Individual household electric power consumption

Machine Learning Pipeline

- Data ingestion
- EDA
- Preprocessing
- Pickling for the preprocessing object(save the preprocessing model)

Regression:linear regression,ridge regression,lasso regression,elastic net, support vector regression

Attribute Information:

Dataset Link:
https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption (https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption)

```
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 kilowa
tt)
4.global_reactive_power: household global minute-averaged reactive power (in ki
lowatt)
5.voltage: minute-averaged voltage (in volt)
6.global_intensity: household global minute-averaged current intensity (in ampe
re)
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 micro
wave (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 [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pymongo
from pymongo import MongoClient
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
df=pd.read_csv(r'C:\Users\prasa\Desktop\Prasath\INeuron\INEURO~1\FSDS_B~1\FSDSBO~1\OCTMA
```

In [4]:
```
data=df.sample(n=50000,ignore_index=True)
```

In [5]:
```
data.head()
```

Out[5]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering
0	9/3/2007	12:31:00	1.396	0.000	241.650	5.800	0.0
1	4/1/2009	01:51:00	0.482	0.096	247.020	2.000	0.0
2	12/11/2010	08:24:00	2.204	0.064	237.72	9.2	
3	9/9/2007	22:26:00	1.030	0.368	239.880	4.600	0.0
4	13/7/2008	16:50:00	0.146	0.000	240.830	0.600	0.0

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Cancel

In [6]:
```
data.shape
```

Out[6]:
```
(50000, 9)
```

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Date                   50000 non-null  object 
 1   Time                   50000 non-null  object 
 2   Global_active_power    50000 non-null  object 
 3   Global_reactive_power  50000 non-null  object 
 4   Voltage                50000 non-null  object 
 5   Global_intensity       50000 non-null  object 
 6   Sub_metering_1         50000 non-null  object 
 7   Sub_metering_2         50000 non-null  object 
 8   Sub_metering_3         49334 non-null  float64
dtypes: float64(1), object(8)
memory usage: 3.4+ MB
```

In [8]:
```
data.isna().sum()
```

Out[8]:
```
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           666
dtype: int64
```

In [9]:
```
data['Date'].unique()
```

Out[9]:
```
array(['9/3/2007', '4/1/2009', '12/11/2010', ..., '8/1/2008', '13/8/2008',
       '12/10/2008'], dtype=object)
```

In [10]:
```
data['Date']
```

Out[10]:
```
0          9/3/2007
1          4/1/2009
2         12/11/2010
3          9/9/2007
4         13/7/2008
             ...
49995     12/3/2008
49996     21/7/2010
49997      8/9/2007
49998      7/1/2010
49999     29/5/2010
Name: Date, Length: 50000, dtype: object
```

In [11]:
```
data['Date']=pd.to_datetime(data['Date'])
```

In [12]:
```
data['Year']=data['Date'].dt.year
data['Month']=data['Date'].dt.month
```

In [13]:
```
data['Day']=data['Date'].dt.day
```

In [14]:
```
data.head()
```

Out[14]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
0	2007-09-03	12:31:00	1.396	0.000	241.650	5.800	0.000
1	2009-04-01	01:51:00	0.482	0.096	247.020	2.000	0.000
2	2010-12-11	08:24:00	2.204	0.064	237.72	9.2	0.0
3	2007-09-09	22:26:00	1.030	0.368	239.880	4.600	0.000
4	2008-07-13	16:50:00	0.146	0.000	240.830	0.600	0.000

In [15]:
```
data['Time'].unique()
```

Out[15]:
```
array(['12:31:00', '01:51:00', '08:24:00', ..., '10:05:00', '08:22:00',
       '21:49:00'], dtype=object)
```

In [16]:
```
data['Time']=pd.to_datetime(data['Time'])
```

In [17]:
```
data['Hour']=data['Time'].dt.hour
```

In [18]:
```
data['Minutes']=data['Time'].dt.minute
```

In [19]:
```
data.head()
```

Out[19]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
0	2007-09-03	2022-11-04 12:31:00	1.396	0.000	241.650	5.800	0.000
1	2009-04-01	2022-11-04 01:51:00	0.482	0.096	247.020	2.000	0.000
2	2010-12-11	2022-11-04 08:24:00	2.204	0.064	237.72	9.2	0.0
3	2007-09-09	2022-11-04 22:26:00	1.030	0.368	239.880	4.600	0.000
4	2008-07-13	2022-11-04 16:50:00	0.146	0.000	240.830	0.600	0.000

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Cancel

In [20]:
```
data.head()
```

Out[20]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
0	2007-09-03	2022-11-04 12:31:00	1.396	0.000	241.650	5.800	0.000
1	2009-04-01	2022-11-04 01:51:00	0.482	0.096	247.020	2.000	0.000
2	2010-12-11	2022-11-04 08:24:00	2.204	0.064	237.72	9.2	0.0
3	2007-09-09	2022-11-04 22:26:00	1.030	0.368	239.880	4.600	0.000
4	2008-07-13	2022-11-04 16:50:00	0.146	0.000	240.830	0.600	0.000

In [21]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   50000 non-null  datetime64[ns]
 1   Time                   50000 non-null  datetime64[ns]
 2   Global_active_power    50000 non-null  object
 3   Global_reactive_power  50000 non-null  object
 4   Voltage                50000 non-null  object
 5   Global_intensity       50000 non-null  object
 6   Sub_metering_1         50000 non-null  object
 7   Sub_metering_2         50000 non-null  object
 8   Sub_metering_3         49334 non-null  float64
 9   Year                   50000 non-null  int64
 10  Month                  50000 non-null  int64
 11  Day                    50000 non-null  int64
 12  Hour                   50000 non-null  int64
 13  Minutes                50000 non-null  int64
dtypes: datetime64[ns](2), float64(1), int64(5), object(6)
memory usage: 5.3+ MB
```

In [22]:
```
data_empty_Index=data[data['Global_active_power']=='?'].index
```

In [23]:
```
data.iloc[data_empty_Index]
```

Out[23]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering
36	2010-09-26	2022-11-04 13:10:00	?	?	?	?	
103	2007-04-28	2022-11-04 01:16:00	?	?	?	?	
168	2010-09-25	2022-11-04 21:45:00	?	?	?	?	
196	2010-01-13	2022-11-04 02:04:00	?	?	?	?	
227	2009-06-13	2022-11-04 04:40:00	?	?	?	?	
...	...	...	...	...	...	...	...
49713	2007-04-29	2022-11-04 19:46:00	?	?	?	?	
49732	2010-08-19	2022-11-04 03:46:00	?	?	?	?	
49772	2010-01-14	2022-11-04 14:08:00	?	?	?	?	
49825	2010-01-13	2022-11-04 17:48:00	?	?	?	?	
49904	2009-06-14	2022-11-04 07:03:00	?	?	?	?	

666 rows × 14 columns

In [24]:
```
data.replace(['?', 'nan', np.nan], -1, inplace=True)
```

In [25]:
```
data.isna().sum()
```

Out[25]:
```
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
Year                   0
Month                  0
Day                    0
Hour                   0
Minutes                0
dtype: int64
```

In [26]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   50000 non-null  datetime64[ns]
 1   Time                   50000 non-null  datetime64[ns]
 2   Global_active_power    50000 non-null  object
 3   Global_reactive_power  50000 non-null  object
 4   Voltage                50000 non-null  object
 5   Global_intensity       50000 non-null  object
 6   Sub_metering_1         50000 non-null  object
 7   Sub_metering_2         50000 non-null  object
 8   Sub_metering_3         50000 non-null  float64
 9   Year                   50000 non-null  int64
 10  Month                  50000 non-null  int64
 11  Day                    50000 non-null  int64
 12  Hour                   50000 non-null  int64
 13  Minutes                50000 non-null  int64
dtypes: datetime64[ns](2), float64(1), int64(5), object(6)
memory usage: 5.3+ MB
```

In [27]:
```
from sklearn.impute import SimpleImputer
num_vars= ['Global_active_power', 'Global_reactive_power', 'Voltage',
           'Global_intensity', 'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']
for i in num_vars:
    data[i] = pd.to_numeric(data[i])
imp = SimpleImputer(missing_values=-1, strategy='mean')
data[num_vars] = imp.fit_transform(data[num_vars])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   50000 non-null  datetime64[ns]
 1   Time                   50000 non-null  datetime64[ns]
 2   Global_active_power    50000 non-null  float64
 3   Global_reactive_power  50000 non-null  float64
 4   Voltage                50000 non-null  float64
 5   Global_intensity       50000 non-null  float64
 6   Sub_metering_1         50000 non-null  float64
 7   Sub_metering_2         50000 non-null  float64
 8   Sub_metering_3         50000 non-null  float64
 9   Year                   50000 non-null  int64
 10  Month                  50000 non-null  int64
 11  Day                    50000 non-null  int64
 12  Hour                   50000 non-null  int64
 13  Minutes                50000 non-null  int64
dtypes: datetime64[ns](2), float64(7), int64(5)
memory usage: 5.3 MB
```

In [28]:
```
data.drop(['Date','Time'],axis=1,inplace=True)
```

In [29]:
```
data.isna().sum()
```

Out[29]:
```
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
Day                      0
Hour                     0
Minutes                  0
dtype: int64
```

In [30]:
```
def merge_Metering(data):
    data['Total_Metering']=data['Sub_metering_1']+data['Sub_metering_2']+data['Sub_meter
    return data
```

In [31]:
```
data=merge_Metering(data)
```

In [32]:
```
data.head()
```

Out[32]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_
0	1.396	0.000	241.65	5.8	0.0	0.
1	0.482	0.096	247.02	2.0	0.0	0.
2	2.204	0.064	237.72	9.2	0.0	0.
3	1.030	0.368	239.88	4.6	0.0	1.
4	0.146	0.000	240.83	0.6	0.0	0.

In [33]:
```
# (global_active_power*1000/60 - sub_metering_1 - sub_metering_2 - sub_metering_3)
data['Power Consumed']=(data['Global_reactive_power']*1000)/60-data['Total_Metering']
```

In [34]:
```
data.head()
```

Out[34]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2
0	1.396	0.000	241.65	5.8	0.0	0.0
1	0.482	0.096	247.02	2.0	0.0	0.0
2	2.204	0.064	237.72	9.2	0.0	0.0
3	1.030	0.368	239.88	4.6	0.0	1.0
4	0.146	0.000	240.83	0.6	0.0	0.0

In [35]:
```
data.describe().T
```

Out[35]:

	count	mean	std	min	25%	50%	75%	
Global_active_power	50000.0	1.093958	1.056850	0.076	0.310000	0.630	1.518000	9
Global_reactive_power	50000.0	0.123732	0.111672	0.000	0.050000	0.102	0.192000	
Voltage	50000.0	240.841642	3.207501	225.450	239.040000	240.970	242.830000	253
Global_intensity	50000.0	4.637885	4.442410	0.200	1.400000	2.800	6.400000	38
Sub_metering_1	50000.0	1.142721	6.138807	0.000	0.000000	0.000	0.000000	78
Sub_metering_2	50000.0	1.307577	5.858565	0.000	0.000000	0.000	1.000000	77
Sub_metering_3	50000.0	6.410670	8.356425	0.000	0.000000	1.000	17.000000	31
Year	50000.0	2008.436220	1.129122	2006.000	2007.000000	2008.000	2009.000000	2010
Month	50000.0	6.513720	3.436517	1.000	4.000000	7.000	9.000000	12
Day	50000.0	15.747520	8.848235	1.000	8.000000	16.000	23.000000	31
Hour	50000.0	11.546220	6.928231	0.000	6.000000	12.000	18.000000	23
Minutes	50000.0	29.510000	17.337448	0.000	14.000000	30.000	44.000000	59
Total_Metering	50000.0	8.860968	12.848745	0.000	0.000000	1.000	18.000000	126
Power Consumed	50000.0	-6.798762	12.649722	-120.800	-15.966667	0.000	1.333333	14

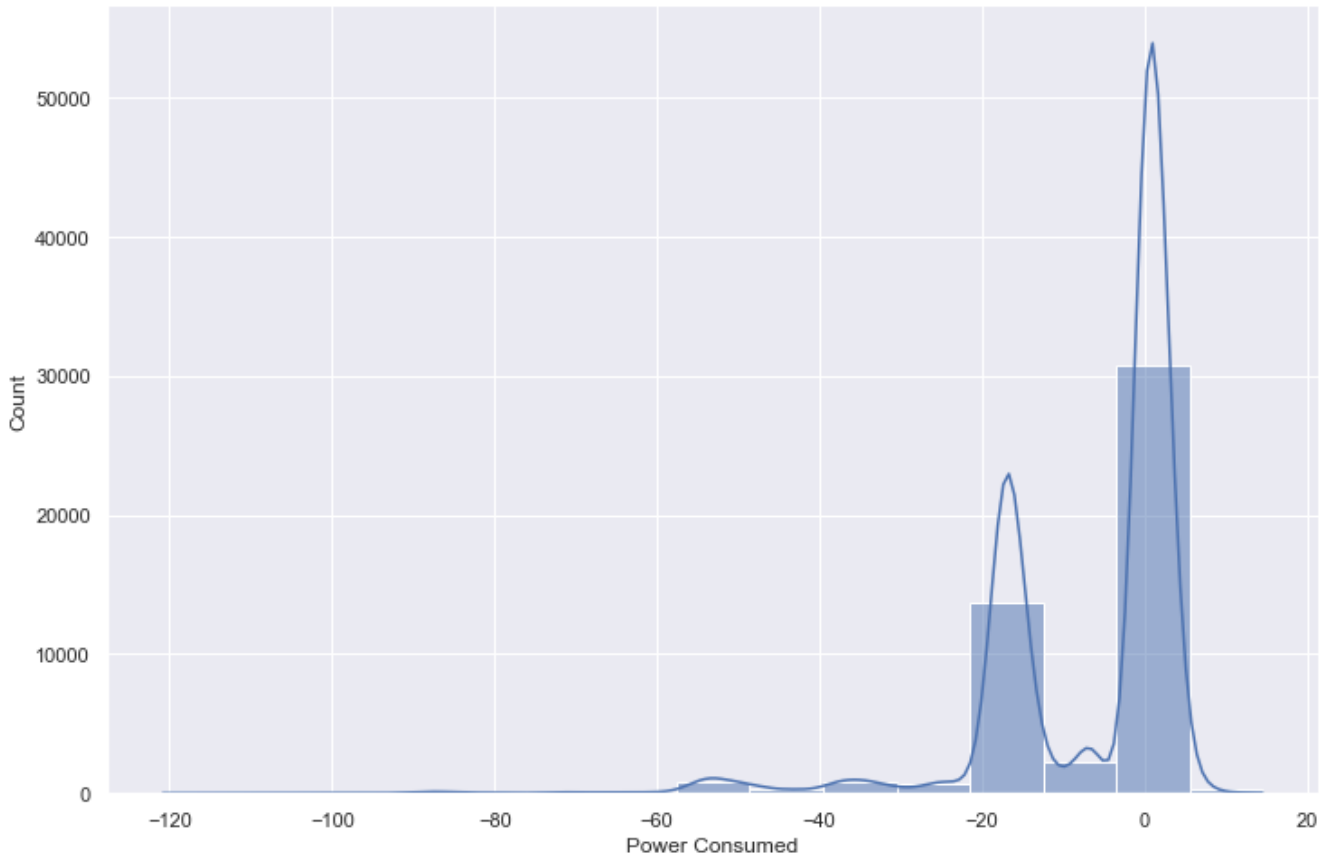
In [36]:
```
data.corr()
```

Out[36]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_meterin
Global_active_power	1.000000	0.245048	-0.408372	0.998910	0.490
Global_reactive_power	0.245048	1.000000	-0.110139	0.264056	0.129
Voltage	-0.408372	-0.110139	1.000000	-0.419450	-0.204
Global_intensity	0.998910	0.264056	-0.419450	1.000000	0.495
Sub_metering_1	0.490378	0.129453	-0.204272	0.495463	1.000
Sub_metering_2	0.439967	0.133839	-0.163492	0.444837	0.061
Sub_metering_3	0.636064	0.085577	-0.274544	0.624219	0.106
Year	-0.036677	0.040431	0.247092	-0.040535	-0.012
Month	0.010138	0.014770	0.037184	0.009455	0.001
Day	-0.014831	-0.002345	0.005556	-0.014965	-0.009
Hour	0.279068	0.119727	-0.180074	0.279169	0.109
Minutes	0.000104	-0.002247	0.014676	0.000036	-0.000
Total_Metering	0.848576	0.178532	-0.350698	0.845522	0.575
Power Consumed	-0.825872	-0.034206	0.340010	-0.819974	-0.565

In [37]:
```
sns.set(rc={'figure.figsize': (12,8)})
sns.histplot(data=data,x='Power Consumed',kde=True,bins=15)
```

Out[37]:
```
<AxesSubplot:xlabel='Power Consumed', ylabel='Count'>
```

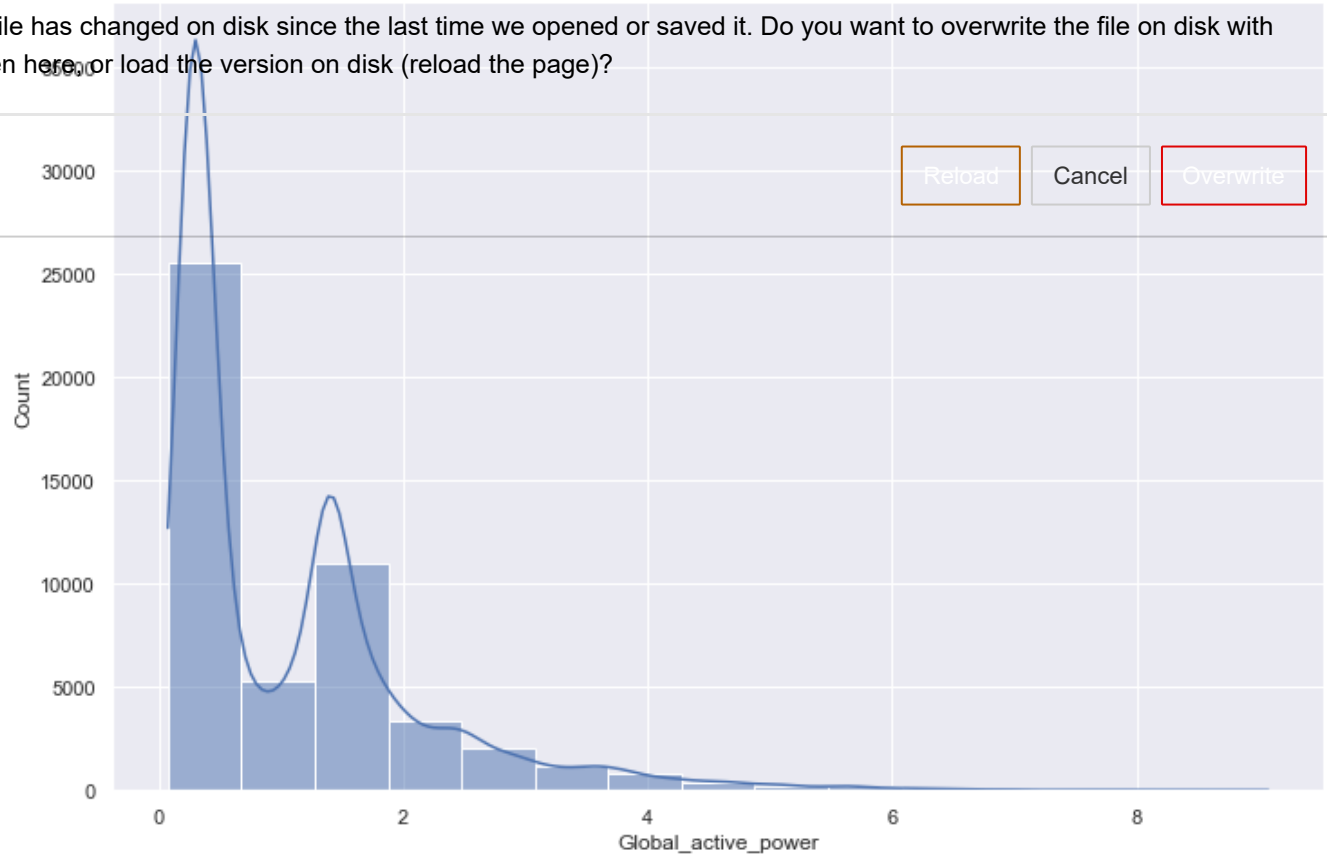


```
In [38]: sns.histplot(data=data,x='Global_active_power',kde=True,bins=15)
Out[38]: <AxesSubplot:xlabel='Global_active_power', ylabel='Count'>
```

Notebook changed

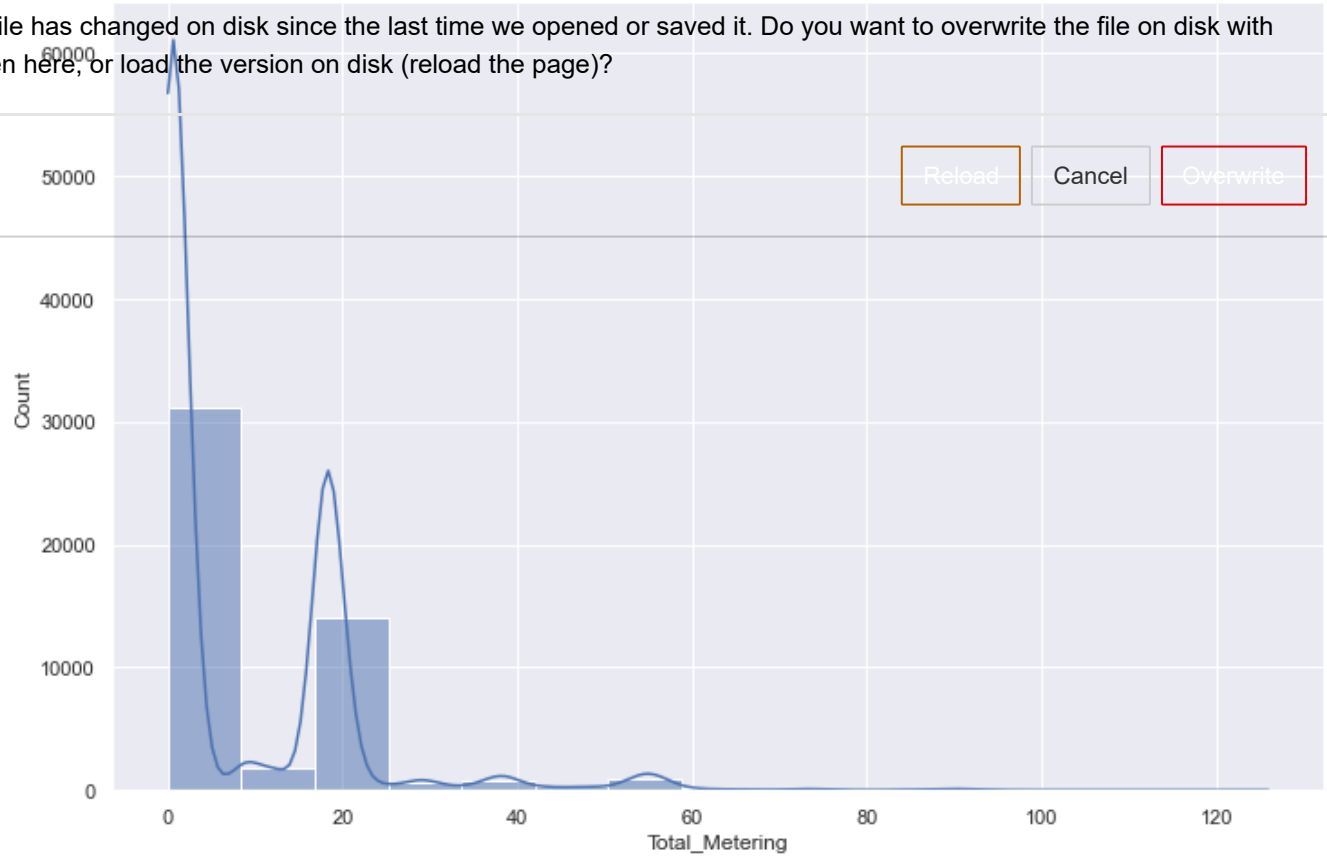
The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite



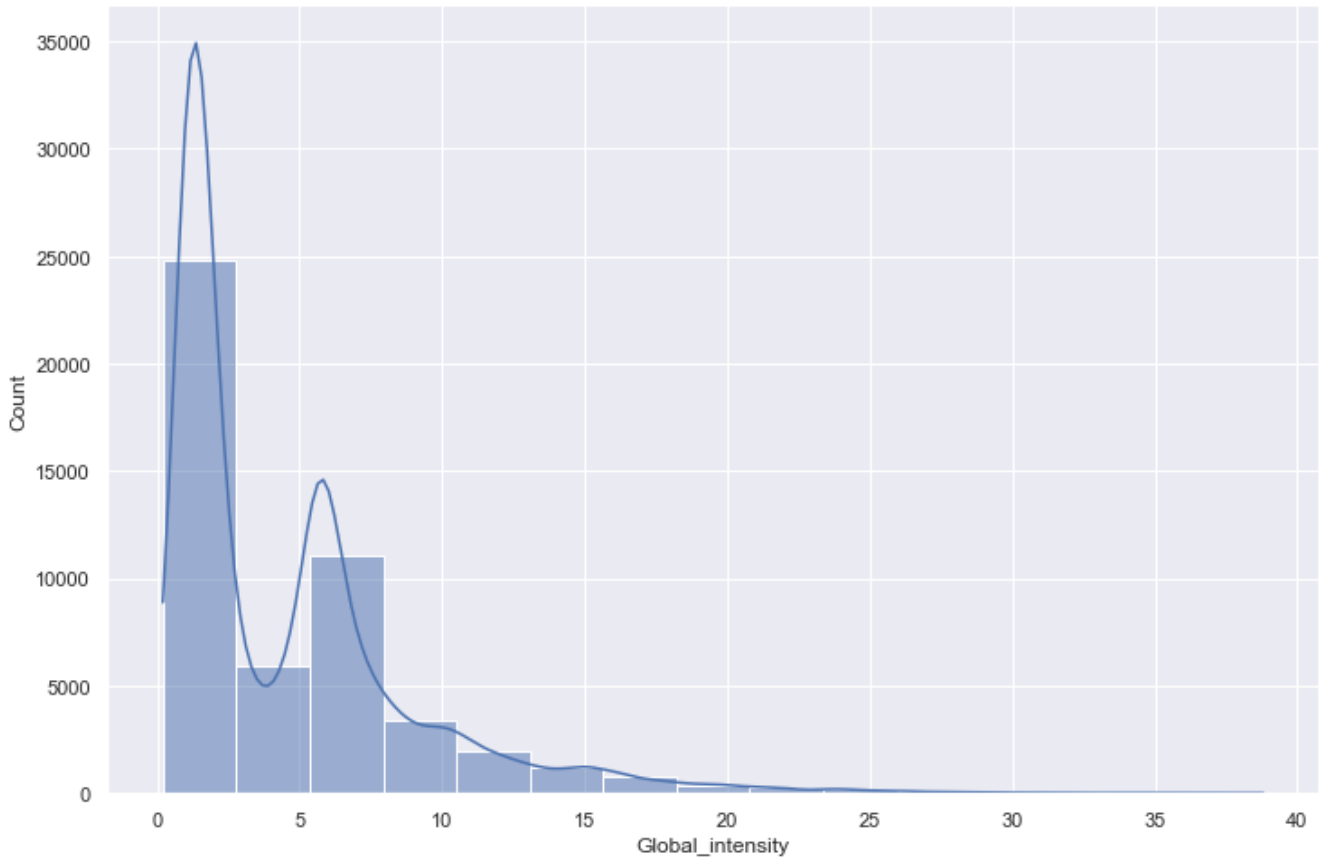
In [39]:
```
sns.histplot(data=data,x='Total_Metering',kde=True,bins=15)
```

Out[39]: <AxesSubplot:xlabel='Total_Metering', ylabel='Count'>



In [40]:
```
sns.histplot(data=data,x='Global_intensity',kde=True,bins=15)
```

Out[40]: <AxesSubplot:xlabel='Global_intensity', ylabel='Count'>

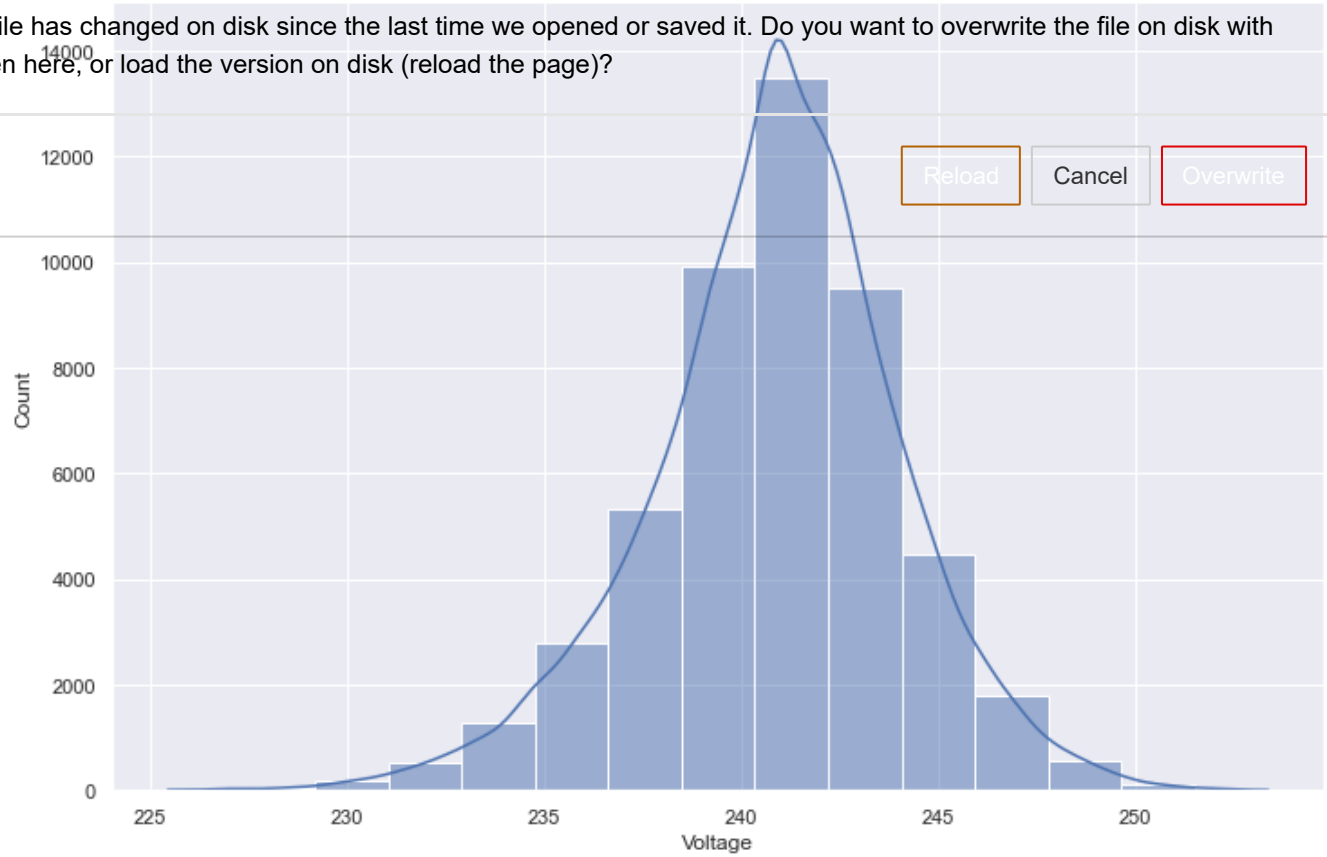


```
In [41]: sns.histplot(data=data,x='Voltage',kde=True,bins=15)
Out[41]: <AxesSubplot:xlabel='Voltage', ylabel='Count'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite



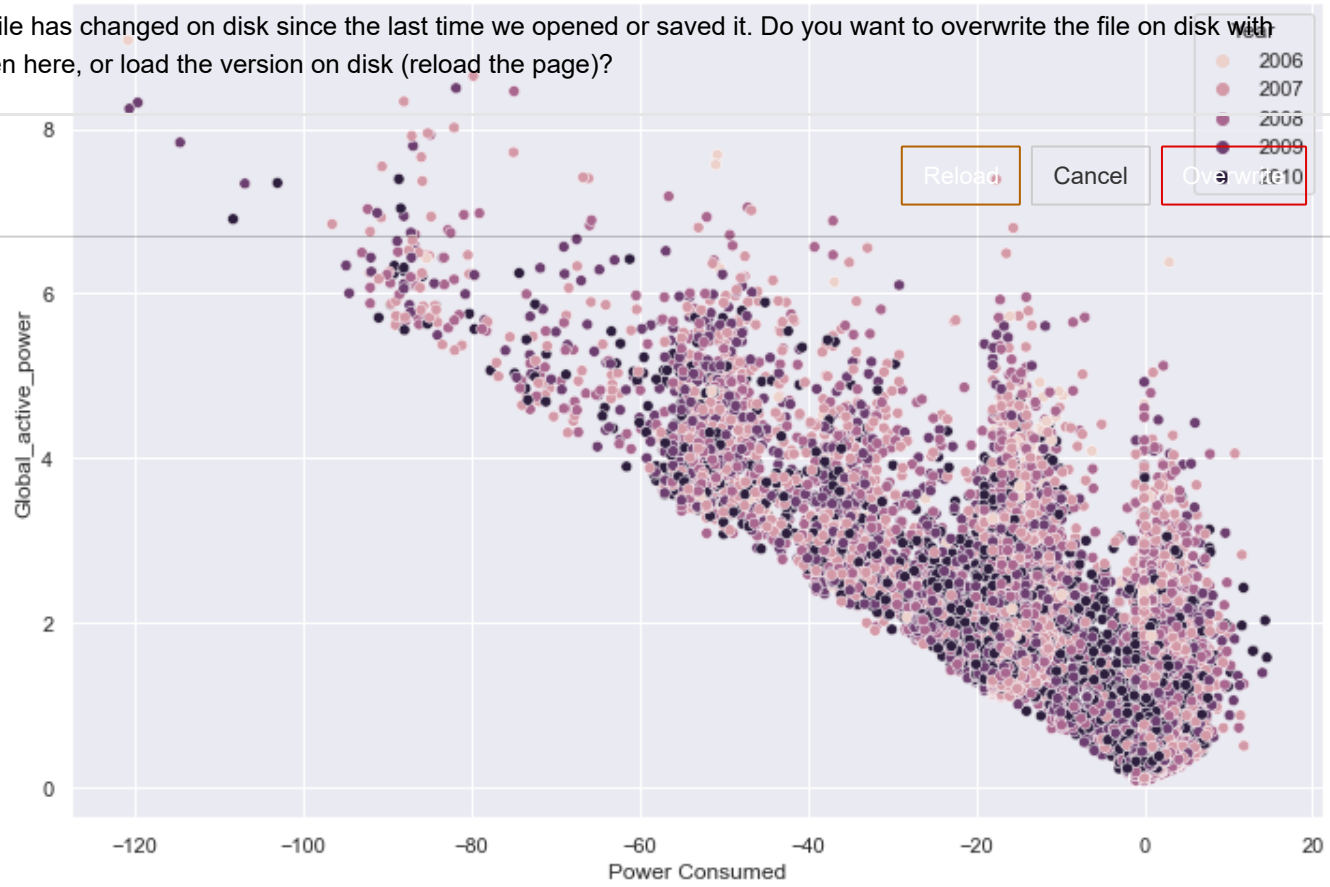
In [42]:
```
sns.scatterplot(x='Power Consumed',y='Global_active_power',data=data,hue='Year')
```

Out[42]: <AxesSubplot:xlabel='Power Consumed', ylabel='Global_active_power'>

Notebook changed

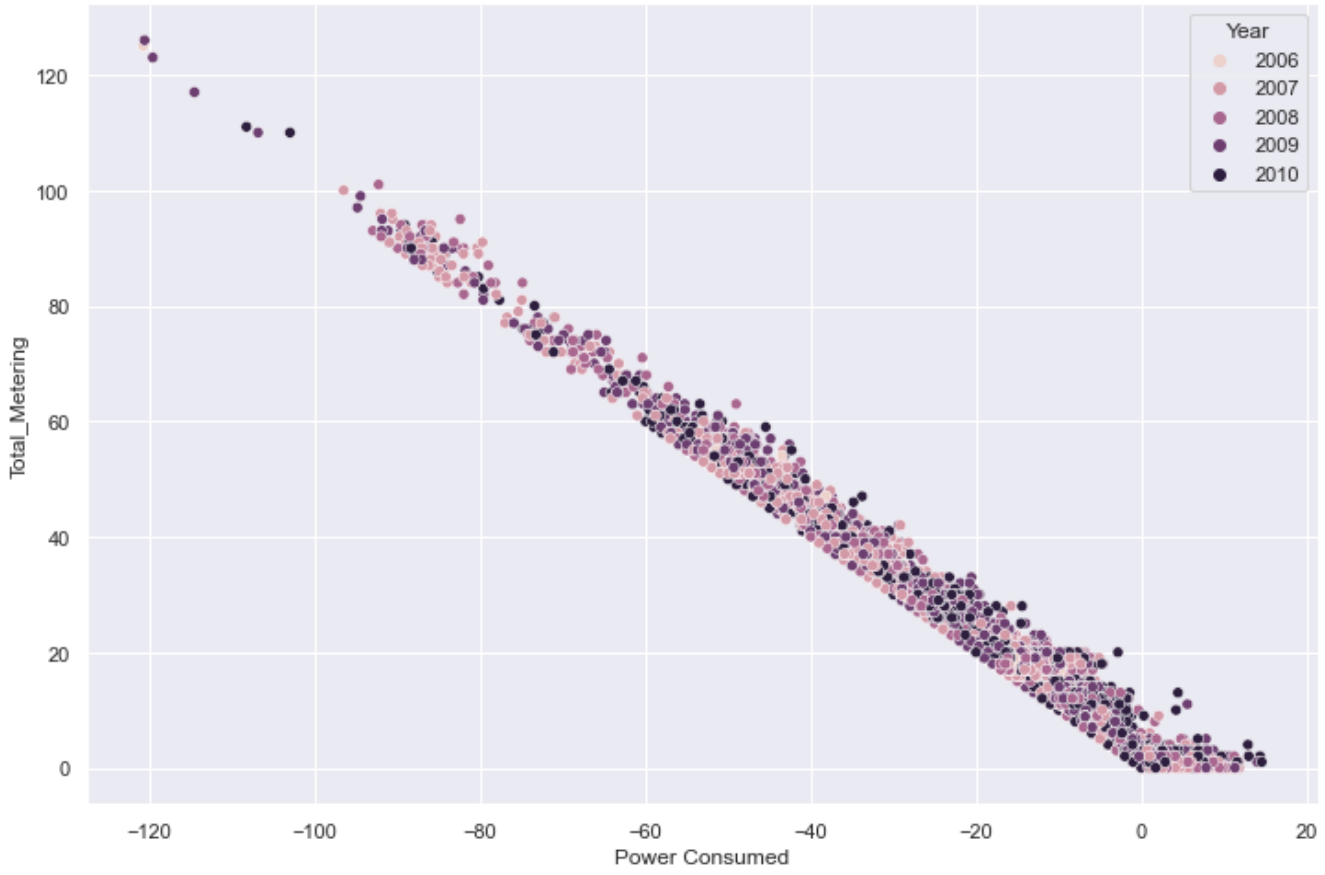
The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite



In [43]:
```
sns.scatterplot(x='Power Consumed',y='Total_Metering',data=data,hue='Year')
```

Out[43]: <AxesSubplot:xlabel='Power Consumed', ylabel='Total_Metering'>



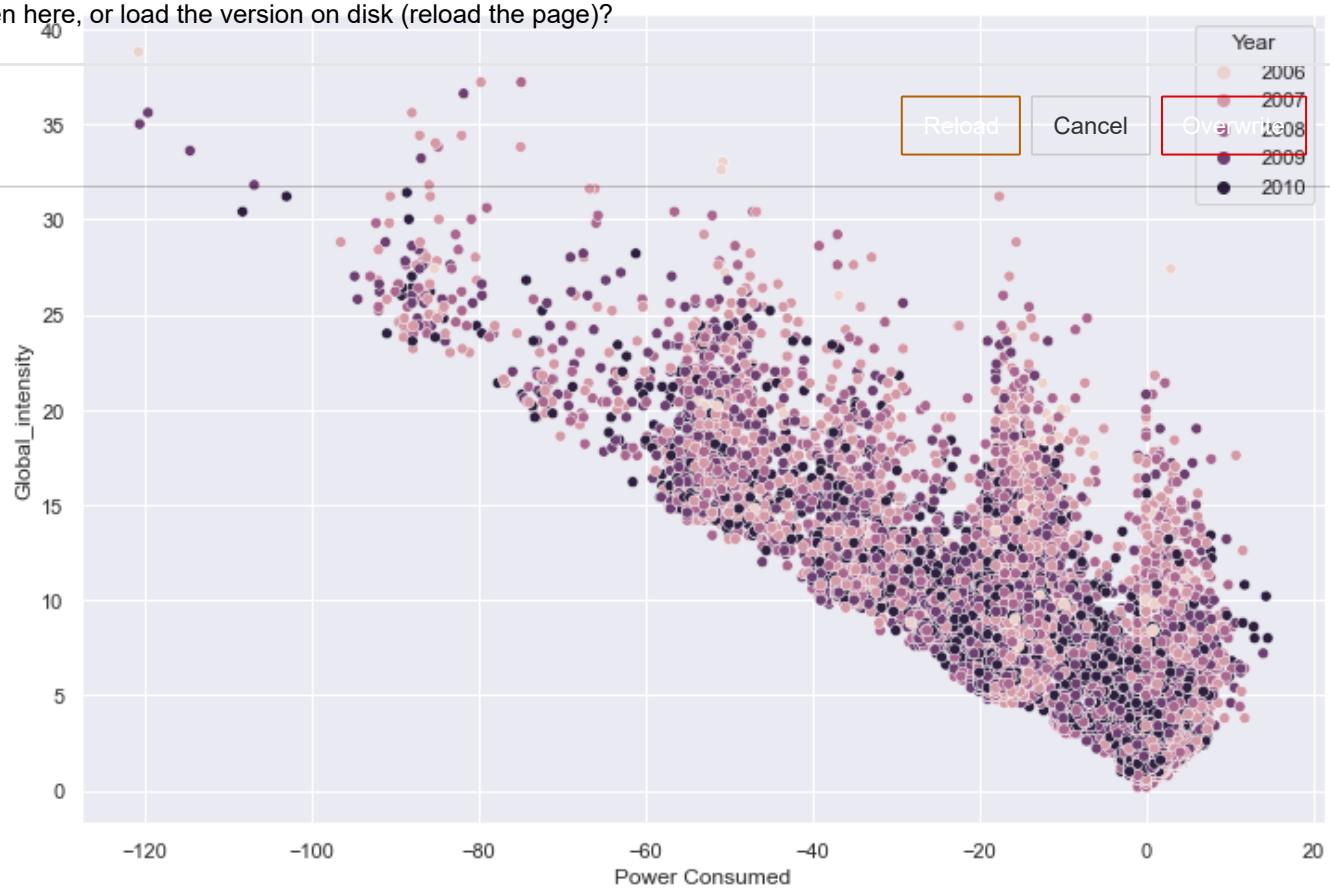
```
In [44]: sns.scatterplot(x='Power Consumed',y='Global_intensity',data=data,hue='Year')
```

Notebook changed

```
Out[44]: <AxesSubplot:xlabel='Power Consumed', ylabel='Global_intensity'>
```

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite



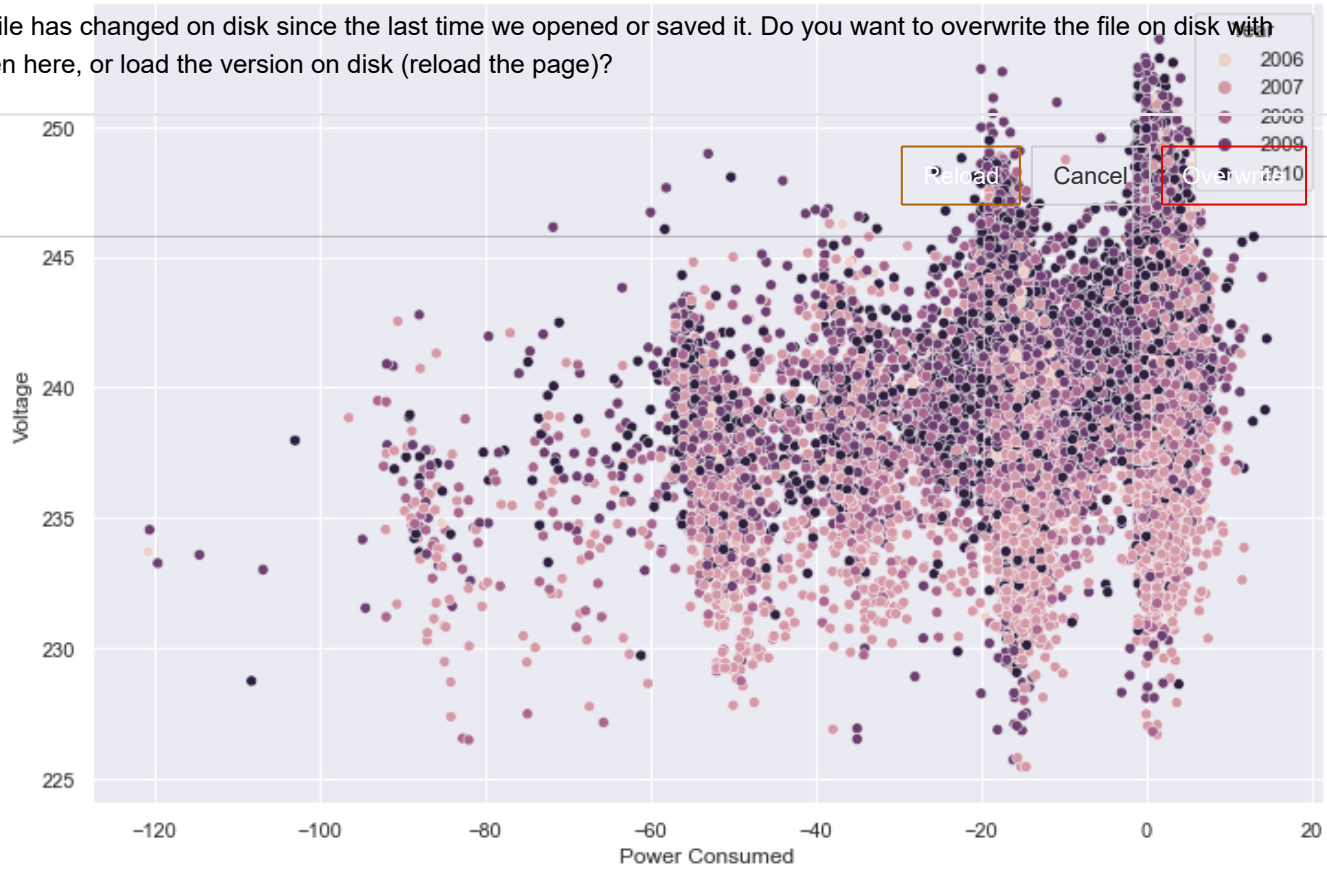
In [45]:
```
sns.scatterplot(x='Power Consumed',y='Voltage',data=data,hue='Year')
```

Out[45]: <AxesSubplot:xlabel='Power Consumed', ylabel='Voltage'>

Notebook changed

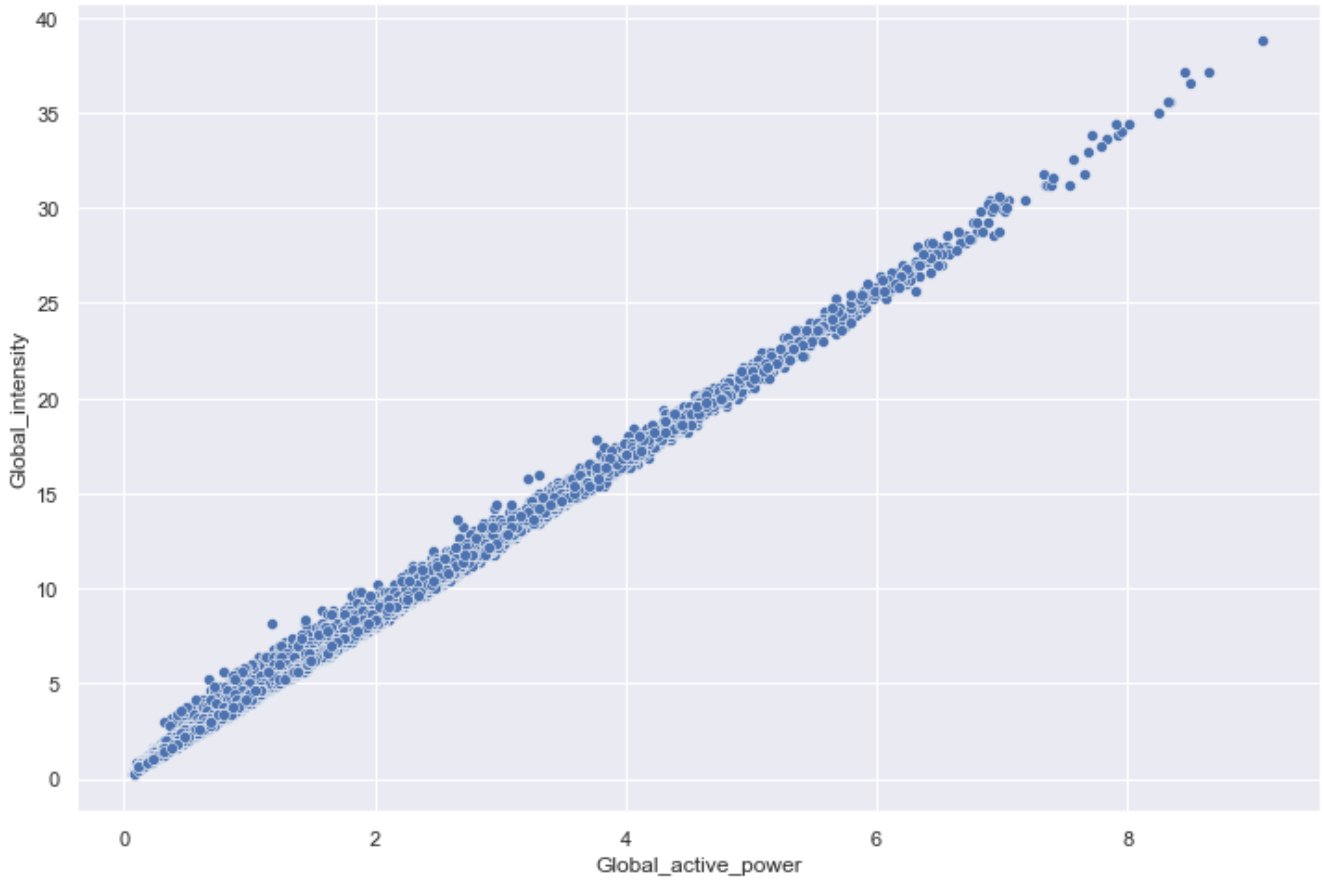
The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload | Cancel | Overwrite



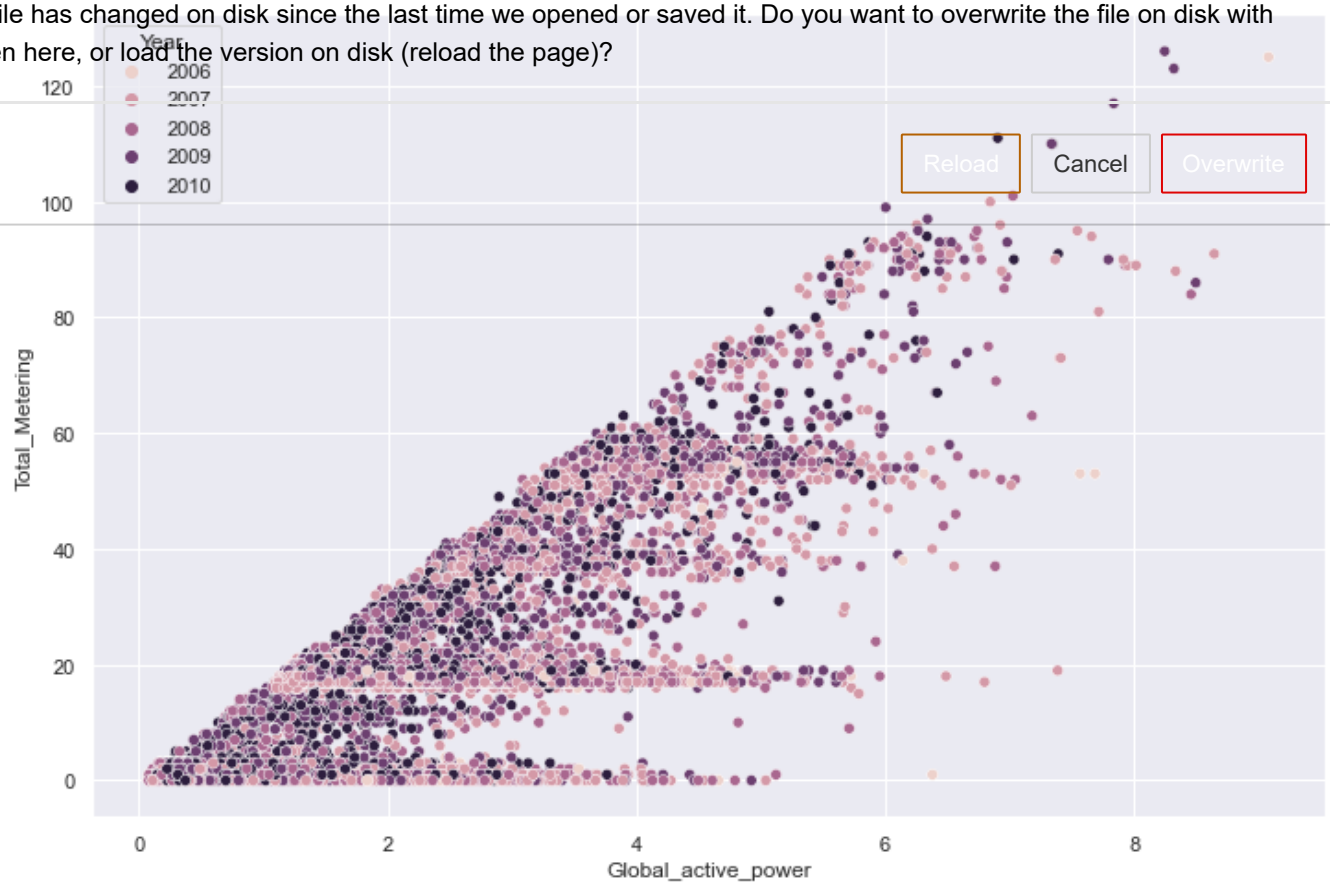
In [46]:
```
sns.scatterplot(x='Global_active_power',y='Global_intensity',data=data)
```

Out[46]: <AxesSubplot:xlabel='Global_active_power', ylabel='Global_intensity'>



In [47]:
```
sns.scatterplot(x='Global_active_power',y='Total_Metering',data=data,hue='Year')
```

Out[47]: <AxesSubplot:xlabel='Global_active_power', ylabel='Total_Metering'>



Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload | Cancel | Overwrite

In [48]:
```
data.head()
```

Out[48]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2
0	1.396	0.000	241.65	5.8	0.0	0.0
1	0.482	0.096	247.02	2.0	0.0	0.0
2	2.204	0.064	237.72	9.2	0.0	0.0
3	1.030	0.368	239.88	4.6	0.0	1.0
4	0.146	0.000	240.83	0.6	0.0	0.0

In [49]:
```
data_1=data.drop(['Sub_metering_1','Sub_metering_2','Sub_metering_3','Year','Month','Day
```

```
In [50]: sns.heatmap(data_1.corr(),cmap="crest",annot=True)
Out[50]: <AxesSubplot:>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

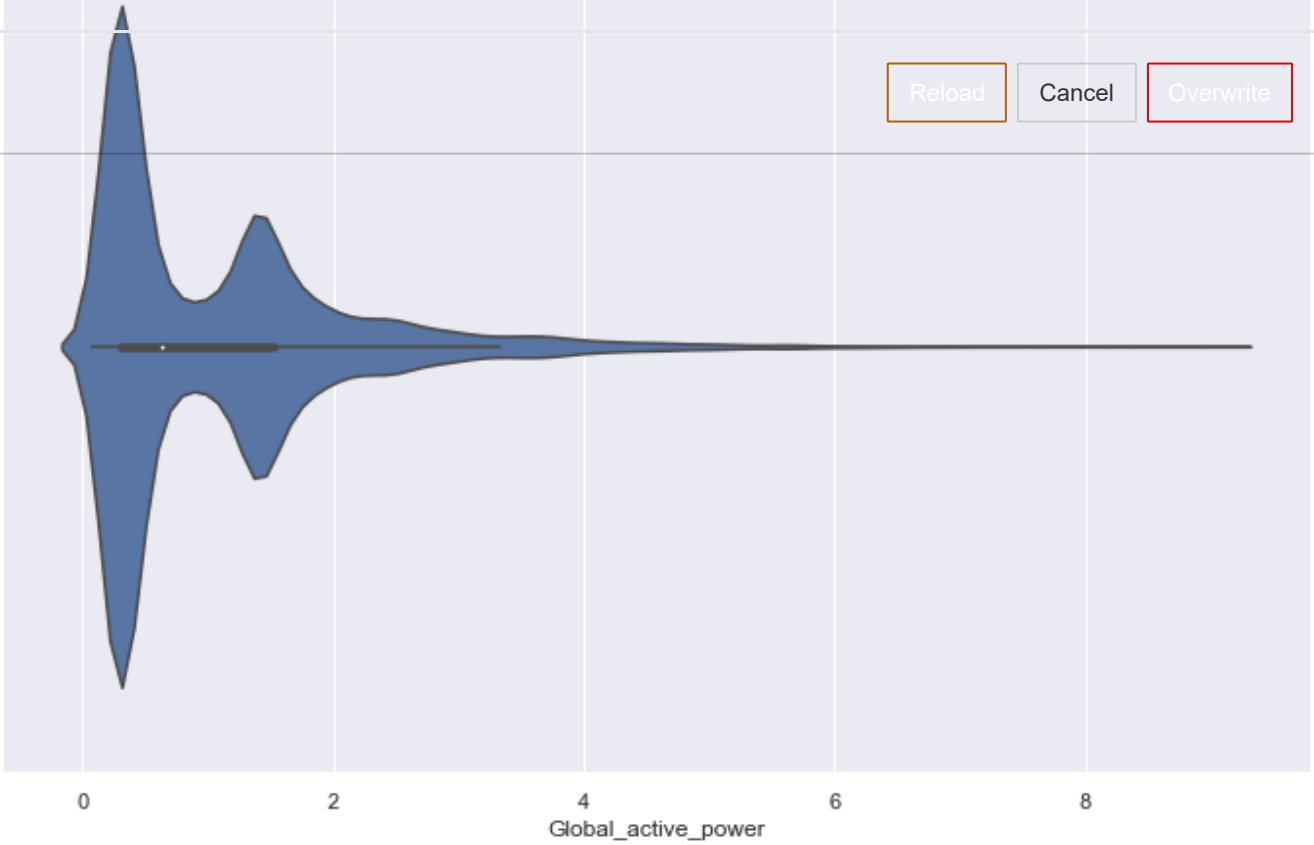
Reload Cancel

```
In [51]: sns.violinplot(data=data_1 ,x="Global_active_power")
Out[51]: <AxesSubplot:xlabel='Global_active_power'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite

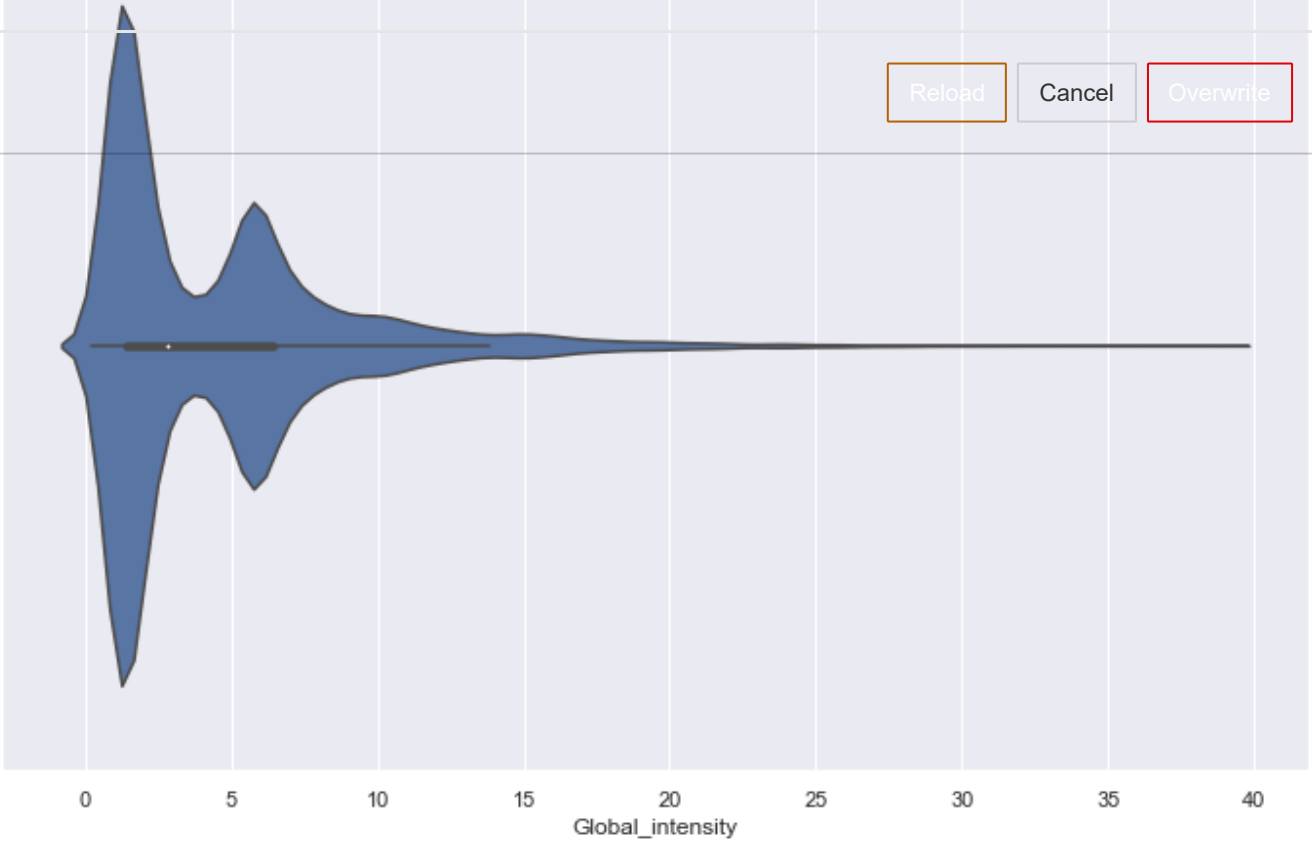


```
In [52]: sns.violinplot(data=data_1 ,x="Global_intensity")
Out[52]: <AxesSubplot:xlabel='Global_intensity'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite

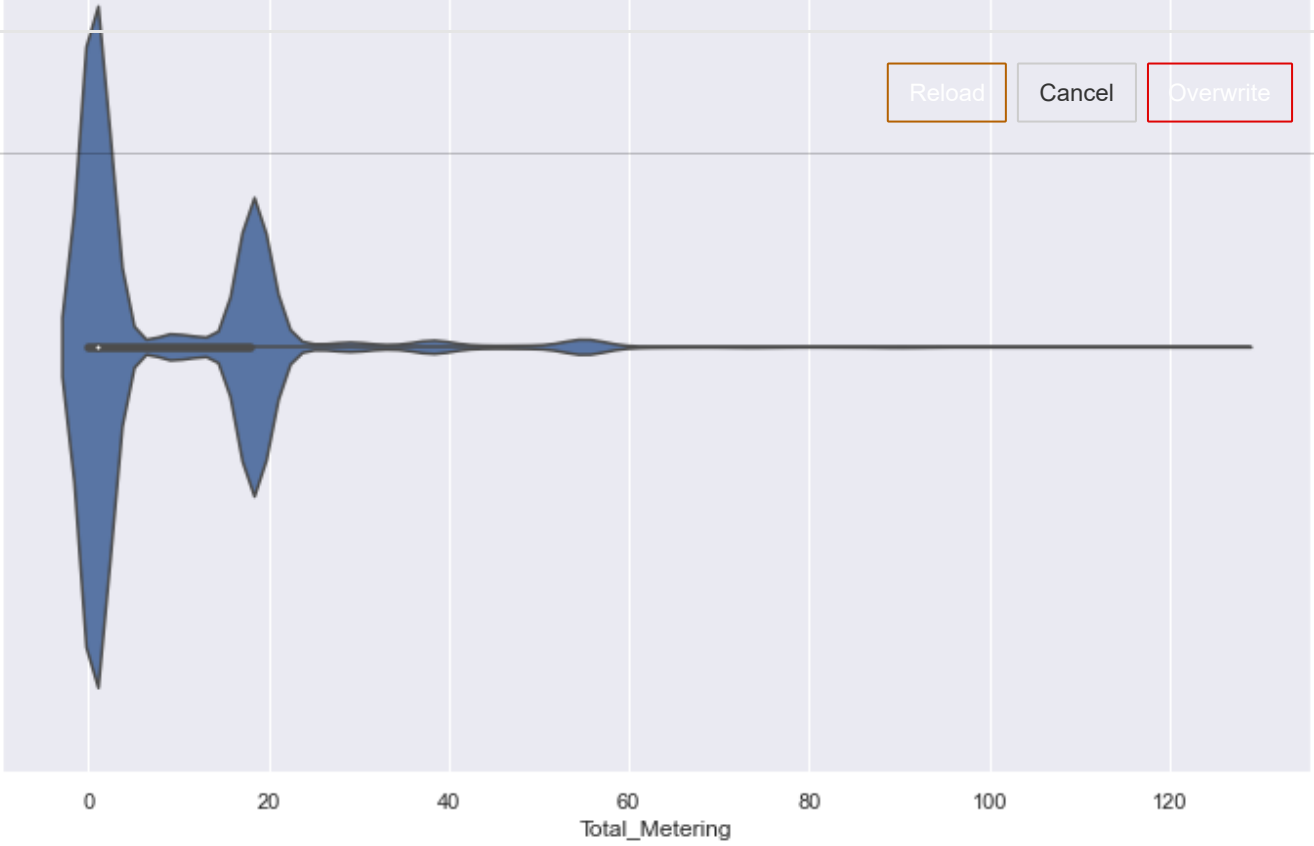


```
In [53]: sns.violinplot(data=data_1,x='Total_Metering')
Out[53]: <AxesSubplot:xlabel='Total_Metering'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite

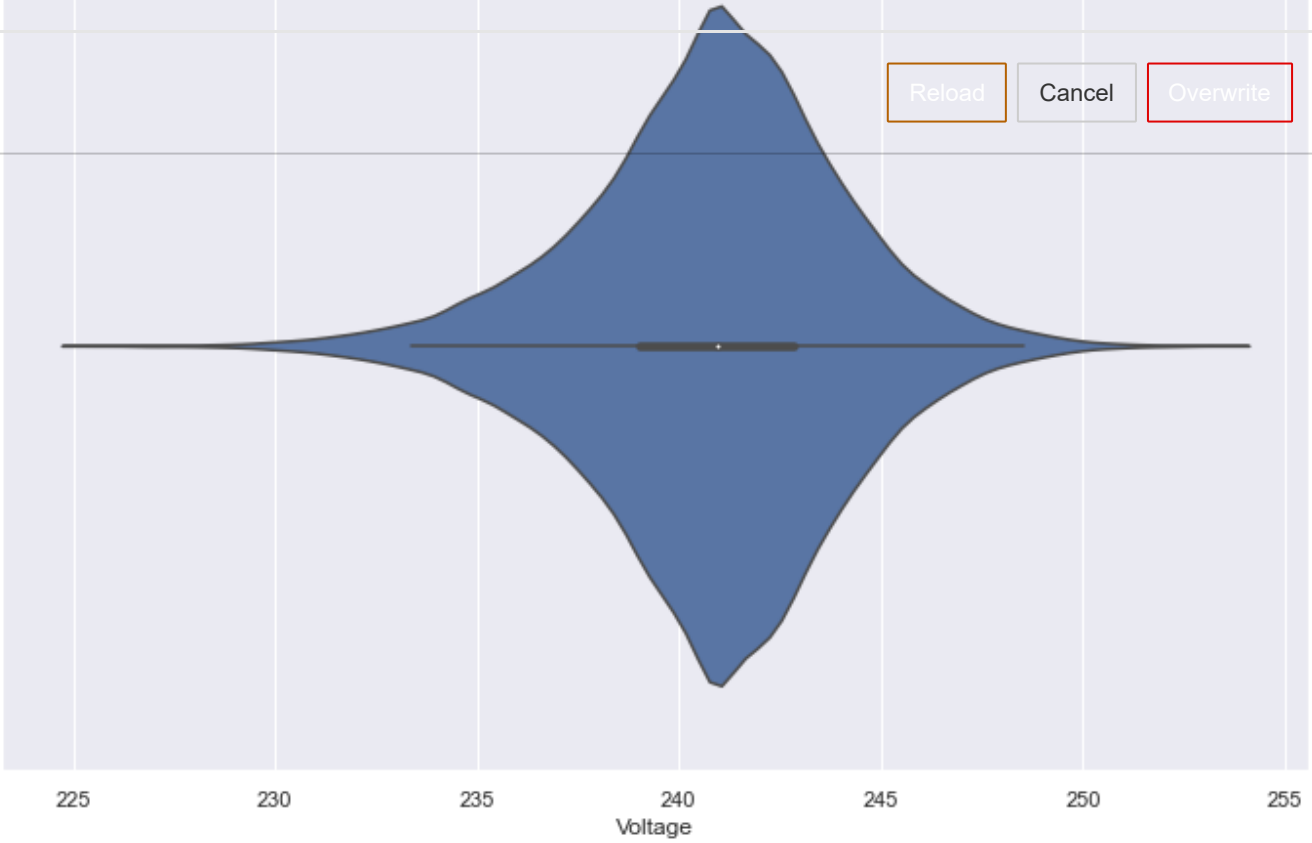


```
In [54]: sns.violinplot(data=data_1,x='Voltage')
Out[54]: <AxesSubplot:xlabel='Voltage'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload | Cancel | Overwrite

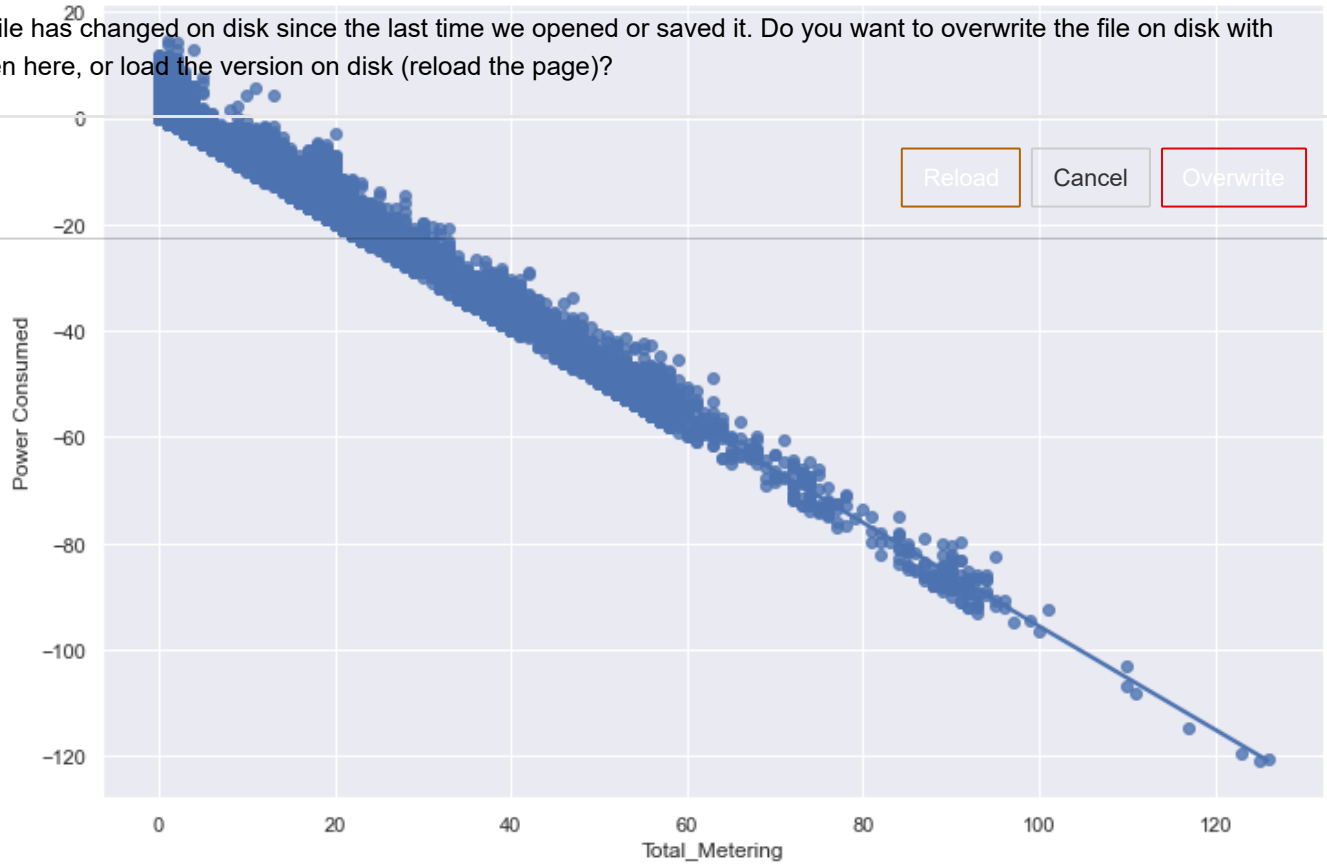


```
In [55]: sns.regplot(x='Total_Metering',y='Power Consumed',data=data_1)
Out[55]: <AxesSubplot:xlabel='Total_Metering', ylabel='Power Consumed'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite

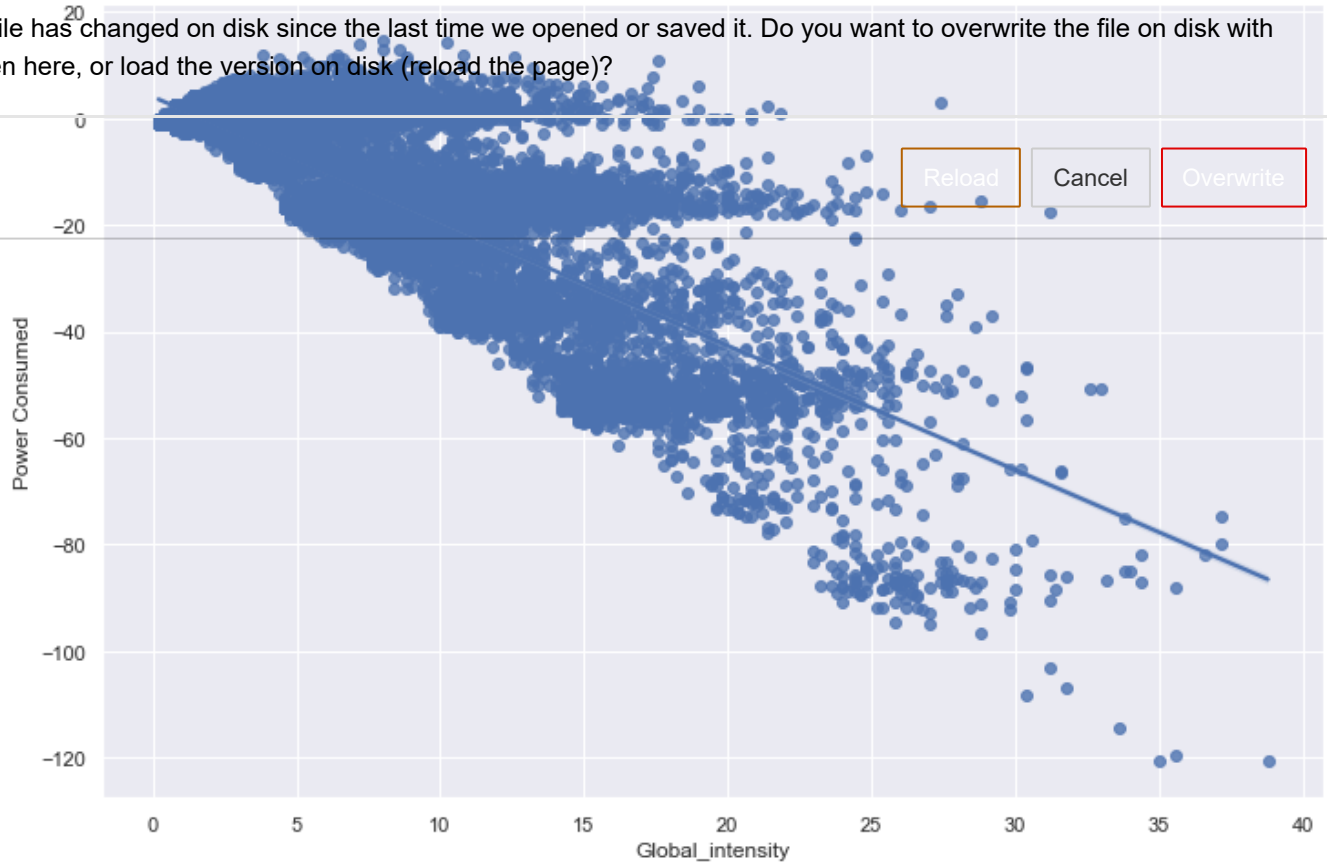


```
In [56]: sns.regplot(x='Global_intensity',y='Power Consumed',data=data_1)
Out[56]: <AxesSubplot:xlabel='Global_intensity', ylabel='Power Consumed'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite

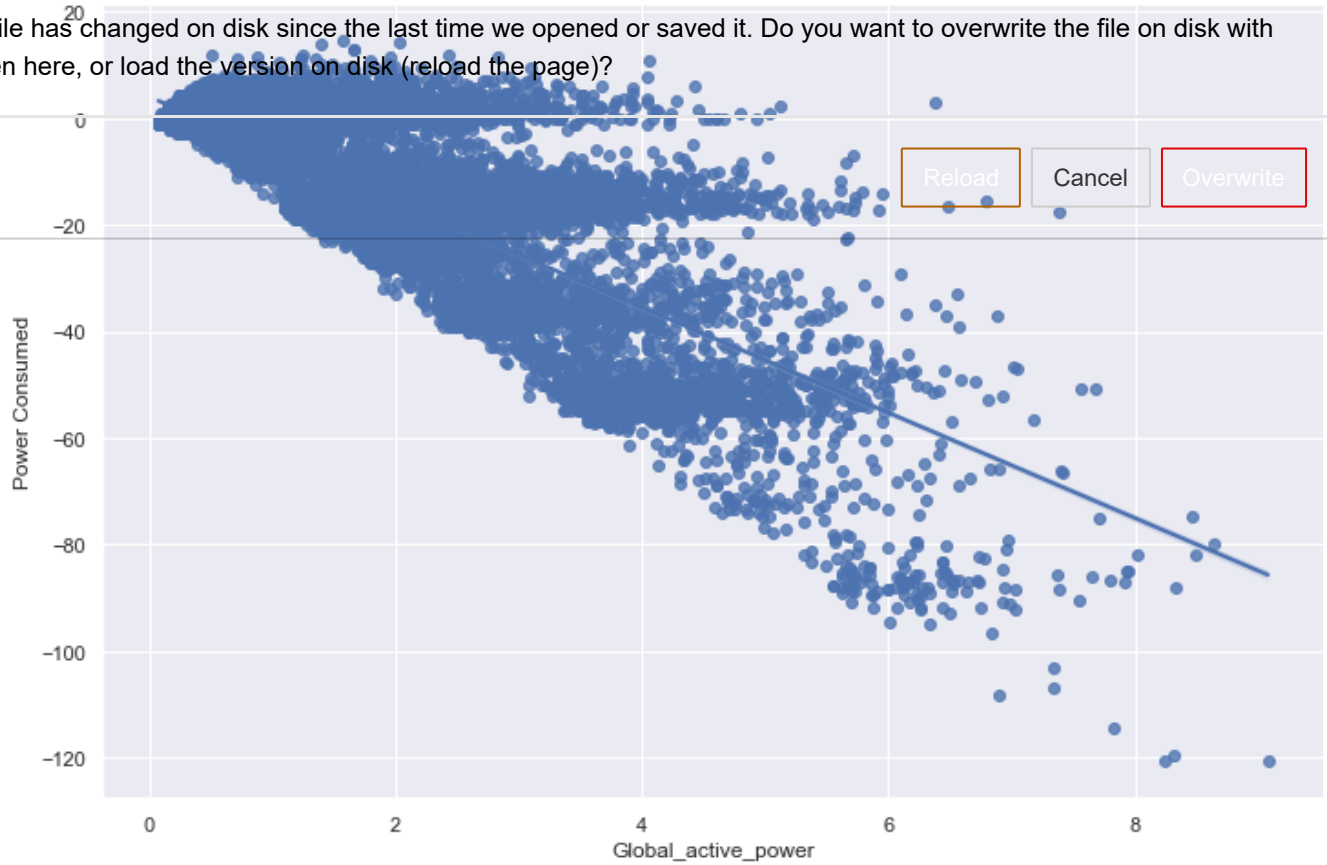


```
In [57]: sns.regplot(x='Global_active_power',y='Power Consumed',data=data_1)
Out[57]: <AxesSubplot:xlabel='Global_active_power', ylabel='Power Consumed'>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite



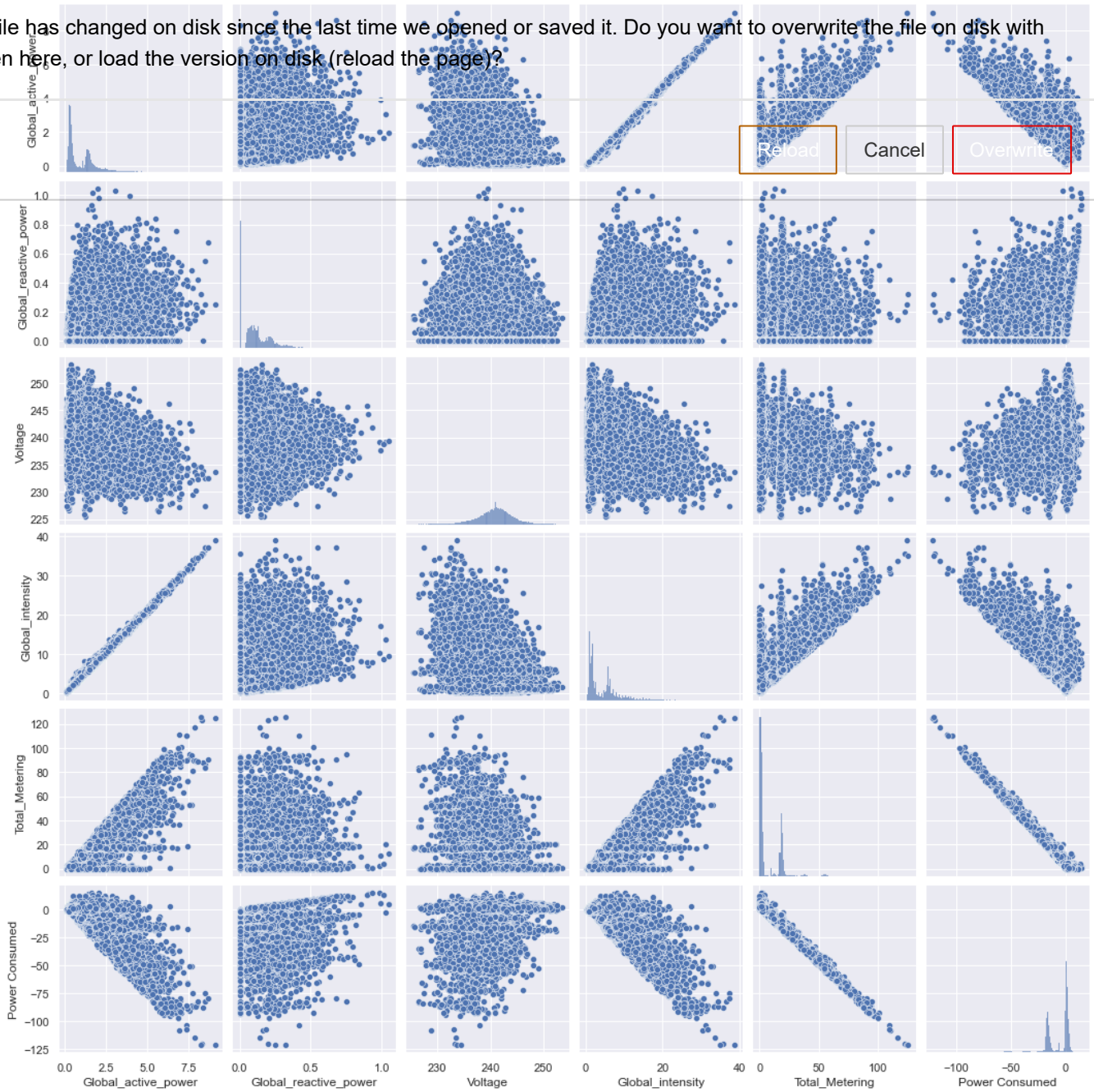
In [58]:
```
sns.pairplot(data_1)
```

Out[58]:
```
<seaborn.axisgrid.PairGrid at 0x24c3be855b0>
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload | Cancel | Overwrite



In [106]:
```
pd.to_pickle(data_1,'Preprocessed.pkl')
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Cancel

In [59]:
```
client = pymongo.MongoClient(
"mongodb+srv://prasathkkps:Luci1108@cluster0.ldqcx.mongodb.net/?retryWrites=true&w=m
db = client.test
print(db)
```

```
Database(MongoClient(host=['cluster0-shard-00-02.ldqcx.mongodb.net:27017', 'cluster0-s
hard-00-01.ldqcx.mongodb.net:27017', 'cluster0-shard-00-00.ldqcx.mongodb.net:27017'],
document_class=dict, tz_aware=False, connect=True, retrywrites=True, w='majority', aut
hsource='admin', replicaset='atlas-2mebps-shard-0', tls=True), 'test')
```

In [60]:
```
database=client['Power_Consumption_Data']
collection=database['PowerConsumed']
```

In [107]:
```
data1=pd.read_pickle('Preprocessed.pkl')
```

In [108]:
```
df1=data_1.to_json('Power_Consumption.json')
```

In [109]:
```
data_1.head()
```

Out[109]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Total_Metering	Power Consumed
0	1.396	0.000	241.65	5.8	18.0	-18.000000
1	0.482	0.096	247.02	2.0	0.0	1.600000
2	2.204	0.064	237.72	9.2	17.0	-15.933333
3	1.030	0.368	239.88	4.6	1.0	5.133333
4	0.146	0.000	240.83	0.6	0.0	0.000000

In [110]:
```
df1=data_1.to_json('test1.json')
```

In [64]:
```
df2=pd.read_json('test1.json')
```

In [65]:
```
df2.head()
```

Out[65]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Total_Metering	Power Consumed
0	1.396	0.000	241.65	5.8	18.0	-18.000000
1	0.482	0.096	247.02	2.0	0.0	1.600000
2	2.204	0.064	237.72	9.2	17.0	-15.933333
3	1.030	0.368	239.88	4.6	1.0	5.133333
4	0.146	0.000	240.83	0.6	0.0	0.000000

In [66]:
```
df3 = df2.to_dict(orient='records')
```

In [67]:
```
print(type(df3))
```

```
<class 'list'>
```

In [68]:
```
# collection.insert_many(df3)
```

In [69]:
```
df = pd.DataFrame(list(collection.find({})))
```


In [70]:
```
df.head()
```

Out[70]:

	_id	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Total_M
0	63654da4b29a6c139caed5ba	1.078	0.320	245.43	4.6	
1	63654da4b29a6c139caed5bb	1.786	0.250	240.64	7.4	
2	63654da4b29a6c139caed5bc	1.926	0.260	236.61	8.2	
3	63654da4b29a6c139caed5bd	1.820	0.122	236.06	7.6	
4	63654da4b29a6c139caed5be	0.334	0.114	242.92	1.4	

In [71]:
```
df.drop('_id',axis=1,inplace=True)
```

In [72]:
```
df.head()
```

Out[72]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Total_Metering	Power Consumed
0	1.078	0.320	245.43	4.6	1.0	4.333333
1	1.786	0.250	240.64	7.4	1.0	3.166667
2	1.926	0.260	236.61	8.2	0.0	4.333333
3	1.820	0.122	236.06	7.6	19.0	-16.966667
4	0.334	0.114	242.92	1.4	1.0	0.900000

In [73]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler
```

Out[73]:
```
StandardScaler()
```

In [74]:
```
X=df.drop('Power Consumed',axis=1)
```

In [75]:
```
y=df['Power Consumed']
```

In [76]:
```
X.head()
```

Out[76]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Total_Metering
0	1.078	0.320	245.43	4.6	1.0
1	1.786	0.250	240.64	7.4	1.0
2	1.926	0.260	236.61	8.2	0.0
3	1.820	0.122	236.06	7.6	19.0
4	0.334	0.114	242.92	1.4	1.0

In [77]:
```
y.head()
```

Out[77]:
```
0     4.333333
1     3.166667
2     4.333333
3   -16.966667
4     0.900000
Name: Power Consumed, dtype: float64
```


In [78]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=42)
```

In [79]:
```
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
(33500, 5) (33500,)
(16500, 5) (16500,)
```

In [80]:
```
from sklearn.linear_model import LinearRegression
regression=LinearRegression()
regression
regression.fit(X_train,y_train)
## Print the Coefficients and the intercept
print(regression.coef_)
print(regression.intercept_)
```

```
[-4.84729517e-13  1.66666667e+01 -3.81916720e-14  2.00728323e-13
 -1.00000000e+00]
1.3847589741544652e-11
```

In [81]:
```
## Prediction for the test data
regression_pred=regression.predict(X_test)
regression_pred
```

Out[81]:
```
array([  0.76666667,   0.9       , -16.36666667, ...,   2.13333333,
         1.6       ,   3.66666667])
```

In [82]:
```
# Calculating the Error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,regression_pre
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,regression_p
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,r
```

```
The Mean Squared Error for the model is 2.6119828716863157e-21
The Mean Absolute Error for the model is 2.4408202070745222e-11
The Root Mean Squared Error for the model is 5.1107561785770173e-11
```

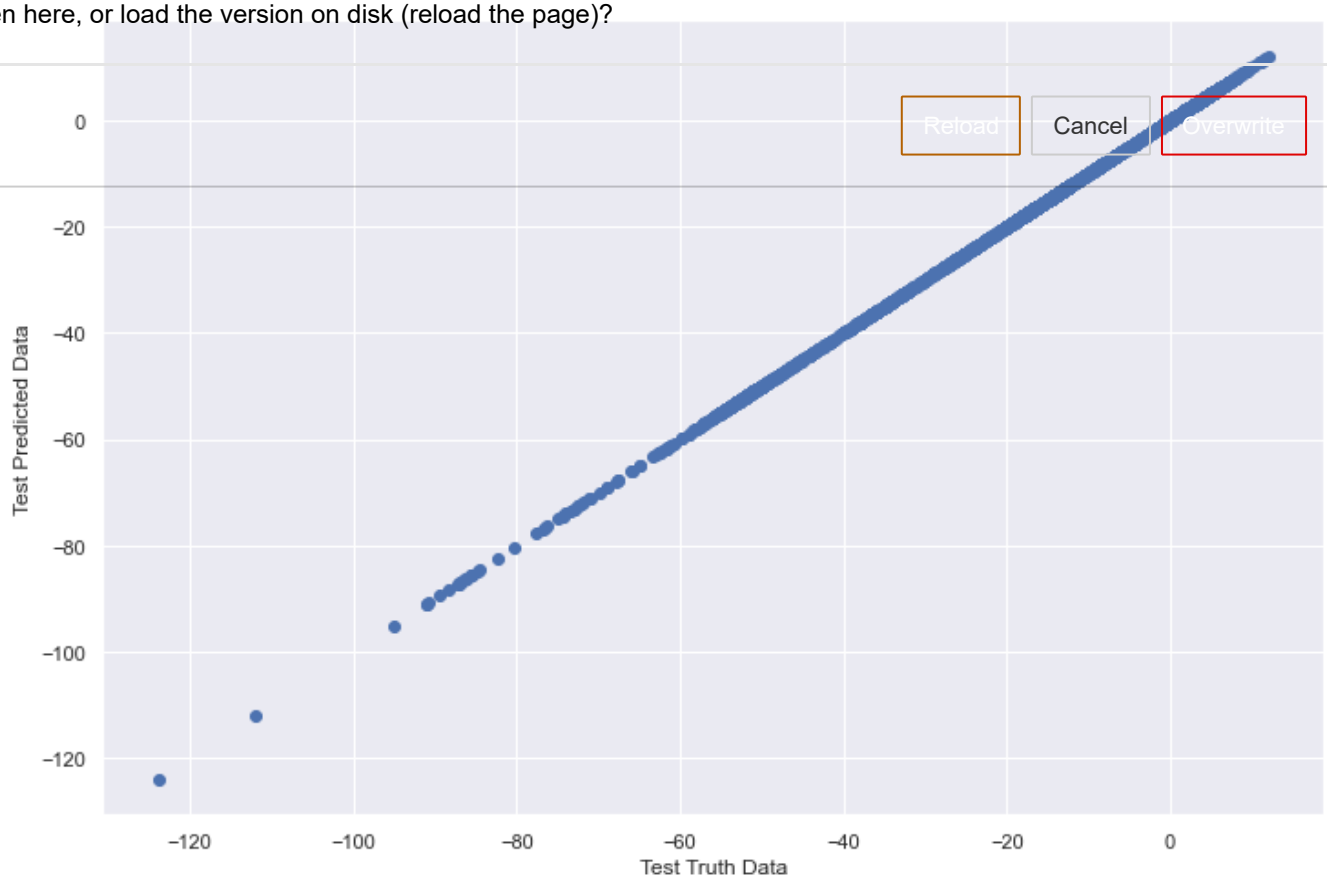
In [83]:
```
plt.scatter(y_test,regression_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[83]:
```
Text(0, 0.5, 'Test Predicted Data')
```

Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Reload Cancel Overwrite



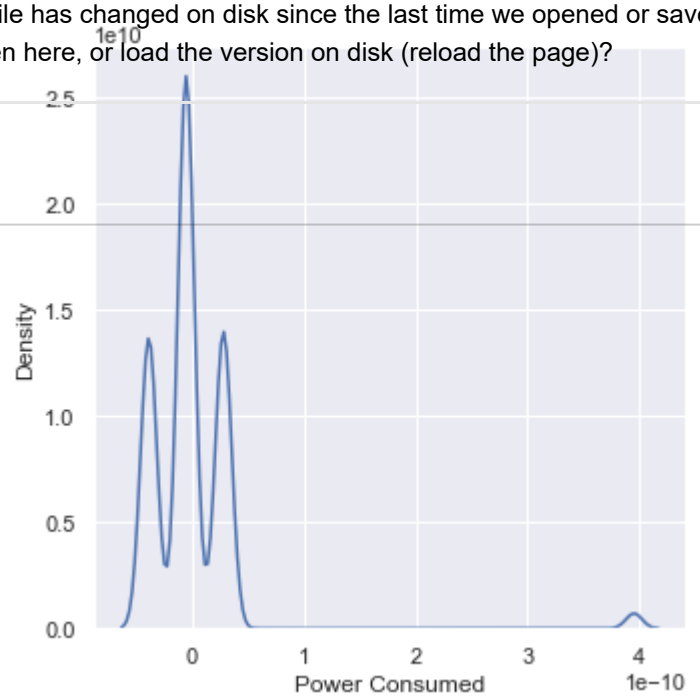
Notebook changed

The notebook file has changed on disk since the last time we opened or saved it. Do you want to overwrite the file on disk with the version open here, or load the version on disk (reload the page)?

Cancel

In [84]:
```
residuals=y_test-regression_pred
sns.displot(residuals,kind="kde")
```

Out[84]: <seaborn.axisgrid.FacetGrid at 0x24c2d328370>



In [85]:
```
# Performance Metrics of the Model
from sklearn.metrics import r2_score
score=r2_score(y_test,regression_pred)
print("The R2 Score for the model builded is",score)
```

```
The R2 Score for the model builded is 1.0
```

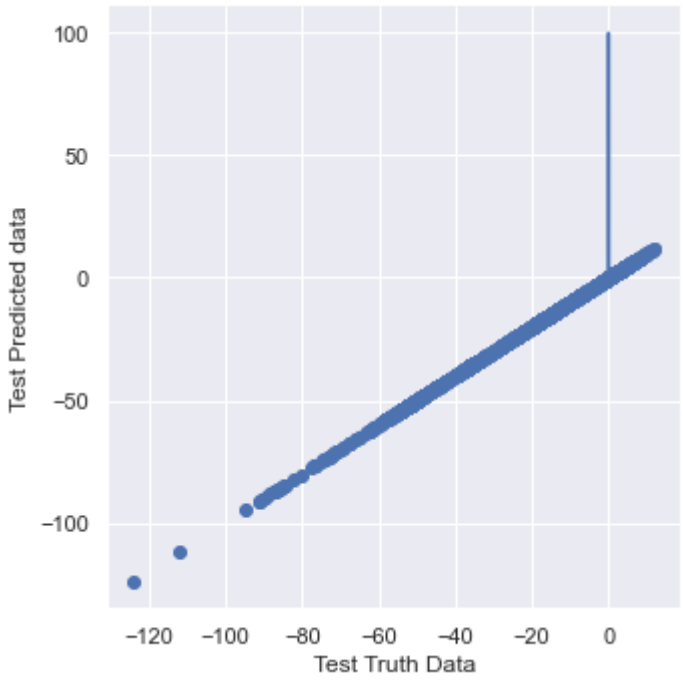
In [86]:
```
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The Adjusted R Square for the model is 1.0
```

In [87]:
```
from sklearn.linear_model import Ridge
ridge=Ridge()
```

In [88]:
```
ridge.fit(X_train,y_train)
ridge_pred=ridge.predict(X_test)
```

In [89]:
```
residual=ridge_pred-y_test
sns.displot(residual,kind='kde')
plt.scatter(y_test,ridge_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted data")
plt.show()
```



In [90]:
```
# Calculating the Error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,ridge_pred))
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,ridge_pred))
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,r
```

```
The Mean Squared Error for the model is 2.64423717500278e-05
The Mean Absolute Error for the model is 0.0038826204828712983
The Root Mean Squared Error for the model is 0.005142214673662293
```

In [91]:
```
# Performance Metrics
from sklearn.metrics import r2_score
score=r2_score(y_test,ridge_pred)
print("The R2 Score for the model builded is",score)
```

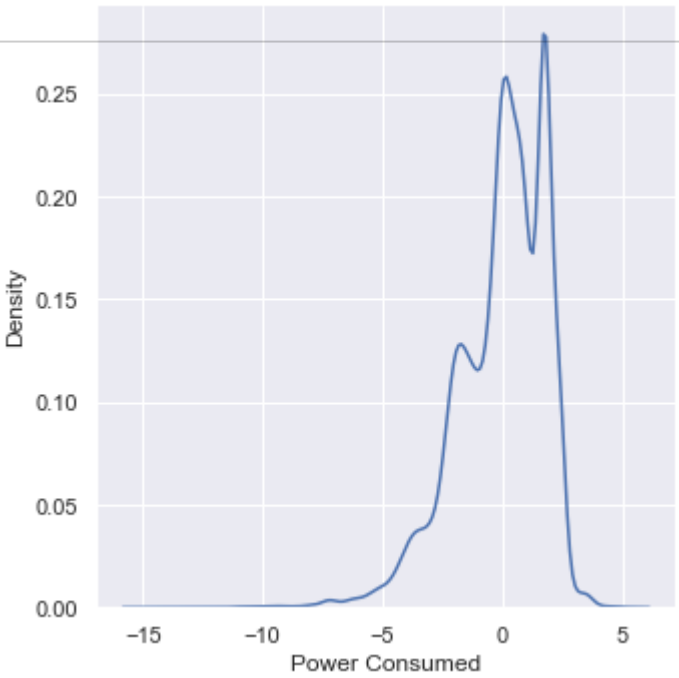
```
The R2 Score for the model builded is 0.9999998271262797
```

In [92]:
```
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The Adjusted R Square for the model is 0.9999998270738746
```

In [93]:
```
from sklearn.linear_model import Lasso
lasso=Lasso()
lasso.fit(X_train,y_train)
lasso_pred=lasso.predict(X_test)
residual=lasso_pred-y_test
sns.displot(residual,kind='kde')
```

Out[93]: <seaborn.axisgrid.FacetGrid at 0x24c2712d400>



In [94]:
```
# Calculation the Error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,lasso_pred))
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,lasso_pred))
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,
```

```
The Mean Squared Error for the model is 3.4289549294160917
The Mean Absolute Error for the model is 1.429987746272272
The Root Mean Squared Error for the model is 1.8517437537132646
```

In [95]:
```
# Performance Metrics
from sklearn.metrics import r2_score
score=r2_score(y_test,lasso_pred)
print("The R2 Score for the model builded is",score)
```

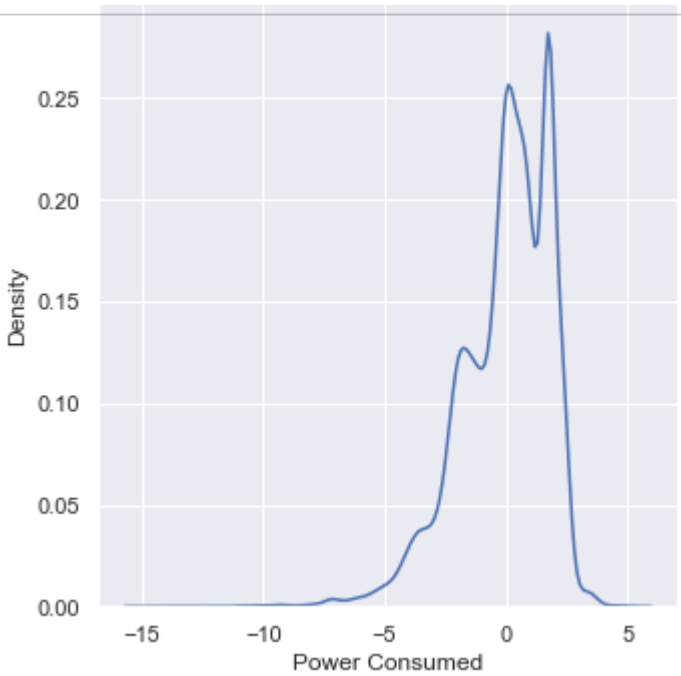
```
The R2 Score for the model builded is 0.9775823363747858
```

In [96]:
```
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The Adjusted R Square for the model is 0.9775755406722196
```

In [97]:
```
# Elastic Net Regression
from sklearn.linear_model import ElasticNet
elastic = ElasticNet(random_state=0)
elastic.fit(X_train,y_train)
elastic_pred = elastic.predict(X_test)
residual = elastic_pred-y_test
sns.displot(residual,kind='kde')
```

Out[97]: <seaborn.axisgrid.FacetGrid at 0x24c2e7545b0>



In [98]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,elastic_pred))
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,elastic_pred))
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,e
```

```
The Mean Squared Error for the model is 3.381839955838527
The Mean Absolute Error for the model is 1.4224019688954508
The Root Mean Squared Error for the model is 1.838977965022563
```

In [99]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,elastic_pred)
print("The R2 Score for the model builded is",score)
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The R2 Score for the model builded is 0.9778903624792737
The Adjusted R Square for the model is 0.977883660151906
```

In [100]:
```
from sklearn.svm import SVR
svr=SVR()
svr
```

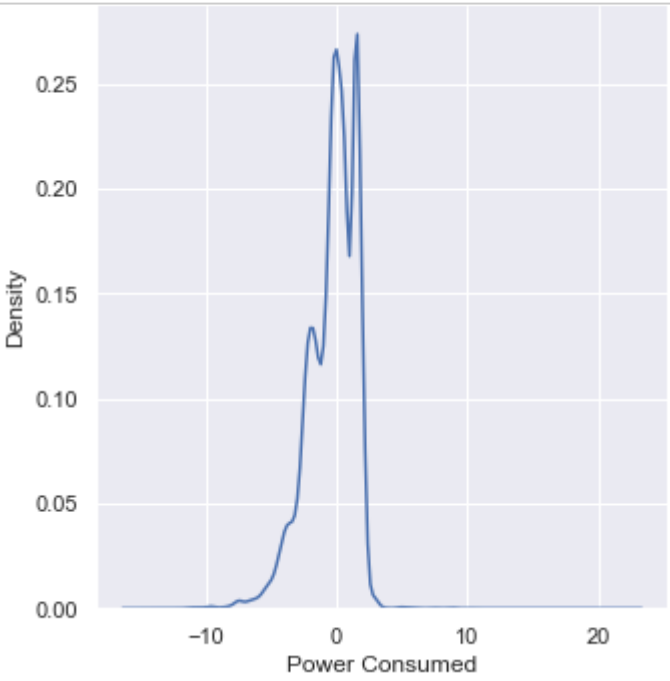
Out[100]:
```
SVR()
```

In [101]:
```
svr.fit(X_train,y_train)
```

Out[101]:
```
SVR()
```

In [102]:
```
svr_pred=svr.predict(X_test)
residual=svr_pred-y_test
sns.displot(residual,kind='kde')
```

Out[102]:
```
<seaborn.axisgrid.FacetGrid at 0x24c4e059580>
```



In [103]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,svr_pred))
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,svr_pred))
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,s
```

```
The Mean Squared Error for the model is 3.515156979994131
The Mean Absolute Error for the model is 1.39604507906014
The Root Mean Squared Error for the model is 1.874875190511126
```

In [104]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,svr_pred)
print("The R2 Score for the model builded is",score)
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The R2 Score for the model builded is 0.977018768578346
The Adjusted R Square for the model is 0.9770118020355361
```