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 [4]:

```
# 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_t
        # Taking only 4 evenly spaced data points, uncomment above line and comment lin
        # data_temp = data_temp[data_temp['date_time'].dt.hour.isin([9,11,13,15])]

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

In [5]:

```
col_name = ['date_time','Minute','DHI','DNI','GHI','Clearsky DHI','Clearsky DNI','Clear
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()
```

Out[5]:

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	I
0	2000-01-01 08:00:00	0	149	375	268	121	465	268	9	16.043264	969.206482	6
1	2000-01-01 09:00:00	0	162	611	455	160	617	456	9	20.477919	969.256287	4

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	H
2	2000-01-01 10:00:00	0	183	698	600	183	698	600	9	23.532446	969.059021	4
3	2000-01-01 11:00:00	0	194	735	682	194	735	682	9	25.100303	968.150452	3
4	2000-01-01 12:00:00	0	195	740	695	195	740	695	9	25.644234	967.158325	3

In [6]:

```
MP_data.shape
```

Out[6]: (49275, 16)

In [7]:

```
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[7]:

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	H
0	2000-01-01 08:00:00	0	147	468	341	117	611	370	17	22.268241	953.526489	7
1	2000-01-01 09:00:00	0	184	587	535	143	737	584	17	23.986125	953.256287	6
2	2000-01-01 10:00:00	0	228	630	692	159	804	751	17	25.217761	952.739014	6
3	2000-01-01 11:00:00	0	177	821	851	167	837	854	17	25.983396	951.830444	5
4	2000-01-01 12:00:00	0	171	816	860	169	845	882	17	26.406886	950.518311	5

In [8]:

```
TamilNadu_data.shape
```

Out[8]: (49275, 16)

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

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	I
0	2000-01-01 08:00:00	0	78	306	135	74	354	139	-8	14.277325	986.486450	2
1	2000-01-01 09:00:00	0	114	597	331	121	600	339	-6	17.677278	986.856323	1
2	2000-01-01 10:00:00	0	144	681	488	147	721	512	-4	21.872848	986.979065	1
3	2000-01-01 11:00:00	0	151	759	608	162	782	633	-2	26.036963	986.390442	1
4	2000-01-01 12:00:00	0	156	788	664	168	805	687	-2	28.260238	985.398315	1

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

Out[11]: (49275, 16)

In [12]:
```
col_name = ['date_time','Minute','DHI','DNI','GHI','Clearsky DHI','Clearsky DNI','Clear
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[12]:

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	I
0	2000-01-01 08:00:00	0	110	602	333	110	602	333	14	18.137191	942.006470	8
1	2000-01-01 09:00:00	0	138	740	547	138	740	547	14	21.422234	941.736328	6
2	2000-01-01 10:00:00	0	155	812	714	155	812	714	14	23.382713	941.219055	5

	date_time	Minute	DHI	DNI	GHI	Clearsky DHI	Clearsky DNI	Clearsky GHI	Dew Point	Temperature	Pressure	I
3	2000-01-01 11:00:00	0	164	847	818	164	847	818	14	24.719346	940.310425	5
4	2000-01-01 12:00:00	0	261	674	797	167	855	847	14	25.476261	938.998352	5

In [13]:
```
AP_data['GHI'].describe()
```

Out[13]:
```
count    49275.000000
mean       623.366900
std        230.797025
min          0.000000
25%        432.000000
50%        644.000000
75%        822.000000
max       1055.000000
Name: GHI, dtype: float64
```

In [14]:
```
AP_data['DHI'].describe()
```

Out[14]:
```
count    49275.000000
mean       229.875677
std         83.114198
min          0.000000
25%        170.000000
50%        212.000000
75%        274.000000
max        904.000000
Name: DHI, dtype: float64
```

In [15]:
```
AP_data['DNI'].describe()
```

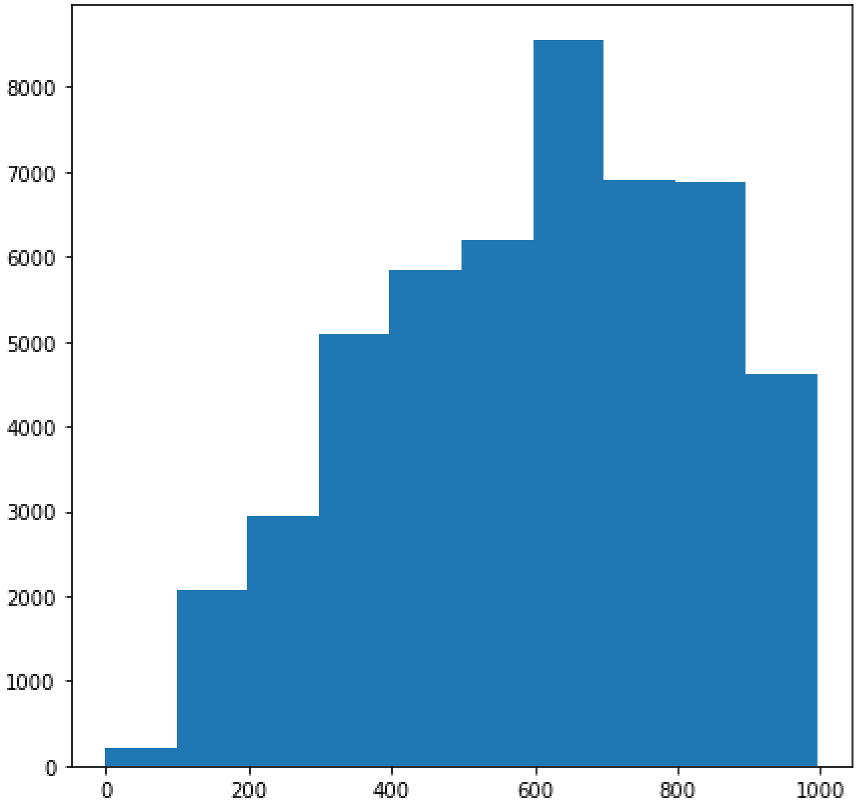
Out[15]:
```
count    49275.000000
mean       522.960284
std        257.188347
min          0.000000
25%        335.500000
50%        599.000000
75%        731.000000
max       1014.000000
Name: DNI, dtype: float64
```

Normality check

In [21]:
```
plt.rcParams["figure.figsize"] = (7,7)
plt.hist(Rajasthan_data['GHI'])
```

Out[21]:
```
(array([ 205., 2063., 2953., 5080., 5847., 6188., 8541., 6891., 6888.,
        4619.]),
```

```
array([  0. ,  99.5, 199. , 298.5, 398. , 497.5, 597. , 696.5, 796. ,
       895.5, 995. ]),
<BarContainer object of 10 artists>)
```



In [17]:

```
def is_normal(df):
    print('Test for state: ' + df.name)
    stat, p = normaltest(df['GHI'])
    print(stat, p)
    print('Statistics=%.3f, p=%.3f' % (stat, p))
    # interpret
    alpha = 0.05
    if p > alpha:
        print('Sample looks Gaussian (fail to reject H0\n\n)')
    else:
        print('Sample does not look Gaussian (reject H0)\n\n')

is_normal(AP_data)
is_normal(MP_data)
is_normal(Rajasthan_data)
is_normal(TamilNadu_data)
```

```
Test for state: Andhra Pradesh
10043.59196153765 0.0
Statistics=10043.592, p=0.000
Sample does not look Gaussian (reject H0)


Test for state: MP
8736.34703024374 0.0
Statistics=8736.347, p=0.000
Sample does not look Gaussian (reject H0)


Test for state: Rajasthan
5378.846626815809 0.0
```

```
Statistics=5378.847, p=0.000
Sample does not look Gaussian (reject H0)


Test for state: Tamil Nadu
12330.258876351612 0.0
Statistics=12330.259, p=0.000
Sample does not look Gaussian (reject H0)
```

Feature Correlation HeatMap

In [18]:

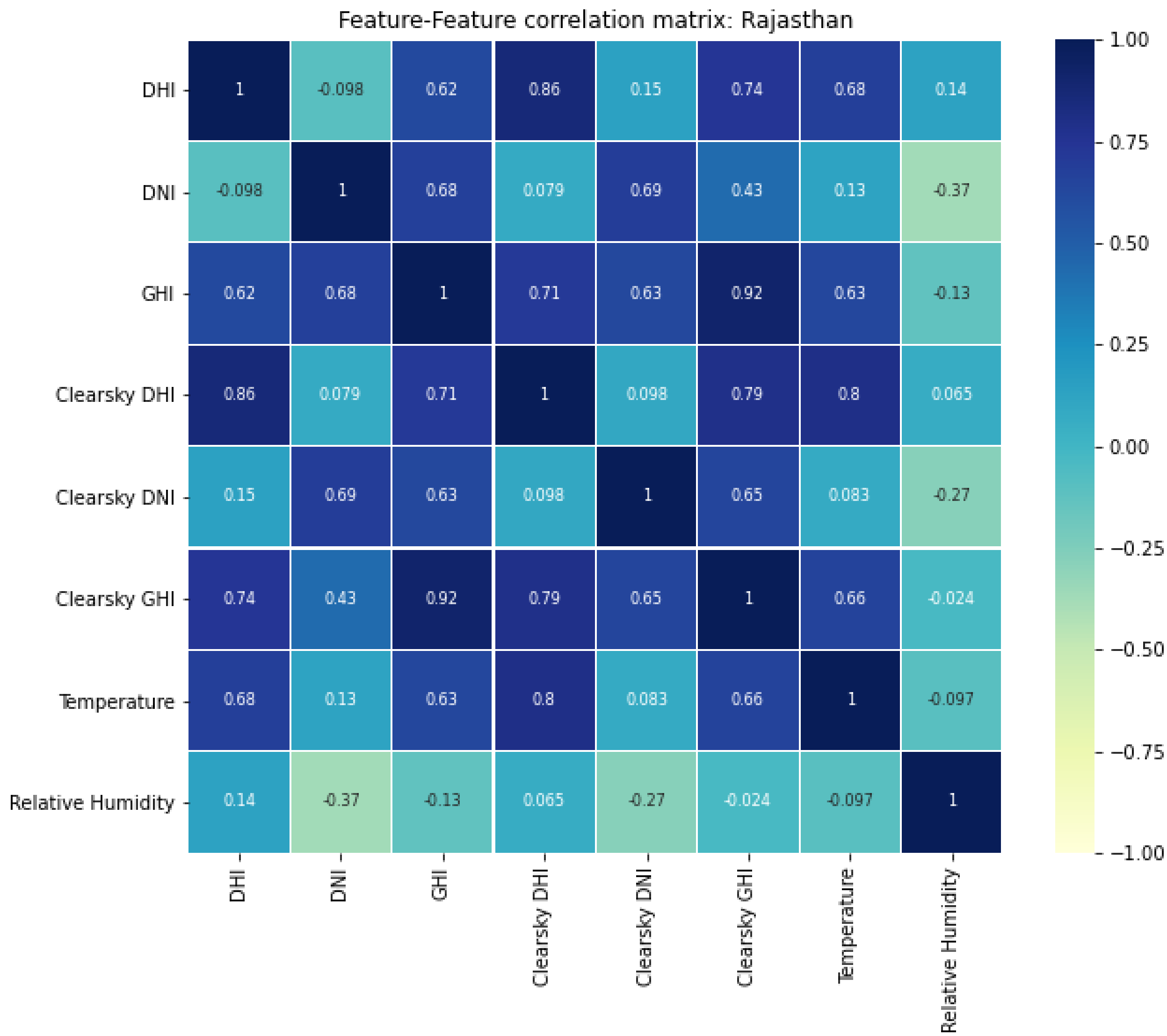
```
#ignoring dewpoint as its value is just around 937 to 940 for every hour
#solar zenith angle and Wind speed also doesnot seem useful
AP_data_attribute = AP_data[['DHI','DNI','GHI','Clearsky DHI','Clearsky DNI','Clearsky
AP_var_correlation = np.corrcoef(AP_data_attribute.T)
AP_var_correlation
```

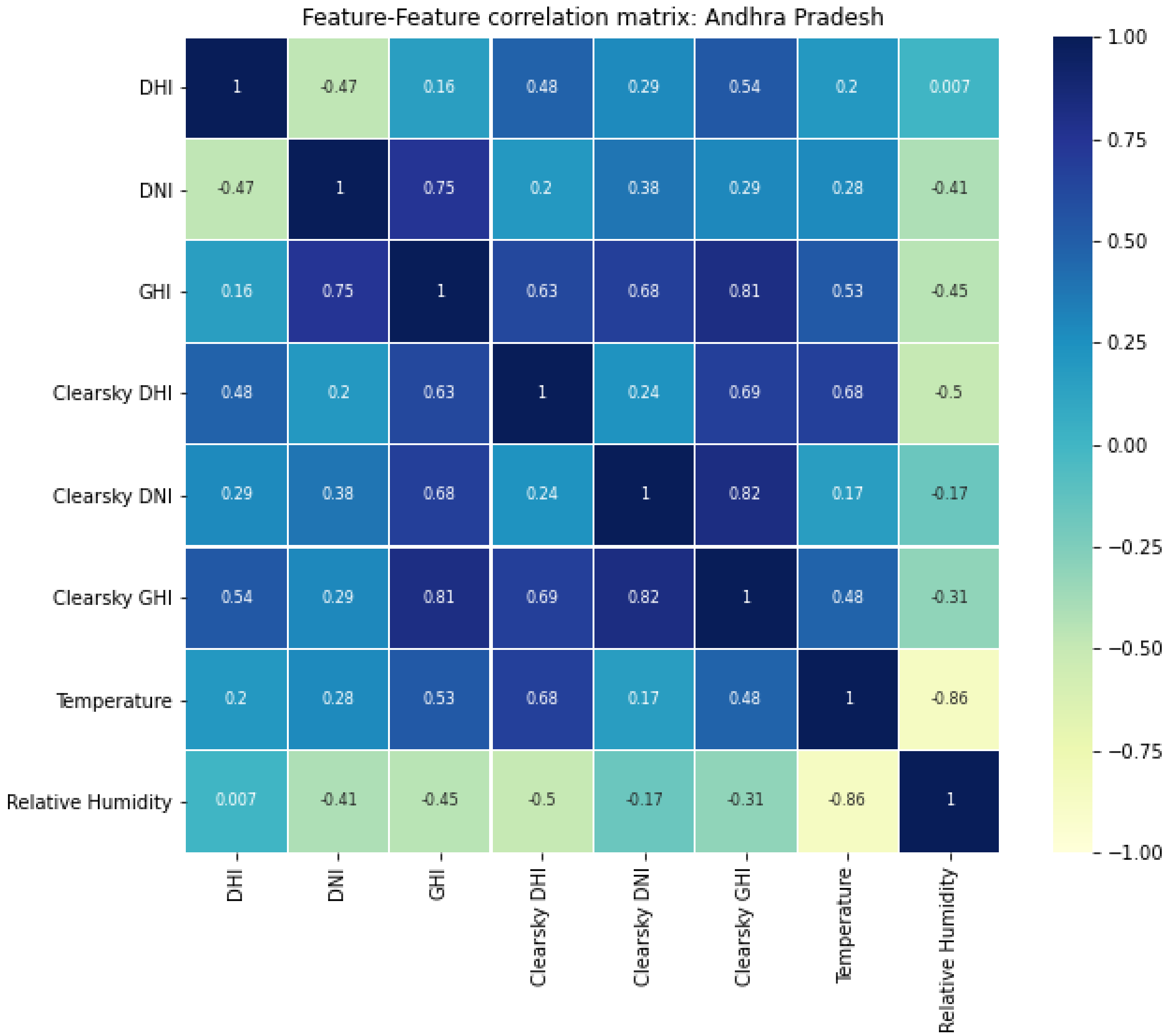
Out[18]:

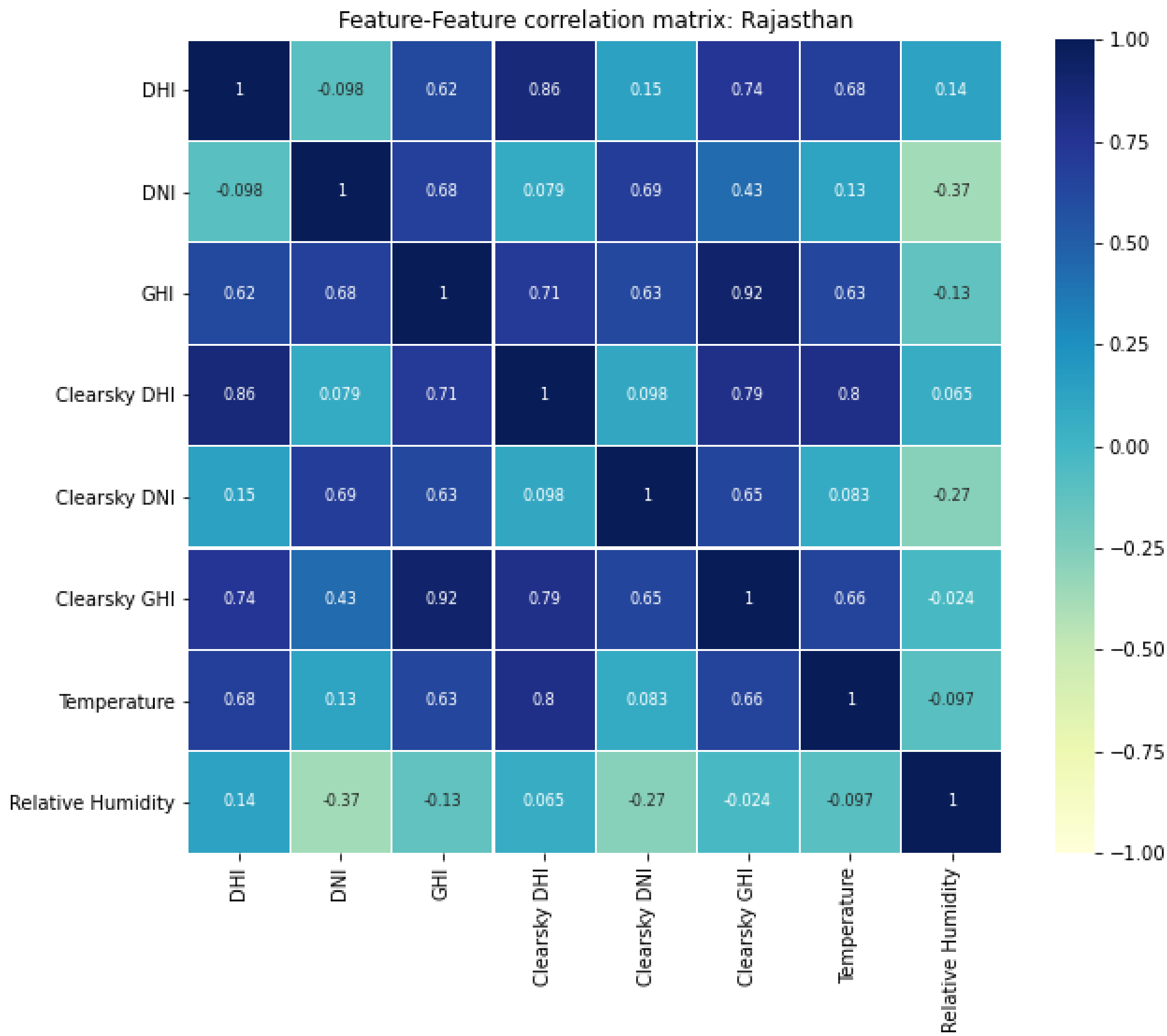
```
array([[ 1.        , -0.46918184,  0.1566238 ,  0.48290679,  0.28627141,
         0.53673211,  0.19879159,  0.00701198],
       [-0.46918184,  1.        ,  0.74869826,  0.19817873,  0.38449552,
         0.29233789,  0.28368719, -0.41080479],
       [ 0.1566238 ,  0.74869826,  1.        ,  0.62689119,  0.68248451,
         0.81260977,  0.52523771, -0.44961834],
       [ 0.48290679,  0.19817873,  0.62689119,  1.        ,  0.24012541,
         0.68683578,  0.67601009, -0.49652217],
       [ 0.28627141,  0.38449552,  0.68248451,  0.24012541,  1.        ,
         0.82162717,  0.16603896, -0.16624852],
       [ 0.53673211,  0.29233789,  0.81260977,  0.68683578,  0.82162717,
         1.        ,  0.47532186, -0.30582594],
       [ 0.19879159,  0.28368719,  0.52523771,  0.67601009,  0.16603896,
         0.47532186,  1.        , -0.85519098],
       [ 0.00701198, -0.41080479, -0.44961834, -0.49652217, -0.16624852,
        -0.30582594, -0.85519098,  1.        ]])
```

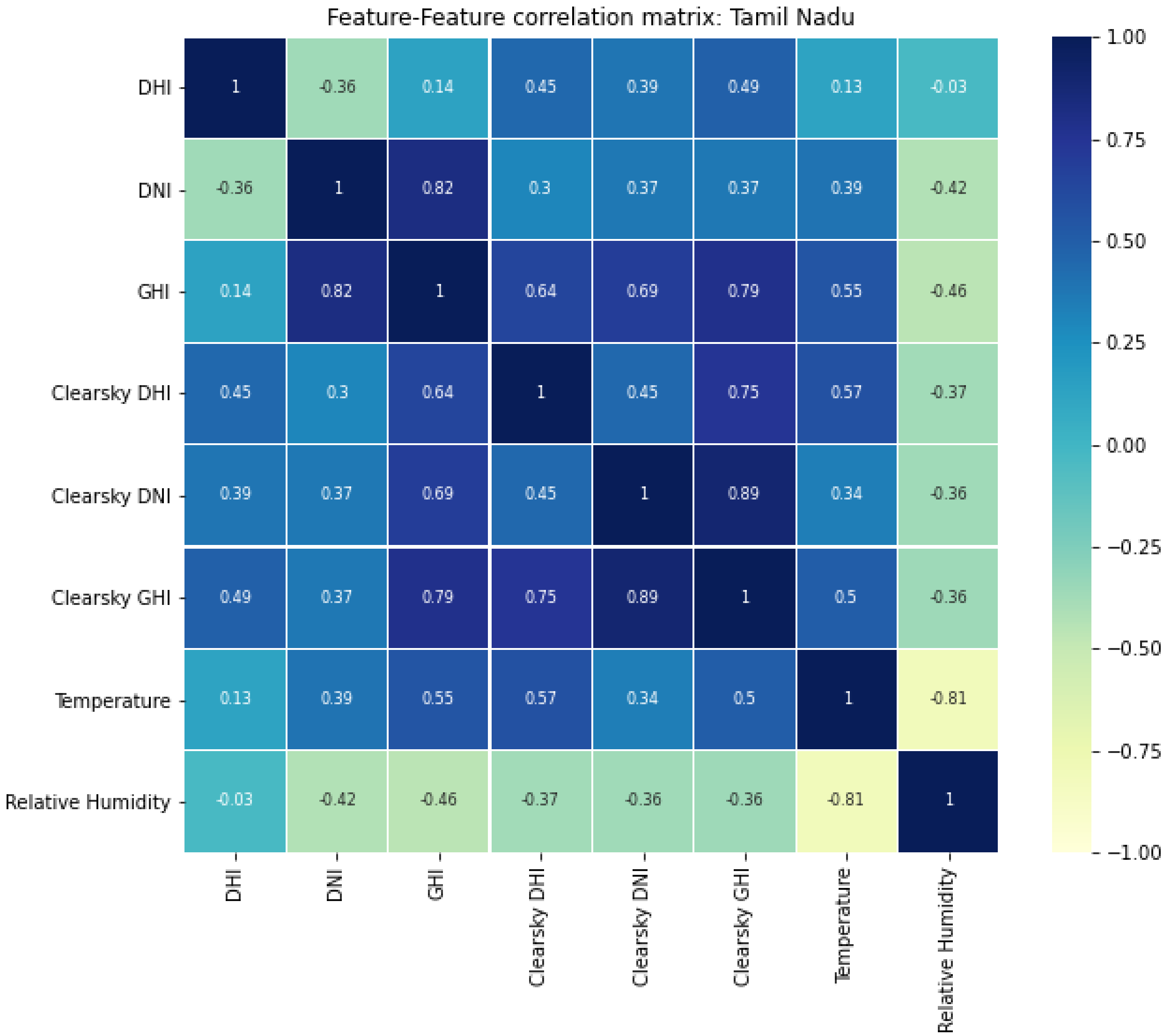
In [19]:

```
dfs = [Rajasthan_data ,AP_data , Rajasthan_data , TamilNadu_data]
for df in dfs:
  df_attributes = df[['DHI','DNI','GHI','Clearsky DHI','Clearsky DNI','Clearsky GHI','T
  corr = df_attributes.corr() # We already examined correlations , drawing a visual plo
  plt.figure(figsize=(10, 8))
  ax = plt.axes()
  ax.set_title('Feature-Feature correlation matrix: ' + str(df.name))
  sns.heatmap(corr,
              cmap='YlGnBu', vmax=1.0, vmin=-1.0, linewidths=0.1,
              annot=True, annot_kws={"size": 8}, square=True);
```


Feature-Feature correlation matrix: Rajasthan
DHI
DNI
GHI
Clearsky DHI
Clearsky DNI
Clearsky GHI
Temperature
Relative Humidity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
1 -0.098 0.62 0.86 0.15 0.74 0.68 0.14
-0.098 1 0.68 0.079 0.69 0.43 0.13 -0.37
0.62 0.68 1 0.71 0.63 0.92 0.63 -0.13
0.86 0.079 0.71 1 0.098 0.79 0.8 0.065
0.15 0.69 0.63 0.098 1 0.65 0.083 -0.27
0.74 0.43 0.92 0.79 0.65 1 0.66 -0.024
0.68 0.13 0.63 0.8 0.083 0.66 1 -0.097
0.14 -0.37 -0.13 0.065 -0.27 -0.024 -0.097 1


Feature-Feature correlation matrix: Andhra Pradesh
DHI
DNI
GHI
Clearsky DHI
Clearsky DNI
Clearsky GHI
Temperature
Relative Humidity
1 -0.47 0.16 0.48 0.29 0.54 0.2 0.007
-0.47 1 0.75 0.2 0.38 0.29 0.28 -0.41
0.16 0.75 1 0.63 0.68 0.81 0.53 -0.45
0.48 0.2 0.63 1 0.24 0.69 0.68 -0.5
0.29 0.38 0.68 0.24 1 0.82 0.17 -0.17
0.54 0.29 0.81 0.69 0.82 1 0.48 -0.31
0.2 0.28 0.53 0.68 0.17 0.48 1 -0.86
0.007 -0.41 -0.45 -0.5 -0.17 -0.31 -0.86 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Feature-Feature correlation matrix: Rajasthan
DHI 1 -0.098 0.62 0.86 0.15 0.74 0.68 0.14
DNI -0.098 1 0.68 0.079 0.69 0.43 0.13 -0.37
GHI 0.62 0.68 1 0.71 0.63 0.92 0.63 -0.13
Clearsky DHI 0.86 0.079 0.71 1 0.098 0.79 0.8 0.065
Clearsky DNI 0.15 0.69 0.63 0.098 1 0.65 0.083 -0.27
Clearsky GHI 0.74 0.43 0.92 0.79 0.65 1 0.66 -0.024
Temperature 0.68 0.13 0.63 0.8 0.083 0.66 1 -0.097
Relative Humidity 0.14 -0.37 -0.13 0.065 -0.27 -0.024 -0.097 1
DHI
DNI
GHI
Clearsky DHI
Clearsky DNI
Clearsky GHI
Temperature
Relative Humidity
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



Highly correlated attributes

- DNI and GHI correlation are 0.7487
- Clearsky GHI and GHI correlation 0.812
- Clearsky GHI and Clearsky DNI correlation 0.8216
- Relative humidity and Temperature correlation -0.855.

Stationarity Check

In [20]:

```
# Now check the data for stationarity by performing the augmented Dickey-Fuller Test.
def stationary_stat_city(test_data):
    print ('Test Statistic: ' + str(test_data[0]))
    print ('P value: ' + str(test_data[1]))
    print ('Number of Observations: '+ str(test_data[3]))
    print ('Critical Value 1%: '+ str(test_data[4]['1%']))
    print ('Critical Value 5%: '+ str(test_data[4]['5%']))
    print ('Critical Value 10%: '+ str(test_data[4]['10%']))
states = [AP_data, TamilNadu_data, Rajasthan_data]
for state in states:
  print("Checking if series stationary for state: " + state.name)
```

```
    stationary_stat_city(adfuller(state['GHI']))
    print('\n')
```

```
Checking if series stationary for state: Andhra Pradesh
Test Statistic: -16.01846079408183
P value: 6.242689338677652e-29
Number of Observations: 49217
Critical Value 1%: -3.4304828736228092
Critical Value 5%: -2.8615987273918306
Critical Value 10%: -2.566801258651965


Checking if series stationary for state: Tamil Nadu
Test Statistic: -17.989265821866113
P value: 2.757893247341827e-30
Number of Observations: 49217
Critical Value 1%: -3.4304828736228092
Critical Value 5%: -2.8615987273918306
Critical Value 10%: -2.566801258651965


Checking if series stationary for state: Rajasthan
Test Statistic: -8.824298704025445
P value: 1.8412993337683353e-14
Number of Observations: 49218
Critical Value 1%: -3.4304828709229724
Critical Value 5%: -2.8615987261985856
Critical Value 10%: -2.566801258016836
```

It is seen that the value of test statistic is greater (more negative) than the 1% critical values for each of the selected studies. Thus, we can reject the null hyothesis of a unit root being present in the data set and we can conclude with more than 99% confidence that this is a stationary series.

In [20]:

```
# Check the same with a non parametric test called KPSS
from statsmodels.tsa.stattools import kpss
def kpss_test(series, **kw):
    statistic, p_value, n_lags, critical_values = kpss(series, **kw)
    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print(f'num lags: {n_lags}')
    print('Critial Values:')
    for key, value in critical_values.items():
        print(f'   {key} : {value}')
    print(f'KPSS Result: The series is {"not " if p_value < 0.05 else ""}stationary')

kpss_test(AP_data['GHI'])
kpss_test(TamilNadu_data['GHI'])
kpss_test(MP_data['GHI'])
kpss_test(Rajasthan_data['GHI'])
```

```
KPSS Statistic: 0.08037541860064312
p-value: 0.1
num lags: 57
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
KPSS Result: The series is stationary
KPSS Statistic: 0.10936170461913289
```

```
p-value: 0.1
num lags: 57
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
KPSS Result: The series is stationary
KPSS Statistic: 0.101001359322854
p-value: 0.1
num lags: 57
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
KPSS Result: The series is stationary
KPSS Statistic: 0.19246040022355407
p-value: 0.1
num lags: 57
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
KPSS Result: The series is stationary
```

```
C:\Users\Dell\anaconda3\envs\Local\lib\site-packages\statsmodels\tsa\stattools.py:1875:
FutureWarning: The behavior of using nlags=None will change in release 0.13.Currently nl
ags=None is the same as nlags="legacy", and so a sample-size lag length is used. After t
he next release, the default will change to be the same as nlags="auto" which uses an au
tomatic lag length selection method. To silence this warning, either use "auto" or "lega
cy"
  warnings.warn(msg, FutureWarning)
C:\Users\Dell\anaconda3\envs\Local\lib\site-packages\statsmodels\tsa\stattools.py:1910:
InterpolationWarning: The test statistic is outside of the range of p-values available i
n the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\Users\Dell\anaconda3\envs\Local\lib\site-packages\statsmodels\tsa\stattools.py:1910:
InterpolationWarning: The test statistic is outside of the range of p-values available i
n the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\Users\Dell\anaconda3\envs\Local\lib\site-packages\statsmodels\tsa\stattools.py:1910:
InterpolationWarning: The test statistic is outside of the range of p-values available i
n the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
C:\Users\Dell\anaconda3\envs\Local\lib\site-packages\statsmodels\tsa\stattools.py:1910:
InterpolationWarning: The test statistic is outside of the range of p-values available i
n the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
```

Finding Distribution Of Data

In [35]:

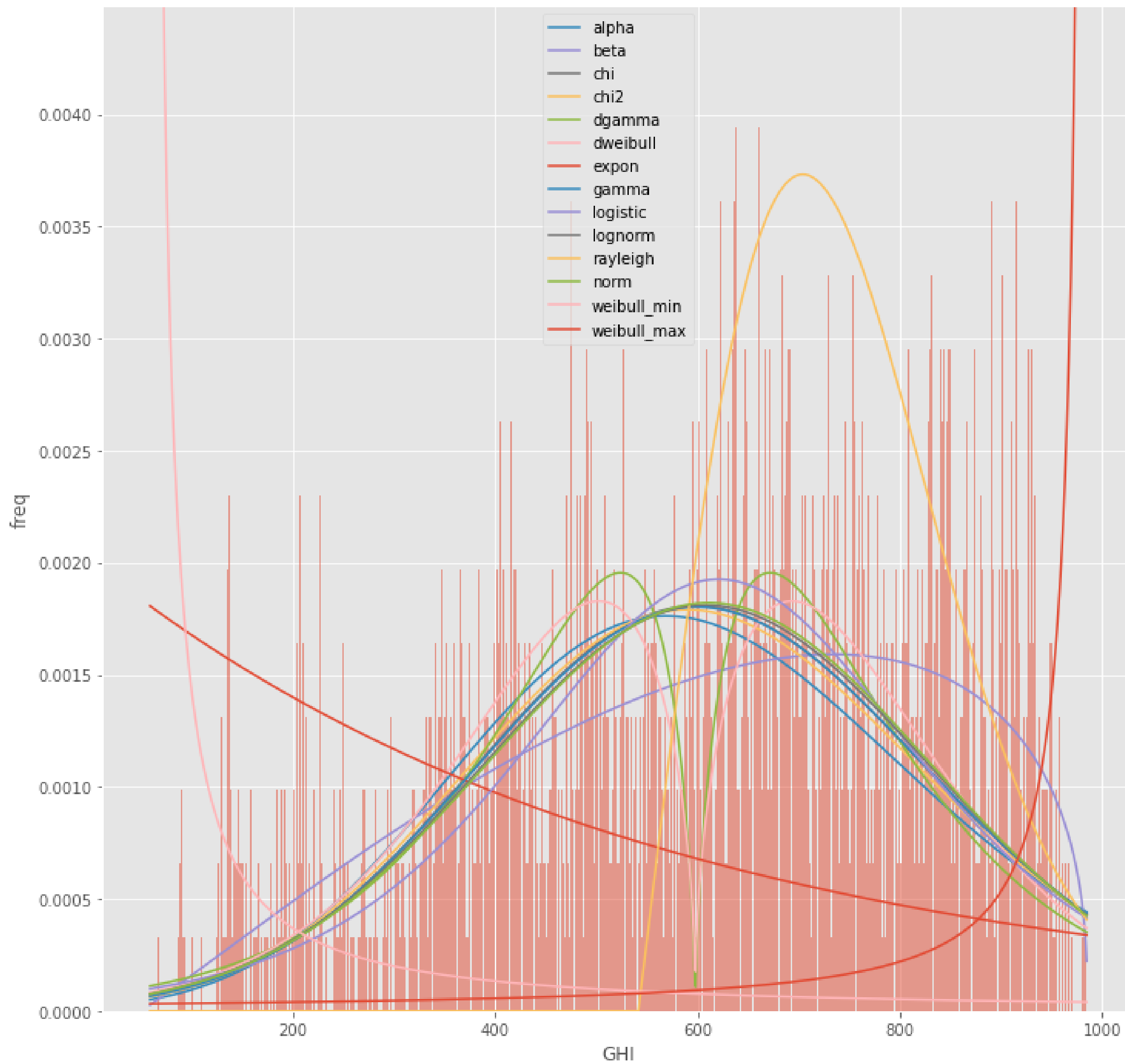
```
%matplotlib inline
```

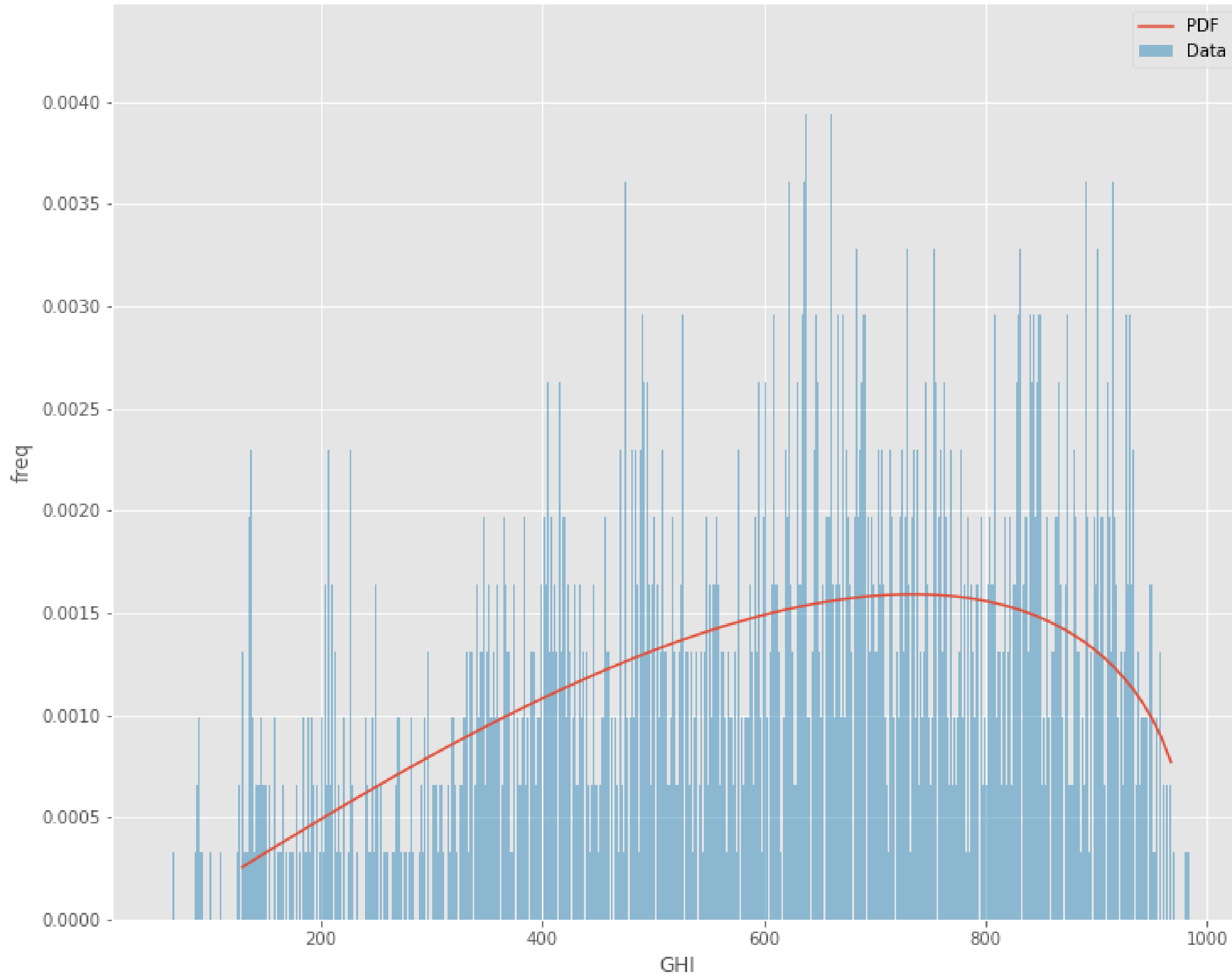
```python
import warnings
import numpy as np
import pandas as pd
import scipy.stats as st
import statsmodels as sm
import matplotlib
import matplotlib.pyplot as plt

matplotlib.rcParams['figure.figsize'] = (16.0, 12.0)
matplotlib.style.use('ggplot')
# Create models from data
bins = 1000
def best_fit_distribution(data, bins=200, ax=None):
    """Model data by finding best fit distribution to data"""
    # Get histogram of original data
    y, x = np.histogram(data, bins=bins, density=True)
    x = (x + np.roll(x, -1))[:-1] / 2.0
    # Main Distributions to check
    # Gamma, Beta, Rayleigh, Logistic, Weibull, Lognormal, Chi-Squared, and Exponential
    DISTRIBUTIONS = [
        st.alpha,st.beta,st.chi,st.chi2,st.dgamma,st.dweibull,st.expon,
        st.gamma,st.logistic,st.lognorm,st.rayleigh,
        st.norm,st.weibull_min,st.weibull_max
    ]

    best_distribution = st.norm
    best_params = (0.0, 1.0)
    best_sse = np.inf
    for distribution in DISTRIBUTIONS:
        try:
            with warnings.catch_warnings():
                warnings.filterwarnings('ignore')
                params = distribution.fit(data)
                arg = params[:-2]
                loc = params[-2]
                scale = params[-1]
                # Calculate fitted PDF and error with fit in distribution
                pdf = distribution.pdf(x, loc=loc, scale=scale, *arg)
                sse = np.sum(np.power(y - pdf, 2.0))
                # if axis pass in add to plot
                try:
                    if ax:
                        pd.Series(pdf, x).plot(ax=ax)
                    end
                except Exception:
                    pass
                # identify if this distribution is better
                if best_sse > sse > 0:
                    best_distribution = distribution
                    best_params = params
                    best_sse = sse

        except Exception:
            pass

    return (best_distribution.name, best_params)

def make_pdf(dist, params, size=100):
    """Generate distributions's Probability Distribution Function """
    # Separate parts of parameters
    arg = params[:-2]
```

```
    loc = params[-2]
    scale = params[-1]
    # Get sane start and end points of distribution
    start = dist.ppf(0.01, *arg, loc=loc, scale=scale) if arg else dist.ppf(0.01, loc=l
    end = dist.ppf(0.99, *arg, loc=loc, scale=scale) if arg else dist.ppf(0.99, loc=loc
    # Build PDF and turn into pandas Series
    x = np.linspace(start, end, size)
    y = dist.pdf(x, loc=loc, scale=scale, *arg)
    pdf = pd.Series(y, x)
    return pdf


plt.figure(figsize=(12,12))
RJ_data_yr_2000 = Rajasthan_data[(Rajasthan_data['date_time'].dt.year).isin([2000])]
data = RJ_data_yr_2000['GHI']
ax = data.plot(kind='hist', density = True, bins=bins, alpha=0.5)
dataYLim = ax.get_ylim()



# Find best fit distribution
best_fit_name, best_fit_params = best_fit_distribution(data, bins, ax)
best_dist = getattr(st, best_fit_name)
print(best_fit_name + " : This is approx distribution")
print(best_fit_params)
# Update plots
ax.set_ylim(dataYLim)
ax.set_xlabel(u'GHI')
ax.set_ylabel('freq')
ax.legend(['alpha','beta','chi','chi2','dgamma','dweibull','expon','gamma','logistic','
print(best_fit_name)
# Make PDF with best params
pdf = make_pdf(best_dist, best_fit_params)

# Display
plt.figure(figsize=(12,10))
ax = pdf.plot(label='PDF', legend=True)
data.plot(kind='hist', density = True,bins=bins,alpha=0.5, label='Data', legend=True, a

param_names = (best_dist.shapes + ', loc, scale').split(', ') if best_dist.shapes else
param_str = ', '.join(['{}={:0.2f}'.format(k,v) for k,v in zip(param_names, best_fit_pa
dist_str = '{}({})'.format(best_fit_name, param_str)

ax.set_ylim(dataYLim)
ax.set_xlabel(u'GHI')
ax.set_ylabel('freq')
```

```
beta : This is approx distribution
(2.075412630106239, 1.3936706573636997, 48.903598077258465, 936.3581006571318)
beta
```

Out[35]:
```
Text(0, 0.5, 'freq')
```


alpha
beta
chi
chi2
dgamma
dweibull
expon
gamma
logistic
lognorm
rayleigh
norm
weibull_min
weibull_max
freq
GHI
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
0.0040
200
400
600
800
1000



Time Series Decomposition

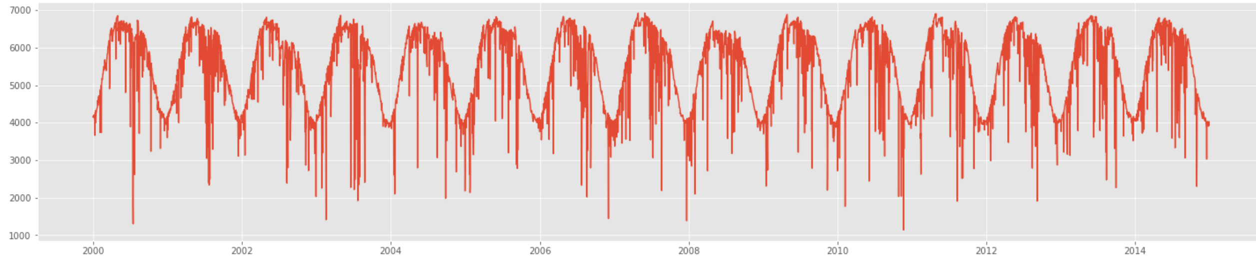
In [30]:

```
# Aggregate data to daily values to remove hourly variations
data = [Rajasthan_data['date_time'],Rajasthan_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_inde
# print(temp_df)
# monthly_AP = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_
daily_AP.set_index('date_time')
# monthly_AP.set_index('date_time')
plt.rcParams["figure.figsize"] = (25,5)

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

```
<ipython-input-30-a9f799bff734>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  daily_AP['GHI'][903] = (daily_AP['GHI'][902]+daily_AP['GHI'][904])/2
```

Out[30]:

```
[<matplotlib.lines.Line2D at 0x28e0628b5e0>]
```



In [31]:

```
data = [AP_data['date_time'],AP_data['GHI']]
headers = ["date_time","GHI"]
temp_df = pd.concat(data,axis=1,keys=headers)
daily_Andhra = temp_df.groupby(temp_df['date_time'].dt.date).agg({'GHI': 'sum'}).reset_
# print(temp_df)
# monthly_AP = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_
daily_Andhra.set_index('date_time')
```

Out[31]:

	GHI
date_time	
2000-01-01	5445
2000-01-02	5525
2000-01-03	5017
2000-01-04	5480
2000-01-05	5477
...	...
2014-12-27	5255
2014-12-28	3669
2014-12-29	4777
2014-12-30	5039
2014-12-31	4930

5475 rows × 1 columns

In [32]:

```
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_inde
# print(temp_df)
# monthly_AP = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_
daily_MP.set_index('date_time')
```

Out[32]:

	GHI
date_time	
2000-01-01	4328
2000-01-02	4358

	GHI
date_time	
2000-01-03	4389
2000-01-04	4382
2000-01-05	4387
...	...
2014-12-27	4287
2014-12-28	4295
2014-12-29	4281
2014-12-30	4300
2014-12-31	1650

5475 rows × 1 columns

In [33]:

```
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_inde
# print(temp_df)
# monthly_AP = temp_df.groupby(temp_df['date_time'].dt.month).agg({'GHI':'sum'}).reset_
daily_TN.set_index('date_time')
```

Out[33]:

	GHI
date_time	
2000-01-01	5559
2000-01-02	5793
2000-01-03	3240
2000-01-04	5678
2000-01-05	5633
...	...
2014-12-27	4254
2014-12-28	4433
2014-12-29	5099
2014-12-30	4689
2014-12-31	3301

5475 rows × 1 columns

In [34]:

```
# Perform Time series decomposition
```

```python
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(daily_AP['GHI'], model='additive',freq=365)
result.plot()
plt.show()
```

```
<ipython-input-34-709bcedd68b2>:3: FutureWarning: the 'freq'' keyword is deprecated, use
'period' instead
  result = seasonal_decompose(daily_AP['GHI'], model='additive',freq=365)
```