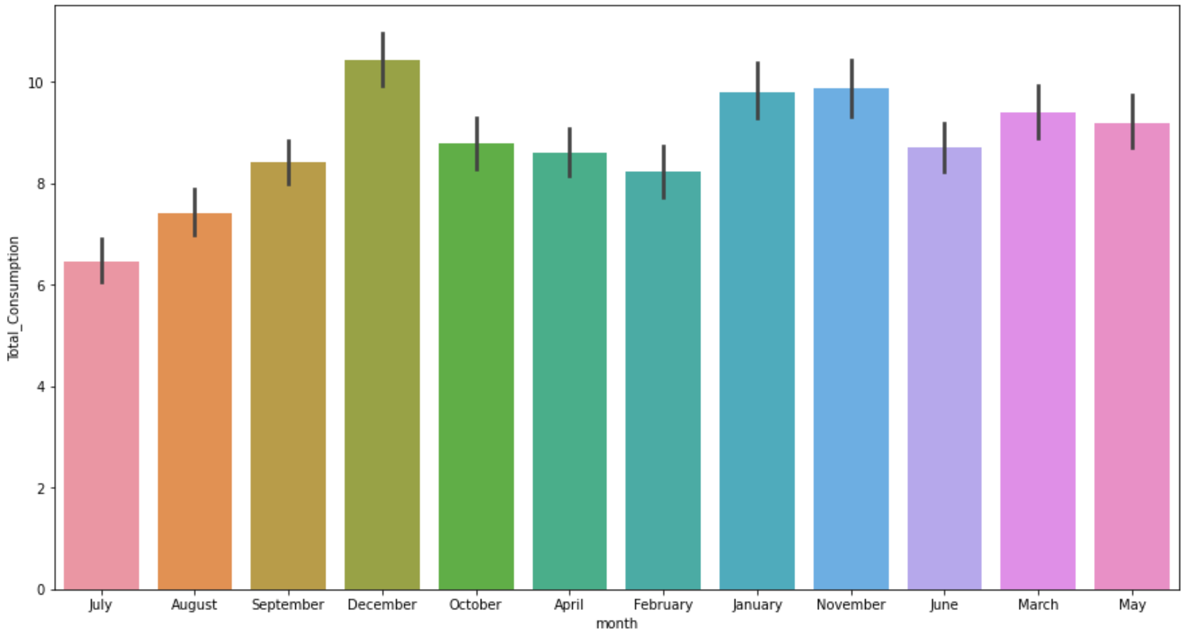
In [32]:
```
# Checking 'time of day' relation with 'total consumption'
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='Time_of_day', y='Total_Consumption')
```

Out[32]: <AxesSubplot:xlabel='Time_of_day', ylabel='Total_Consumption'>



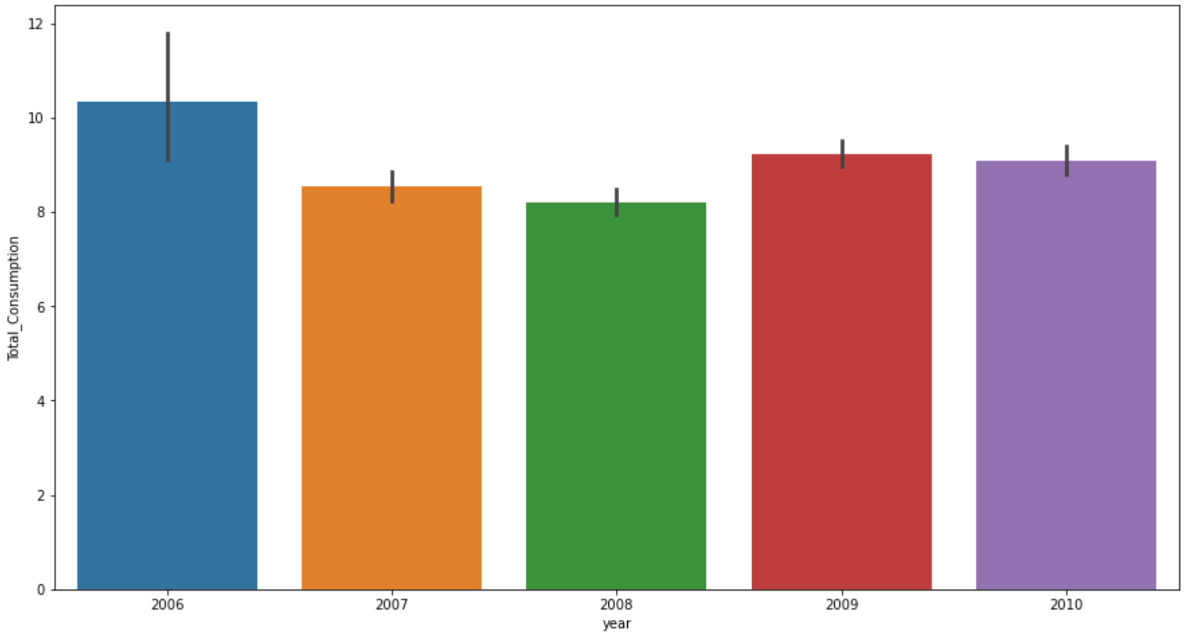
In [33]:
```
# Checking 'month' relation with 'total consumption'
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='month', y='Total_Consumption')
```

Out[33]: <AxesSubplot:xlabel='month', ylabel='Total_Consumption'>




In [34]:
```
# Checking 'year' relation with 'total consumption'
plt.figure(figsize=(15,8))
sns.barplot(data=df, x='year', y='Total_Consumption')
```

Out[34]: <AxesSubplot:xlabel='year', ylabel='Total_Consumption'>

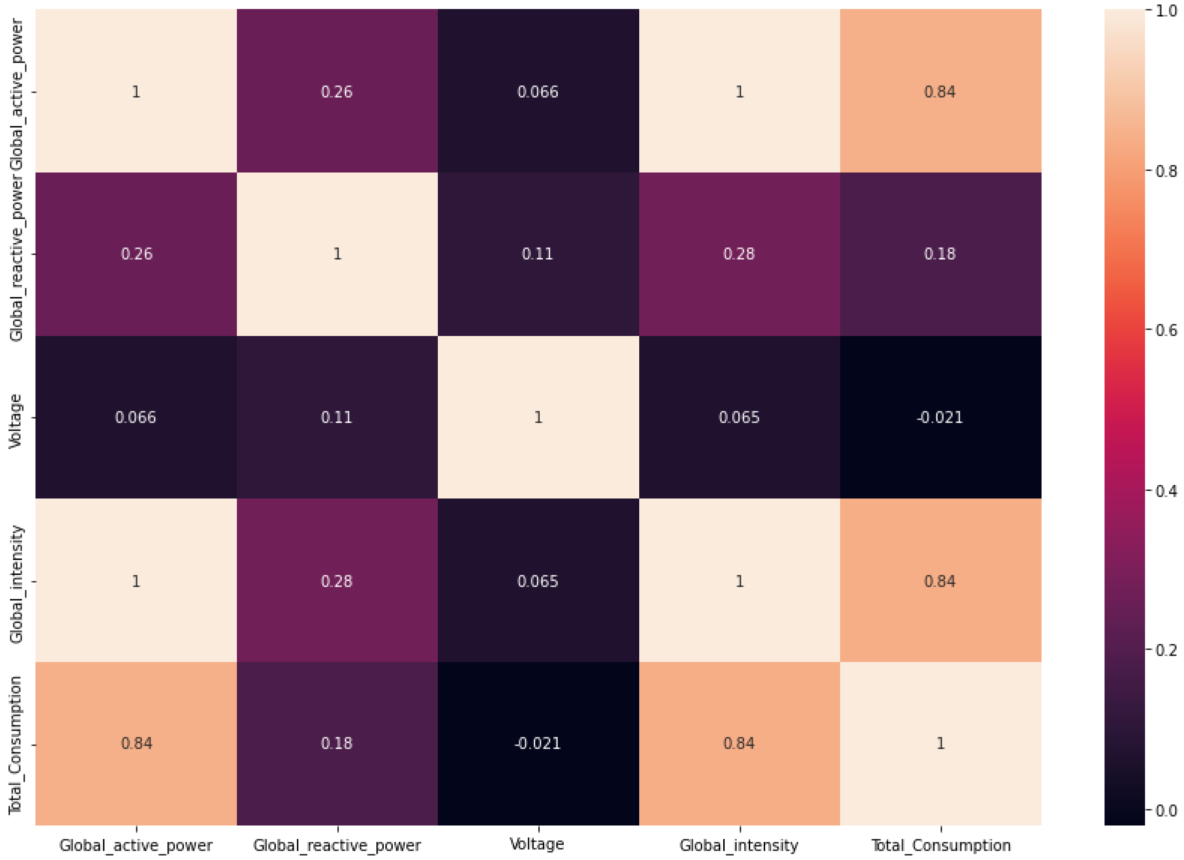



## 👉 Feature Selection

In [35]:
```
# Dropping not important features that will not use in model building
data = df.drop(['Date','Time', 'Time_of_day','month', 'year', 'Sub_metering_1', 'Su
```

In [36]:
```
# Ploting heatmap of features correlation
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(), annot=True)
```

Out[36]: <AxesSubplot:>




👉 Observation: 'Global_active_power', 'Global_intensity' column is highly correlated to each other. In that case we can drop one feature.

In [37]:
```
# For now, we are dropping 'Global_intensity' feature
data.drop(['Global_intensity'], axis=1, inplace=True)
```

In [38]:
```
# After dropping 'Global_intensity' feature
data.head()
```

Out[38]:

| | Global_active_power | Global_reactive_power | Voltage | Total_Consumption |
|---|---|---|---|---|
| **0** | 0.196 | 0.074 | 234.17 | 0.0 |
| **1** | 1.262 | 0.054 | 241.42 | 18.0 |
| **2** | 0.344 | 0.076 | 242.46 | 0.0 |
| **3** | 2.672 | 0.000 | 238.84 | 18.0 |
| **4** | 0.254 | 0.000 | 250.22 | 0.0 |

# 👉 Inserting data to mongodb

In [39]:
```
import pymongo
```

In [40]:
```
# Initializing db features
client = pymongo.MongoClient("mongodb+srv://samarpancoder2002:practice_test@practi
db = client['HouseHold_Data_Database']
data_collection = db['moderated_data']
```

In [41]:
```
# Converting the data to json format
```

```
moderated_data_json = data.to_dict('records')
```

In [42]:
```
# Inserting data into MongoDB
# data_collection.insert_many(moderated_data_json)
```

# 👉 Loading data from mongodb

In [43]:
```
# Getting all records from mongodb
imported_data = data_collection.find()
```

In [44]:
```
# Converting to dataframe
imported_data = pd.DataFrame(imported_data)
imported_data.head()
```

Out[44]:

| | _id | Global_active_power | Global_reactive_power | Voltage | Total_Consum |
|---|---|---|---|---|---|
| 0 | 636a1788b919b6f87bae9dfc | 3.736 | 0.384 | 237.09 | |
| 1 | 636a1788b919b6f87bae9dfa | 0.254 | 0.000 | 250.22 | |
| 2 | 636a1788b919b6f87bae9df7 | 1.262 | 0.054 | 241.42 | |
| 3 | 636a1788b919b6f87bae9e07 | 0.552 | 0.192 | 242.62 | |
| 4 | 636a1788b919b6f87bae9df9 | 2.672 | 0.000 | 238.84 | |

In [45]:
```
# Data coming from mongodb size checking
imported_data.shape
```

Out[45]: (40000, 5)

# 👉 Dropping not important columns from data comimg from mongodb

In [46]:
```
imported_data.drop(['_id'], axis=1, inplace=True)
imported_data.head()
```

Out[46]:

| | Global_active_power | Global_reactive_power | Voltage | Total_Consumption |
|---|---|---|---|---|
| 0 | 3.736 | 0.384 | 237.09 | 49.0 |
| 1 | 0.254 | 0.000 | 250.22 | 0.0 |
| 2 | 1.262 | 0.054 | 241.42 | 18.0 |
| 3 | 0.552 | 0.192 | 242.62 | 1.0 |
| 4 | 2.672 | 0.000 | 238.84 | 18.0 |

# 👉 Getting Independent and Dependent Features

In [47]:
```
# Getting independent features
X = imported_data.iloc[:,0:3]
X
```

Out[47]:

| | Global_active_power | Global_reactive_power | Voltage |
|---|---|---|---|
| **0** | 3.736 | 0.384 | 237.09 |
| **1** | 0.254 | 0.000 | 250.22 |
| **2** | 1.262 | 0.054 | 241.42 |
| **3** | 0.552 | 0.192 | 242.62 |
| **4** | 2.672 | 0.000 | 238.84 |
| **...** | ... | ... | ... |
| **39995** | 0.578 | 0.000 | 240.14 |
| **39996** | 0.414 | 0.344 | 248.25 |
| **39997** | 0.210 | 0.098 | 243.07 |
| **39998** | 0.364 | 0.084 | 242.34 |
| **39999** | 0.348 | 0.072 | 241.78 |

40000 rows × 3 columns

In [48]:
```
# Getting dependent features
y = imported_data.iloc[:, -1]
y
```

Out[48]:
```
0        49.0
1         0.0
2        18.0
3         1.0
4        18.0
         ...
39995     0.0
39996     2.0
39997     3.0
39998     0.0
39999     1.0
Name: Total_Consumption, Length: 40000, dtype: float64
```

# Spliting Training and Test Data

In [49]:
```
from sklearn.model_selection import train_test_split
```

In [50]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=244, test_s
```

In [51]:
```
# Independent training data size
X_train.shape
```

Out[51]:
```
(26000, 3)
```

In [52]:
```
# Dependent training data size
y_train.shape
```

Out[52]:
```
(26000,)
```

In [53]:
```
# Independent test data size
X_test.shape
```

Out[53]:
```
(14000, 3)
```

In [54]:
```
# Dependent test data size
y_test.shape
```

Out[54]:
```
(14000,)
```

# 👉 Feature Scaling

In [55]:
```
from sklearn.preprocessing import StandardScaler
```

In [56]:
```
scaler = StandardScaler()
```

In [57]:
```
# Apply scaler on training dataset
X_train = scaler.fit_transform(X_train)
```

In [58]:
```
# Apply scaler on test dataset
X_test = scaler.transform(X_test)
```

### Important: to avoid data leaking we use transform instead of fit_transform in case of test data

# 👉 Pickling the scaler and moderated data

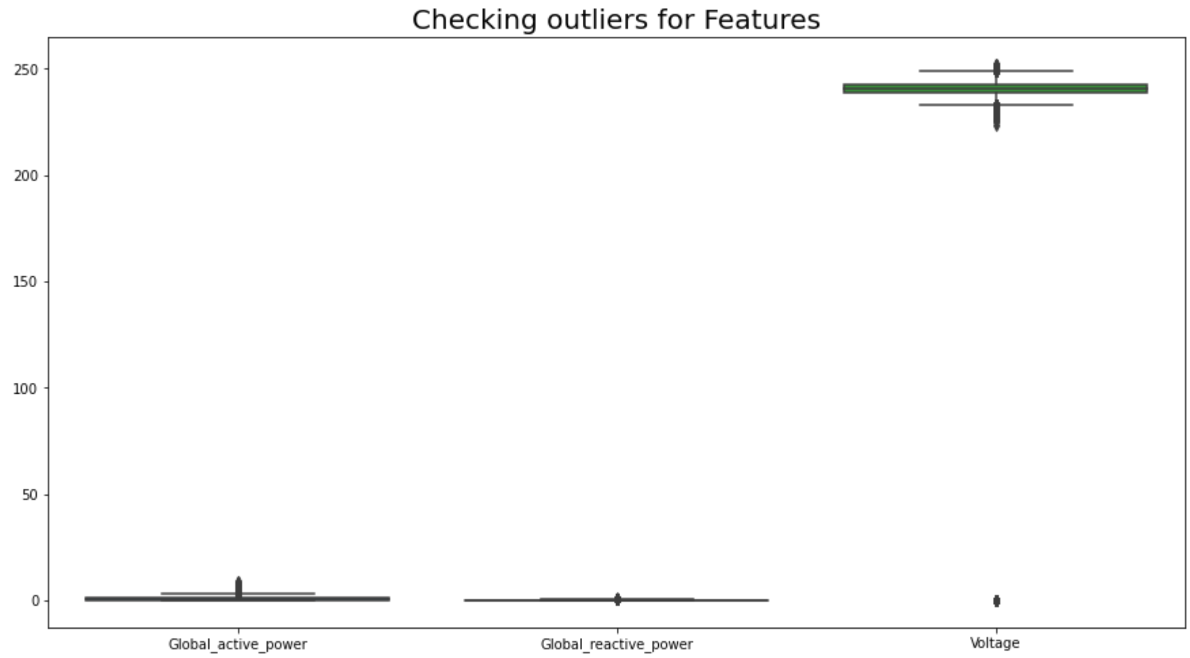
In [59]:
```
import pickle

with open('standard_scalar.sav', 'wb') as scaler_f:
    pickle.dump(scaler, scaler_f)

with open('preprocessed_data.sav', 'wb') as data_f:
    pickle.dump(imported_data, data_f)
```

# 👉 Model building features outliers check

In [60]:
```
plt.figure(figsize=(15,8))
plt.title("Checking outliers for Features",fontsize=20)
sns.boxplot(data=imported_data.iloc[:,0:3])
plt.show()
```




In [61]:
```
from sklearn.metrics import r2_score
```

In [62]:
```
model_predictions = {};
```

# Bagging Regressor

In [63]:
```
from sklearn.ensemble import BaggingRegressor
```

In [64]:
```
raw_bag_model = BaggingRegressor() # By default best estimator takes is DecisionTre
```

In [65]:
```
raw_bag_model.fit(X_train, y_train)
```

Out[65]:
```
BaggingRegressor()
```

### When training with train data, accuracy

In [66]:
```
raw_bag_model.score(X_train, y_train)
```

Out[66]:
```
0.952859458681099
```

### When training with test data

In [67]:
```
raw_bag_y_pred = raw_bag_model.predict(X_test)
```

In [68]:
```
score = r2_score(y_test, raw_bag_y_pred)
score
```

Out[68]:
```
0.7275306124943237
```

In [69]:
```
adjusted_r2_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test) - X_test.shape[1]-1
print(f'Adjusted R Square: {adjusted_r2_score}')
```

```
Adjusted R Square: 0.7274722095104342
```

### Observation:

- Without pre-pruning with proper params, gives an unstable and overfitted model. Because testing with training dataset, it's score is very high. But with prediction with test data, it's score is very low.

# Try with BaggingRegressor best_estimator as Linear Regression

In [70]:
```
from sklearn.linear_model import LinearRegression
```

In [71]:
```
raw_linear_model=BaggingRegressor(base_estimator=LinearRegression()) # By default l
```

In [72]:
```
raw_linear_model.fit(X_train,y_train)
```

Out[72]:
```
BaggingRegressor(base_estimator=LinearRegression())
```

### When training with train data, accuracy

In [73]:
```
raw_linear_model.score(X_train, y_train)
```

Out[73]:
```
0.7147176849838259
```

### When training with test data

In [74]:
```
raw_linear_y_pred = raw_linear_model.predict(X_test)
```

In [75]:
```
score = r2_score(y_test, raw_linear_y_pred)
score
```

Out[75]:
```
0.714153109275862
```

In [76]:
```
adjusted_r2_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test) - X_test.shape[1]-1
print(f'Adjusted R Square: {adjusted_r2_score}')
```

```
Adjusted R Square: 0.7140918388648752
```

In [77]:
```
model_predictions[raw_linear_model] = adjusted_r2_score
```

### Observation:

- In BaggingRegressor, with best_estimator as LinearRegression, score with training and test data giving almost same score. So we can call model is well-trained.

# Extra Trees Regressor

In [78]:
```
from sklearn.ensemble import ExtraTreesRegressor
```

In [79]:
```
raw_model = ExtraTreesRegressor()
```

In [80]:
```
raw_model.fit(X_train, y_train)
```

Out[80]:
```
ExtraTreesRegressor()
```

In [81]:
```
raw_model.score(X_train, y_train)
```

Out[81]:
```
0.999995375910015
```

### Observation: Looks like a overfitted model

# Training with test data

In [82]:
```
raw_y_pred = raw_model.predict(X_test)
```

In [83]:
```
score = r2_score(y_test, raw_y_pred)
score
```

Out[83]:
```
0.7366792876871511
```

In [84]:
```
adjusted_r2_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test) - X_test.shape[1]-1
print(f'Adjusted R Square: {adjusted_r2_score}')
```

```
Adjusted R Square: 0.7366228456939431
```

### Observation:

- Without pre-pruning with proper params, gives an unstable and overfitted model. Because testing with training dataset, it's score is very high. But with prediction with test data, it's score is very low.

# HyperParameter Tuning with HalvingGridSearchCV

In [85]:
```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
```

In [86]:
```
param_grid_collection = {
    'criterion': ["squared_error", "absolute_error"],
    'max_depth': [2,3,5,7,9,15,20,25],
    'min_samples_split': [2,6,15,20,24,35],
    'min_samples_leaf': [2,3,4,9,10,15],
}
```

In [87]:
```
tuned_model = HalvingGridSearchCV(estimator=ExtraTreesRegressor(), param_grid=parar
```

In [88]:
```
tuned_model.fit(X_train, y_train)
```

```
n_iterations: 6
n_required_iterations: 6
n_possible_iterations: 6
min_resources_: 106
max_resources_: 26000
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 576
n_resources: 106
Fitting 5 folds for each of 576 candidates, totalling 2880 fits
----------
iter: 1
n_candidates: 192
n_resources: 318
Fitting 5 folds for each of 192 candidates, totalling 960 fits
----------
iter: 2
n_candidates: 64
n_resources: 954
Fitting 5 folds for each of 64 candidates, totalling 320 fits
----------
iter: 3
n_candidates: 22
n_resources: 2862
Fitting 5 folds for each of 22 candidates, totalling 110 fits
----------
iter: 4
n_candidates: 8
n_resources: 8586
Fitting 5 folds for each of 8 candidates, totalling 40 fits
----------
iter: 5
n_candidates: 3
n_resources: 25758
Fitting 5 folds for each of 3 candidates, totalling 15 fits
```

Out[88]:

```
HalvingGridSearchCV(estimator=ExtraTreesRegressor(),
                    param_grid={'criterion': ['squared_error',
                                              'absolute_error'],
                                'max_depth': [2, 3, 5, 7, 9, 15, 20, 25],
                                'min_samples_leaf': [2, 3, 4, 9, 10, 15],
                                'min_samples_split': [2, 6, 15, 20, 24, 35]},
                    verbose=1)
```

In [89]:

```
tuned_model.best_params_
```

Out[89]:

```
{'criterion': 'squared_error',
 'max_depth': 20,
 'min_samples_leaf': 2,
 'min_samples_split': 35}
```

In [90]:

```
# # Best pre-pruned params for getting better model
# (criterion= 'squared_error',
#  max_depth= 20,
#  min_samples_leaf= 2,
#  min_samples_split= 35)
```

In [91]:

```
best_fit_model = ExtraTreesRegressor(criterion= 'squared_error',
 max_depth= 20,
 min_samples_leaf= 2,
 min_samples_split= 35)
```

In [92]:
```
best_fit_model.fit(X_train,y_train)
```

Out[92]:
```
ExtraTreesRegressor(max_depth=20, min_samples_leaf=2, min_samples_split=35)
```

## Score while testing with training dataset

In [93]:
```
best_fit_model.score(X_train,y_train) # It gives r2_score
```

Out[93]:
```
0.807660405985378
```

In [94]:
```
best_fit_y_pred = best_fit_model.predict(X_test)
```

## Score while testing with test dataset

In [95]:
```
score = r2_score(y_test, best_fit_y_pred)
score
```

Out[95]:
```
0.7654419576533412
```

In [96]:
```
adjusted_r2_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test) - X_test.shape[1]-1
print(f'Adjusted R Square: {adjusted_r2_score}')
```

```
Adjusted R Square: 0.7653916808508948
```

In [97]:
```
model_predictions[best_fit_model] = adjusted_r2_score
```

### Observation:

- After using Hyperparameter Tuning we get some best_params. With that pre-pruning, we can improve model performance.
- Also with HyperParameter Tuning, we increased accuracy with test dataset from 73% to 76%

# Voting Regressor

In [98]:
```
from sklearn.ensemble import VotingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
```

In [99]:
```
ensemble = [
    ('dtr', DecisionTreeRegressor(criterion= 'squared_error',
 max_depth= 3,
 min_samples_leaf= 6,
 min_samples_split= 3,
 splitter= 'best')),
    ('etr', ExtraTreesRegressor(criterion= 'squared_error',
 max_depth= 25,
 min_samples_leaf= 2,
 min_samples_split= 35)),
     ('lnr', LinearRegression()),
     ('svr', SVR())
]
```

In [100…
```
voting_model = VotingRegressor(estimators=ensemble, n_jobs=-1,verbose=1)
```

In [101…
```
voting_model.fit(X_train, y_train)
```

Out[101]:
```
VotingRegressor(estimators=[('dtr',
                             DecisionTreeRegressor(max_depth=3,
                                                   min_samples_leaf=6,
                                                   min_samples_split=3)),
                            ('etr',
                             ExtraTreesRegressor(max_depth=25,
                                                 min_samples_leaf=2,
                                                 min_samples_split=35)),
                            ('lnr', LinearRegression()), ('svr', SVR())],
                n_jobs=-1, verbose=1)
```

## Score with training data

In [102…
```
voting_model.score(X_train, y_train)
```

Out[102]:
```
0.7677690496952341
```

## Score with test data

In [103…
```
voting_y_pred=voting_model.predict(X_test)
```

In [104…
```
score = r2_score(y_test, voting_y_pred)
score
```

Out[104]:
```
0.7509523501344152
```

In [105…
```
adjusted_r2_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test) - X_test.shape[1]-1
print(f'Adjusted R Square: {adjusted_r2_score}')
```

```
Adjusted R Square: 0.7508989675286994
```

In [106…
```
model_predictions[voting_model] = adjusted_r2_score
```

### Observation:

- As the score between training and test data is almost near.
- Compare to others, score with test data is almost similar with other successful predictions. We can say that model is well-trained.

# Random Forest Regressor

In [107…
```
from sklearn.ensemble import RandomForestRegressor
```

In [108…
```
raw_model = RandomForestRegressor()
```

In [109…
```
raw_model.fit(X_train, y_train)
```

Out[109]:
```
RandomForestRegressor()
```

## Score with training data

In [110…
```
raw_model.score(X_train, y_train)
```

Out[110]: 0.9641325726445291

# Score with test data

In [111…
```
y_raw_pred=raw_model.predict(X_test)
```

In [112…
```
score = r2_score(y_test, y_raw_pred)
score
```

Out[112]: 0.7491109421384107

In [113…
```
adjusted_r2_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test) - X_test.shape[1]-1
print(f'Adjusted R Square: {adjusted_r2_score}')
```

```
Adjusted R Square: 0.7490571648324957
```

### Observation:

- Without pre-pruning with proper params, gives an unstable and overfitted model. Because testing with training dataset, it's score is very high. But with prediction with test data, it's score is very low.

# HyperParameter Tuning for get best params with HalvingGridSearchCV

In [114…
```
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
```

In [115…
```
param_grid_collection = {
    'criterion': ["squared_error", "absolute_error"],
    'max_depth': [2,3,5,7,9,15,20,25],
    'min_samples_split': [2,6,15,20,24,35],
    'min_samples_leaf': [2,3,4,9,10,15],
}
```

In [116…
```
tuned_model = HalvingGridSearchCV(estimator=RandomForestRegressor(), param_grid=pa
```

In [117…
```
tuned_model.fit(X_train, y_train)
```

```
n_iterations: 6
n_required_iterations: 6
n_possible_iterations: 6
min_resources_: 106
max_resources_: 26000
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 576
n_resources: 106
Fitting 3 folds for each of 576 candidates, totalling 1728 fits
----------
iter: 1
n_candidates: 192
n_resources: 318
Fitting 3 folds for each of 192 candidates, totalling 576 fits
----------
iter: 2
n_candidates: 64
n_resources: 954
Fitting 3 folds for each of 64 candidates, totalling 192 fits
----------
iter: 3
n_candidates: 22
n_resources: 2862
Fitting 3 folds for each of 22 candidates, totalling 66 fits
----------
iter: 4
n_candidates: 8
n_resources: 8586
Fitting 3 folds for each of 8 candidates, totalling 24 fits
----------
iter: 5
n_candidates: 3
n_resources: 25758
Fitting 3 folds for each of 3 candidates, totalling 9 fits
```

Out[117]:

```
HalvingGridSearchCV(cv=3, estimator=RandomForestRegressor(),
                    param_grid={'criterion': ['squared_error',
                                              'absolute_error'],
                                'max_depth': [2, 3, 5, 7, 9, 15, 20, 25],
                                'min_samples_leaf': [2, 3, 4, 9, 10, 15],
                                'min_samples_split': [2, 6, 15, 20, 24, 35]},
                    verbose=1)
```

In [118…

```
tuned_model.best_params_
```

Out[118]:

```
{'criterion': 'squared_error',
 'max_depth': 9,
 'min_samples_leaf': 2,
 'min_samples_split': 24}
```

In [120…

```
# # Best params for RandomForestRegressor
# (criterion= 'squared_error',
#  max_depth= 9,
#  min_samples_leaf= 2,
#  min_samples_split= 24)
```

In [119…

```
rf_best_model = RandomForestRegressor(criterion= 'squared_error',
 max_depth= 9,
 min_samples_leaf= 2,
 min_samples_split= 24)
```

In [121…
```
rf_best_model.fit(X_train, y_train)
```
Out[121]:
```
RandomForestRegressor(max_depth=9, min_samples_leaf=2, min_samples_split=24)
```

## Score with training data

In [122…
```
rf_best_model.score(X_train, y_train)
```
Out[122]:
```
0.800240144313553
```

## Score with test data

In [123…
```
y_rf_pred = rf_best_model.predict(X_test)
```
In [124…
```
score = r2_score(y_test, y_rf_pred)
score
```
Out[124]:
```
0.7624438341974867
```
In [125…
```
adjusted_r2_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test) - X_test.shape[1]-1
print(f'Adjusted R Square: {adjusted_r2_score}')
```
```
Adjusted R Square: 0.762392914756403
```
In [126…
```
model_predictions[rf_best_model] = adjusted_r2_score
```

### Observation:

- with HyperParameter Tuning, model accuracy increase from 74.67% to 75.62%
- Tuned model giving nearest score with train and test data. So model is not overfitted or underfitted.

In [127…
```
# Trained models
trained_models_collection = list(model_predictions.keys())
```
In [128…
```
# Trained models adjusted r2_score
trained_models_score = list(model_predictions.values())
```
In [129…
```
# Converting trained model score into List collection
ready_for_df = []

for model in trained_models_collection:
    ready_for_df.append({
        'trained_model': model,
        'score': model_predictions[model]
    })

ready_for_df
```

Out[129]:
```
[{'trained_model': BaggingRegressor(base_estimator=LinearRegression()),
  'score': 0.7140918388648752},
 {'trained_model': ExtraTreesRegressor(max_depth=20, min_samples_leaf=2, min_sampl
es_split=35),
  'score': 0.7653916808508948},
 {'trained_model': VotingRegressor(estimators=[('dtr',
                                DecisionTreeRegressor(max_depth=3,
                                                      min_samples_leaf=6,
                                                      min_samples_split=3)),
                               ('etr',
                                ExtraTreesRegressor(max_depth=25,
                                                    min_samples_leaf=2,
                                                    min_samples_split=35)),
                               ('lnr', LinearRegression()), ('svr', SVR())],
                   n_jobs=-1, verbose=1),
  'score': 0.7508989675286994},
 {'trained_model': RandomForestRegressor(max_depth=9, min_samples_leaf=2, min_samp
les_split=24),
  'score': 0.762392914756403}]
```

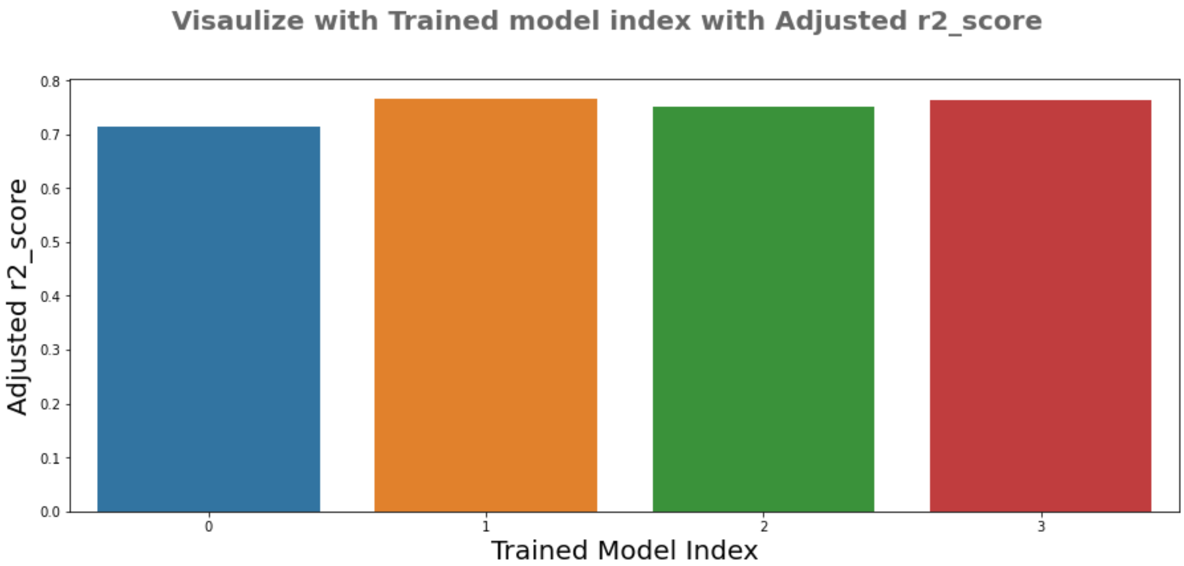
In [130…
```python
# Converting trained model score to DataFrame
model_df = pd.DataFrame(ready_for_df)
model_df.reset_index(inplace=True)
model_df
```

Out[130]:

| | index | trained_model | score |
|---|---|---|---|
| **0** | 0 | (LinearRegression(), LinearRegression(), Linea... | 0.714092 |
| **1** | 1 | (ExtraTreeRegressor(max_depth=20, min_samples_... | 0.765392 |
| **2** | 2 | VotingRegressor(estimators=[('dtr',\n ... | 0.750899 |
| **3** | 3 | (DecisionTreeRegressor(max_depth=9, max_featur... | 0.762393 |

In [131…
```python
# Visaulize with Trained model index with Adjusted r2_score

plt.figure(figsize=(15,6))
plt.suptitle('Visaulize with Trained model index with Adjusted r2_score', fontsize
sns.barplot(data=model_df, x='index', y='score')
plt.xlabel('Trained Model Index', fontdict={'fontsize': 20})
plt.ylabel('Adjusted r2_score', fontdict={'fontsize': 20})
plt.show()
```




## Observation:

- Between all trained model, ExtraTreesRegressor with pre-pruned params got by HyperParameter Tuning gives highest adjusted r2_score.
- So, we can called this is best trained model between all the trained models.

# Storing best model to use in future

In [132…

```
import pickle

with open('best_model_household_reg.sav', 'wb') as best_model_f:
    pickle.dump(model_df['trained_model'][1], best_model_f)
```

In [ ]: