In [65]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pprint
%matplotlib inline
```

In [66]:
```
df= pd.read_excel("Building energy consumption record1.xlsx")
df
```

Out[66]:

	Datetime	building 1	building 2
0	2016-01-01 01:00:00	23.783228	23.783228
1	2016-01-01 02:00:00	23.783228	23.783228
2	2016-01-01 03:00:00	23.783228	23.783228
3	2016-01-01 04:00:00	23.783228	23.783228
4	2016-01-01 05:00:00	23.783228	23.783228
...	...	...	...
26298	2018-12-31 19:00:00	18.602723	18.602723
26299	2018-12-31 20:00:00	18.838200	18.838200
26300	2018-12-31 21:00:00	18.602723	18.602723
26301	2018-12-31 22:00:00	18.131768	18.131768
26302	2018-12-31 23:00:00	18.602723	18.602723

26303 rows × 3 columns

In [67]:
```
print("="*50)
print("Information About Dataset","\n")
print(df.info(),"\n")

print("="*50)
print("Describe the Dataset ","\n")
print(df.describe(),"\n")

print("="*50)
print("Null Values t ","\n")
print(df.isnull().sum(),"\n")
```

```
==================================================
Information About Dataset

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26303 entries, 0 to 26302
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   Datetime     26303 non-null  datetime64[ns]
 1    building 1  26303 non-null  float64       
 2    building 2  26303 non-null  float64       
dtypes: datetime64[ns](1), float64(2)
memory usage: 616.6 KB
None

==================================================
Describe the Dataset

         building 1    building 2
count  26303.000000  26303.000000
mean      25.694969     25.694969
std        6.317738      6.317738
min       15.541515     15.541515
25%       20.957498     20.957498
50%       23.783228     23.783228
75%       28.728255     28.728255
max       59.340330     59.340330

==================================================
Null Values t

Datetime       0
 building 1    0
 building 2    0
dtype: int64
```

In [68]:
```
# Extract all Data Like Year MOnth Day Time etc
dataset = df
dataset["Month"] = pd.to_datetime(df["Datetime"]).dt.month
dataset["Year"] = pd.to_datetime(df["Datetime"]).dt.year
dataset["Date"] = pd.to_datetime(df["Datetime"]).dt.date
dataset["Time"] = pd.to_datetime(df["Datetime"]).dt.time
dataset["Week"] = pd.to_datetime(df["Datetime"]).dt.week
dataset["Day"] = pd.to_datetime(df["Datetime"]).dt.day_name()
dataset = df.set_index("Datetime")
dataset.index = pd.to_datetime(dataset.index)
dataset.head(2)
```

```
C:\Users\ritik\AppData\Local\Temp\ipykernel_23372\2741105229.py:7: FutureWarn
ing: Series.dt.weekofyear and Series.dt.week have been deprecated. Please use
Series.dt.isocalendar().week instead.
  dataset["Week"] = pd.to_datetime(df["Datetime"]).dt.week
```

Out[68]:

Datetime	building 1	building 2	Month	Year	Date	Time	Week	Day
2016-01-01 01:00:00	23.783228	23.783228	1	2016	2016-01-01	01:00:00	53	Friday
2016-01-01 02:00:00	23.783228	23.783228	1	2016	2016-01-01	02:00:00	53	Friday

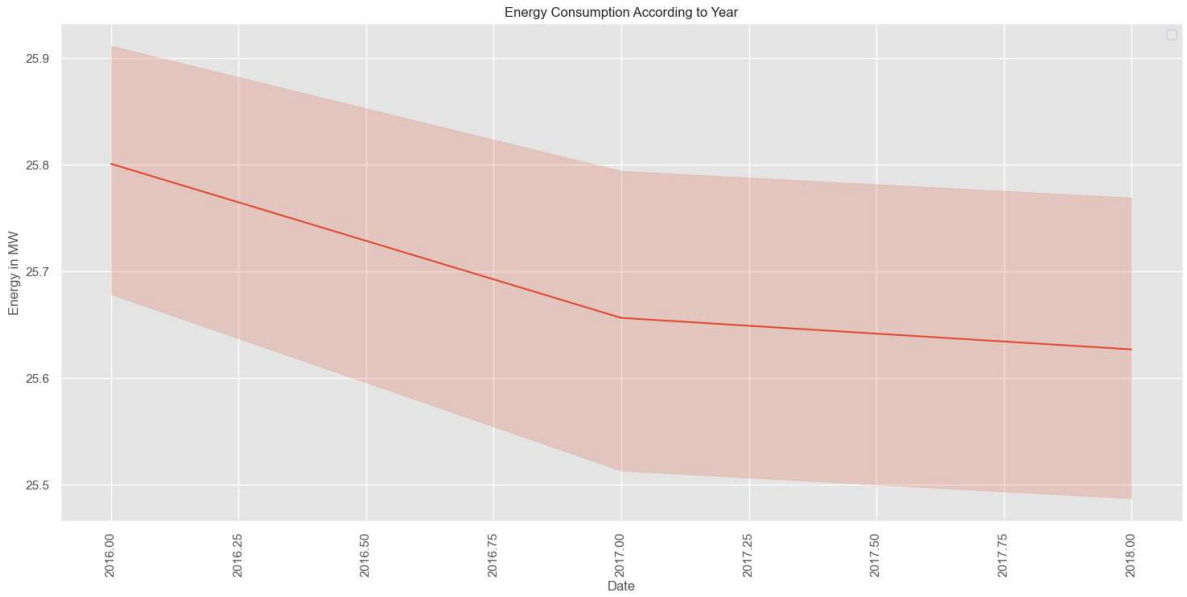
#How many Unique Year do we Have in Dataset print(df.Year.unique(),"\n") print("Total Number of Unique Year", df.Year.nunique(), "\n")

energy consumption Each Year

In [70]:
```
from matplotlib import style

fig = plt.figure()
ax1 = plt.subplot2grid((1,1), (0,0))

style.use('ggplot')

sns.lineplot(x=dataset["Year"], y=dataset[" building 1"], data=df)
sns.set(rc={'figure.figsize':(15,6)})

plt.title("Energy consumptionnin Year 2004")
plt.xlabel("Date")
plt.ylabel("Energy in MW")
plt.grid(True)
plt.legend()

for label in ax1.xaxis.get_ticklabels():
    label.set_rotation(90)


plt.title("Energy Consumption According to Year")
```

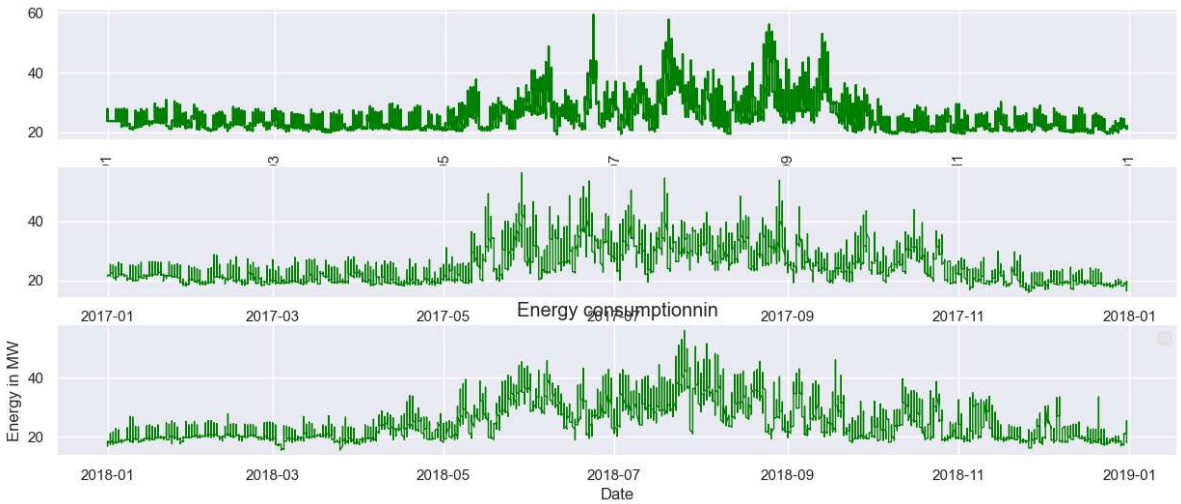
```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```

Out[70]: Text(0.5, 1.0, 'Energy Consumption According to Year')



In [71]:
```
from matplotlib import style


fig = plt.figure()

ax1= fig.add_subplot(311)
ax2= fig.add_subplot(312)
ax3= fig.add_subplot(313)


style.use('ggplot')

y_2004 = dataset["2016"][" building 1"].to_list()
x_2004 = dataset["2016"]["Date"].to_list()
ax1.plot(x_2004,y_2004, color="green", linewidth=1.7)


y_2005 = dataset["2017"][" building 1"].to_list()
x_2005 = dataset["2017"]["Date"].to_list()
ax2.plot(x_2005, y_2005, color="green", linewidth=1)


y_2006 = dataset["2018"][" building 1"].to_list()
x_2006 = dataset["2018"]["Date"].to_list()
ax3.plot(x_2006, y_2006, color="green", linewidth=1)


plt.rcParams["figure.figsize"] = (18,8)
plt.title("Energy consumptionnin")
plt.xlabel("Date")
plt.ylabel("Energy in unit)
plt.grid(True, alpha=1)
plt.legend()

for label in ax1.xaxis.get_ticklabels():
    label.set_rotation(90)
```

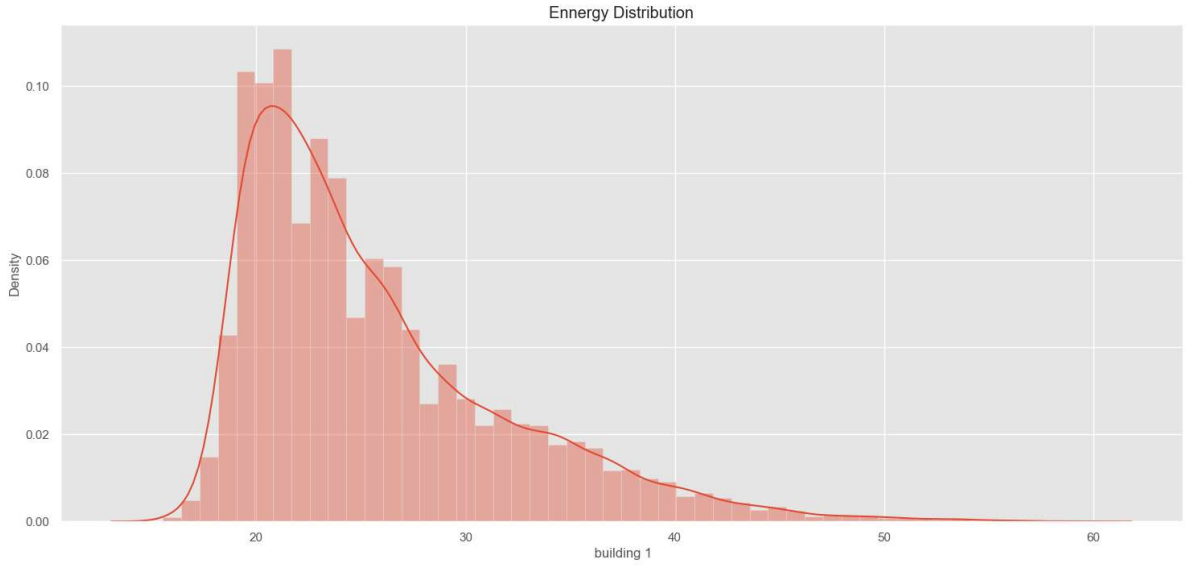
```
C:\Users\ritik\AppData\Local\Temp\ipykernel_23372\2183828767.py:13: FutureWar
ning: Indexing a DataFrame with a datetimelike index using a single string to
slice the rows, like `frame[string]`, is deprecated and will be removed in a
future version. Use `frame.loc[string]` instead.
  y_2004 = dataset["2016"][" building 1"].to_list()
C:\Users\ritik\AppData\Local\Temp\ipykernel_23372\2183828767.py:14: FutureWar
ning: Indexing a DataFrame with a datetimelike index using a single string to
slice the rows, like `frame[string]`, is deprecated and will be removed in a
future version. Use `frame.loc[string]` instead.
  x_2004 = dataset["2016"]["Date"].to_list()
C:\Users\ritik\AppData\Local\Temp\ipykernel_23372\2183828767.py:18: FutureWar
ning: Indexing a DataFrame with a datetimelike index using a single string to
slice the rows, like `frame[string]`, is deprecated and will be removed in a
future version. Use `frame.loc[string]` instead.
  y_2005 = dataset["2017"][" building 1"].to_list()
C:\Users\ritik\AppData\Local\Temp\ipykernel_23372\2183828767.py:19: FutureWar
ning: Indexing a DataFrame with a datetimelike index using a single string to
slice the rows, like `frame[string]`, is deprecated and will be removed in a
future version. Use `frame.loc[string]` instead.
  x_2005 = dataset["2017"]["Date"].to_list()
C:\Users\ritik\AppData\Local\Temp\ipykernel_23372\2183828767.py:23: FutureWar
ning: Indexing a DataFrame with a datetimelike index using a single string to
slice the rows, like `frame[string]`, is deprecated and will be removed in a
future version. Use `frame.loc[string]` instead.
  y_2006 = dataset["2018"][" building 1"].to_list()
C:\Users\ritik\AppData\Local\Temp\ipykernel_23372\2183828767.py:24: FutureWar
ning: Indexing a DataFrame with a datetimelike index using a single string to
slice the rows, like `frame[string]`, is deprecated and will be removed in a
future version. Use `frame.loc[string]` instead.
  x_2006 = dataset["2018"]["Date"].to_list()
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```



In [72]:
```
sns.distplot(dataset[" building 1"])
plt.title("Ennergy Distribution")
```

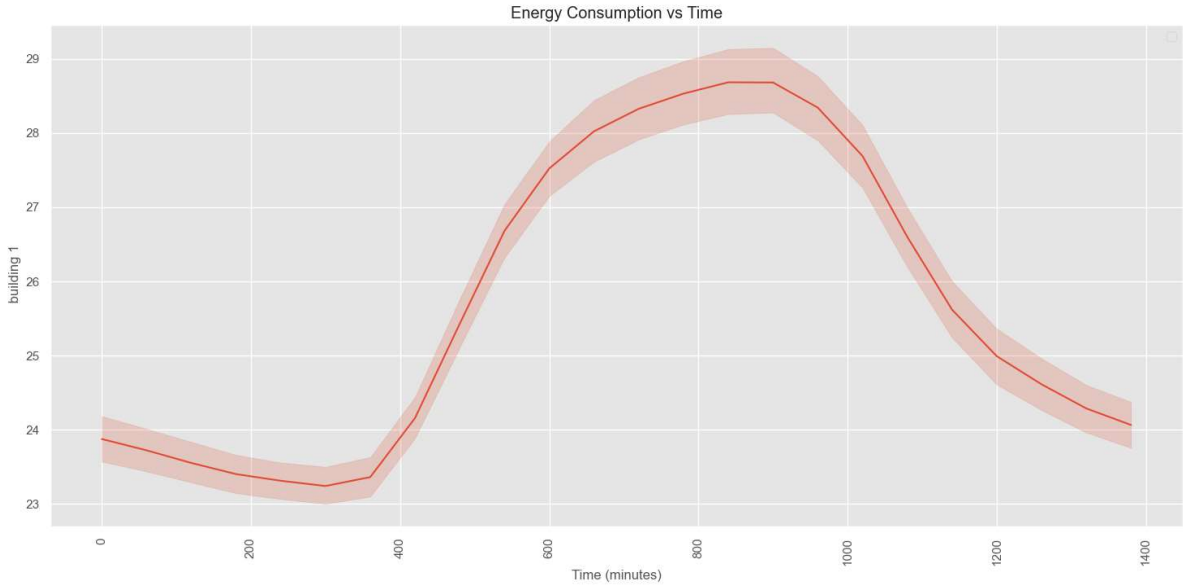
```
C:\Users\ritik\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[72]:
```
Text(0.5, 1.0, 'Ennergy Distribution')
```



In [101]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming "Time" column is in the format of datetime.time
# If it's a string, you might need to adjust the conversion accordingly
dataset["Time_numeric"] = dataset["Time"].apply(lambda x: x.hour * 60 + x.minu

fig = plt.figure()
ax1 = fig.add_subplot(111)

sns.lineplot(x="Time_numeric", y=" building 1", data=dataset, ax=ax1)  # Adjus
plt.title("Energy Consumption vs Time")
plt.xlabel("Time (minutes)")
plt.grid(True, alpha=1)
plt.legend()

for label in ax1.xaxis.get_ticklabels():
    label.set_rotation(90)

plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```



In [73]:
```
NewDataSet = dataset.resample('D').mean()
```

In [74]:
```
print("Old Dataset ",dataset.shape )
print("New  Dataset ",NewDataSet.shape )
```

```
Old Dataset  (26303, 8)
New  Dataset  (1096, 5)
```

In [75]:
```
TestData = NewDataSet.tail(100)

Training_Set = NewDataSet.iloc[:,0:1]

Training_Set = Training_Set[:-60]
```

In [76]:
```
print("Training Set Shape ", Training_Set.shape)
print("Test Set Shape ", TestData.shape)
```

```
Training Set Shape  (1036, 1)
Test Set Shape  (100, 5)
```

In [77]:
```
from sklearn.preprocessing import MinMaxScaler

Training_Set = Training_Set.values
sc = MinMaxScaler(feature_range=(0, 1))
Train = sc.fit_transform(Training_Set)
```

In [78]:
```
X_Train = []
Y_Train = []

# Range should be fromm 60 Values to END
for i in range(60, Train.shape[0]):

    # X_Train 0-59
    X_Train.append(Train[i-60:i])

    # Y Would be 60 th Value based on past 60 Values
    Y_Train.append(Train[i])

# Convert into Numpy Array
X_Train = np.array(X_Train)
Y_Train = np.array(Y_Train)

print(X_Train.shape)
print(Y_Train.shape)
```

```
(976, 60, 1)
(976, 1)
```

In [79]:
```
# Shape should be Number of [Datapoints , Steps , 1 )
# we convert into 3-d Vector or #rd Dimesnsion
X_Train = np.reshape(X_Train, newshape=(X_Train.shape[0], X_Train.shape[1], 1))
X_Train.shape
```

Out[79]:
```
(976, 60, 1)
```

In [82]:
```
from keras.models import Sequential
from keras.layers import LSTM, Dropout, Dense
regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_Train
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a third LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units = 1))

# Compiling the RNN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
WARNING:tensorflow:From C:\Users\ritik\anaconda3\lib\site-packages\keras\src
\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Pleas
e use tf.compat.v1.train.Optimizer instead.
```

In [83]:
```
regressor.fit(X_Train, Y_Train, epochs = 50, batch_size = 32)
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\ritik\anaconda3\lib\site-packages\keras\src
\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. P
lease use tf.compat.v1.ragged.RaggedTensorValue instead.

31/31 [==============================] - 15s 119ms/step - loss: 0.0379
Epoch 2/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0206
Epoch 3/50
31/31 [==============================] - 4s 120ms/step - loss: 0.0191
Epoch 4/50
31/31 [==============================] - 4s 121ms/step - loss: 0.0186
Epoch 5/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0184
Epoch 6/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0185
Epoch 7/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0185
Epoch 8/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0176
Epoch 9/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0184
Epoch 10/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0175
Epoch 11/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0178
Epoch 12/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0171
Epoch 13/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0180
Epoch 14/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0169
Epoch 15/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0162
Epoch 16/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0170
Epoch 17/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0172
Epoch 18/50
31/31 [==============================] - 4s 120ms/step - loss: 0.0157
Epoch 19/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0160
Epoch 20/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0158
Epoch 21/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0157
Epoch 22/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0162
Epoch 23/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0160
Epoch 24/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0154
Epoch 25/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0143
Epoch 26/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0143
Epoch 27/50
```

```
31/31 [==============================] - 4s 119ms/step - loss: 0.0145
Epoch 28/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0135
Epoch 29/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0129
Epoch 30/50
31/31 [==============================] - 4s 119ms/step - loss: 0.0123
Epoch 31/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0124
Epoch 32/50
31/31 [==============================] - 4s 118ms/step - loss: 0.0119
Epoch 33/50
31/31 [==============================] - 4s 120ms/step - loss: 0.0111
Epoch 34/50
31/31 [==============================] - 2s 76ms/step - loss: 0.0108
Epoch 35/50
31/31 [==============================] - 2s 69ms/step - loss: 0.0098
Epoch 36/50
31/31 [==============================] - 2s 69ms/step - loss: 0.0104
Epoch 37/50
31/31 [==============================] - 2s 69ms/step - loss: 0.0096
Epoch 38/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0098
Epoch 39/50
31/31 [==============================] - 2s 70ms/step - loss: 0.0101
Epoch 40/50
31/31 [==============================] - 2s 70ms/step - loss: 0.0098
Epoch 41/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0099
Epoch 42/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0089
Epoch 43/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0093
Epoch 44/50
31/31 [==============================] - 2s 72ms/step - loss: 0.0095
Epoch 45/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0088
Epoch 46/50
31/31 [==============================] - 2s 72ms/step - loss: 0.0092
Epoch 47/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0094
Epoch 48/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0092
Epoch 49/50
31/31 [==============================] - 2s 70ms/step - loss: 0.0093
Epoch 50/50
31/31 [==============================] - 2s 71ms/step - loss: 0.0092
```

Out[83]: <keras.src.callbacks.History at 0x2aed9bc7760>

In [84]:
```
TestData.head(2)
```

Out[84]:

Datetime	building 1	building 2	Month	Year	Week
2018-09-23	22.203566	22.203566	9.0	2018.0	38.0
2018-09-24	21.556003	21.556003	9.0	2018.0	39.0

In [85]:
```
TestData.shape
```

Out[85]:
```
(100, 5)
```

In [86]:
```
NewDataSet.shape
```

Out[86]:
```
(1096, 5)
```

In [89]:
```
Df_Total = pd.concat((NewDataSet[[" building 1"]], TestData[[" building 1"]]),
```

In [90]:
```
Df_Total.shape
```

Out[90]:
```
(1196, 1)
```

In [91]:
```
inputs = Df_Total[len(Df_Total) - len(TestData) - 60:].values
inputs.shape
```

Out[91]:
```
(160, 1)
```

In [92]:
```
inputs = Df_Total[len(Df_Total) - len(TestData) - 60:].values

# We need to Reshape
inputs = inputs.reshape(-1,1)

# Normalize the Dataset
inputs = sc.transform(inputs)

X_test = []
for i in range(60, 160):
    X_test.append(inputs[i-60:i])

# Convert into Numpy Array
X_test = np.array(X_test)

# Reshape before Passing to Network
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Pass to Model
predicted_stock_price = regressor.predict(X_test)

# Do inverse Transformation to get Values
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
4/4 [==============================] - 1s 24ms/step
```

In [94]:
```
True_MegaWatt = TestData[" building 1"].to_list()
Predicted_MegaWatt  = predicted_stock_price
dates = TestData.index.to_list()
```

In [95]:
```
Machine_Df = pd.DataFrame(data={
    "Date":dates,
    "TrueMegaWatt": True_MegaWatt,
    "PredictedMeagWatt":[x[0] for x in Predicted_MegaWatt ]
})
```

In [96]:
```
Machine_Df
```

Out[96]:

	Date	TrueMegaWatt	PredictedMeagWatt
0	2018-09-23	22.203566	20.001995
1	2018-09-24	21.556003	21.286577
2	2018-09-25	21.703176	21.671352
3	2018-09-26	22.625463	21.808964
4	2018-09-27	24.371921	22.292931
...	...	...	...
95	2018-12-27	19.044243	19.856487
96	2018-12-28	19.171793	19.913385
97	2018-12-29	17.641189	19.966343
98	2018-12-30	17.690247	19.446136
99	2018-12-31	21.291091	19.011095

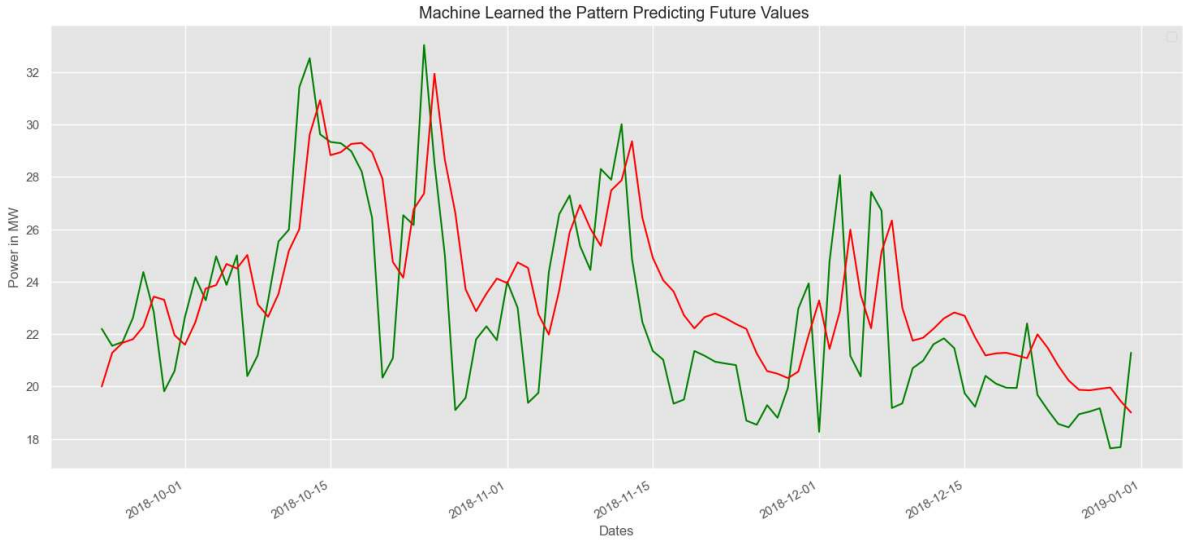
100 rows × 3 columns

In [97]:
```
True_MegaWatt = TestData[" building 1"].to_list()
Predicted_MegaWatt  = [x[0] for x in Predicted_MegaWatt ]
dates = TestData.index.to_list()
```

In [98]:
```
fig = plt.figure()
ax1= fig.add_subplot(111)
x = dates
y = True_MegaWatt
y1 = Predicted_MegaWatt
plt.plot(x,y, color="green")
plt.plot(x,y1, color="red")
# beautify the x-labels
plt.gcf().autofmt_xdate()
plt.xlabel('Dates')
plt.ylabel("Power in MW")
plt.title("Machine Learned the Pattern Predicting Future Values ")
plt.legend()
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argumen
t.
```

Out[98]:
```
<matplotlib.legend.Legend at 0x2aee4859df0>
```



In [ ]: