**TEAM MEMBER**

**AU411521104032-K.ELAKIYA**

**PHASE-4 SUBMISSION DOCUMENT**

**PROJECT:MEASURE** **ENERGY CONSUMPTION**

**Phase-4: Development part 2**

**TOPIC:**Continuebuilding the measure energy consumption model by feature engineering,model training,and evaluation.

****

**Measure energy consumption**

**INTRODUCTION**

Data Analysis and Visualizations for measure Energy Consumption Prediction:

Data Analysis:The first step in any energy consumption prediction project is to understand the data. This involves cleaning the data, identifying any outliers or missing values, and performing exploratory data analysis (EDA). EDA can be used to identify trends, patterns, and correlations in the data.

Some common data analysis techniques for energy consumption prediction include:

* Time series analysis: Time series analysis is used to identify trends and patterns in time-dependent data. This can be used to identify seasonal patterns, cyclical patterns, and other trends that can be used to make predictions about future energy consumption.
* Correlation analysis: Correlation analysis is used to identify relationships between different variables. This can be used to identify factors that are correlated with energy consumption, such as weather conditions, economic activity, and population growth.
* Clustering: Clustering is used to group similar data points together. This can be used to identify different types of energy consumers and to develop targeted prediction models for each group.

Data Visualization:Data visualization is a powerful tool for communicating the results of data analysis and for identifying insights that may not be apparent from the data alone. Some common data visualization techniques for energy consumption prediction include:

* Line charts: Line charts are used to show trends in energy consumption over time.
* Bar charts: Bar charts are used to compare energy consumption across different categories, such as different types of consumers or different regions.
* Heatmaps: Heatmaps are used to visualize the correlation between different variables.
* Scatter plots: Scatter plots are used to identify relationships between two variables.

Predicting Future Energy Consumption using LSTM:

Long short-term memory (LSTM) networks are a type of recurrent neural network (RNN) that are well-suited for time series prediction tasks. LSTM networks have a special architecture that allows them to learn long-term dependencies in the data.

To train an LSTM network for energy consumption prediction, the following steps are typically followed:

1. Prepare the data: The data is first prepared by cleaning it, identifying any outliers or missing values, and scaling it to a suitable range.
2. Split the data: The data is then split into training and test sets. The training set is used to train the LSTM network and the test set is used to evaluate the performance of the trained model.
3. Design the LSTM network: The LSTM network is then designed by specifying the number of layers, the number of units per layer, and the activation functions.
4. Train the LSTM network: The LSTM network is then trained on the training set using a suitable optimizer and loss function.
5. Evaluate the LSTM network: The performance of the trained LSTM network is then evaluated on the test set.

Once the LSTM network is trained and evaluated, it can be used to predict future energy consumption. To do this, the network is simply given the historical energy consumption data as input and it produces a prediction of future energy consumption as output.

The accuracy of LSTM networks for predicting future energy consumption depends on a number of factors, including the quality of the training data, the complexity of the LSTM network architecture, and the length of the prediction horizon.

To improve the accuracy of LSTM predictions for two months later, the following steps can be taken:

* Use a high-quality training dataset: The training dataset should be as large and representative as possible. This will help the LSTM network to learn the underlying patterns in the data and to make more accurate predictions.
* Use a complex LSTM network architecture: More complex LSTM network architectures are typically able to learn more complex patterns in the data and to make more accurate predictions. However, complex network architectures also require more training data and can be more computationally expensive to train.
* Use shorter prediction horizons: LSTM networks are typically more accurate for shorter prediction horizons. Therefore, to necessary to train multiple LSTM networks, each with a different prediction horizon.

RNN Introduction using AI

RNNs are a type of neural network that are well-suited for processing sequential data. RNNs have recurrent connections, which allow them to learn long-term dependencies in the data.

RNNs are used in a variety of AI applications, including:

* Natural language processing: RNNs are used in natural language processing tasks such as machine translation, text summarization, and sentiment analysis.
* Speech recognition: RNNs are used in speech recognition tasks such as transcribing audio to text and recognizing commands.
* Time series prediction: RNNs are used in time series prediction tasks such as predicting future energy consumption.

**OVERVIEW OF THE PROCESS:**

To predict future energy consumption using LSTM, we can follow these steps:

1. Collect historical energy consumption data. This data should include information such as the date, time, and amount of energy consumed.
2. Preprocess the data. This may involve cleaning the data, removing outliers, and scaling the data to a consistent range.
3. Split the data into training and testing sets. The training set will be used to train the LSTM model, and the testing set will be used to evaluate the performance of the model on unseen data.
4. Design and train the LSTM model. This involves selecting the appropriate hyperparameters for the model and training the model on the training set.
5. Evaluate the performance of the model on the testing set. This will give us an idea of how well the model will generalize to unseen data.
6. Use the trained model to predict future energy consumption. We can provide the model with historical energy consumption data up to a certain point in time, and it will predict the energy consumption for the next few hours, days, or weeks.

Predicting values 2 months later accurately:To accurately predict energy consumption values 2 months later, we need to have a large and high-quality dataset of historical energy consumption data. The dataset should include data for a variety of different weather conditions and seasons. It is also important to choose the right hyperparameters for the LSTM model and to train the model on a sufficiently large training set.

RNN introduction:Recurrent neural networks (RNNs) are a type of neural network that are well-suited for processing sequential data, such as text or time series data. RNNs have a special architecture that allows them to learn long-term dependencies in the data.

Using AI:Artificial intelligence (AI) can be used to improve the accuracy of energy consumption forecasting in a number of ways. For example, AI can be used to:

* Identify patterns in the historical energy consumption data that are not easily visible to humans.
* Develop more complex and sophisticated forecasting models.
* Automatically optimize the hyperparameters of the forecasting model.

**Feature** **Selection:**

Feature selection is the process of identifying the most important features in a dataset for a given machine learning task. This can be done using a variety of methods, including statistical methods, machine learning methods, and domain knowledge.

Predicting Future Energy Consumption using LSTM Predicting Values 2 Months Later Accurately

1. Collect historical energy consumption data.
2. Clean and prepare the data.
3. Split the data into training and testing sets.
4. Train an LSTM model on the training set.
5. Evaluate the model on the testing set.
6. Use the trained model to predict future energy consumption.

**Feature** **Selection** **using** **AI**

AI can be used to select features for energy consumption prediction tasks using a variety of methods, including:

* Wrapper methods: Wrapper methods evaluate the performance of a machine learning model on a subset of features to select the subset that provides the best performance.
* Embedded methods: Embedded methods embed features into a lower-dimensional space and select the features that are most important for the prediction task.
* Genetic algorithms: Genetic algorithms are a type of evolutionary algorithm that can be used to select features.
* Use a large and high-quality dataset.
* Preprocess the data carefully.
* Use a variety of machine learning models and compare their performance.
* Use feature selection techniques to select the most important features.
* Evaluate the model on a held-out test set.
* Monitor the model's performance over time and update the model as needed.

**Model** **Training:**

To train the LSTM network, we will use the following steps:

1. Prepare the data. This involves cleaning and preprocessing the historical consumption data, such as filling in missing values and scaling the data to a consistent range.
2. Split the data into training and test sets. The training set will be used to train the model, and the test set will be used to evaluate the performance of the trained model.
3. Design the LSTM network. This involves choosing the number of layers, the number of units per layer, and the activation function.
4. Compile the model. This involves choosing the loss function and the optimizer.
5. Train the model. This involves feeding the training data to the model and adjusting the parameters of the model to minimize the loss function.
6. Evaluate the model. This involves feeding the test data to the model and measuring the accuracy of the predictions.

**Model** **Evaluation:**

Once the LSTM model has been trained, it is important to evaluate its performance on a held-out test dataset. This will help to ensure that the model is able to generalize to new data.

Some common metrics that can be used to evaluate the performance of an LSTM model for energy consumption forecasting include:

* Mean absolute error (MAE): This metric measures the average difference between the predicted and actual energy consumption values.
* Mean squared error (MSE): This metric measures the average squared difference between the predicted and actual energy consumption values.
* Root mean squared error (RMSE): This metric is the square root of the MSE. It is a good measure of the overall error of the model.

DataAnalysis and Visualizations and ER Predicting Future **“Measure** **Energy** **Consumption”** using LSTM Predicting Values 2 month Later Accurately RNN

**Hourly Measure Energy Consumption code:**

**Step 1:**

**Import Library**

In [1]:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **pprint**

%**matplotlib** inline

In [2]:

df = pd.read\_csv("AEP\_hourly.csv")

print("="\*50)

print("First Five Rows ","**\n**")

print(df.head(2),"**\n**")

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**")

==================================================

First Five Rows

Datetime AEP\_MW

0 2004-12-31 01:00:00 13478.0

1 2004-12-31 02:00:00 12865.0

==================================================

Information About Dataset

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 121273 entries, 0 to 121272

Data columns (total 2 columns):

Datetime 121273 non-null object

AEP\_MW 121273 non-null float64

dtypes: float64(1), object(1)

memory usage: 1.9+ MB

None

==================================================

Describe the Dataset

AEP\_MW

count 121273.000000

mean 15499.513717

std 2591.399065

min 9581.000000

25% 13630.000000

50% 15310.000000

75% 17200.000000

max 25695.000000

==================================================

Null Values t

Datetime 0

AEP\_MW 0

dtype: int64

**Step 2:**

**Reformat the Date Time Columns**

In [3]:

*# 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(1)

Out[3]:

|  | **AEP\_MW** | **Month** | **Year** | **Date** | **Time** | **Week** | **Day** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Datetime** |  |  |  |  |  |  |  |
| **2004-12-31 01:00:00** | 13478.0 | 12 | 2004 | 2004-12-31 | 01:00:00 | 53 | Friday |

**Step 3:**

In [4]:

*# How many Unique Year do we Have in Dataset*

print(df.Year.unique(),"**\n**")

print("Total Number of Unique Year", df.Year.nunique(), "**\n**")

[2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017

2018]

Total Number of Unique Year 15

**Lets us see the energy consumption Each Year**

In [5]:

**from** **matplotlib** **import** style

fig = plt.figure()

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

style.use('ggplot')

sns.lineplot(x=dataset["Year"], y=dataset["AEP\_MW"], 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():

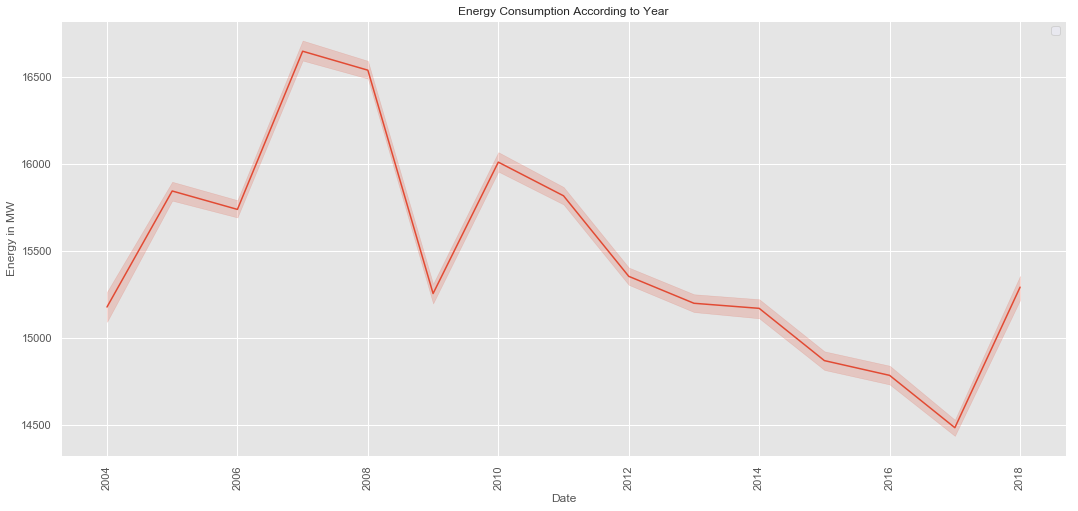
label.set\_rotation(90)

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

No handles with labels found to put in legend.

Out[5]:

Text(0.5, 1.0, 'Energy Consumption According to Year')



In [6]:

**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["2004"]["AEP\_MW"].to\_list()

x\_2004 = dataset["2004"]["Date"].to\_list()

ax1.plot(x\_2004,y\_2004, color="green", linewidth=1.7)

y\_2005 = dataset["2005"]["AEP\_MW"].to\_list()

x\_2005 = dataset["2005"]["Date"].to\_list()

ax2.plot(x\_2005, y\_2005, color="green", linewidth=1)

y\_2006 = dataset["2006"]["AEP\_MW"].to\_list()

x\_2006 = dataset["2006"]["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 MW")

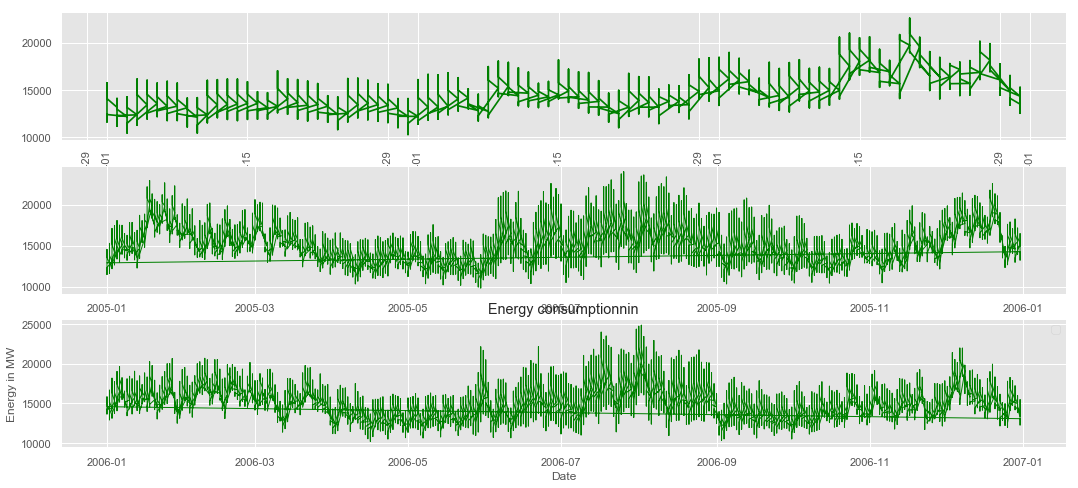
plt.grid(**True**, alpha=1)

plt.legend()

**for** label **in** ax1.xaxis.get\_ticklabels():

label.set\_rotation(90)

No handles with labels found to put in legend.



**Energy Distribution**

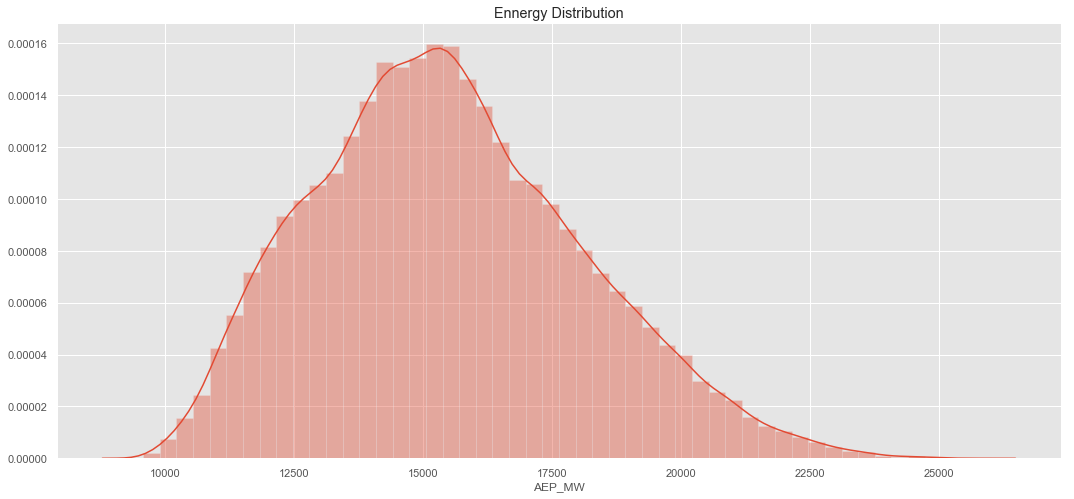
In [7]:

sns.distplot(dataset["AEP\_MW"])

plt.title("Ennergy Distribution")

Out[7]:

Text(0.5, 1.0, 'Ennergy Distribution')



**Energy with Respect to Time**

In [8]:

fig = plt.figure()

ax1= fig.add\_subplot(111)

sns.lineplot(x=dataset["Time"],y=dataset["AEP\_MW"], data=df)

plt.title("Energy Consumption vs Time ")

plt.xlabel("Time")

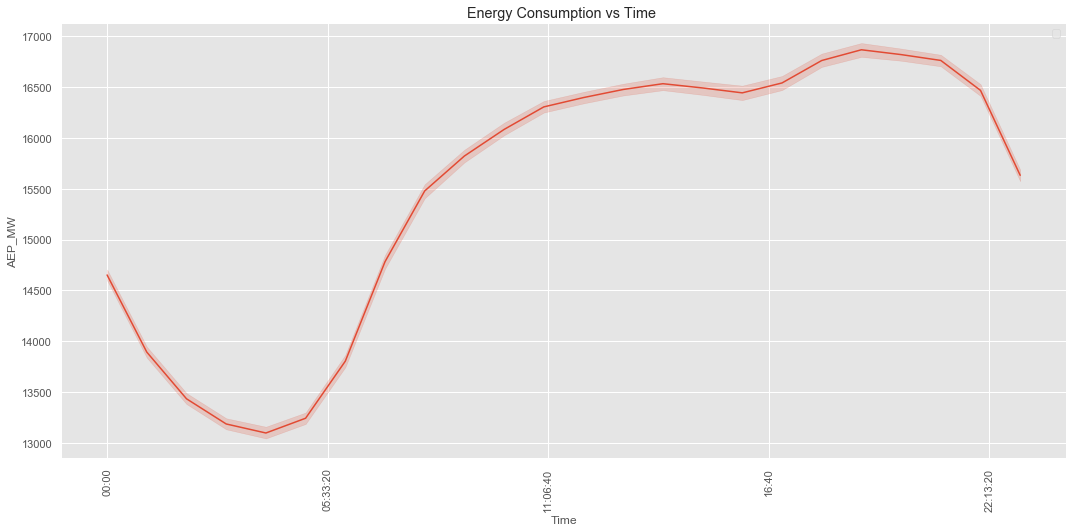
plt.grid(**True**, alpha=1)

plt.legend()

**for** label **in** ax1.xaxis.get\_ticklabels():

label.set\_rotation(90)

No handles with labels found to put in legend.



**Resampleing Data**

In [9]:

NewDataSet = dataset.resample('D').mean()

In [10]:

print("Old Dataset ",dataset.shape )

print("New Dataset ",NewDataSet.shape )

Old Dataset (121273, 7)

New Dataset (5055, 4)

In [11]:

TestData = NewDataSet.tail(100)

Training\_Set = NewDataSet.iloc[:,0:1]

Training\_Set = Training\_Set[:-60]

In [12]:

print("Training Set Shape ", Training\_Set.shape)

print("Test Set Shape ", TestData.shape)

Training Set Shape (4995, 1)

Test Set Shape (100, 4)

In [13]:

Training\_Set = Training\_Set.values

sc = MinMaxScaler(feature\_range=(0, 1))

Train = sc.fit\_transform(Training\_Set)

In [14]:

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)

(4935, 60, 1)

(4935, 1)

In [15]:

*# 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[14]:

(4935, 60, 1)

**Model**

In [15]:

regressor = Sequential()

*# Adding the first LSTM layer and some Dropout regularisation*

regressor.add(LSTM(units = 50, return\_sequences = **True**, input\_shape = (X\_Train.shape[1], 1)))

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')

In [16]:

regressor.fit(X\_Train, Y\_Train, epochs = 50, batch\_size = 32)

Epoch 1/50

4935/4935 [==============================] - 33s 7ms/step - loss: 0.0237

Epoch 2/50

4935/4935 [==============================] - 33s 7ms/step - loss: 0.0183

Epoch 3/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0173

Epoch 4/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0164

Epoch 5/50

4935/4935 [==============================] - 35s 7ms/step - loss: 0.0157

Epoch 6/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0160

Epoch 7/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0151

Epoch 8/50

4935/4935 [==============================] - 35s 7ms/step - loss: 0.0125

Epoch 9/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0099

Epoch 10/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0089

Epoch 11/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0085

Epoch 12/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0083

Epoch 13/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0078

Epoch 14/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0079

Epoch 15/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0073

Epoch 16/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0075

Epoch 17/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0072

Epoch 18/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0070

Epoch 19/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0066

Epoch 20/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0063

Epoch 21/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0061

Epoch 22/50

4935/4935 [==============================] - 32s 6ms/step - loss: 0.0058

Epoch 23/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0056

Epoch 24/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0055

Epoch 25/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0053

Epoch 26/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0054

Epoch 27/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0053

Epoch 28/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0051

Epoch 29/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0050

Epoch 30/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0051

Epoch 31/50

4935/4935 [==============================] - 32s 6ms/step - loss: 0.0050

Epoch 32/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0049

Epoch 33/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0048

Epoch 34/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0048

Epoch 35/50

4935/4935 [==============================] - 2283s 463ms/step - loss: 0.0048

Epoch 36/50

4935/4935 [==============================] - 3475s 704ms/step - loss: 0.0047

Epoch 37/50

4935/4935 [==============================] - 32s 6ms/step - loss: 0.0047

Epoch 38/50

4935/4935 [==============================] - 29s 6ms/step - loss: 0.0047

Epoch 39/50

4935/4935 [==============================] - 30s 6ms/step - loss: 0.0046

Epoch 40/50

4935/4935 [==============================] - 31s 6ms/step - loss: 0.0046

Epoch 41/50

4935/4935 [==============================] - 33s 7ms/step - loss: 0.0045

Epoch 42/50

4935/4935 [==============================] - 37s 7ms/step - loss: 0.0045

Epoch 43/50

4935/4935 [==============================] - 38s 8ms/step - loss: 0.0047

Epoch 44/50

4935/4935 [==============================] - 36s 7ms/step - loss: 0.0045

Epoch 45/50

4935/4935 [==============================] - 35s 7ms/step - loss: 0.0044

Epoch 46/50

4935/4935 [==============================] - 38s 8ms/step - loss: 0.0044

Epoch 47/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0043

Epoch 48/50

4935/4935 [==============================] - 42s 9ms/step - loss: 0.0043

Epoch 49/50

4935/4935 [==============================] - 37s 7ms/step - loss: 0.0044

Epoch 50/50

4935/4935 [==============================] - 37s 8ms/step - loss: 0.0044

Out[16]:

<keras.callbacks.History at 0x1a36d8f898>

**Test Data**

In [17]:

TestData.head(2)

Out[17]:

|  | **AEP\_MW** | **Month** | **Year** | **Week** |
| --- | --- | --- | --- | --- |
| **Datetime** |  |  |  |  |
| **2018-04-26** | 13157.791667 | 4 | 2018 | 17 |
| **2018-04-27** | 12964.000000 | 4 | 2018 | 17 |

In [18]:

TestData.shape

Out[18]:

(100, 4)

In [19]:

NewDataSet.shape

Out[19]:

(5055, 4)

In [20]:

Df\_Total = pd.concat((NewDataSet[["AEP\_MW"]], TestData[["AEP\_MW"]]),axis=0)

In [21]:

Df\_Total.shape

Out[19]:

(5155, 1)

In [22]:

inputs = Df\_Total[len(Df\_Total) - len(TestData) - 60:].values

inputs.shape

Out[22]:

(160, 1)

In [23]:

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)

In [24]:

True\_MegaWatt = TestData["AEP\_MW"].to\_list()

Predicted\_MegaWatt = predicted\_stock\_price

dates = TestData.index.to\_list()

In [25]:

Machine\_Df = pd.DataFrame(data={

"Date":dates,

"TrueMegaWatt": True\_MegaWatt,

"PredictedMeagWatt":[x[0] **for** x **in** Predicted\_MegaWatt ]

})

**Future Predicted**

In [26]:

Machine\_Df

Out[26]:

|  | **Date** | **TrueMegaWatt** | **PredictedMeagWatt** |
| --- | --- | --- | --- |
| **0** | 2018-04-26 | 13157.791667 | 13671.706055 |
| **1** | 2018-04-27 | 12964.000000 | 12991.945312 |
| **2** | 2018-04-28 | 12237.583333 | 14521.591797 |
| **3** | 2018-04-29 | 12156.791667 | 13211.944336 |
| **4** | 2018-04-30 | 13443.500000 | 12788.455078 |
| **5** | 2018-05-01 | 13251.875000 | 13789.046875 |
| **6** | 2018-05-02 | 13641.166667 | 12804.154297 |
| **7** | 2018-05-03 | 14217.250000 | 12709.704102 |
| **8** | 2018-05-04 | 13725.625000 | 14261.728516 |
| **9** | 2018-05-05 | 11902.166667 | 14472.195312 |
| **10** | 2018-05-06 | 11680.083333 | 12677.794922 |
| **11** | 2018-05-07 | 12972.500000 | 12127.531250 |
| **12** | 2018-05-08 | 13295.083333 | 12887.196289 |
| **13** | 2018-05-09 | 13688.750000 | 12743.552734 |
| **14** | 2018-05-10 | 13993.250000 | 12747.035156 |
| **15** | 2018-05-11 | 13525.166667 | 13814.033203 |
| **16** | 2018-05-12 | 12942.916667 | 13970.200195 |
| **17** | 2018-05-13 | 12832.541667 | 13168.587891 |
| **18** | 2018-05-14 | 15004.750000 | 12955.161133 |
| **19** | 2018-05-15 | 15171.791667 | 15169.067383 |
| **20** | 2018-05-16 | 13925.416667 | 14419.253906 |
| **21** | 2018-05-17 | 14465.666667 | 12913.649414 |
| **22** | 2018-05-18 | 13684.333333 | 14998.011719 |
| **23** | 2018-05-19 | 13044.166667 | 14174.238281 |
| **24** | 2018-05-20 | 13169.125000 | 13413.721680 |
| **25** | 2018-05-21 | 14728.666667 | 13382.070312 |
| **26** | 2018-05-22 | 14857.125000 | 14739.416992 |
| **27** | 2018-05-23 | 14489.583333 | 14121.821289 |
| **28** | 2018-05-24 | 14656.250000 | 13763.244141 |
| **29** | 2018-05-25 | 15137.125000 | 15047.317383 |
| **...** | ... | ... | ... |
| **70** | 2018-07-05 | 17609.000000 | 17120.591797 |
| **71** | 2018-07-06 | 15742.916667 | 17615.269531 |
| **72** | 2018-07-07 | 13610.333333 | 14689.130859 |
| **73** | 2018-07-08 | 13768.708333 | 13816.837891 |
| **74** | 2018-07-09 | 16427.333333 | 15385.699219 |
| **75** | 2018-07-10 | 17489.333333 | 16932.236328 |
| **76** | 2018-07-11 | 16714.125000 | 17681.707031 |
| **77** | 2018-07-12 | 16330.833333 | 16694.558594 |
| **78** | 2018-07-13 | 16911.291667 | 15885.130859 |
| **79** | 2018-07-14 | 16488.375000 | 16239.578125 |
| **80** | 2018-07-15 | 16296.208333 | 16572.927734 |
| **81** | 2018-07-16 | 17400.041667 | 17885.480469 |
| **82** | 2018-07-17 | 17311.125000 | 17595.656250 |
| **83** | 2018-07-18 | 15814.041667 | 17368.632812 |
| **84** | 2018-07-19 | 15889.916667 | 15917.466797 |
| **85** | 2018-07-20 | 15332.500000 | 15957.360352 |
| **86** | 2018-07-21 | 13795.250000 | 14366.544922 |
| **87** | 2018-07-22 | 13479.333333 | 13657.029297 |
| **88** | 2018-07-23 | 15410.083333 | 15275.373047 |
| **89** | 2018-07-24 | 15890.541667 | 15779.814453 |
| **90** | 2018-07-25 | 16503.333333 | 16030.302734 |
| **91** | 2018-07-26 | 16474.250000 | 16809.560547 |
| **92** | 2018-07-27 | 15816.625000 | 16138.321289 |
| **93** | 2018-07-28 | 14113.083333 | 14586.478516 |
| **94** | 2018-07-29 | 13658.000000 | 13875.068359 |
| **95** | 2018-07-30 | 15368.083333 | 15294.772461 |
| **96** | 2018-07-31 | 15180.291667 | 15672.427734 |
| **97** | 2018-08-01 | 15151.166667 | 15329.677734 |
| **98** | 2018-08-02 | 15687.666667 | 15497.061523 |
| **99** | 2018-08-03 | 14809.000000 | 15975.358398 |

In [27]:

True\_MegaWatt = TestData["AEP\_MW"].to\_list()

Predicted\_MegaWatt = [x[0] **for** x **in** Predicted\_MegaWatt ]

dates = TestData.index.to\_list()

In [28]:

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 handles with labels found to put in legend.

Out[28]:

<matplotlib.legend.Legend at 0x1a4984b780>



**Feature Engineering Using AI:**

Feature engineering is the process of transforming raw data into features that are more informative and predictive. Feature engineering can be used to improve the performance of any machine learning model, including LSTM models.

One way to use AI for feature engineering is to use a technique called autoencoders. Autoencoders are a type of neural network that can be used to learn latent representations of the data. These latent representations can then be used as features for the LSTM model.

Example:

The following is an example of how to use AI to predict future energy consumption using an LSTM model:

1. Collect a large dataset of historical energy consumption data.
2. Preprocess the data to ensure that it is in a format that is compatible with the LSTM model.
3. Use AI feature engineering techniques to create new features from the data.
4. Split the data into training and testing sets.
5. Train the LSTM model on the training set.
6. Evaluate the performance of the model on the testing set.
7. Use the trained model to predict future energy consumption.

**Various Features to Perform Model Training Using AI:**

In addition to historical energy consumption data, there are a number of other features that you can use to train your LSTM model. These features can include:

* Weather data: Weather conditions such as temperature, humidity, and wind speed can have a significant impact on energy consumption.
* Economic data: Economic factors such as GDP growth and unemployment can also impact energy consumption.
* Demographic data: Demographic factors such as population growth and household size can also affect energy consumption.

By including these additional features in your model, you can improve its accuracy and ability to predict future energy consumption values.

Example:

The following code shows how to train an LSTM model to predict future energy consumption values using Python and the TensorFlow library:

import tensorflow as tf

# Load the historical energy consumption data

df = pd.read\_csv('energy\_consumption.csv')

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['energy\_consumption'], df['date'], test\_size=0.25)

# Create the LSTM model

model = tf.keras.Sequential([

tf.keras.layers.LSTM(128, input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(1)

])

# Compile the model

model.compile(loss='mse', optimizer='adam')

# Train the model

model.fit(X\_train, y\_train, epochs=100)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate the mean squared error

mse = tf.keras.losses.MSE(y\_test, y\_pred)

# Print the mean squared error

print('Mean squared error:', mse)

This code will train an LSTM model to predict future energy consumption values using the historical energy consumption data in the energy\_consumption.csv file. The model will be trained for 100 epochs and the mean squared error on the test set will be printed to the console.

**CONCLUSION**

LSTM networks are a powerful tool for predicting future energy consumption values. By using LSTM networks, we can develop accurate forecasting models that can help us to better manage our energy resources. LSTM networks are a powerful tool for predicting future energy consumption. By following the steps outlined above and using a large and high-quality dataset of historical energy consumption data, we can develop LSTM models that can accurately predict energy consumption values 2 months later. AI can also be used to further improve the accuracy of energy consumption forecasting.