```
In [2]:
# Import relevant libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
import os
from datetime import datetime
from scipy.stats import normaltest
%matplotlib inline
```

```
In [3]:
# Read data
state_list = ["MP","Rajasthan1","Tamil Nadu","Andhra Pradesh"]
data = {}
for state in state_list:
    data[state] = []

def last_8_char(x):
  return x[-8:]
for state in state_list:
    for file in sorted(os.listdir(state), key=last_8_char):
        data_temp = pd.read_csv(state+'/'+file, skiprows=2,
                    parse_dates={'date_time': ['Year','Month','Day','Hour']},
                    date_parser=lambda x: datetime.strptime(x, '%Y %m %d %H'))
        data_temp = data_temp[(data_temp['date_time'].dt.hour >=8) & (data_temp['date_time'].dt.hour <=16)]
        # Taking only 4 evenly spaced data points, uncomment above line and comment line below to take all data
        # data_temp = data_temp[data_temp['date_time'].dt.hour.isin([9,11,13,15])]

        data[state].extend(data_temp.values)
```

```
In [4]:
col_name = ['date_time','Minute','DHI','DNI','GHI','Clearsky DHI','Clearsky DNI','Clearsky GHI','Dew Point','Te
MP_data = pd.DataFrame(data['MP'],columns=col_name)
# MP_data = MP_data.drop(labels='NAN', axis=1)
MP_data = MP_data[MP_data['GHI'].notna()]
MP_data.name = 'MP'
MP_data.head(10)
```

Out[4]:

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	Relative Humidity	Solar Zenith Angle	Snow depth	Wind Speed
0	2000-01-01 08:00:00	0	149	375	268	121	465	268	9	16.043264	969.206482	65.736004	71.561409	0	1.404870
1	2000-01-01 09:00:00	0	162	611	455	160	617	456	9	20.477919	969.256287	48.926508	61.406733	0	1.177964
2	2000-01-01 10:00:00	0	183	698	600	183	698	600	9	23.532446	969.059021	41.357676	53.319769	0	0.574517
3	2000-01-01 11:00:00	0	194	735	682	194	735	682	9	25.100303	968.150452	37.809076	48.357745	0	0.293565
4	2000-01-01 12:00:00	0	195	740	695	195	740	695	9	25.644234	967.158325	36.318170	47.514574	0	0.457926
5	2000-01-01 13:00:00	0	188	714	637	188	714	637	9	25.414283	966.396484	36.261497	50.995718	0	0.626590
6	2000-01-01 14:00:00	0	170	650	515	170	650	515	9	24.456031	965.862549	37.480632	58.025734	0	0.898525
7	2000-01-01 15:00:00	0	138	530	341	138	530	341	9	22.709661	966.012390	42.444601	67.500131	0	1.375460
8	2000-01-01 16:00:00	0	94	205	135	82	307	143	10	19.482297	966.293945	55.230587	78.532527	0	2.145357
9	2000-01-02 08:00:00	0	148	374	267	121	468	269	8	16.042273	970.506714	62.628500	71.598231	0	1.116331

```
In [5]:
MP_data.shape
```

Out[5]: (49275, 16)

```
In [6]:
TamilNadu_data = pd.DataFrame(data['Tamil Nadu'],columns=col_name)
# TamilNadu_data = TamilNadu_data.drop(labels='NAN', axis=1)
TamilNadu_data = TamilNadu_data[TamilNadu_data['GHI'].notna()]
TamilNadu_data.name = 'Tamil Nadu'
TamilNadu_data.head()
```

Out[6]:

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	Relative Humidity	Solar Zenith Angle	Snow depth	Wind Speed
0	2000-01-01 08:00:00	0	147	468	341	117	611	370	17	22.268241	953.526489	73.018861	65.523141	0	0.716501
1	2000-01-01 09:00:00	0	184	587	535	143	737	584	17	23.986125	953.256287	66.480553	53.289244	0	0.776021
2	2000-01-01 10:00:00	0	228	630	692	159	804	751	17	25.217761	952.739014	61.667143	42.592118	0	0.819869
3	2000-01-01 11:00:00	0	177	821	851	167	837	854	17	25.983396	951.830444	59.480310	34.901158	0	0.824011
4	2000-01-01 12:00:00	0	171	816	860	169	845	882	17	26.406886	950.518311	58.353354	32.445055	0	0.811820

```
In [7]:
TamilNadu_data.shape
```

Out[7]: (49275, 16)

```
In [8]:
Rajasthan_data = pd.DataFrame(data['Rajasthan1'],columns=col_name)
# Rajasthan_data = Rajasthan_data.drop(labels='NAN', axis=1)
Rajasthan_data = Rajasthan_data[Rajasthan_data['GHI'].notna()]
Rajasthan_data.name = 'Rajasthan'
Rajasthan_data.head()
```

Out[8]:

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	Relative Humidity	Solar Zenith Angle	Snow depth	Wind Speed
0	2000-01-01 08:00:00	0	78	306	135	74	354	139	-8	14.277325	986.486450	20.120814	79.450627	0	3.323988
1	2000-01-01 09:00:00	0	114	597	331	121	600	339	-6	17.677278	986.856323	18.365240	68.755938	0	2.818877
2	2000-01-01 10:00:00	0	144	681	488	147	721	512	-4	21.872848	986.979065	16.310936	59.670536	0	2.604525
3	2000-01-01 11:00:00	0	151	759	608	162	782	633	-2	26.036963	986.390442	14.891855	53.040145	0	2.763721
4	2000-01-01 12:00:00	0	156	788	664	168	805	687	-2	28.260238	985.398315	13.308055	49.861770	0	2.677435

```
In [9]:
Rajasthan_data.shape
```

Out[9]: (49275, 16)

```
In [10]:
col_name = ['date_time','Minute','DHI','DNI','GHI','Clearsky DHI','Clearsky DNI','Clearsky GHI','Dew Point','Te
AP_data = pd.DataFrame(data['Andhra Pradesh'],columns=col_name)
# AP_data = AP_data.drop(labels='NAN', axis=1)
AP_data = AP_data[AP_data['GHI'].notna()]
AP_data.name = 'Andhra Pradesh'
AP_data.head()
```

Out[10]:

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	Relative Humidity	Solar Zenith Angle	Wind Speed	NAN
0	2000-01-01 08:00:00	0	110	602	333	110	602	333	14	18.137191	942.006470	80.201618	68.258226	2.482057	NaN
1	2000-01-01 09:00:00	0	138	740	547	138	740	547	14	21.422234	941.736328	66.567218	56.554706	2.902531	NaN
2	2000-01-01 10:00:00	0	155	812	714	155	812	714	14	23.382713	941.219055	58.555843	46.546520	2.866336	NaN
3	2000-01-01 11:00:00	0	164	847	818	164	847	818	14	24.719346	940.310425	53.863563	39.556979	2.829693	NaN
4	2000-01-01 12:00:00	0	261	674	797	167	855	847	14	25.476261	938.998352	51.624979	37.330650	2.829407	NaN

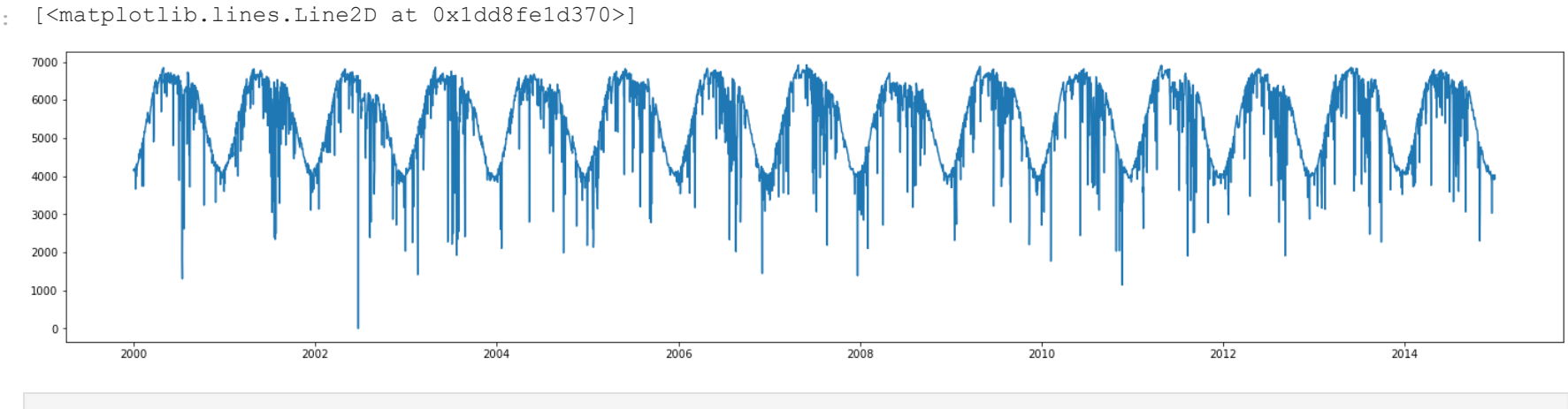
```
In [11]:
AP_data.shape
```

Out[11]: (49275, 15)

Time series Analysis

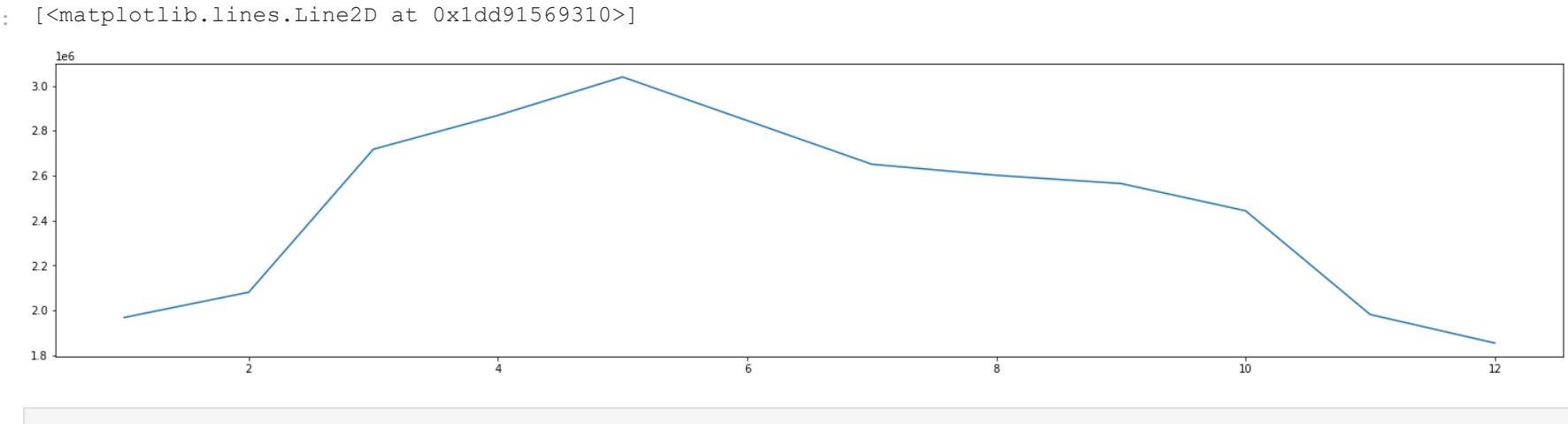
```
In [17]:
# Aggregate data to daily values
data = [Rajasthan_data['date_time'],Rajasthan_data['GHI']]
headers = ["date_time","GHI"]
temp_df = pd.concat(data,axis=1,keys=headers)
daily_RJ = temp_df.groupby(temp_df['date_time'].dt.date).agg({'GHI': 'sum'}).reset_index()
daily_RJ.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

# daily_RJ['GHI'][903] = (daily_RJ['GHI'][902]+daily_RJ['GHI'][904])/2
plt.plot(daily_RJ['date_time'],daily_RJ['GHI'])
```

Out[17]: [<matplotlib.lines.Line2D at 0x1dd8fe1d370>]



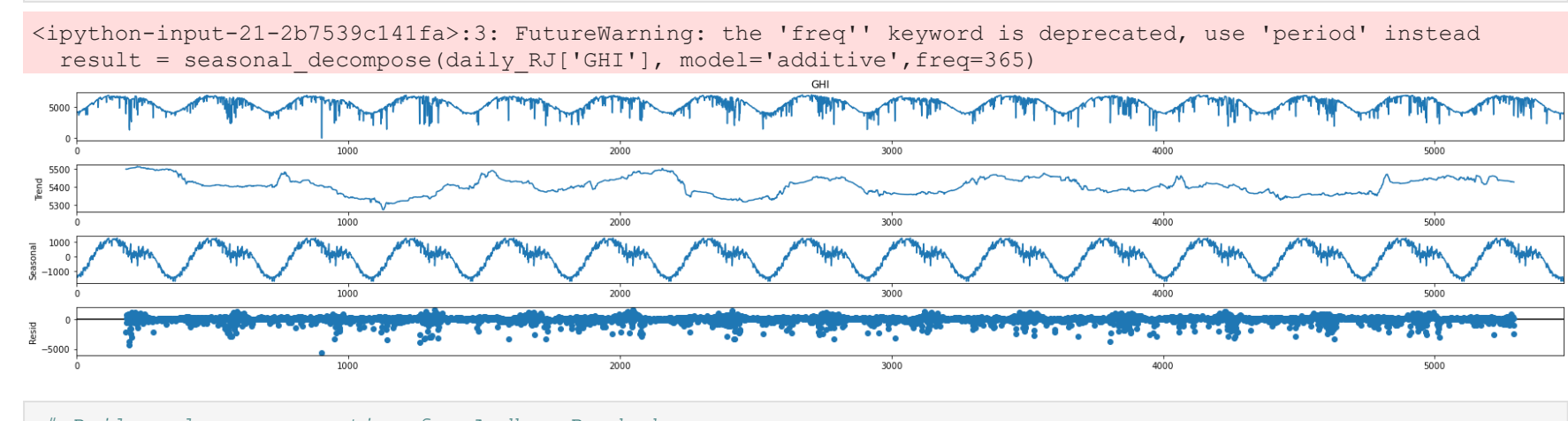
```
In [20]:
# Aggregate data to monthly values
monthly_RJ = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_index()
monthly_RJ.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

plt.plot(monthly_RJ['date_time'],monthly_RJ['GHI'])
```

Out[20]: [<matplotlib.lines.Line2D at 0x1dd91569310>]



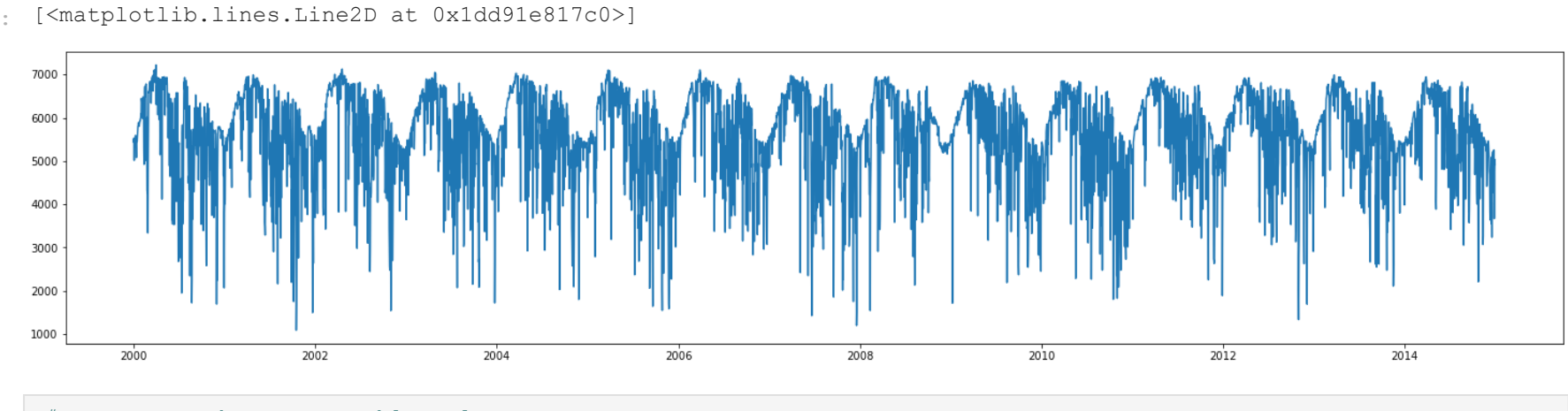
```
In [21]:
# Perform Time series decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(daily_RJ['GHI'], model='additive',freq=365)
result.plot()
plt.show()
```

```
<ipython-input-21-2b7539c141fa>:3: FutureWarning: the 'freq'' keyword is deprecated, use 'period' instead
  result = seasonal_decompose(daily_RJ['GHI'], model='additive',freq=365)
```



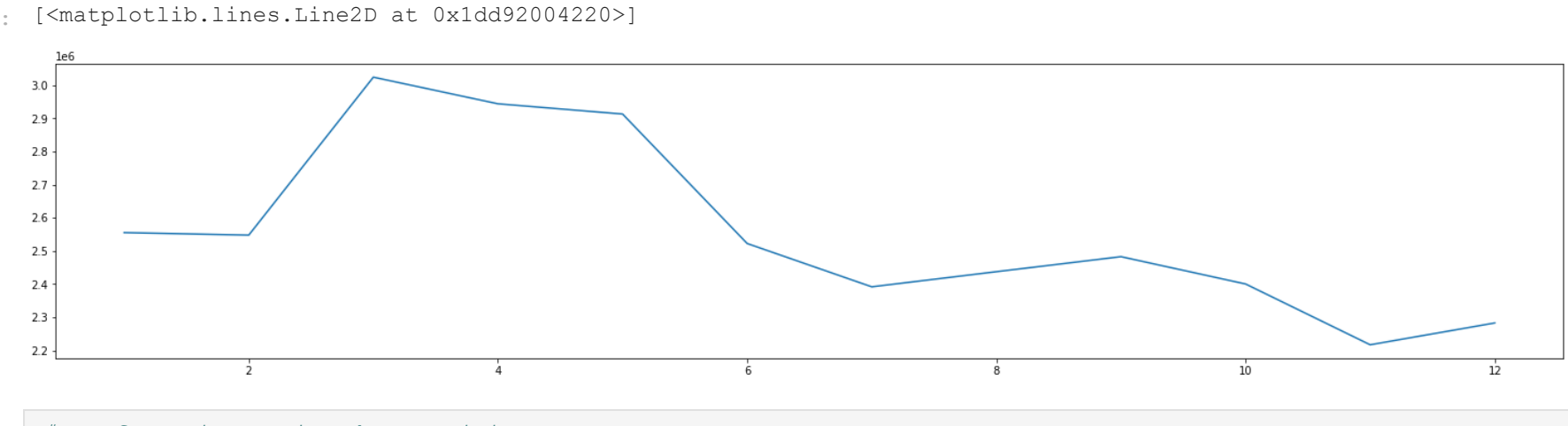
```
In [22]:
# Daily values aggregation for Andhra Pradesh
data = [AP_data['date_time'],AP_data['GHI']]
headers = ["date_time","GHI"]
temp_df = pd.concat(data,axis=1,keys=headers)
daily_AP = temp_df.groupby(temp_df['date_time'].dt.date).agg({'GHI': 'sum'}).reset_index()
daily_AP.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

plt.plot(daily_AP['date_time'],daily_AP['GHI'])
```

Out[22]: [<matplotlib.lines.Line2D at 0x1dd91e817c0>]



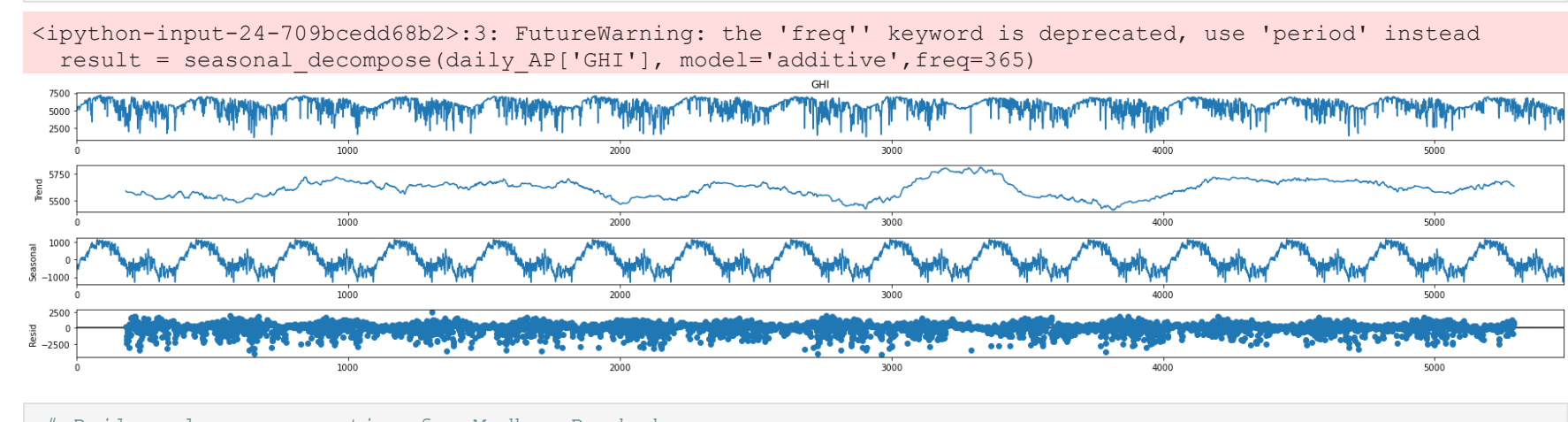
```
In [23]:
# Aggregate data to monthly values
monthly_AP = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_index()
monthly_AP.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

plt.plot(monthly_AP['date_time'],monthly_AP['GHI'])
```

Out[23]: [<matplotlib.lines.Line2D at 0x1dd92004220>]



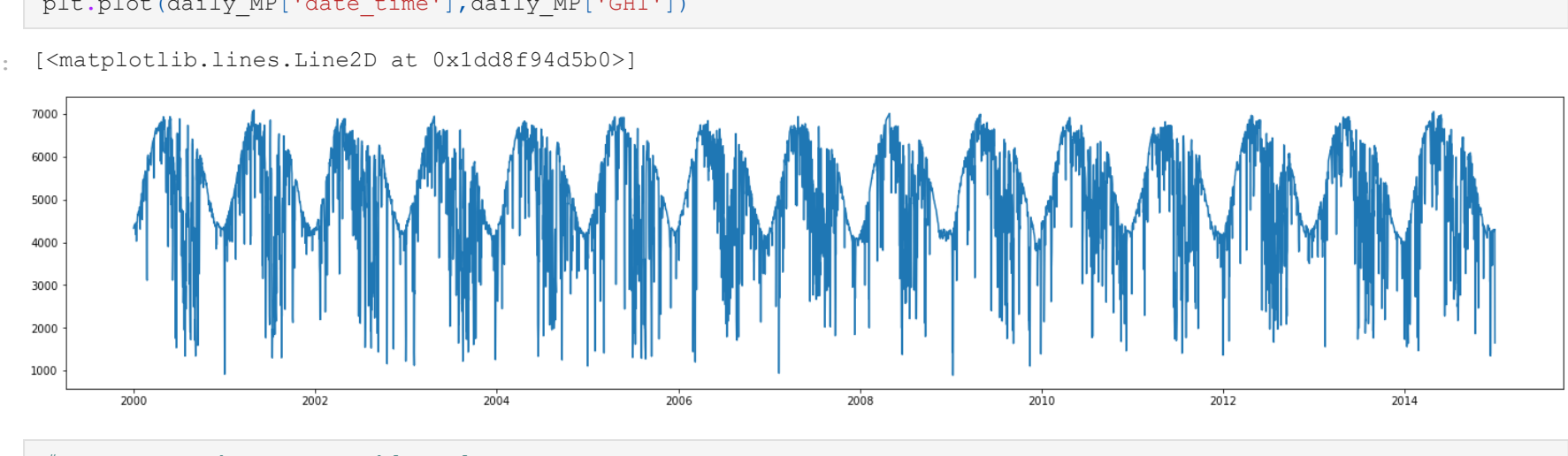
```
In [24]:
# Perform Time series decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(daily_AP['GHI'], model='additive',freq=365)
result.plot()
plt.show()
```

```
<ipython-input-24-709bcedd68b2>:3: FutureWarning: the 'freq'' keyword is deprecated, use 'period' instead
  result = seasonal_decompose(daily_AP['GHI'], model='additive',freq=365)
```



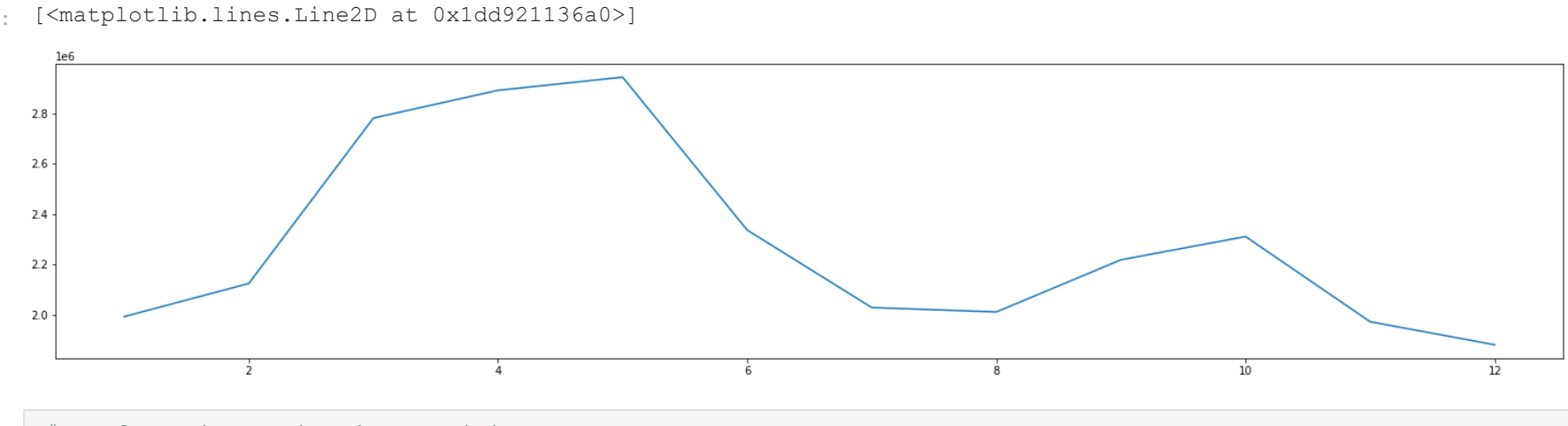
```
In [25]:
# Daily values aggregation for Madhya Pradesh
data = [MP_data['date_time'],MP_data['GHI']]
headers = ["date_time","GHI"]
temp_df = pd.concat(data,axis=1,keys=headers)
daily_MP = temp_df.groupby(temp_df['date_time'].dt.date).agg({'GHI': 'sum'}).reset_index()
daily_MP.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

plt.plot(daily_MP['date_time'],daily_MP['GHI'])
```

Out[25]: [<matplotlib.lines.Line2D at 0x1dd8f94d5b0>]



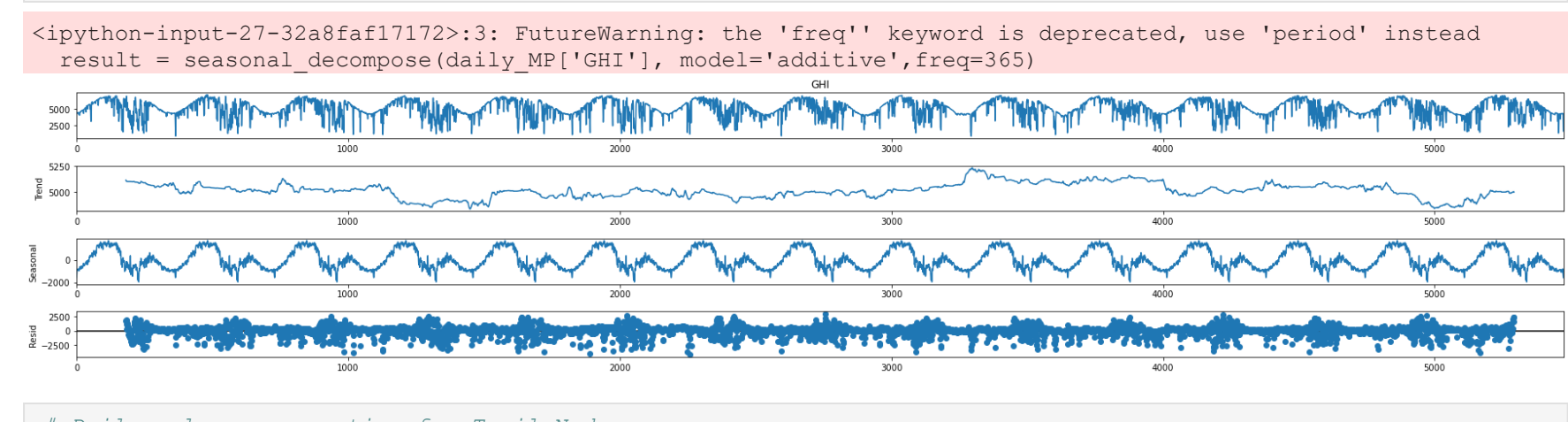
```
In [26]:
# Aggregate data to monthly values
monthly_MP = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_index()
monthly_MP.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

plt.plot(monthly_MP['date_time'],monthly_MP['GHI'])
```

Out[26]: [<matplotlib.lines.Line2D at 0x1dd921136a0>]



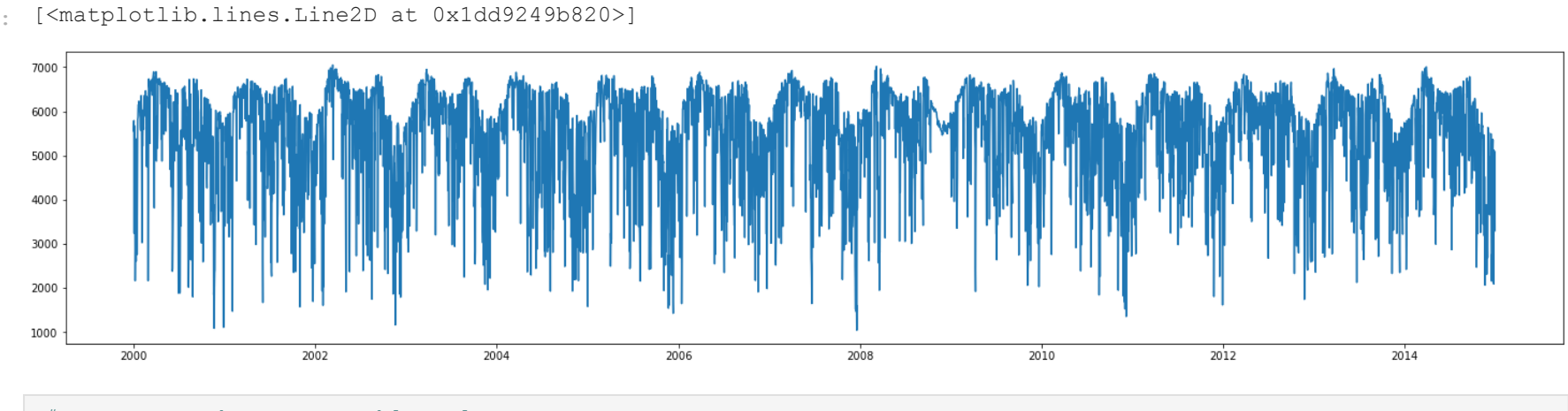
```
In [27]:
# Perform Time series decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(daily_MP['GHI'], model='additive',freq=365)
result.plot()
plt.show()
```

```
<ipython-input-27-32a8faf17172>:3: FutureWarning: the 'freq'' keyword is deprecated, use 'period' instead
  result = seasonal_decompose(daily_MP['GHI'], model='additive',freq=365)
```



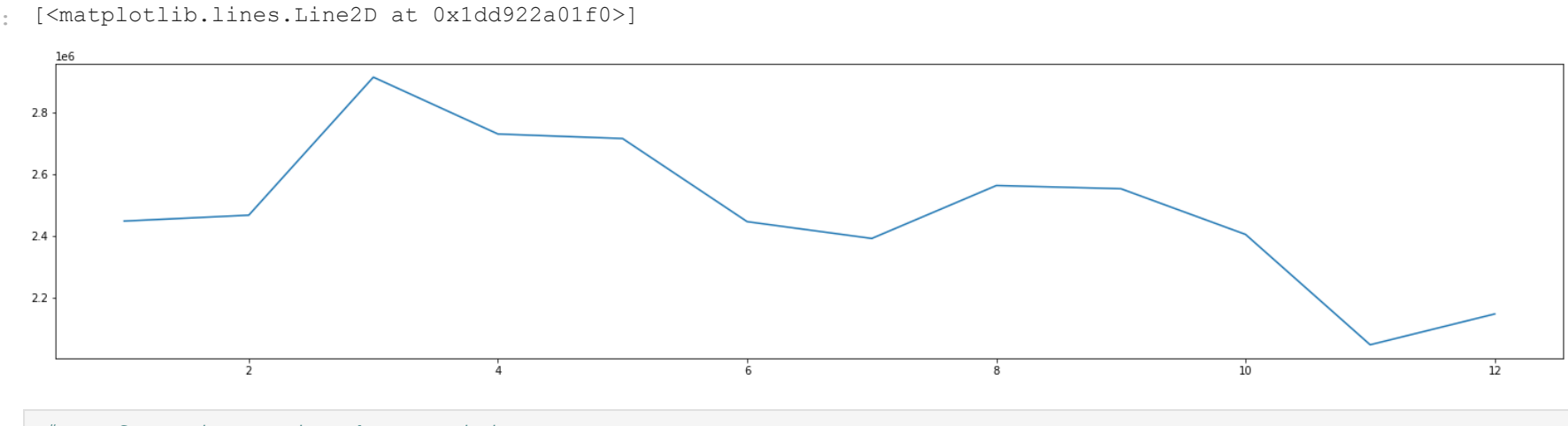
```
In [28]:
# Daily values aggregation for Tamil Nadu
data = [TamilNadu_data['date_time'],TamilNadu_data['GHI']]
headers = ["date_time","GHI"]
temp_df = pd.concat(data,axis=1,keys=headers)
daily_TN = temp_df.groupby(temp_df['date_time'].dt.date).agg({'GHI': 'sum'}).reset_index()
daily_TN.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

plt.plot(daily_TN['date_time'],daily_TN['GHI'])
```

Out[28]: [<matplotlib.lines.Line2D at 0x1dd9249b820>]



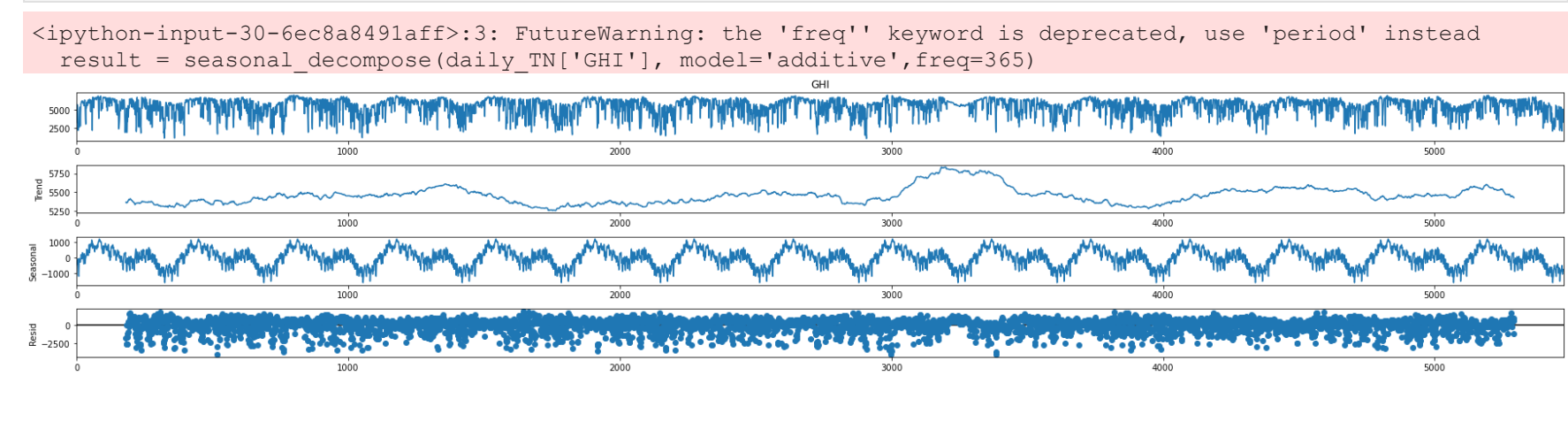
```
In [29]:
# Aggregate data to monthly values
monthly_TN = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_index()
monthly_TN.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

plt.plot(monthly_TN['date_time'],monthly_TN['GHI'])
```

Out[29]: [<matplotlib.lines.Line2D at 0x1dd922a01f0>]



```
In [30]:
# Perform Time series decomposition
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(daily_TN['GHI'], model='additive',freq=365)
result.plot()
plt.show()
```

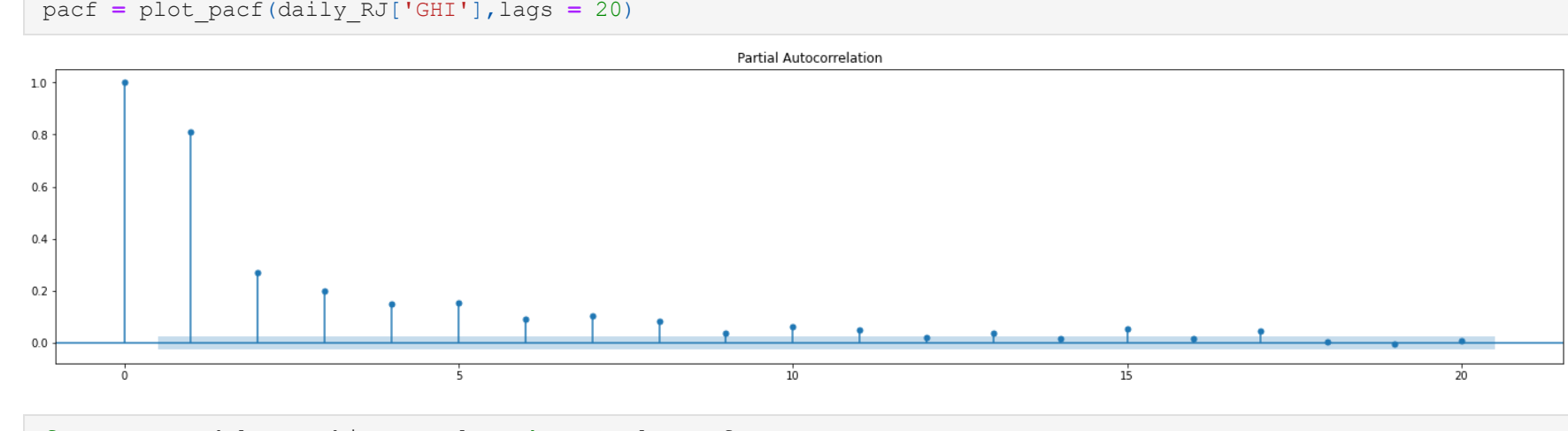
```
<ipython-input-30-6ec8a8491aff>:3: FutureWarning: the 'freq'' keyword is deprecated, use 'period' instead
  result = seasonal_decompose(daily_TN['GHI'], model='additive',freq=365)
```



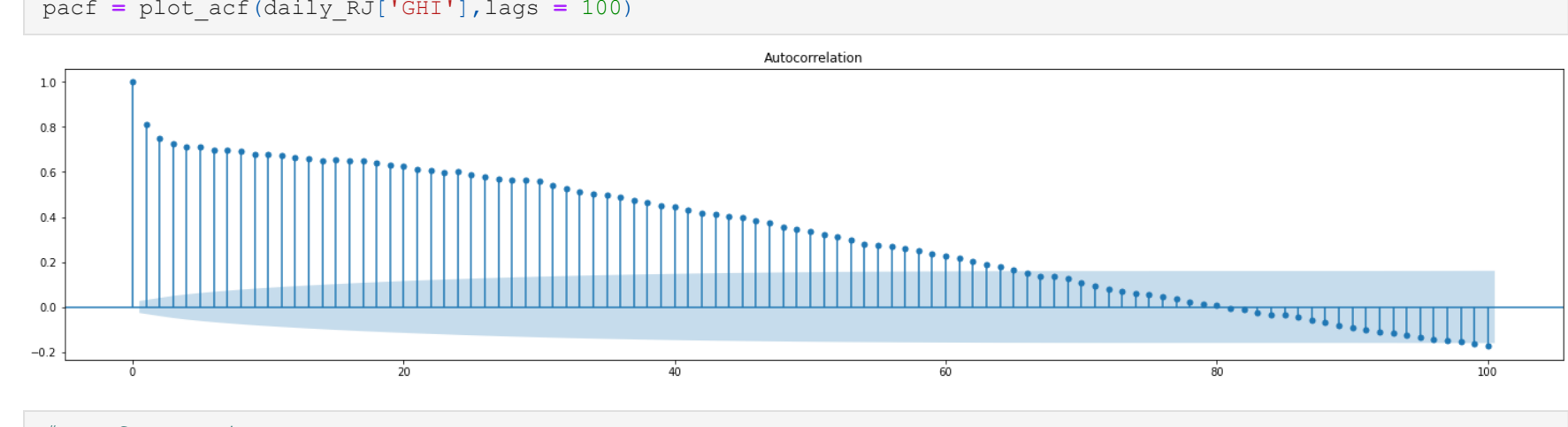
Daily/Weekly forecasting

```
In [31]:
from statsmodels.tsa.arima_model import ARIMA
```

```
In [40]:
# In time series analysis, the partial autocorrelation function (PACF) gives the partial correlation
# of a stationary time series with its own lagged values, regressed the values of the time series at all shorte
# It contrasts with the autocorrelation function, which does not control for other lags.
from statsmodels.graphics.tsaplots import plot_pacf

pacf = plot_pacf(daily_RJ['GHI'],lags = 20)
```



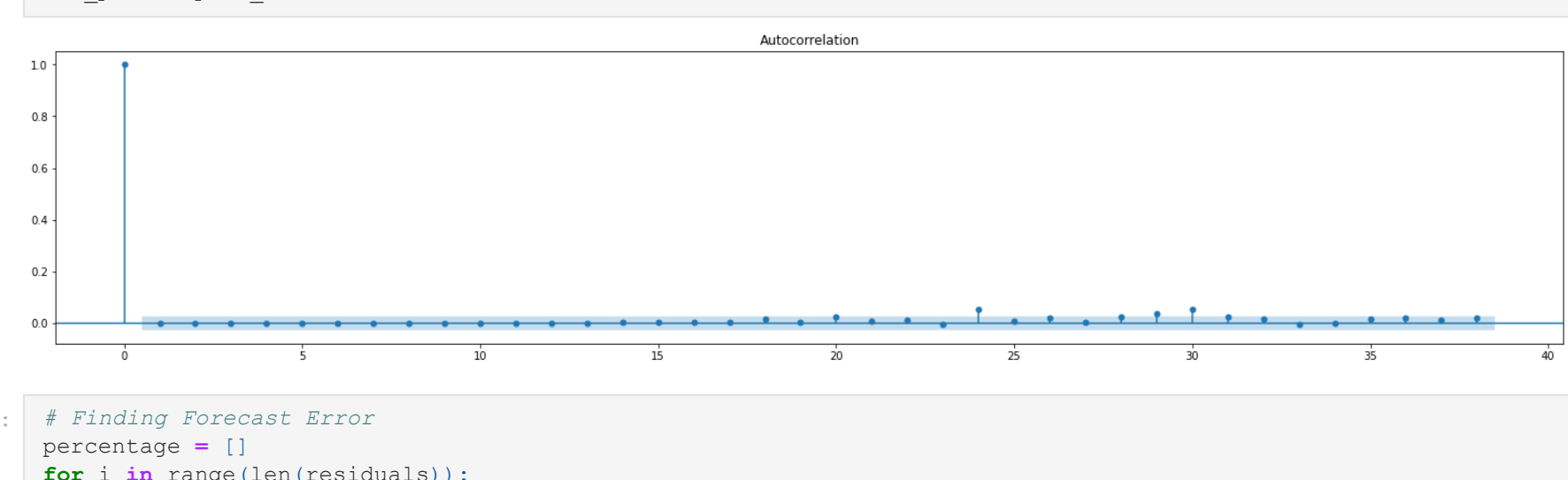

```
In [42]:
from statsmodels.graphics.tsaplots import plot_acf

pacf = plot_acf(daily_RJ['GHI'],lags = 100)
```




```
In [45]:
# AR forecasting
ar_model = ARIMA(daily_RJ['GHI'],order=(17,0,0))
ar_fit_model = ar_model.fit()
print(ar_fit_model.summary())
residuals = pd.DataFrame(ar_fit_model.resid)
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    GHI   No. Observations:                 5475
Model:               ARIMA(17, 0, 0)   Log Likelihood              -42175.074
Date:               Sun, 25 Apr 2021   AIC                          84388.149
Time:                       15:42:15   BIC                          84513.700
Sample:                            0   HQIC                         84431.951
                              - 5475
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       5408.8571    243.776     22.188      0.000    4931.065    5886.649
ar.L1          0.4365      0.007     58.282      0.000       0.422       0.451
ar.L2          0.0750      0.010      7.611      0.000       0.056       0.094
ar.L3          0.0647      0.011      6.038      0.000       0.044       0.086
ar.L4          0.0345      0.009      3.639      0.000       0.016       0.053
ar.L5          0.0786      0.010      7.691      0.000       0.059       0.099
ar.L6          0.0189      0.010      1.861      0.063      -0.001       0.039
ar.L7          0.0453      0.010      4.329      0.000       0.025       0.066
ar.L8          0.0505      0.012      4.284      0.000       0.027       0.074
ar.L9         -0.0036      0.014     -0.265      0.791      -0.030       0.023
ar.L10         0.0305      0.012      2.570      0.010       0.007       0.054
ar.L11         0.0306      0.011      2.772      0.006       0.009       0.052
ar.L12        -0.0037      0.011     -0.330      0.741      -0.025       0.018
ar.L13         0.0259      0.012      2.249      0.024       0.003       0.048
ar.L14        -0.0131      0.012     -1.139      0.255      -0.036       0.009
ar.L15         0.0450      0.012      3.873      0.000       0.022       0.068
ar.L16        -0.0014      0.013     -0.112      0.911      -0.027       0.024
ar.L17         0.0454      0.011      4.283      0.000       0.025       0.066
sigma2       2.88e+05   2599.973    110.764      0.000    2.83e+05    2.93e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             52629.47
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               0.99   Skew:                            -2.38
Prob(H) (two-sided):                  0.87   Kurtosis:                        17.42
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
In [46]:
acf_plot = plot_acf(residuals)
```

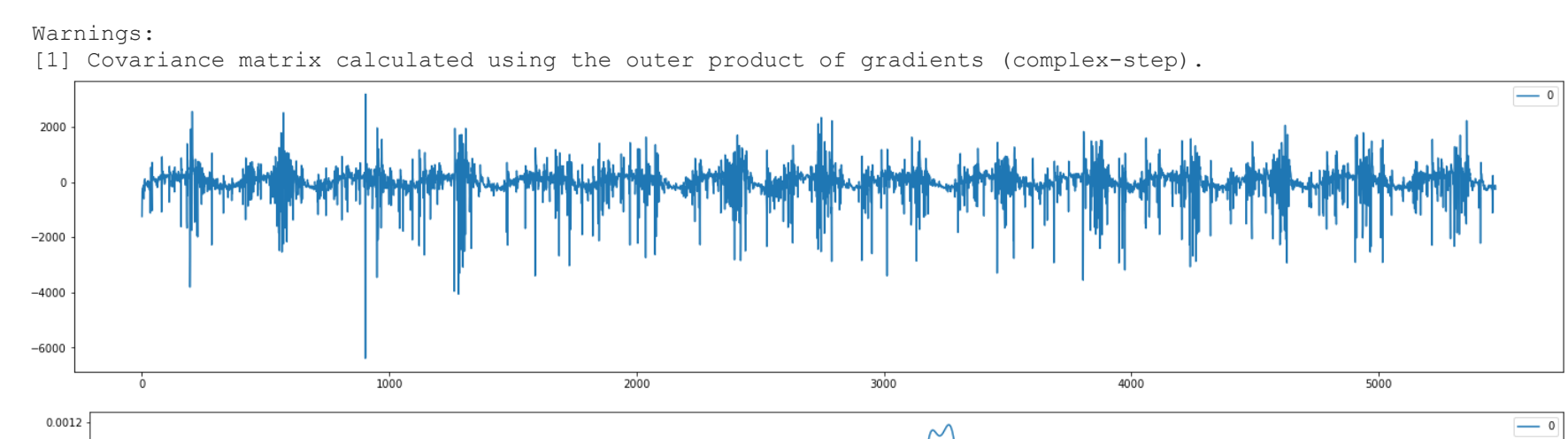


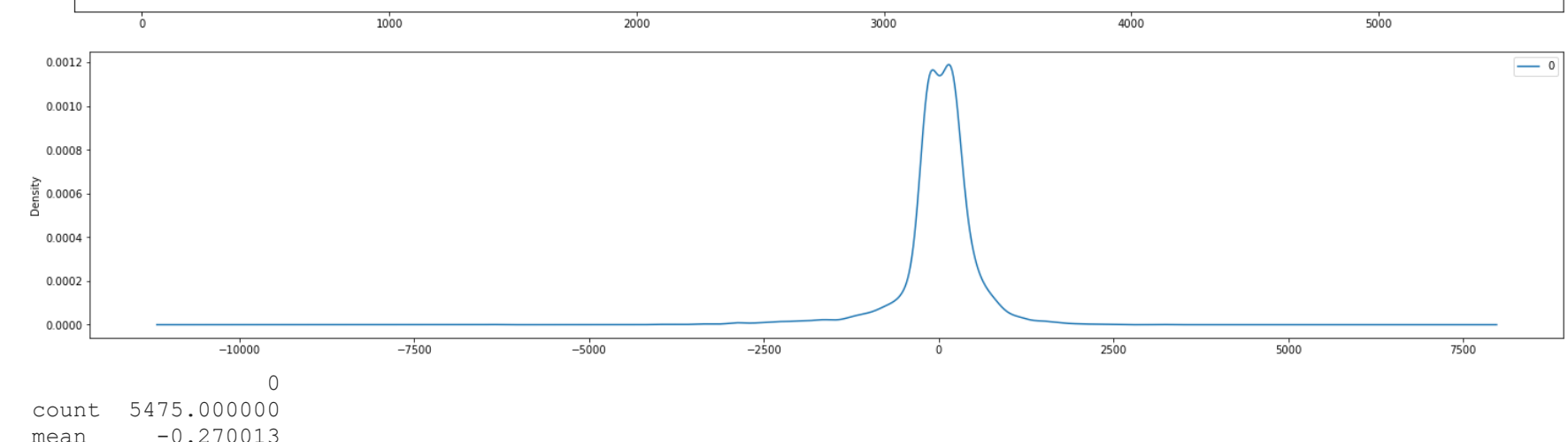

```
In [47]:
# Finding Forecast Error
percentage = []
for i in range(len(residuals)):
  if (daily_RJ['GHI'][i]==0):
    continue
  percentage.append((abs(residuals[0][i])/(daily_RJ['GHI'][i]))*100)
mape = sum(percentage)/len(percentage)
print("Mean Absolute Percentage Error: {0} %".format(mape))
```

Mean Absolute Percentage Error: 6.9001792971454385 %

```
In [48]:
# from statsmodels.tsa.arima_model import ARIMA
modelMA = ARIMA(daily_RJ['GHI'],order=(0,0,30))
fit_modelMA = modelMA.fit()
print(fit_modelMA.summary())
residualsMA = pd.DataFrame(fit_modelMA.resid)
residualsMA.plot()
plt.show()
residualsMA.plot(kind='kde')
plt.show()
print(residualsMA.describe())
```

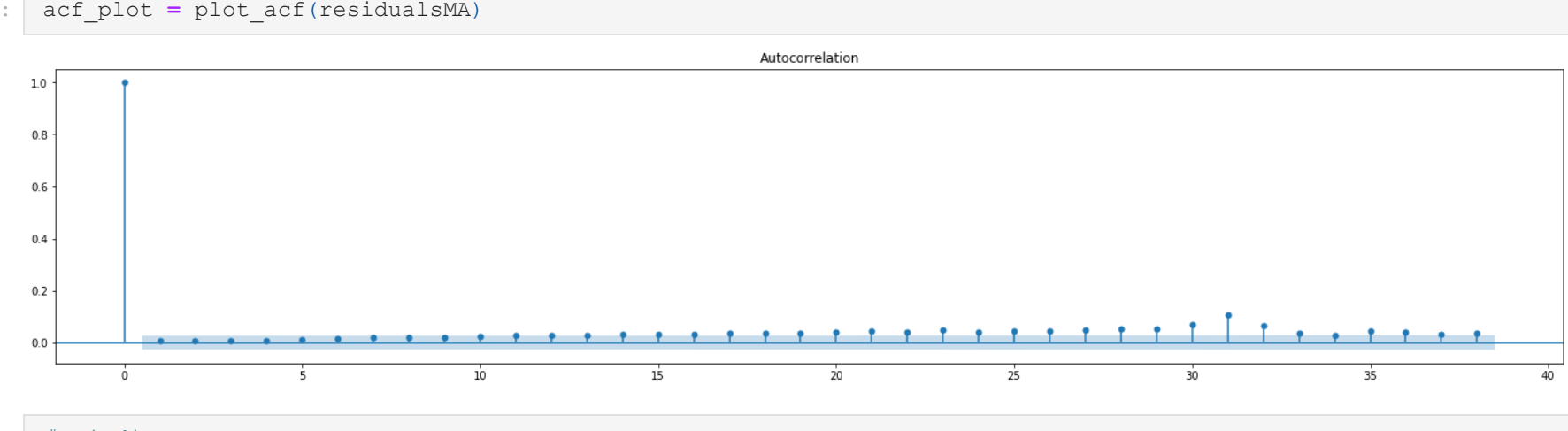
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    GHI   No. Observations:                 5475
Model:               ARIMA(0, 0, 30)   Log Likelihood              -42293.561
Date:               Sun, 25 Apr 2021   AIC                          84651.123
Time:                       15:51:21   BIC                          84862.577
Sample:                            0   HQIC                         84724.894
                              - 5475
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       5408.7891     69.772     77.520      0.000    5272.038    5545.540
ma.L1          0.4767      0.008     62.205      0.000       0.462       0.492
ma.L2          0.3177      0.011     30.187      0.000       0.297       0.338
ma.L3          0.2741      0.011     24.328      0.000       0.252       0.296
ma.L4          0.2422      0.012     19.763      0.000       0.218       0.266
ma.L5          0.2708      0.013     20.340      0.000       0.245       0.297
ma.L6          0.2443      0.014     17.721      0.000       0.217       0.271
ma.L7          0.2484      0.014     17.412      0.000       0.220       0.276
ma.L8          0.2671      0.015     17.269      0.000       0.237       0.297
ma.L9          0.2314      0.016     14.117      0.000       0.199       0.264
ma.L10         0.2370      0.016     14.564      0.000       0.205       0.269
ma.L11         0.2373      0.017     14.105      0.000       0.204       0.270
ma.L12         0.2117      0.017     12.669      0.000       0.179       0.244
ma.L13         0.2178      0.016     13.221      0.000       0.185       0.250
ma.L14         0.1849      0.017     10.988      0.000       0.152       0.218
ma.L15         0.2138      0.018     12.078      0.000       0.179       0.249
ma.L16         0.1957      0.018     10.884      0.000       0.160       0.231
ma.L17         0.2147      0.018     11.912      0.000       0.179       0.250
ma.L18         0.2090      0.017     12.018      0.000       0.175       0.243
ma.L19         0.1780      0.017     10.432      0.000       0.145       0.211
ma.L20         0.1796      0.017     10.475      0.000       0.146       0.213
ma.L21         0.1506      0.017      8.747      0.000       0.117       0.184
ma.L22         0.1442      0.017      8.607      0.000       0.111       0.177
ma.L23         0.1073      0.016      6.527      0.000       0.075       0.140
ma.L24         0.1496      0.015      9.772      0.000       0.120       0.180
ma.L25         0.1165      0.015      7.572      0.000       0.086       0.147
ma.L26         0.1051      0.015      6.964      0.000       0.076       0.135
ma.L27         0.0783      0.015      5.226      0.000       0.049       0.108
ma.L28         0.0766      0.013      5.754      0.000       0.050       0.103
ma.L29         0.0780      0.013      5.831      0.000       0.052       0.104
ma.L30         0.0771      0.013      6.134      0.000       0.052       0.102
sigma2          3e+05   2858.539    104.949      0.000    2.94e+05    3.06e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.14   Jarque-Bera (JB):             37882.65
Prob(Q):                              0.70   Prob(JB):                         0.00
Heteroskedasticity (H):               1.01   Skew:                            -1.94
Prob(H) (two-sided):                  0.89   Kurtosis:                        15.29
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```





```
                 0
count  5475.000000
mean     -0.270013
std     547.932518
min   -6392.203110
25%    -164.934036
50%      44.224383
75%     239.486316
max    3188.411413
```

```
In [49]:
acf_plot = plot_acf(residualsMA)
```



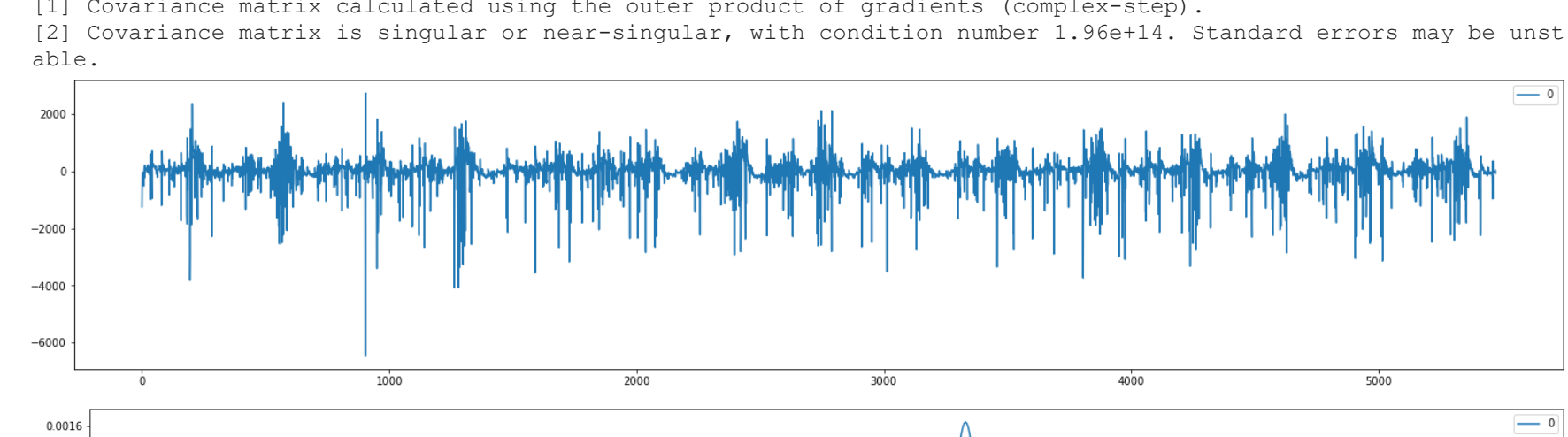

```
In [50]:
# Finding Forecast Error
percentage = []
for i in range(len(residuals)):
  if daily_RJ['GHI'][i]==0:
    continue
  percentage.append((abs(residualsMA[0][i])/(daily_RJ['GHI'][i]))*100)
mape = sum(percentage)/len(percentage)
print("Mean Absolute Percentage Error: {0} %".format(mape))
```

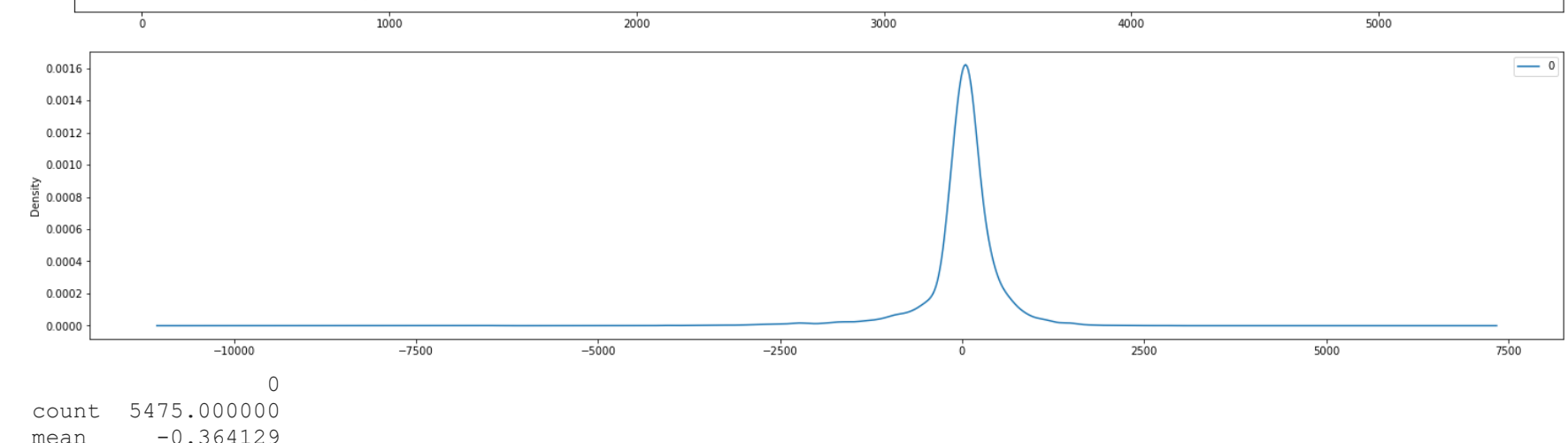
Mean Absolute Percentage Error: 7.323987998268238 %

```
In [54]:
modelARMA = ARIMA(daily_RJ['GHI'],order=(12,0,10))
fit_modelARMA = modelARMA.fit()
print(fit_modelARMA.summary())
residualsARMA = pd.DataFrame(fit_modelARMA.resid)
residualsARMA.plot()
plt.show()
residualsARMA.plot(kind='kde')
plt.show()
print(residualsARMA.describe())
```

```
C:\Users\dipam\anaconda3\lib\site-packages\statsmodels\base\model.py:566: ConvergenceWarning: Maximum Likelihoo
d optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

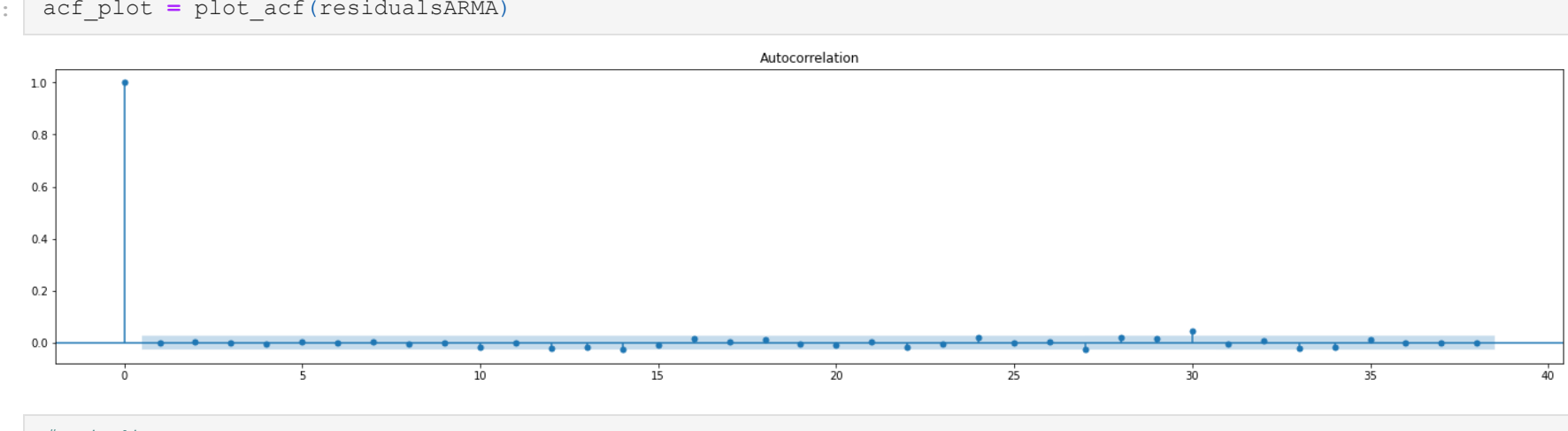
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    GHI   No. Observations:                 5475
Model:              ARIMA(12, 0, 10)   Log Likelihood              -42135.108
Date:               Sun, 25 Apr 2021   AIC                          84318.216
Time:                       17:50:02   BIC                          84476.807
Sample:                            0   HQIC                         84373.545
                              - 5475
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       5408.8569     98.534     54.893      0.000    5215.734    5601.980
ar.L1         -0.6974      0.214     -3.257      0.001      -1.117      -0.278
ar.L2          0.9062      0.091      9.958      0.000       0.728       1.085
ar.L3          0.9542      0.243      3.928      0.000       0.478       1.430
ar.L4         -0.1117      0.113     -0.989      0.323      -0.333       0.110
ar.L5         -0.0398      0.122     -0.325      0.745      -0.280       0.200
ar.L6          0.6013      0.118      5.096      0.000       0.370       0.833
ar.L7          0.8865      0.164      5.400      0.000       0.565       1.208
ar.L8         -0.3198      0.105     -3.043      0.002      -0.526      -0.114
ar.L9         -1.1054      0.155     -7.123      0.000      -1.410      -0.801
ar.L10        -0.3356      0.128     -2.626      0.009      -0.586      -0.085
ar.L11         0.2399      0.048      4.946      0.000       0.145       0.335
ar.L12         0.0143      0.020      0.707      0.479      -0.025       0.054
ma.L1          1.1190      0.214      5.228      0.000       0.700       1.539
ma.L2         -0.3712      0.121     -3.073      0.002      -0.608      -0.134
ma.L3         -0.9825      0.193     -5.091      0.000      -1.361      -0.604
ma.L4         -0.2311      0.128     -1.799      0.072      -0.483       0.021
ma.L5         -0.0406      0.092     -0.443      0.657      -0.220       0.139
ma.L6         -0.6100      0.097     -6.275      0.000      -0.800      -0.419
ma.L7         -1.1574      0.163     -7.114      0.000      -1.476      -0.839
ma.L8         -0.1882      0.144     -1.303      0.193      -0.471       0.095
ma.L9          0.9260      0.102      9.107      0.000       0.727       1.125
ma.L10         0.6031      0.139      4.333      0.000       0.330       0.876
sigma2      2.892e+05   2711.505    106.660      0.000    2.84e+05    2.95e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             52333.83
Prob(Q):                              0.95   Prob(JB):                         0.00
Heteroskedasticity (H):               1.01   Skew:                            -2.34
Prob(H) (two-sided):                  0.89   Kurtosis:                        17.40
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.96e+14. Standard errors may be unst
able.
```





```
                 0
count  5475.000000
mean     -0.364129
std     532.230002
min   -6460.165456
25%    -108.629731
50%      47.861021
75%     200.593195
max    2740.215365
```

```
In [55]:
acf_plot = plot_acf(residualsARMA)
```



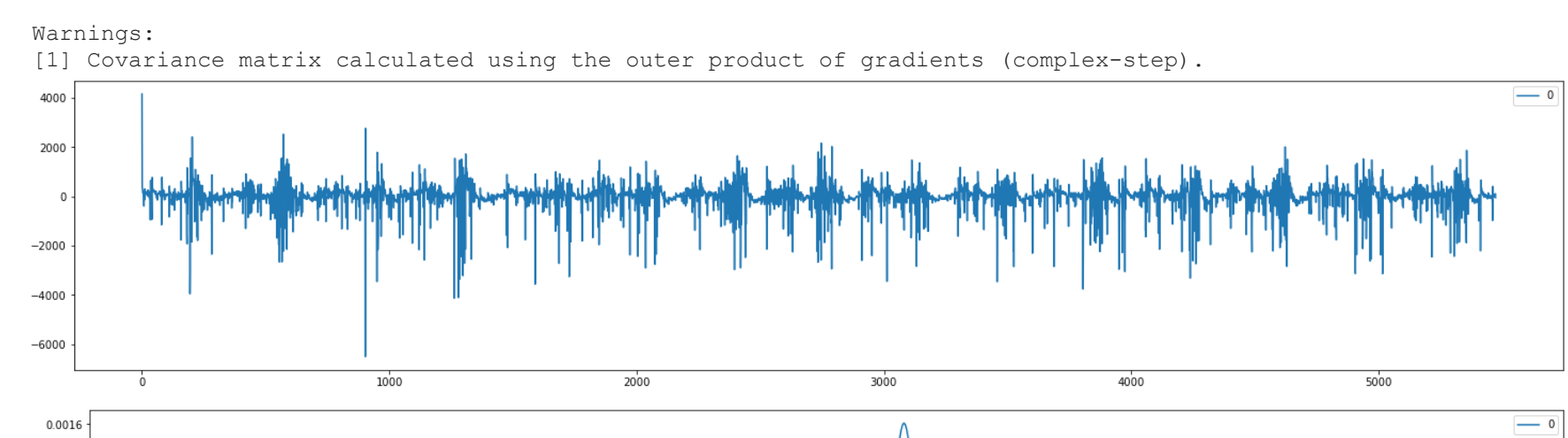

```
In [56]:
# Finding Forecast Error
percentage = []
for i in range(len(residualsARMA)):
  if daily_RJ['GHI'][i]==0:
    continue
  percentage.append((abs(residualsARMA[0][i])/(daily_RJ['GHI'][i]))*100)
mape = sum(percentage)/len(percentage)
print("Mean Absolute Percentage Error: {0} %".format(mape))
```

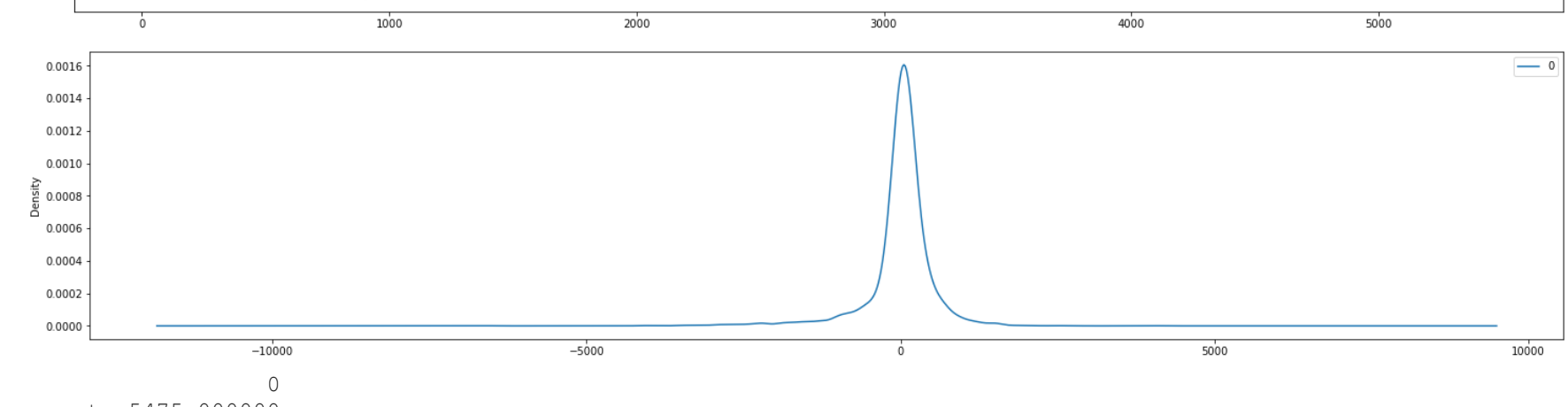
Mean Absolute Percentage Error: 6.705384235118743 %

```
In [59]:
# d = 1 gave a better fit over the data(MAPE) than d = 2.
modelARIMA = ARIMA(daily_RJ['GHI'],order=(12,1,10))
fit_modelARIMA = modelARIMA.fit()
print(fit_modelARIMA.summary())
residualsARIMA = pd.DataFrame(fit_modelARIMA.resid)
residualsARIMA.plot()
plt.show()
residualsARIMA.plot(kind='kde')
plt.show()
print(residualsARIMA.describe())
```

```
C:\Users\dipam\anaconda3\lib\site-packages\statsmodels\base\model.py:566: ConvergenceWarning: Maximum Likelihoo
d optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    GHI   No. Observations:                 5475
Model:              ARIMA(12, 1, 10)   Log Likelihood              -42163.259
Date:               Sun, 25 Apr 2021   AIC                          84372.517
Time:                       17:53:32   BIC                          84524.496
Sample:                            0   HQIC                         84425.540
                              - 5475
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3495      0.734      0.476      0.634      -1.089       1.788
ar.L2          0.8206      0.549      1.495      0.135      -0.255       1.897
ar.L3         -0.7063      1.078     -0.655      0.512      -2.819       1.406
ar.L4          0.1036      0.887      0.117      0.907      -1.634       1.841
ar.L5          0.6410      0.843      0.760      0.447      -1.011       2.293
ar.L6         -0.7135      0.773     -0.924      0.356      -2.228       0.801
ar.L7          0.2377      0.556      0.427      0.669      -0.853       1.328
ar.L8          0.4709      0.426      1.105      0.269      -0.364       1.306
ar.L9         -0.2806      0.331     -0.847      0.397      -0.930       0.369
ar.L10        -0.0024      0.123     -0.020      0.984      -0.244       0.239
ar.L11         0.0076      0.039      0.197      0.844      -0.068       0.083
ar.L12        -0.0335      0.026     -1.312      0.190      -0.084       0.017
ma.L1         -0.9122      0.734     -1.243      0.214      -2.350       0.526
ma.L2         -0.7996      0.659     -1.214      0.225      -2.090       0.491
ma.L3          1.1797      1.238      0.953      0.341      -1.246       3.606
ma.L4         -0.3713      1.169     -0.318      0.751      -2.662       1.920
ma.L5         -0.6443      1.068     -0.604      0.546      -2.737       1.448
ma.L6          1.0603      1.081      0.981      0.327      -1.058       3.179
ma.L7         -0.5273      0.762     -0.692      0.489      -2.021       0.967
ma.L8         -0.4087      0.642     -0.637      0.524      -1.666       0.849
ma.L9          0.5326      0.379      1.403      0.160      -0.211       1.276
ma.L10        -0.0765      0.316     -0.242      0.809      -0.696       0.542
sigma2      2.902e+05   2107.340    137.711      0.000    2.86e+05    2.94e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):             53832.32
Prob(Q):                              0.85   Prob(JB):                         0.00
Heteroskedasticity (H):               1.00   Skew:                            -2.41
Prob(H) (two-sided):                  0.97   Kurtosis:                        17.59
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

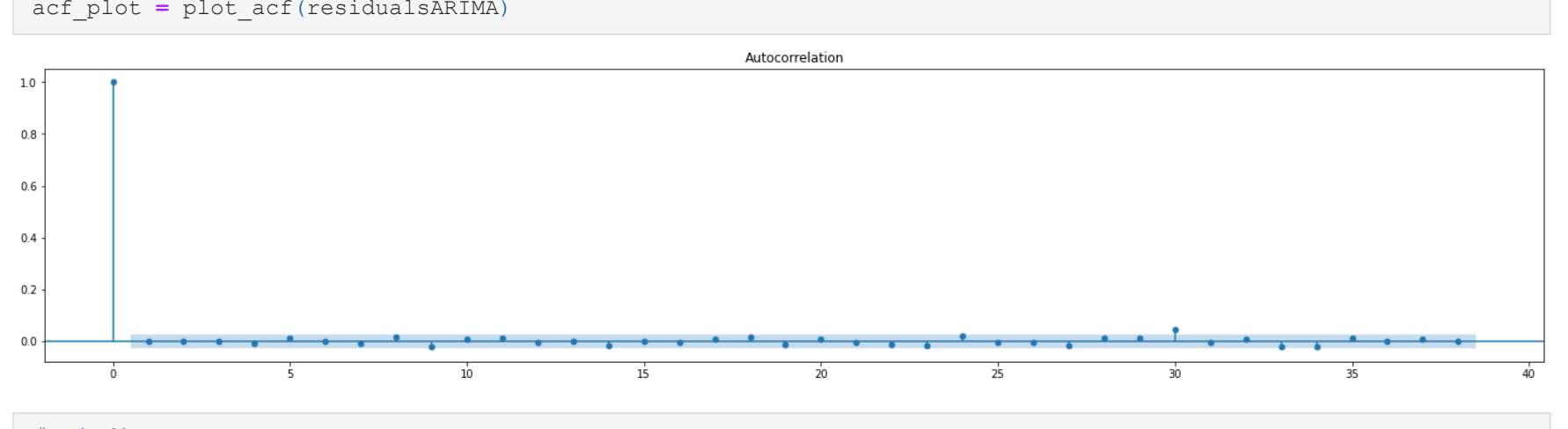




```
                 0
count  5475.000000
mean      0.789912
std     538.554206
min   -6508.306535
25%    -100.333928
50%      54.572492
75%     207.398773
max    4162.000000
```

In [60]:

```
acf_plot = plot_acf(residualsARIMA)
```



In [61]:

```
# Finding Forecast Error
percentage = []
for i in range(len(residualsARIMA)):
  if daily_RJ['GHI'][i]==0:
    continue
  percentage.append((abs(residualsARIMA[0][i])/(daily_RJ['GHI'][i]))*100)
mape = sum(percentage)/len(percentage)
print("Mean Absolute Percentage Error: {0} %".format(mape))
```

```
Mean Absolute Percentage Error: 6.794710456878758 %
```

In [63]:

```
train_data_len = int((len(daily_RJ)*13)/15)
train_data = daily_RJ['GHI'][0:train_data_len]
test_data = daily_RJ['GHI'][train_data_len:len(daily_RJ)]
```

In [64]:

```
# Rolling forecast for AR
history = train_data.copy()

predictions = []
weekly = []
monthly = []
for t in range(len(test_data)):
  modelAR = ARIMA(history,order = (8,0,0))
  model_fit = modelAR.fit()

  if (t)%7==0:
    forecast = model_fit.predict(start=t+train_data_len, end=t+train_data_len+6)
    # print(forecast)
    for i in range(7):
      weekly.append(forecast[train_data_len+t+i])

  if (t)%30==0:
    forecast = model_fit.predict(start=t+train_data_len, end=t+train_data_len+29)
    # print(forecast)
    for i in range(30):
      monthly.append(forecast[train_data_len+t+i])

  output = model_fit.forecast()
  predVal = output[train_data_len+t]
  predictions.append(predVal)
  history[len(history)] = test_data[train_data_len+t]

# ar_model = ARIMA(train_data,order=(12,0,0))
# ar_fit_model = ar_model.fit()
# print(ar_fit_model.summary())
# residuals = pd.DataFrame(ar_fit_model.resid)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-64-339c9b881b23> in <module>
      7 for t in range(len(test_data)):
      8   modelAR = ARIMA(history,order = (8,0,0))
----> 9   model_fit = modelAR.fit()
     10 
     11   if (t)%7==0:

~\anaconda3\lib\site-packages\statsmodels\tsa\arima\model.py in fit(self, start_params, transformed, includes_fixed, method, method_kwargs, gls, gls_kwargs, cov_type, cov_kwds, return_params, low_memory)
    340                 method_kwargs.setdefault('disp', 0)
    341 
--> 342                 res = super(ARIMA, self).fit(
    343                     return_params=return_params, low_memory=low_memory,
    344                     cov_type=cov_type, cov_kwds=cov_kwds, **method_kwargs)

~\anaconda3\lib\site-packages\statsmodels\tsa\statespace\mlemodel.py in fit(self, start_params, transformed, includes_fixed, cov_type, cov_kwds, method, maxiter, full_output, disp, callback, return_params, optim_score, optim_complex_step, optim_hessian, flags, low_memory, **kwargs)
    688                 flags['hessian_method'] = optim_hessian
    689             fargs = (flags,)
--> 690             mlefit = super(MLEModel, self).fit(start_params, method=method,
    691                                                fargs=fargs,
    692                                                maxiter=maxiter,

~\anaconda3\lib\site-packages\statsmodels\base\model.py in fit(self, start_params, method, maxiter, full_output, disp, fargs, callback, retall, skip_hessian, **kwargs)
    517         warn_convergence = kwargs.pop('warn_convergence', True)
    518         optimizer = Optimizer()
--> 519         xopt, retvals, optim_settings = optimizer._fit(f, score, start_params,
    520                                                        fargs, kwargs,
    521                                                        hessian=hess,

~\anaconda3\lib\site-packages\statsmodels\base\optimizer.py in _fit(self, objective, gradient, start_params, fargs, kwargs, hessian, method, maxiter, full_output, disp, callback, retall)
    213 
    214         func = fit_funcs[method]
--> 215         xopt, retvals = func(objective, gradient, start_params, fargs, kwargs,
    216                              disp=disp, maxiter=maxiter, callback=callback,
    217                              retall=retall, full_output=full_output,

~\anaconda3\lib\site-packages\statsmodels\base\optimizer.py in _fit_lbfgs(f, score, start_params, fargs, kwargs, disp, maxiter, callback, retall, full_output, hess)
    450         func = f
    451 
--> 452     retvals = optimize.fmin_l_bfgs_b(func, start_params, maxiter=maxiter,
    453                                      callback=callback, args=fargs,
    454                                      bounds=bounds, disp=disp,

~\anaconda3\lib\site-packages\scipy\optimize\lbfgsb.py in fmin_l_bfgs_b(func, x0, fprime, args, approx_grad, bounds, m, factr, pgtol, epsilon, iprint, maxfun, maxiter, disp, callback, maxls)
    195             'maxls': maxls}
    196 
--> 197     res = _minimize_lbfgsb(fun, x0, args=args, jac=jac, bounds=bounds,
    198                            **opts)
    199     d = {'grad': res['jac'],

~\anaconda3\lib\site-packages\scipy\optimize\lbfgsb.py in _minimize_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps, maxfun, maxiter, iprint, callback, maxls, finite_diff_rel_step, **unknown_options)
    304             iprint = disp
    305 
--> 306     sf = _prepare_scalar_function(fun, x0, jac=jac, args=args, epsilon=eps,
    307                                   bounds=new_bounds,
    308                                   finite_diff_rel_step=finite_diff_rel_step)

~\anaconda3\lib\site-packages\scipy\optimize\optimize.py in _prepare_scalar_function(fun, x0, jac, args, bounds, epsilon, finite_diff_rel_step, hess)
    259     # ScalarFunction caches. Reuse of fun(x) during grad
    260     # calculation reduces overall function evaluations.
--> 261     sf = ScalarFunction(fun, x0, args, grad, hess,
    262                         finite_diff_rel_step, bounds, epsilon=epsilon)
    263 

~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in __init__(self, fun, x0, args, grad, hess, finite_diff_rel_step, finite_diff_bounds, epsilon)
     93 
     94         self._update_grad_impl = update_grad
---> 95         self._update_grad()
     96 
     97         # Hessian Evaluation

~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in _update_grad(self)
    169     def _update_grad(self):
    170         if not self.g_updated:
--> 171             self._update_grad_impl()
    172             self.g_updated = True
    173 

~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in update_grad()
     89                 self._update_fun()
     90                 self.ngev += 1
---> 91                 self.g = approx_derivative(fun_wrapped, self.x, f0=self.f,
     92                                            **finite_diff_options)
     93 

~\anaconda3\lib\site-packages\scipy\optimize\_numdiff.py in approx_derivative(fun, x0, method, rel_step, abs_step, f0, bounds, sparsity, as_linear_operator, args, kwargs)
    424 
    425         if sparsity is None:
--> 426             return _dense_difference(fun_wrapped, x0, f0, h,
    427                                      use_one_sided, method)
    428         else:

~\anaconda3\lib\site-packages\scipy\optimize\_numdiff.py in _dense_difference(fun, x0, f0, h, use_one_sided, method)
    495             x = x0 + h_vecs[i]
    496             dx = x[i] - x0[i]  # Recompute dx as exactly representable number.
--> 497             df = fun(x) - f0
    498         elif method == '3-point' and use_one_sided[i]:
    499             x1 = x0 + h_vecs[i]

~\anaconda3\lib\site-packages\scipy\optimize\_numdiff.py in fun_wrapped(x)
    375 
    376     def fun_wrapped(x):
--> 377         f = np.atleast_1d(fun(x, *args, **kwargs))
    378         if f.ndim > 1:
    379             raise RuntimeError("`fun` return value has "

~\anaconda3\lib\site-packages\scipy\optimize\_differentiable_functions.py in fun_wrapped(x)
     68         def fun_wrapped(x):
     69             self.nfev += 1
---> 70             return fun(x, *args)
     71 
     72         def update_fun():

~\anaconda3\lib\site-packages\statsmodels\base\model.py in f(params, *args)
    499 
    500         def f(params, *args):
--> 501             return -self.loglike(params, *args) / nobs
    502 
    503         if method == 'newton':

~\anaconda3\lib\site-packages\statsmodels\tsa\statespace\mlemodel.py in loglike(self, params, *args, **kwargs)
    923             kwargs['inversion_method'] = INVERT_UNIVARIATE | SOLVE_LU
    924 
--> 925         loglike = self.ssm.loglike(complex_step=complex_step, **kwargs)
    926 
    927         # Koopman, Shephard, and Doornik recommend maximizing the average

~\anaconda3\lib\site-packages\statsmodels\tsa\statespace\kalman_filter.py in loglike(self, **kwargs)
    981         kwargs.setdefault('conserve_memory',
    982                           MEMORY_CONSERVE ^ MEMORY_NO_LIKELIHOOD)
--> 983         kfilter = self._filter(**kwargs)
    984         loglikelihood_burn = kwargs.get('loglikelihood_burn',
    985                                         self.loglikelihood_burn)

KeyboardInterrupt: 
```

In [ ]: