In [1]:

```
#importing required libraries
import numpy as np
import pandas as pd
import pandas.util.testing as tm
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
```

```
<ipython-input-1-71e87fd5a599>:4: FutureWarning: pandas.util.testing is depr
ecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

```
df = pd.read_csv('household_power_consumption.txt', sep=';',
                 parse_dates={'dt' : ['Date', 'Time']}, infer_datetime_format=True,
                 low_memory=False, na_values=['nan','?'], index_col='dt')

df.shape
```

Out[2]:

```
(2075259, 7)
```

In [3]:

```
df.head()
```

Out[3]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1
dt					
2006-12-16 17:24:00	4.216	0.418	234.84	18.4	0.0
2006-12-16 17:25:00	5.360	0.436	233.63	23.0	0.0
2006-12-16 17:26:00	5.374	0.498	233.29	23.0	0.0
2006-12-16 17:27:00	5.388	0.502	233.74	23.0	0.0
2006-12-16 17:28:00	3.666	0.528	235.68	15.8	0.0

In [4]:

```
df.isnull().sum()
```

Out[4]:

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

In [5]:

```
df['Global_active_power'].fillna(df['Global_active_power'].mean(),inplace=True)
df['Global_reactive_power'].fillna(df['Global_reactive_power'].mean(),inplace=True)
df['Voltage'].fillna(df['Voltage'].mean(),inplace=True)
df['Global_intensity'].fillna(df['Global_intensity'].mean(),inplace=True)
df['Sub_metering_1'].fillna(df['Sub_metering_1'].mean(),inplace=True)
df['Sub_metering_2'].fillna(df['Sub_metering_2'].mean(),inplace=True)
df['Sub_metering_3'].fillna(df['Sub_metering_3'].mean(),inplace=True)
```

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
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 [7]:

```
sum(df.duplicated())
```

Out[7]:

```
168560
```

In [8]:

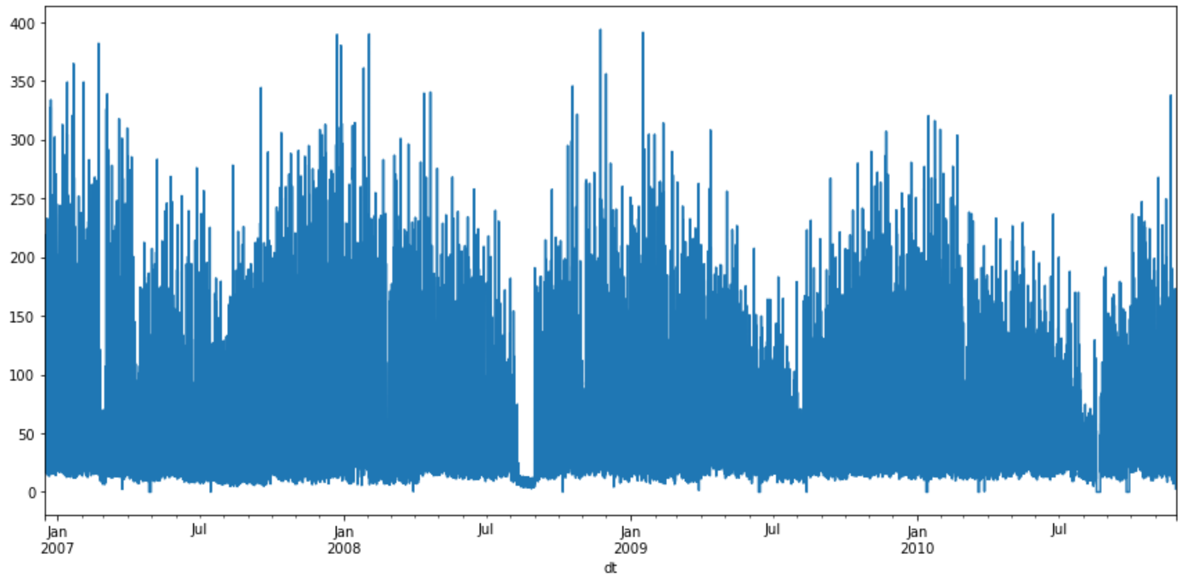
```
df.drop_duplicates(inplace=True)
sum(df.duplicated())
```

Out[8]:

```
0
```

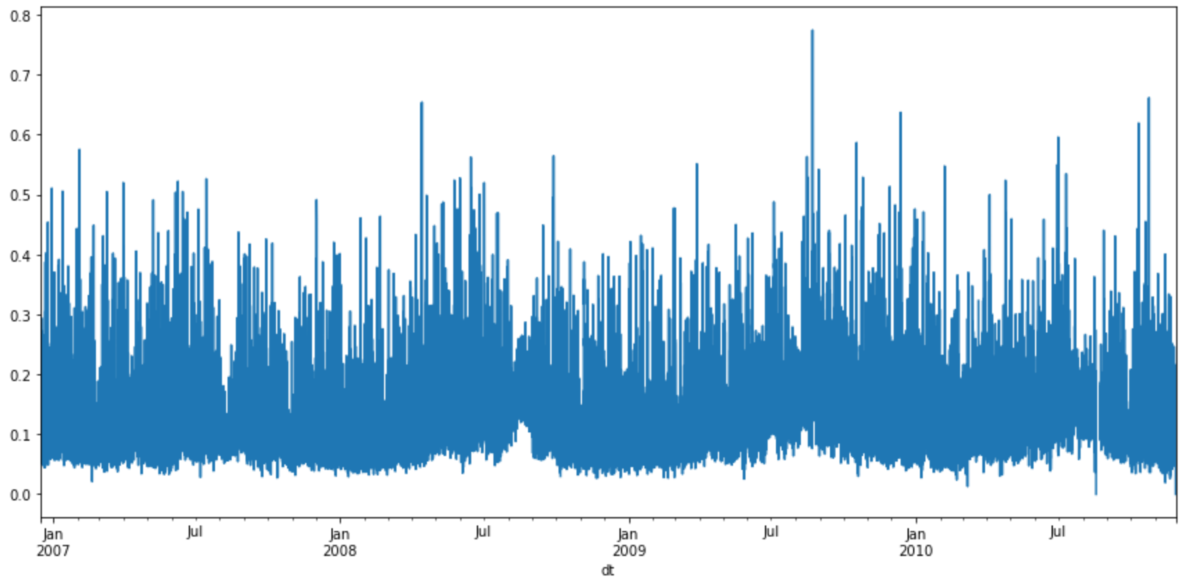
In [9]:

```
#resampling the data for every hour
plt.figure(figsize=(12,6))
df.Global_active_power.resample('1H').sum().plot()  #sum() gives the sum of global_active_p
plt.tight_layout()
```



In [10]:

```
plt.figure(figsize=(12,6))
df.Global_reactive_power.resample('1H').mean().plot()  #mean() gives the sum of global_acti
plt.tight_layout()
```

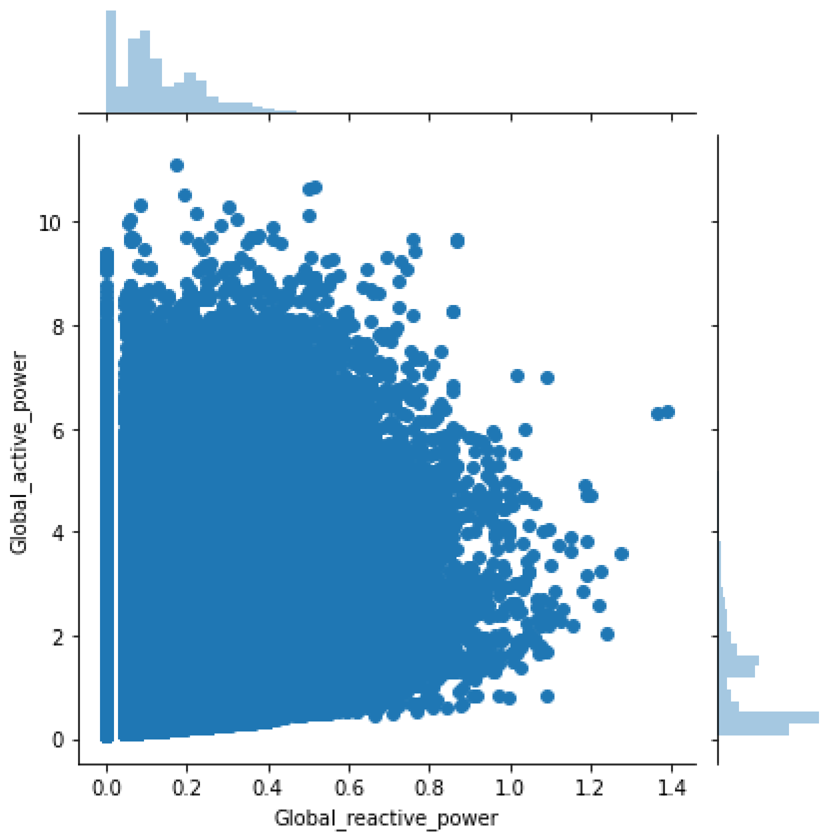


In [11]:

```
#scatter charts
sns.jointplot(x='Global_reactive_power',y='Global_active_power',data=df,kind='scatter')
```

Out[11]:

```
<seaborn.axisgrid.JointGrid at 0x1eb22c6ee80>
```

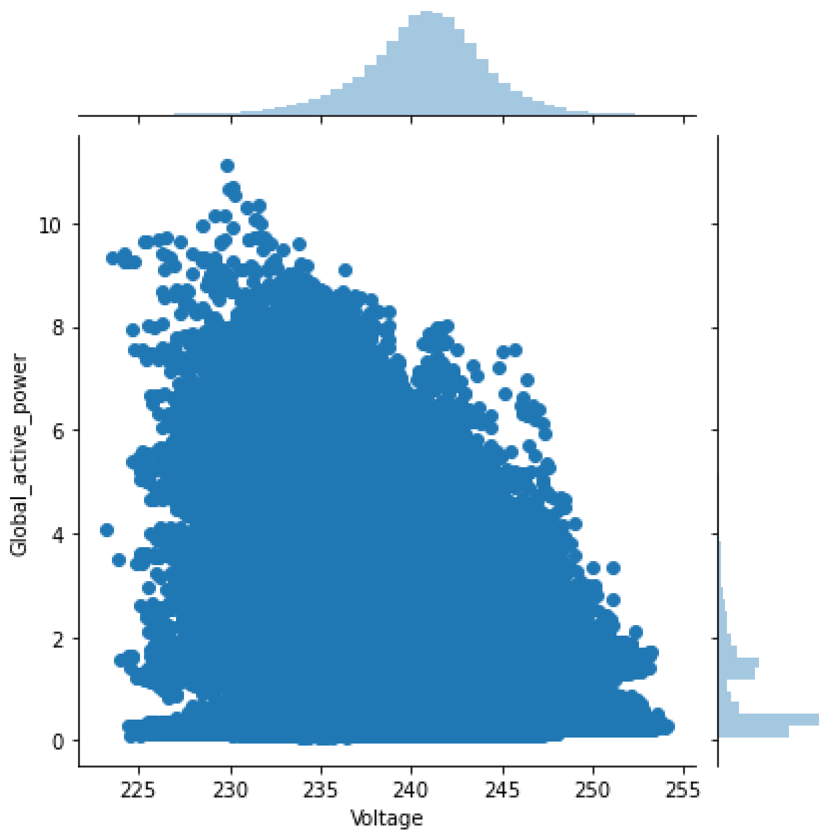


In [12]:

```
sns.jointplot(x='Voltage',y='Global_active_power',data=df,kind='scatter')
```

Out[12]:

```
<seaborn.axisgrid.JointGrid at 0x1eb23232130>
```

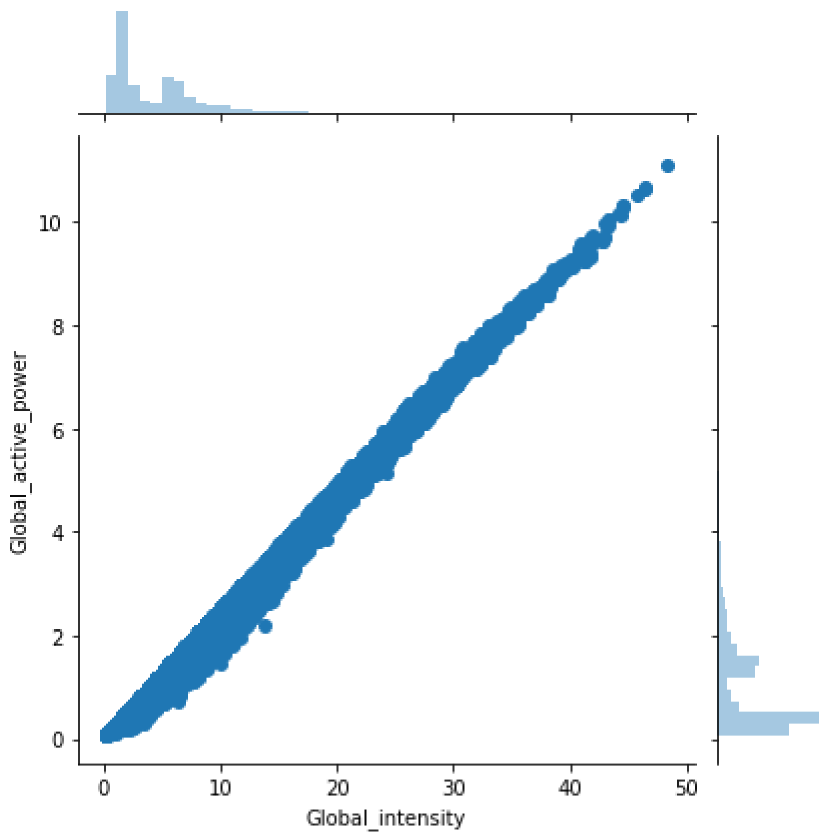


In [13]:

```
sns.jointplot(x='Global_intensity',y='Global_active_power',data=df,kind='scatter')
```

Out[13]:

```
<seaborn.axisgrid.JointGrid at 0x1eb232c3b80>
```

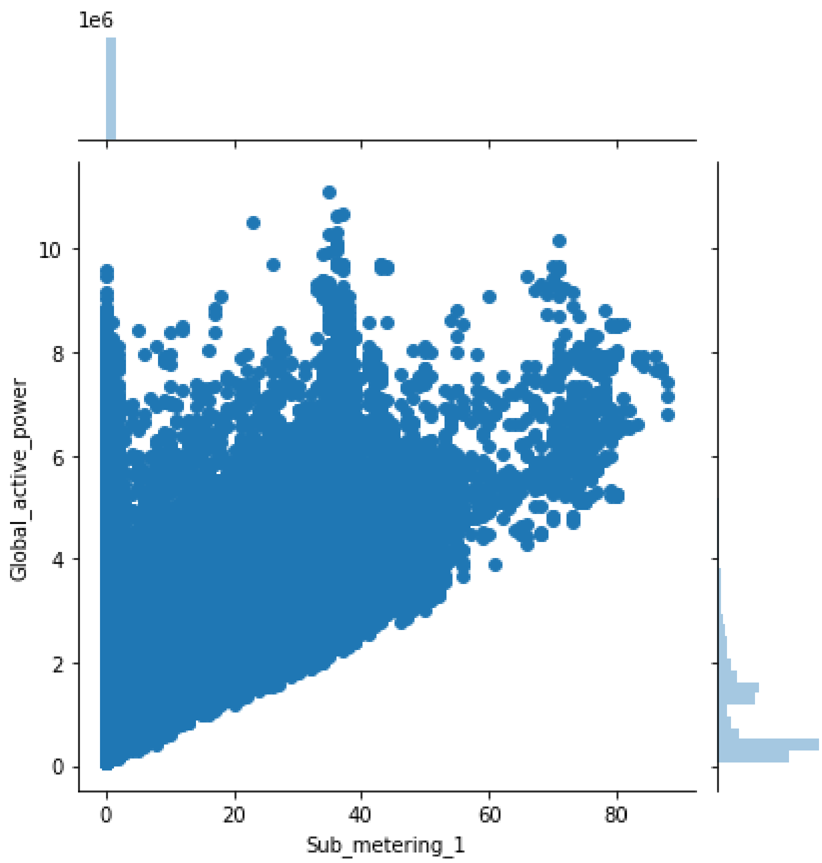


In [14]:

```
sns.jointplot(x='Sub_metering_1',y='Global_active_power',data=df,kind='scatter')
```

Out[14]:

```
<seaborn.axisgrid.JointGrid at 0x1eb23391400>
```

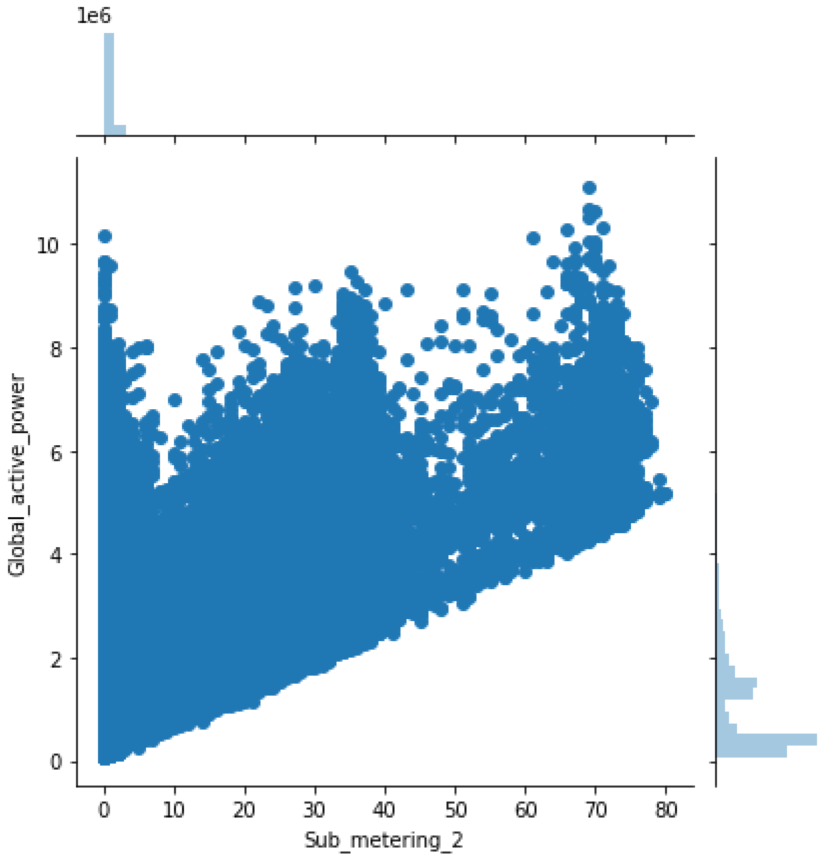


In [15]:

```
sns.jointplot(x='Sub_metering_2',y='Global_active_power',data=df,kind='scatter')
```

Out[15]:

```
<seaborn.axisgrid.JointGrid at 0x1eb22f14af0>
```

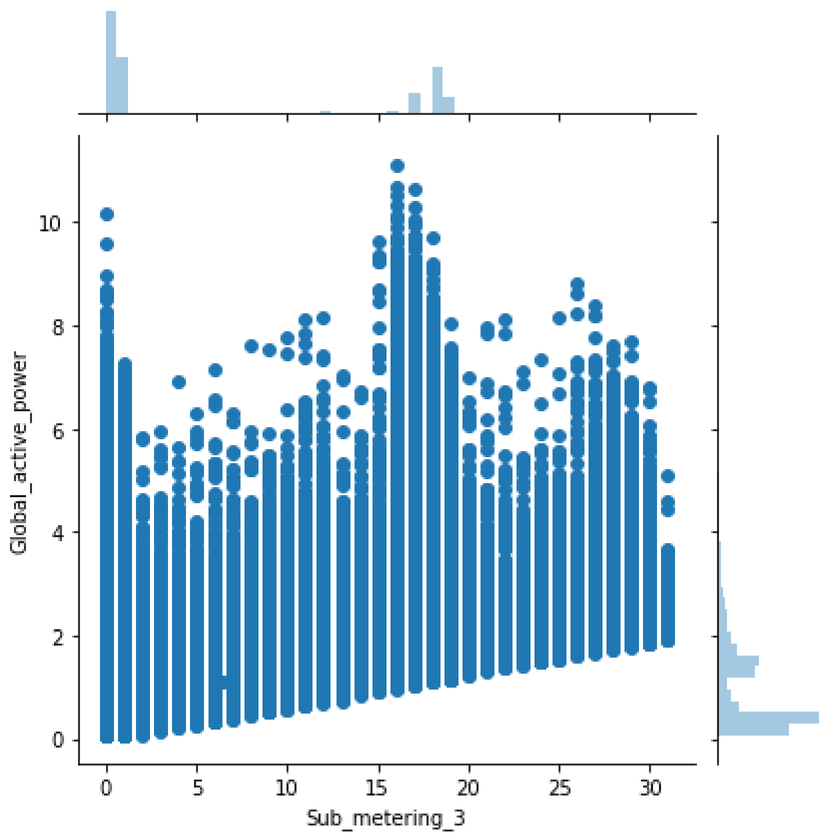


In [16]:

```
sns.jointplot(x='Sub_metering_3',y='Global_active_power',data=df,kind='scatter')
```

Out[16]:

```
<seaborn.axisgrid.JointGrid at 0x1eb230e09a0>
```

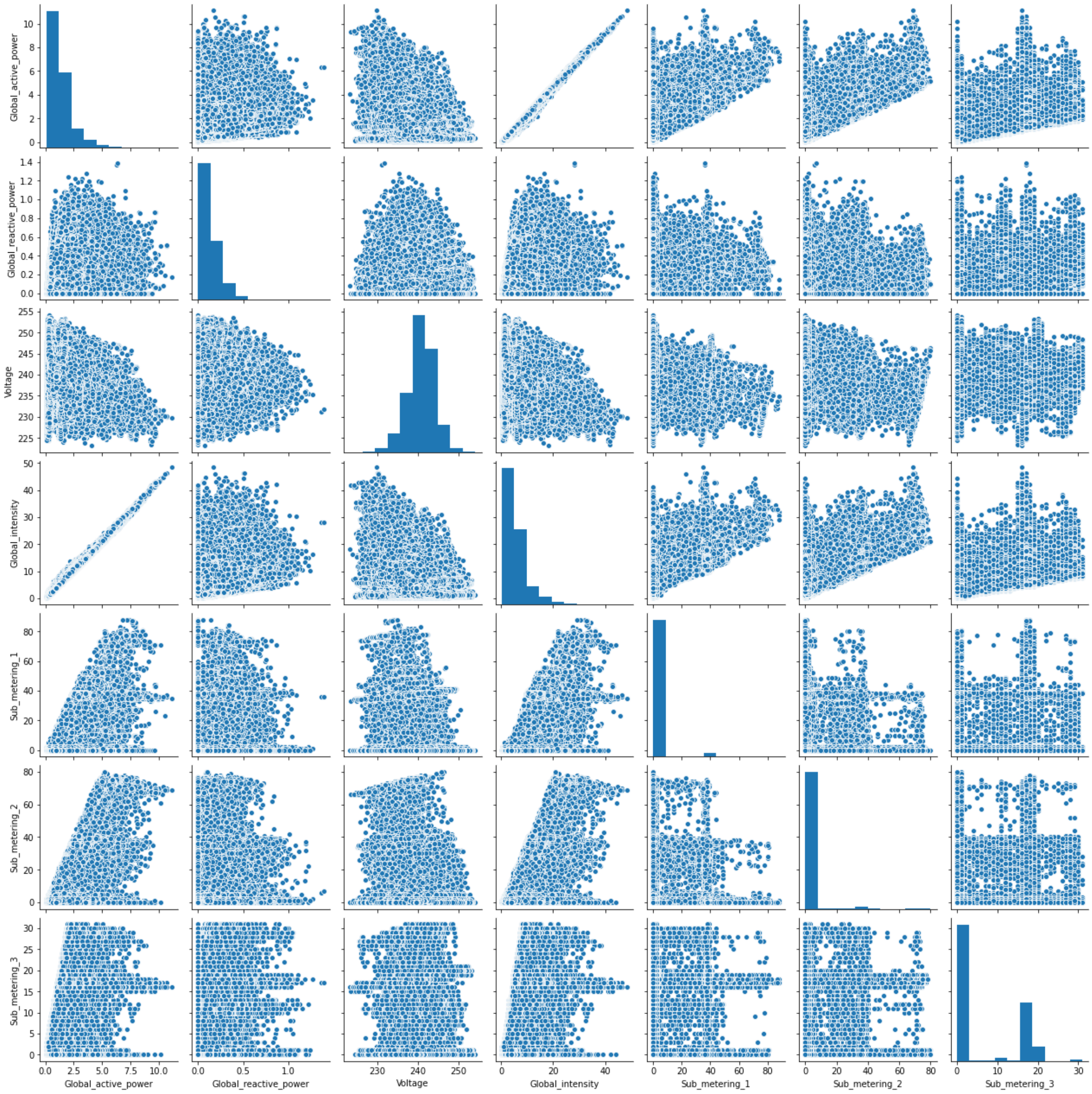


In [17]:

```
#pairwise relationship across the whole dataset
sns.pairplot(df)
```

Out[17]:

```
<seaborn.axisgrid.PairGrid at 0x1eb22f73c40>
```



In [19]:

```
X=df.drop('Global_active_power',axis=1)
y=df['Global_active_power']
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=101)
```

In [20]:

```
#feature scaling
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
X_train2 = pd.DataFrame(scaler.fit_transform(X_train))
X_test2 = pd.DataFrame(scaler.transform(X_test))
X_train2.columns = X_train.columns.values
X_test2.columns = X_test.columns.values
X_train2.index = X_train.index.values
X_test2.index = X_test.index.values
X_train = X_train2
X_test = X_test2
```

In [21]:

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train, y_train)
```

Out[21]:

```
LinearRegression()
```

In [22]:

```
predictions_linear = lm.predict(X_test)
```

In [23]:

```
lm.intercept_
```

Out[23]:

```
-0.03587199119110562
```

In [24]:

```
lm.coef_
```

Out[24]:

```
array([-0.24481595,  0.14365636, 11.47713272, -0.02944447, -0.03632183,
        0.06742105])
```

In [25]:

```
from sklearn import metrics
print('Results of Linear Regression:\n')
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,predictions_linear))
print('Mean Squared Error:',metrics.mean_squared_error(y_test,predictions_linear))
print('Root Mean Squared Error:',np.sqrt(metrics.mean_squared_error(y_test,predictions_line
```

```
Results of Linear Regression:

Mean Absolute Error: 0.026567798555369868
Mean Squared Error: 0.001739136603590701
Root Mean Squared Error: 0.04170295677276014
```

In [26]:

```
df_result = pd.DataFrame({'Actual': y_test, 'Predicted': predictions_linear})
df_result
```

Out[26]:

	Actual	Predicted
dt		
2008-04-06 13:48:00	1.520	1.531844
2009-07-20 00:06:00	0.222	0.227919
2009-07-07 15:11:00	1.058	1.059497
2010-06-04 19:44:00	0.346	0.351732
2008-07-27 22:01:00	2.392	2.401320
...	...	...
2007-09-07 18:53:00	0.490	0.541664
2006-12-29 13:53:00	1.704	1.682286
2010-08-31 02:37:00	0.226	0.235857
2010-01-27 08:40:00	1.444	1.472283
2009-06-04 17:13:00	0.238	0.215650

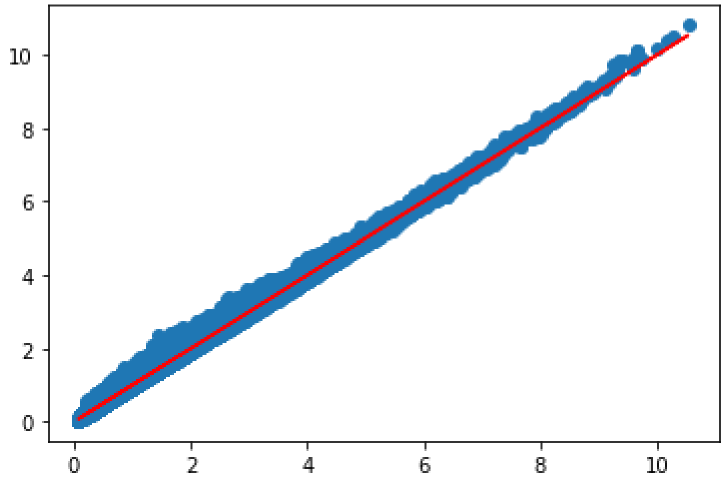
629211 rows × 2 columns

In [27]:

```
plt.scatter(df_result['Actual'],df_result['Predicted'])
plt.plot(y_test,y_test,'r')
```

Out[27]:

```
[<matplotlib.lines.Line2D at 0x1eb4f0b8730>]
```



In [1]:

```
class Priority:

    def processData(self, no_of_processes):
        process_data = []
        for i in range(no_of_processes):
            temporary = []
            process_id = int(input("Enter sub_metering ID: "))
            burst_time = int(input(f"Enter Time duration for Process {process_id}: "))
            priority = int(input(f"Enter Priority  {process_id}: "))
            temporary.extend([process_id, 0, burst_time, priority, 0, burst_time])
            '''
            '0' is the state of the process. 0 means not executed and 1 means execution com
            '''
            process_data.append(temporary)
        Priority.schedulingProcess(self, process_data)

    def schedulingProcess(self, process_data):
        start_time = []
        exit_time = []
        s_time = 0
        sequence_of_process = []
        while 1:
            ready_queue = []
            temp = []
            for i in range(len(process_data)):
                if process_data[i][1] <= s_time and process_data[i][4] == 0:
                    temp.extend([process_data[i][0], process_data[i][1], process_data[i][2]
                                 process_data[i][5]])
                    ready_queue.append(temp)
                    temp = []
            if len(ready_queue) == 0:
                break
            if len(ready_queue) != 0:
                ready_queue.sort(key=lambda x: x[3], reverse=True)
                start_time.append(s_time)
                s_time = s_time + 1
                e_time = s_time
                exit_time.append(e_time)
                sequence_of_process.append(ready_queue[0][0])
                for k in range(len(process_data)):
                    if process_data[k][0] == ready_queue[0][0]:
                        break
                process_data[k][2] = process_data[k][2] - 1
                if process_data[k][2] == 0:
                    process_data[k][4] = 1
                    process_data[k].append(e_time)
        t_time = Priority.calculateTurnaroundTime(self, process_data)
        w_time = Priority.calculateWaitingTime(self, process_data)
        Priority.printData(self, process_data, t_time, w_time, sequence_of_process)

    def calculateTurnaroundTime(self, process_data):
        total_turnaround_time = 0
        for i in range(len(process_data)):
            turnaround_time = process_data[i][6] - process_data[i][1]
            '''
            turnaround_time = completion_time - arrival_time
            '''
            total_turnaround_time = total_turnaround_time + turnaround_time
            process_data[i].append(turnaround_time)
```

```
        average_turnaround_time = total_turnaround_time / len(process_data)
        '''
        average_turnaround_time = total_turnaround_time / no_of_processes
        '''
        return average_turnaround_time

    def calculateWaitingTime(self, process_data):
        total_waiting_time = 0
        for i in range(len(process_data)):
            waiting_time = process_data[i][6] - process_data[i][5]
            '''
            waiting_time = turnaround_time - burst_time
            '''
            total_waiting_time = total_waiting_time + waiting_time
            process_data[i].append(waiting_time)
        average_waiting_time = total_waiting_time / len(process_data)
        '''
        average_waiting_time = total_waiting_time / no_of_processes
        '''
        return average_waiting_time

    def printData(self, process_data, average_turnaround_time, average_waiting_time, sequen
        process_data.sort(key=lambda x: x[0])
        '''
        Sort processes according to the Process ID
        '''
        print("Process_ID  Arrival_Time Time_duration   Priority        Completed  Orig_Tim
        for i in range(len(process_data)):
            for j in range(len(process_data[i])):
                print(process_data[i][j], end="\t\t\t\t")
            print()
        print(f'Average Turnaround Time: {average_turnaround_time}')
        print(f'Average Waiting Time: {average_waiting_time}')
        print(f'Sequence of Process: {sequence_of_process}')


if __name__ == "__main__":
    no_of_processes = int(input("Enter number of sub_meterings: "))
    priority = Priority()
    priority.processData(no_of_processes)
```

```
Enter number of sub_meterings: 3
Enter sub_metering ID: 01
Enter Time duration for Process 1: 1
Enter Priority  1: 3
Enter sub_metering ID: 02
Enter Time duration for Process 2: 2
Enter Priority  2: 2
Enter sub_metering ID: 03
Enter Time duration for Process 3: 4
Enter Priority  3: 1
Process_ID  Arrival_Time Time_duration   Priority        Completed  Orig_T
ime_duration Completion_Time  Turnaround_Time  Waiting_Time
1                               0                               0
3                               1                               1
1                               1                               0
2                               0                               0
2                               1                               2
```

```
3                               3                               1
3                               0                               0
1                               1                               4
7                               7                               3
Average Turnaround Time: 3.6666666666666665
Average Waiting Time: 1.3333333333333333
Sequence of Process: [1, 2, 2, 3, 3, 3, 3]
```