Power Consumption Analysis For House Holds Using ML

INTRODUCTION

Overview:

Electricity sector in India. India is the world's third largest producer and third largest consumer of electricity. The gross electricity consumption in 2018-19 was 1,181 kWh per capita. Energy use can be viewed as a function of total GDP, structure of the economy and technology. The increase in household energy consumption is more significant than that in the industrial sector. To achieve reduction in electricity consumption, it is vital to have current information about household electricity use. This Guided Project mainly focuses on applying a machine-learning algorithm to calculate the power consumed by all appliances.

Purpose:

This will help you track the power consumed on regular intervals for all kinds of appliances which use heavy loads such as Air Conditioners, Oven or a washing machine etc.

LITERATURE SURVEY

To calculate the power consumed by all appliances. To solve this problem, we use linear regression machine learning algorithm.

THEORITICAL ANALYSIS

Hardware / Software designing

To develop this project, we need to install the following software/packages:

1. Anaconda Navigator :

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with great tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, RStudio, Visual Studio Code.

For this project, we will be using **Jupyter** notebook and **Spyder**

2. To build Machine learning models you must require the following packages

Sklearn: Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.

NumPy: NumPy is a Python package that stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object

Pandas: pandas is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.

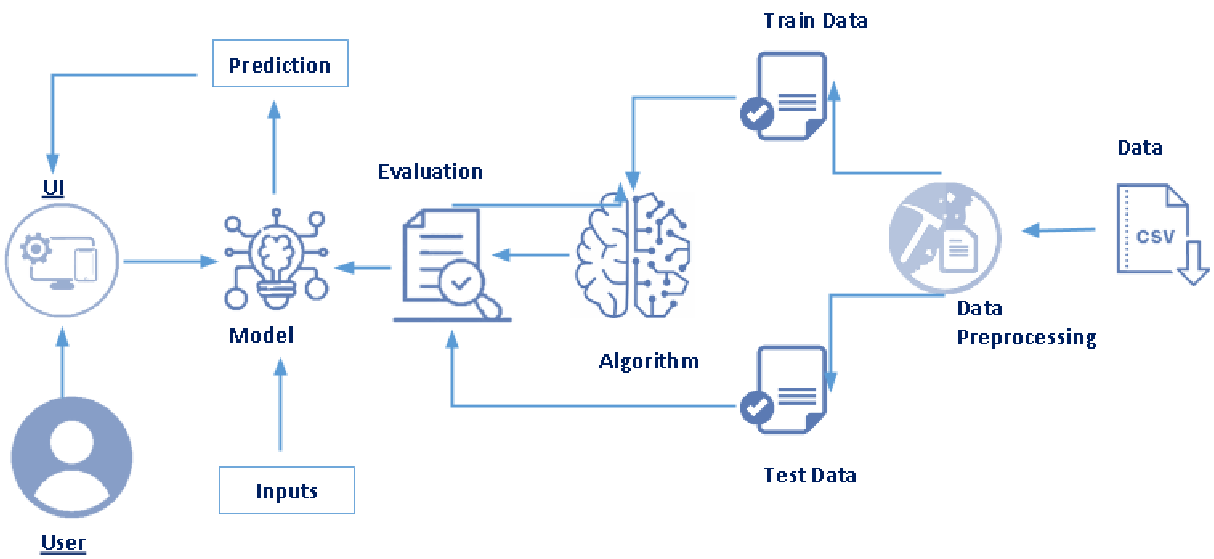
Matplotlib: It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits

Flask: Web framework used for building Web applications.

EXPERIMENTAL INVESTIGATIONS

The dataset which contains a set of features through which power consumption can be calculated, is to be collected. You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository etc.

FLOWCHART

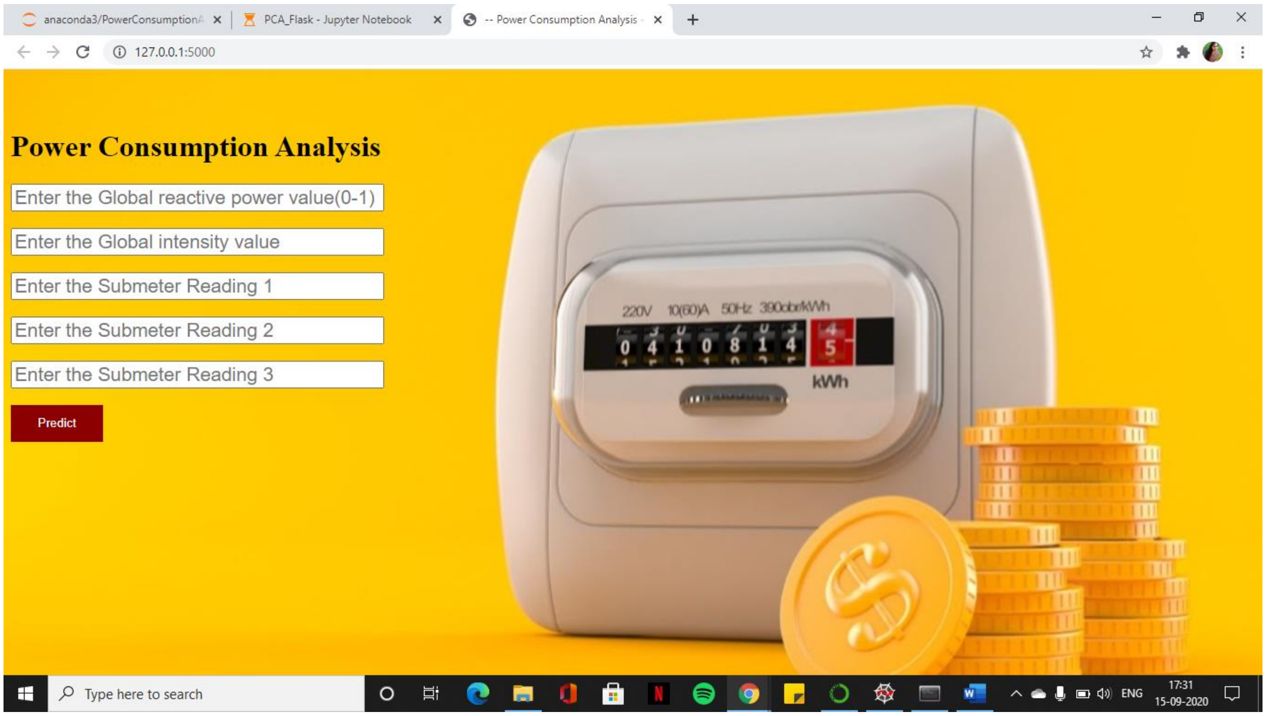


RESULT

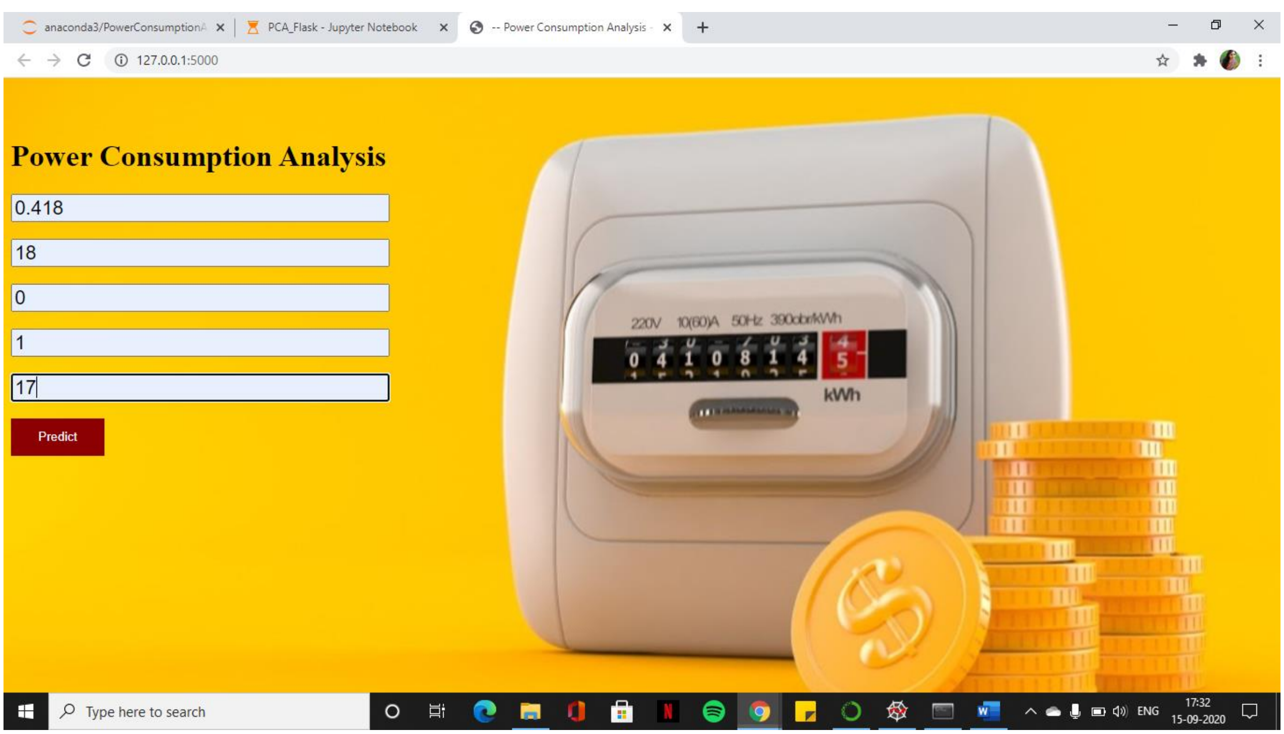
Execute the python code and after the module is running, open index.html page and scroll down to find the buttons to test with.

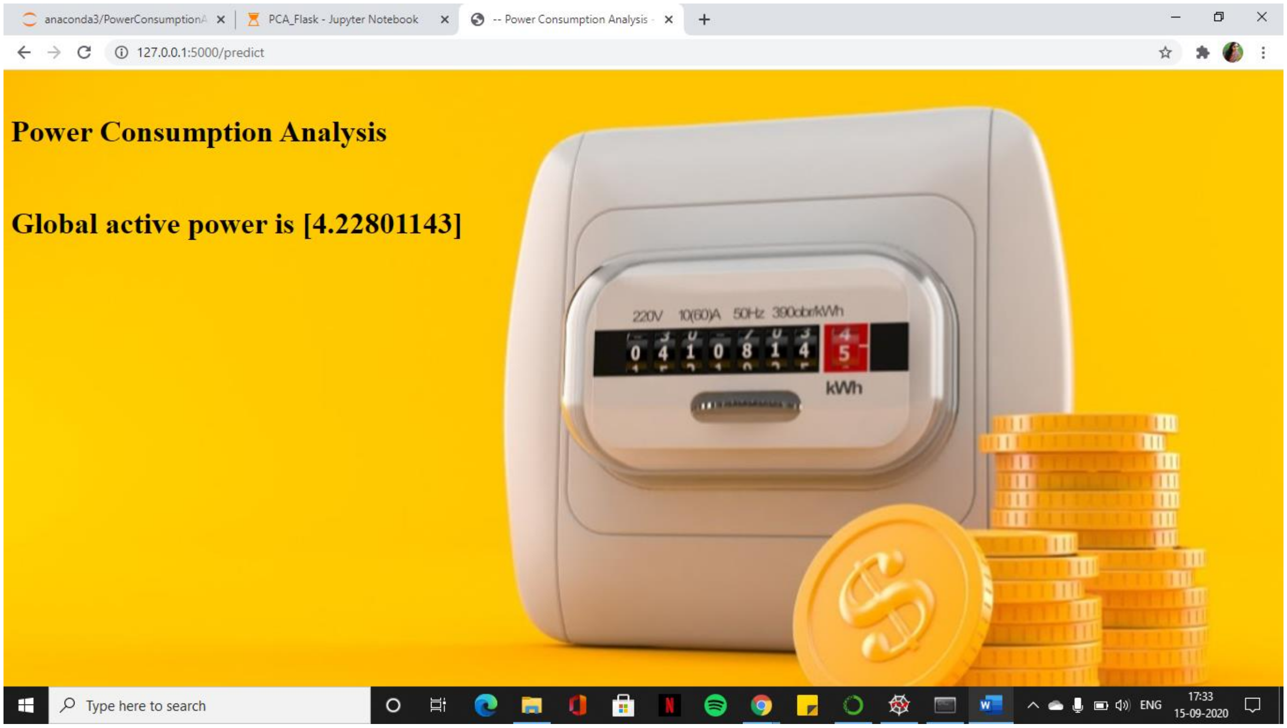
- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type "python app.py" command.
- It will show the local host where your app is running on http://127.0.0.1.5000/
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

Let’s see how our output page looks like:



Enter the values and click on Predict button to view the result on “result1.html”.





Finally, total power consumption by all the appliances is calculated and displayed.

APPLICATIONS

Household-Power-Consumption-Analysis Project was done to understand the advantages of big data applications. Analyzed the amount of energy consumed in a household which is given to us as a timeseries, and our objective is to derive patterns from the obtained real time data. Imported data into Databricks Azure.

Conclusion:

By the end of the project, I have understood that

- I have understood the problem to classify if it is a regression or a classification kind of problem.
- I can know how to pre-process/clean the data using different data pre-processing techniques.
- Applying different algorithms according to the dataset
- I can know how to find the accuracy of the model.

- I can build web applications using the Flask framework

BIBILOGRAPHY:

References:

1.The Hundred Pages Machine Learning Book(Author – Andriy Burkov)

2. SMART INTERNZ

Source Code:

Importing necessary libraries

```
In [1]:
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing dataset

```
In [2]:
dataset = pd.read_csv("C:/Users/rincy/OneDrive/Desktop/PowerConsumptionAnalysis
                      /household_power_consumption.txt"
                      , sep=';', header=0, infer_datetime_format=True,
                      parse_dates={'datetime':[0,1]}, index_col=['datetime'])
```

```
C:\Users\rincy\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns (2,3,4,5,6,7) have mixed types.Spe
cify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Understand the dataset

```
In [3]:
dataset.head()
```

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
datetime							
2006-12-16 17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	17.0
2006-12-16 17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	16.0
2006-12-16 17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	17.0
2006-12-16 17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	17.0
2006-12-16 17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	17.0

In [4]:
```
dataset.tail()
```

datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
2010-11-26 20:58:00	0.946	0	240.43	4	0	0	0.0
2010-11-26 20:59:00	0.944	0	240	4	0	0	0.0
2010-11-26 21:00:00	0.938	0	239.82	3.8	0	0	0.0
2010-11-26 21:01:00	0.934	0	239.7	3.8	0	0	0.0
2010-11-26 21:02:00	0.932	0	239.55	3.8	0	0	0.0

In [5]:
```
print(f"The Dataset has {dataset.shape[0]} rows and {dataset.shape[1]} columns")
```

```
The Dataset has 2075259 rows and 7 columns
```

In [6]:
```
dataset.columns
```

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

Checking total null values in each column

In [8]:
```
dataset.isnull().sum()
```

```
Global_active_power          0
Global_reactive_power        0
Voltage                      0
Global_intensity             0
Sub_metering_1               0
Sub_metering_2               0
Sub_metering_3           25979
dtype: int64
```

Understanding percent of data missing

In [9]:
```
percent_missing = dataset.isnull().sum() * 100 / len(dataset)
missing_value_df = pd.DataFrame({'percent_missing': percent_missing})
```

In [10]:
```
missing_value_df
```

	percent_missing
Global_active_power	0.000000
Global_reactive_power	0.000000
Voltage	0.000000
Global_intensity	0.000000
Sub_metering_1	0.000000
Sub_metering_2	0.000000
Sub_metering_3	1.251844

Handling missing values

In [13]:
```
dataset.loc[dataset.Sub_metering_3.isnull()].head()
```

datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
2006-12-21 11:23:00	?	?	?	?	?	?	NaN
2006-12-21 11:24:00	?	?	?	?	?	?	NaN
2006-12-30 10:08:00	?	?	?	?	?	?	NaN
2006-12-30 10:09:00	?	?	?	?	?	?	NaN
2007-01-14 18:36:00	?	?	?	?	?	?	NaN

In [14]:
```
dataset.replace('?', np.nan, inplace=True)
```

In [15]:

```
dataset.loc[dataset.Sub_metering_3.isnull()].head()
```

datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
2006-12-21 11:23:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2006-12-21 11:24:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2006-12-30 10:08:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2006-12-30 10:09:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2007-01-14 18:36:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [16]:

```
dataset = dataset.dropna(how = 'all')
```

In [17]:

```
for i in dataset.columns:
    dataset[i] = dataset[i].astype('float64')
#dataset = dataset.astype('float32')
```

In [18]:

```
dataset.shape
```

```
(2049280, 7)
```

Adding another sub_metering_4 column

In [19]:

```
values = dataset.values
dataset['sub_metering_4'] = (values[:,0] * 1000 / 60) - (values[:,4] + values[:,5] + values[:,6])
```

In [20]:

```
dataset.dtypes
```

```
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
sub_metering_4           float64
dtype: object
```

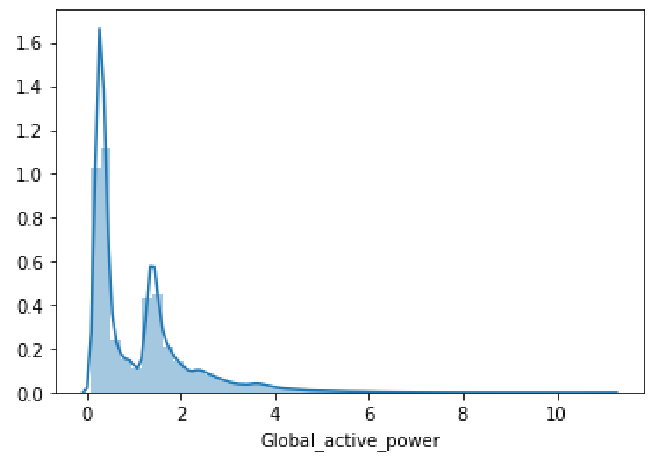
In [21]:
```python
dataset.describe()
```

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_mete
count	2.049280e+06	2.049280e+06	2.049280e+06	2.049280e+06	2.049280e+06	2.049280e+06	2.049280e-
mean	1.091615e+00	1.237145e-01	2.408399e+02	4.627759e+00	1.121923e+00	1.298520e+00	6.458447e-
std	1.057294e+00	1.127220e-01	3.239987e+00	4.444396e+00	6.153031e+00	5.822026e+00	8.437154e-
min	7.600000e-02	0.000000e+00	2.232000e+02	2.000000e-01	0.000000e+00	0.000000e+00	0.000000e-
25%	3.080000e-01	4.800000e-02	2.389900e+02	1.400000e+00	0.000000e+00	0.000000e+00	0.000000e-
50%	6.020000e-01	1.000000e-01	2.410100e+02	2.600000e+00	0.000000e+00	0.000000e+00	1.000000e-
75%	1.528000e+00	1.940000e-01	2.428900e+02	6.400000e+00	0.000000e+00	1.000000e+00	1.700000e-
max	1.112200e+01	1.390000e+00	2.541500e+02	4.840000e+01	8.800000e+01	8.000000e+01	3.100000e-

Data Visualization

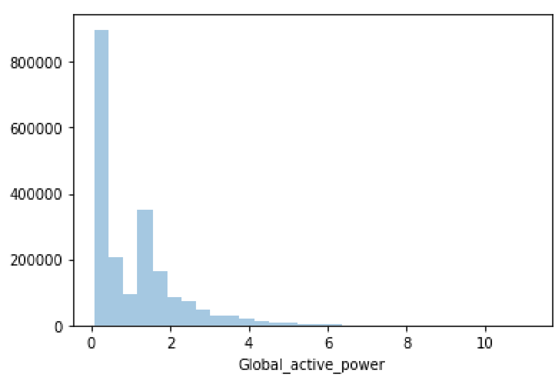
In [22]:
```python
sns.distplot(dataset['Global_active_power'])
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x236ddd3cd88>

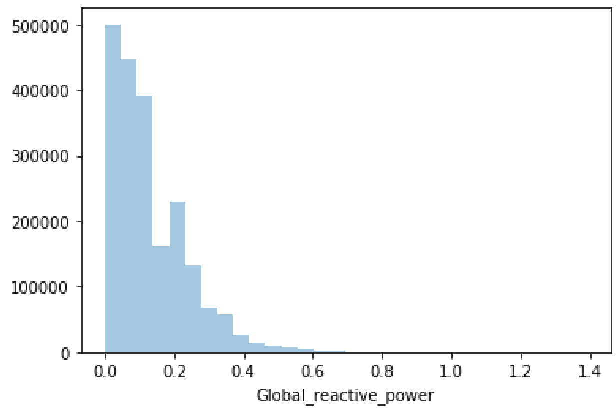


In [23]:
```python
sns.distplot(dataset['Global_active_power'],kde=False,bins=30)
```

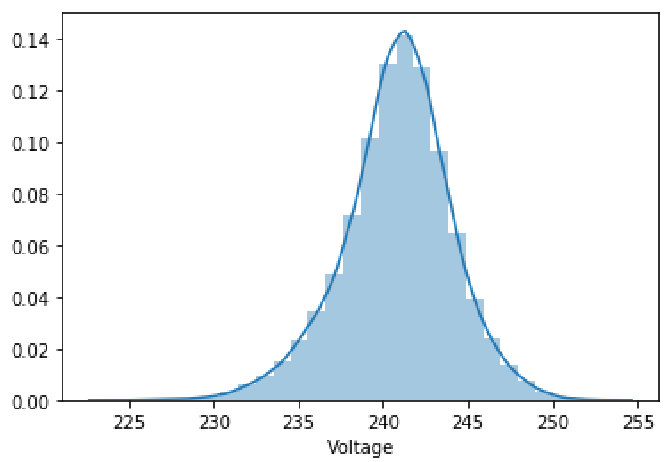
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x236dd9e2448>



```
In [24]: sns.distplot(dataset['Global_reactive_power'],kde=False,bins=30)
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x236df31fd48>
```

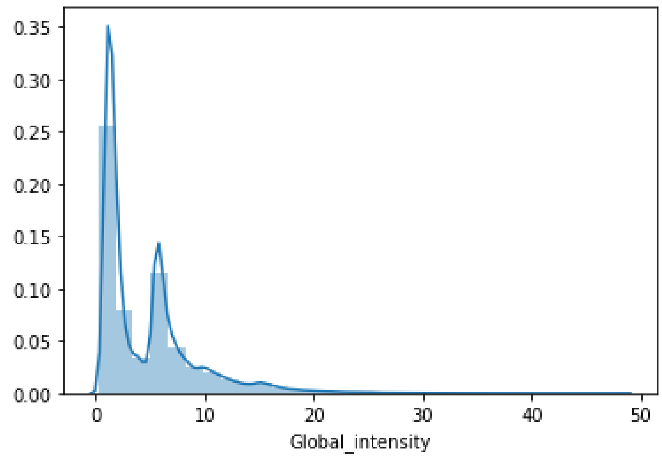



```
In [25]: sns.distplot(dataset['Voltage'],kde=True,bins=30)
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x236df307688>
```



In [26]:
```
sns.distplot(dataset['Global_intensity'],kde=True,bins=30)
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x2368791aa88>



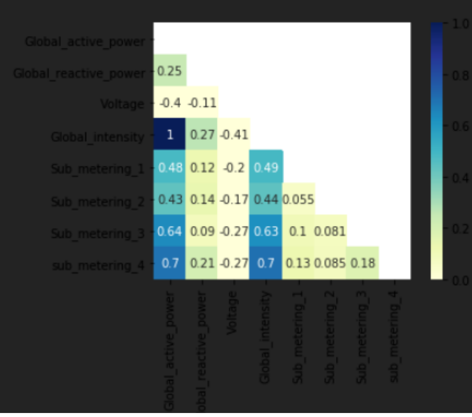
Correlation of dataset values

In [27]:
```
dataset.corr()
```

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2
Global_active_power	1.000000	0.247017	-0.399762	0.998889	0.484401	0.434569
Global_reactive_power	0.247017	1.000000	-0.112246	0.266120	0.123111	0.139231
Voltage	-0.399762	-0.112246	1.000000	-0.411363	-0.195976	-0.167405
Global_intensity	0.998889	0.266120	-0.411363	1.000000	0.489298	0.440347
Sub_metering_1	0.484401	0.123111	-0.195976	0.489298	1.000000	0.054721
Sub_metering_2	0.434569	0.139231	-0.167405	0.440347	0.054721	1.000000
Sub_metering_3	0.638555	0.089617	-0.268172	0.626543	0.102571	0.080872
sub_metering_4	0.701380	0.211624	-0.271371	0.703258	0.125067	0.085201

Analysis using heatmap

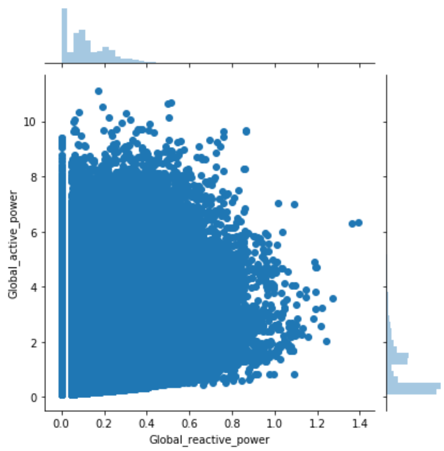
[28]:
```
pearson = dataset.corr(method='pearson')
mask = np.zeros_like(pearson)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(pearson, vmax=1, vmin=0, square=True, cbar=True, annot=True, cmap="YlGnBu", mask=mask);
```



Data Visualization

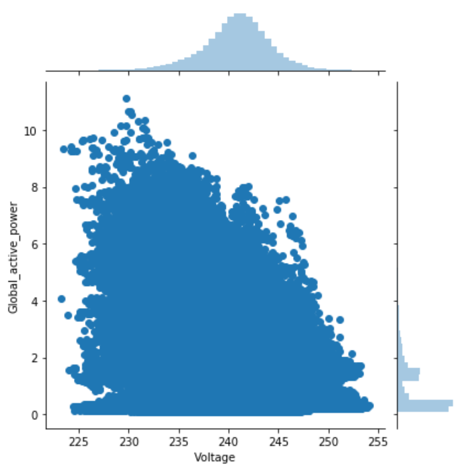
In [29]:
```
sns.jointplot( x = 'Global_reactive_power' , y = 'Global_active_power' , data = dataset , kind = 'scatter')
```

Out[29]: <seaborn.axisgrid.JointGrid at 0x23687af36c8>



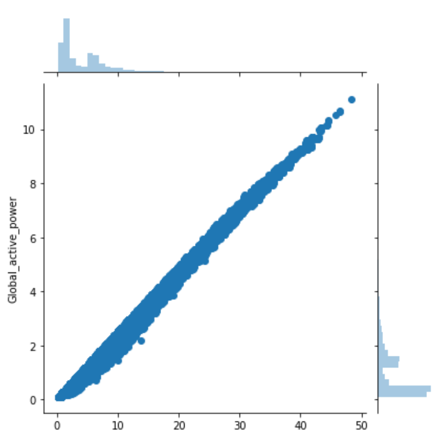
In [30]:
```
sns.jointplot( x = 'Voltage' , y = 'Global_active_power' , data = dataset , kind = 'scatter')
```

Out[30]: <seaborn.axisgrid.JointGrid at 0x23687a75488>



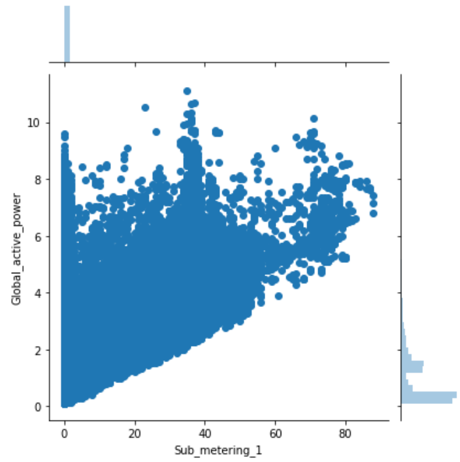
In [31]:
```
sns.jointplot( x = 'Global_intensity' , y = 'Global_active_power' , data = dataset , kind = 'scatter')
```

Out[31]: <seaborn.axisgrid.JointGrid at 0x23687dfb848>



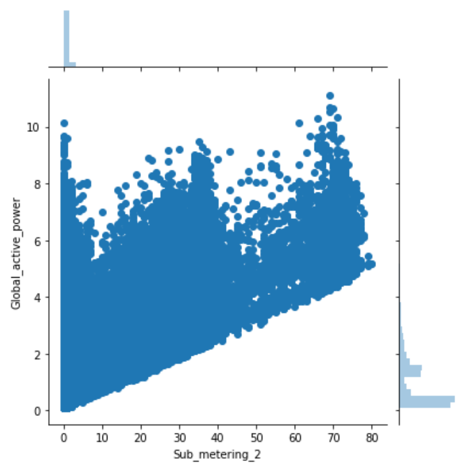
```
In [32]: sns.jointplot( x = 'Sub_metering_1' , y = 'Global_active_power' , data = dataset , kind = 'scatter')
```

```
Out[32]: <seaborn.axisgrid.JointGrid at 0x23688060548>
```



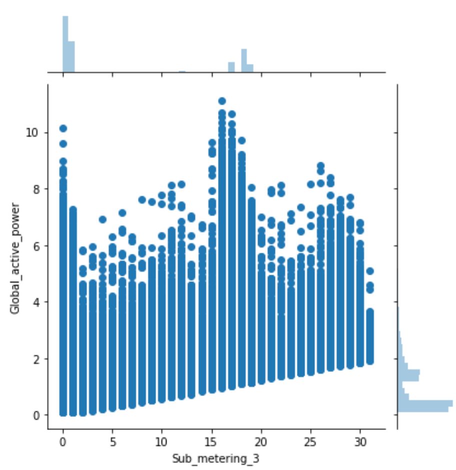

```
In [33]: sns.jointplot( x = 'Sub_metering_2' , y = 'Global_active_power' , data = dataset , kind = 'scatter')
```

```
Out[33]: <seaborn.axisgrid.JointGrid at 0x23688050748>
```




```
In [34]: sns.jointplot( x = 'Sub_metering_3' , y = 'Global_active_power' , data = dataset , kind = 'scatter')
```

```
Out[34]: <seaborn.axisgrid.JointGrid at 0x23689c180c8>
```



independent and depended variable

In [35]:
```
x=dataset.iloc[:,[1,3,4,5,6]]
y=dataset.iloc[:,1]
```

In [36]:
```
x.head()
```

Out[36]:

datetime	Global_reactive_power	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
2006-12-16 17:24:00	0.418	18.4	0.0	1.0	17.0
2006-12-16 17:25:00	0.436	23.0	0.0	1.0	16.0
2006-12-16 17:26:00	0.498	23.0	0.0	2.0	17.0
2006-12-16 17:27:00	0.502	23.0	0.0	1.0	17.0
2006-12-16 17:28:00	0.528	15.8	0.0	1.0	17.0

In [37]:
```
y.head()
```

Out[37]:
```
datetime
2006-12-16 17:24:00    0.418
2006-12-16 17:25:00    0.436
2006-12-16 17:26:00    0.498
2006-12-16 17:27:00    0.502
2006-12-16 17:28:00    0.528
Name: Global_reactive_power, dtype: float64
```

splitting and testing

In [38]:
```
from sklearn.model_selection import train_test_split
```

In [39]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=100)
```

In [40]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_train.shape)
```

```
(1452681, 5)
(622578, 5)
(1452681,)
(1452681,)
```

training the model

In [41]:
```
from sklearn.linear_model import LinearRegression
```

In [42]:
```
lm=LinearRegression()
```

In [43]:
```
lm.fit(x_train,y_train)
```

Out[43]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [44]:
```
predictions = lm.predict(x_test)
```

In [45]:
```
predictions
```

Out[45]:
```
array([-1.13390211e-14, -1.13396001e-14,  1.84000000e-01, ...,
        5.40000000e-02, -1.13418063e-14,  1.52000000e-01])
```

In [46]:
```
from sklearn import metrics
print('MAE:',metrics.mean_absolute_error(y_test,predictions))
print('MSE:',metrics.mean_squared_error(y_test,predictions))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,predictions)))
print('RSquarevalue:',metrics.r2_score(y_test,predictions))
```

```
MAE: 8.00933730516358e-15
MSE: 1.0677939853116176e-28
RMSE: 1.0333411756586581e-14
RSquarevalue: 1.0
```

In [47]:

```
import pickle
filename = 'PCASSS_model.pkl'
pickle.dump(lm,open(filename,'wb'))
```