electrical-power194160consumption

October 17, 2024

1 Exploring Time Series Analysis of Residential Electrical Power Consumption

[1]:
```
# Import necessary libraries and packages
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Set floating point precision option for pandas
#pd.set_option('display.float_format', lambda x: '%.4f' % x)

# Import seaborn library and set context and style
import seaborn as sns
#sns.set_context("paper", font_scale=1.3)
#sns.set_style('white')

# Import warnings and set filter to ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Import time library
from time import time

# Import matplotlib ticker and scipy stats
import matplotlib.ticker as tkr
from scipy import stats

# Import statistical tools for time series analysis
from statsmodels.tsa.stattools import adfuller

# Import preprocessing from sklearn
from sklearn import preprocessing

# Import partial autocorrelation function from statsmodels
from statsmodels.tsa.stattools import pacf

# Enable inline plotting in Jupyter Notebook
```

```
%matplotlib inline

# Import math library
import math

# Import necessary functions from keras
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import *
import warnings
warnings.filterwarnings('ignore')

# Import MinMaxScaler from sklearn
from sklearn.preprocessing import MinMaxScaler

# Import mean squared error and mean absolute error from sklearn
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# Import early stopping from keras callbacks
from keras.callbacks import EarlyStopping
```

```
Using TensorFlow backend.
```

[2]:
```
# Load the data from the file 'household_power_consumption.txt' using pandas
# and specify the delimiter as ';'
data = pd.read_csv('household_power_consumption.txt', delimiter=';')
```

[3]:
```
# Print the number of rows and columns in the data
print('Number of rows and columns:', data.shape)

# Display the first 5 rows of the data
data.head(5)
```

```
Number of rows and columns: (5195, 9)
```

[3]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage	\
0	16/12/2006	17:24:00	4.216	0.418	234.84	
1	16/12/2006	17:25:00	5.360	0.436	233.63	
2	16/12/2006	17:26:00	5.374	0.498	233.29	
3	16/12/2006	17:27:00	5.388	0.502	233.74	
4	16/12/2006	17:28:00	3.666	0.528	235.68	

	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3

```
0              18.4              0.0              1.0              17.0
1              23.0              0.0              1.0              16.0
2              23.0              0.0              2.0              17.0
3              23.0              0.0              1.0              17.0
4              15.8              0.0              1.0              17.0
```

[4]:
```
# Display the last 5 rows of the data
data.tail(5)
```

[4]:
```
            Date      Time  Global_active_power  Global_reactive_power  \
5190  20/12/2006  07:54:00                2.532                  0.196
5191  20/12/2006  07:55:00                2.522                  0.196
5192  20/12/2006  07:56:00                2.832                  0.192
5193  20/12/2006  07:57:00                3.050                  0.208
5194  20/12/2006  07:58:00                2.982                  0.220

      Voltage  Global_intensity  Sub_metering_1  Sub_metering_2  \
5190   239.66              10.6             0.0             0.0
5191   239.66              10.6             0.0             0.0
5192   239.47              11.8             0.0             0.0
5193   240.31              12.6             0.0             0.0
5194   240.63              12.4             0.0             0.0

      Sub_metering_3
5190            17.0
5191            18.0
5192            17.0
5193            18.0
5194            18.0
```

[5]:
```
# Get the information about the dataframe
print("\nInformation about the dataframe:\n\n")
print(data.info())
```

```
Information about the dataframe:


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5195 entries, 0 to 5194
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Date                   5195 non-null   object
 1   Time                   5195 non-null   object
 2   Global_active_power    5195 non-null   float64
 3   Global_reactive_power  5195 non-null   float64
```

```
 4   Voltage                5195 non-null   float64
 5   Global_intensity       5195 non-null   float64
 6   Sub_metering_1         5195 non-null   float64
 7   Sub_metering_2         5195 non-null   float64
 8   Sub_metering_3         5195 non-null   float64
dtypes: float64(7), object(2)
memory usage: 365.4+ KB
None
```

[6]:
```
# Get the data type of each column in the dataframe
print("\nData type of each column in the dataframe:")
print(data.dtypes)
```

```
Data type of each column in the dataframe:
Date                    object
Time                    object
Global_active_power    float64
Global_reactive_power  float64
Voltage                float64
Global_intensity       float64
Sub_metering_1         float64
Sub_metering_2         float64
Sub_metering_3         float64
dtype: object
```

2 FEATURE ENGINEERING

[7]:
```
# Convert the 'Date' and 'Time' columns to a single 'date_time' column
# by combining the two columns and converting to datetime format
data['date_time'] = pd.to_datetime(data['Date'] + ' ' + data['Time'])
```

[8]:
```
# Convert the 'Global_active_power' column to numeric format
# and remove any rows with NaN values
data['Global_active_power'] = pd.to_numeric(data['Global_active_power'],␣
 ↪errors='coerce')
data = data.dropna(subset=['Global_active_power'])
```

[9]:
```
# Convert the 'date_time' column to datetime format
data['date_time'] = pd.to_datetime(data['date_time'])
data['date_time']
```

[9]:
```
0      2006-12-16 17:24:00
1      2006-12-16 17:25:00
2      2006-12-16 17:26:00
3      2006-12-16 17:27:00
4      2006-12-16 17:28:00
```

```
                ...
5190   2006-12-20 07:54:00
5191   2006-12-20 07:55:00
5192   2006-12-20 07:56:00
5193   2006-12-20 07:57:00
5194   2006-12-20 07:58:00
Name: date_time, Length: 5195, dtype: datetime64[ns]
```

[10]:
```
# Create new columns for year, quarter, month, and day
data['year'] = data['date_time'].apply(lambda x: x.year)
data['quarter'] = data['date_time'].apply(lambda x: x.quarter)
data['month'] = data['date_time'].apply(lambda x: x.month)
data['day'] = data['date_time'].apply(lambda x: x.day)
```

[11]:
```
# Keep only the columns 'date_time', 'Global_active_power', 'year', 'quarter',␣
 ↪'month', 'day'
data = data.loc[:,['date_time','Global_active_power',␣
 ↪'year','quarter','month','day']]
```

[12]:
```
# Sort the data by date_time in ascending order
data.sort_values('date_time', inplace=True, ascending=True)
```

[13]:
```
# Reset the index of the data
data = data.reset_index(drop=True)
```

[14]:
```
# Create a new column 'weekday' that indicates if the day is a weekday (1) or␣
 ↪weekend (0)
data['weekday'] = data['date_time'].apply(lambda x: x.weekday() < 5).astype(int)
```

[15]:
```
# Print the number of rows and columns in the data
print('Number of rows and columns:', data.shape)

# Print the minimum and maximum date_time values
print('Minimum date_time:', data.date_time.min())
print('Maximum date_time:', data.date_time.max())

# Display the last 5 rows of the data
data.tail(5)
```

```
Number of rows and columns: (5195, 7)
Minimum date_time: 2006-12-16 17:24:00
Maximum date_time: 2006-12-20 07:58:00
```

[15]:

	date_time	Global_active_power	year	quarter	month	day	\
5190	2006-12-20 07:54:00	2.532	2006	4	12	20	
5191	2006-12-20 07:55:00	2.522	2006	4	12	20	
5192	2006-12-20 07:56:00	2.832	2006	4	12	20	

```
5193 2006-12-20 07:57:00                3.050  2006        4     12   20
5194 2006-12-20 07:58:00                2.982  2006        4     12   20

      weekday
5190        1
5191        1
5192        1
5193        1
5194        1
```

[16]:
```
data
```

[16]:
```
                date_time  Global_active_power  year  quarter  month  day  \
0     2006-12-16 17:24:00                4.216  2006        4     12   16
1     2006-12-16 17:25:00                5.360  2006        4     12   16
2     2006-12-16 17:26:00                5.374  2006        4     12   16
3     2006-12-16 17:27:00                5.388  2006        4     12   16
4     2006-12-16 17:28:00                3.666  2006        4     12   16
…                     …                  …     …      …     …  …
5190 2006-12-20 07:54:00                2.532  2006        4     12   20
5191 2006-12-20 07:55:00                2.522  2006        4     12   20
5192 2006-12-20 07:56:00                2.832  2006        4     12   20
5193 2006-12-20 07:57:00                3.050  2006        4     12   20
5194 2006-12-20 07:58:00                2.982  2006        4     12   20

      weekday
0           0
1           0
2           0
3           0
4           0
…       …
5190        1
5191        1
5192        1
5193        1
5194        1

[5195 rows x 7 columns]
```

[17]:
```
data1=data.loc[:,['date_time','Global_active_power']]
data1.set_index('date_time',inplace=True)
```

3 Modelling and Evaluation

[18]:
```
#Transform the Global_active_power column of the data DataFrame into a numpy␣
↪array of float values

dataset = data.Global_active_power.values.astype('float32')
#Reshape the numpy array into a 2D array with 1 column

dataset = np.reshape(dataset, (-1, 1))
#Create an instance of the MinMaxScaler class to scale the values between 0 and␣
↪1

scaler = MinMaxScaler(feature_range=(0, 1))
#Fit the MinMaxScaler to the transformed data and transform the values

dataset = scaler.fit_transform(dataset)
#Split the transformed data into a training set (80%) and a test set (20%)

train_size = int(len(dataset) * 0.80)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
```

[19]:
```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)
```

[20]:
```
# reshape into X=t and Y=t+1
look_back = 30
X_train, Y_train = create_dataset(train, look_back)
X_test, Y_test = create_dataset(test, look_back)
```

[21]:
```
X_train.shape
```

[21]:
```
(4125, 30)
```

[22]:
```
Y_train.shape
```

[22]:
```
(4125,)
```

[23]:
```
# reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

[24]:
```
X_train
```

[24]:
```
array([[[0.5260267 , 0.67564744, 0.67747843, …, 0.3968088 ,
         0.4007324 , 0.39026943]],

       [[0.67564744, 0.67747843, 0.6793095 , …, 0.4007324 ,
         0.39026943, 0.33036885]],

       [[0.67747843, 0.6793095 , 0.45409364, …, 0.39026943,
         0.33036885, 0.46612608]],

       …,

       [[0.02328015, 0.01569448, 0.01307873, …, 0.0238033 ,
         0.0185718 , 0.01569448]],

       [[0.01569448, 0.01307873, 0.01569448, …, 0.0185718 ,
         0.01569448, 0.01804865]],

       [[0.01307873, 0.01569448, 0.01360188, …, 0.01569448,
         0.01804865, 0.01490976]]], dtype=float32)
```

[25]:
```
Y_train
```

[25]:
```
array([0.33036885, 0.46612608, 0.5425059 , …, 0.01804865, 0.01490976,
       0.01726393], dtype=float32)
```

4 LSTM model

[26]:
```
# Defining the LSTM model
model = Sequential()

# Adding the first layer with 100 LSTM units and input shape of the data
model.add(LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2])))

# Adding a dropout layer to avoid overfitting
model.add(Dropout(0.2))

# Adding a dense layer with 1 unit to make predictions
model.add(Dense(1))

# Compiling the model with mean squared error as the loss function and using␣
 ↪Adam optimizer
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
# Fitting the model on training data and using early stopping to avoid␣
 ↪overfitting
history = model.fit(X_train, Y_train, epochs=50, batch_size=1240,␣
 ↪validation_data=(X_test, Y_test),
                    callbacks=[EarlyStopping(monitor='val_loss', patience=4)],␣
 ↪verbose=1, shuffle=False)

# Displaying a summary of the model
model.summary()
```

```
WARNING:tensorflow:From C:\Users\abhil\anaconda3\envs\tensorflow\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 4125 samples, validate on 1008 samples
Epoch 1/50
4125/4125 [==============================] - 1s 157us/step - loss: 0.0543 -
val_loss: 0.0100
Epoch 2/50
4125/4125 [==============================] - 0s 20us/step - loss: 0.0217 -
val_loss: 0.0052
Epoch 3/50
4125/4125 [==============================] - 0s 20us/step - loss: 0.0129 -
val_loss: 0.0065
Epoch 4/50
4125/4125 [==============================] - 0s 18us/step - loss: 0.0157 -
val_loss: 0.0069
Epoch 5/50
4125/4125 [==============================] - 0s 18us/step - loss: 0.0153 -
val_loss: 0.0054
Epoch 6/50
4125/4125 [==============================] - 0s 18us/step - loss: 0.0117 -
val_loss: 0.0044
Epoch 7/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0106 -
val_loss: 0.0043
Epoch 8/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0109 -
val_loss: 0.0042
Epoch 9/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0108 -
val_loss: 0.0039
Epoch 10/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0099 -
val_loss: 0.0036
Epoch 11/50
4125/4125 [==============================] - 0s 17us/step - loss: 0.0090 -
```

```
val_loss: 0.0035
Epoch 12/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0088 -
val_loss: 0.0034
Epoch 13/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0086 -
val_loss: 0.0033
Epoch 14/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0084 -
val_loss: 0.0031
Epoch 15/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0080 -
val_loss: 0.0029
Epoch 16/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0080 -
val_loss: 0.0028
Epoch 17/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0076 -
val_loss: 0.0027
Epoch 18/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0076 -
val_loss: 0.0027
Epoch 19/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0073 -
val_loss: 0.0026
Epoch 20/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0072 -
val_loss: 0.0025
Epoch 21/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0070 -
val_loss: 0.0025
Epoch 22/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0071 -
val_loss: 0.0024
Epoch 23/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0070 -
val_loss: 0.0024
Epoch 24/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0068 -
val_loss: 0.0023
Epoch 25/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0068 -
val_loss: 0.0023
Epoch 26/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0067 -
val_loss: 0.0023
Epoch 27/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0066 -
```

```
val_loss: 0.0022
Epoch 28/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0066 -
val_loss: 0.0022
Epoch 29/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0065 -
val_loss: 0.0022
Epoch 30/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0063 -
val_loss: 0.0021
Epoch 31/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0062 -
val_loss: 0.0021
Epoch 32/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0062 -
val_loss: 0.0021
Epoch 33/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0062 -
val_loss: 0.0021
Epoch 34/50
4125/4125 [==============================] - 0s 17us/step - loss: 0.0062 -
val_loss: 0.0020
Epoch 35/50
4125/4125 [==============================] - 0s 16us/step - loss: 0.0061 -
val_loss: 0.0020
Epoch 36/50
4125/4125 [==============================] - 0s 15us/step - loss: 0.0061 -
val_loss: 0.0020
Epoch 37/50
4125/4125 [==============================] - 0s 17us/step - loss: 0.0060 -
val_loss: 0.0020
Epoch 38/50
4125/4125 [==============================] - 0s 18us/step - loss: 0.0059 -
val_loss: 0.0020
Epoch 39/50
4125/4125 [==============================] - 0s 19us/step - loss: 0.0059 -
val_loss: 0.0019
Epoch 40/50
4125/4125 [==============================] - 0s 20us/step - loss: 0.0057 -
val_loss: 0.0019
Epoch 41/50
4125/4125 [==============================] - 0s 18us/step - loss: 0.0058 -
val_loss: 0.0019
Epoch 42/50
4125/4125 [==============================] - 0s 19us/step - loss: 0.0057 -
val_loss: 0.0019
Epoch 43/50
4125/4125 [==============================] - 0s 20us/step - loss: 0.0056 -
```

```
val_loss: 0.0019
Epoch 44/50
4125/4125 [==============================] - 0s 20us/step - loss: 0.0056 -
val_loss: 0.0019
Epoch 45/50
4125/4125 [==============================] - 0s 18us/step - loss: 0.0056 -
val_loss: 0.0019
Epoch 46/50
4125/4125 [==============================] - 0s 18us/step - loss: 0.0055 -
val_loss: 0.0018
Epoch 47/50
4125/4125 [==============================] - 0s 22us/step - loss: 0.0056 -
val_loss: 0.0018
Epoch 48/50
4125/4125 [==============================] - 0s 19us/step - loss: 0.0054 -
val_loss: 0.0018
Epoch 49/50
4125/4125 [==============================] - 0s 17us/step - loss: 0.0055 -
val_loss: 0.0018
Epoch 50/50
4125/4125 [==============================] - 0s 17us/step - loss: 0.0054 -
val_loss: 0.0018
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               52400
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 52,501
Trainable params: 52,501
Non-trainable params: 0
_________________________________________________________________
```

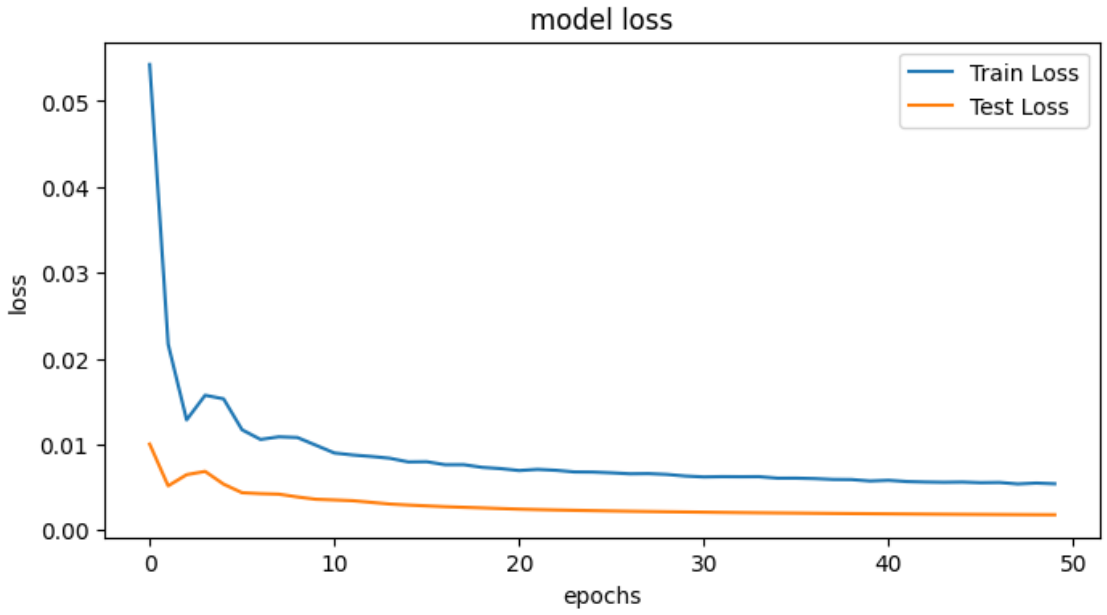
5 Evaluation

[27]:
```
# make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
# invert predictions
train_predict = scaler.inverse_transform(train_predict)
Y_train = scaler.inverse_transform([Y_train])
test_predict = scaler.inverse_transform(test_predict)
Y_test = scaler.inverse_transform([Y_test])
```

```
print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0],␣
 ↪train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0],␣
 ↪train_predict[:,0])))
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:
 ↪,0]))
print('Test Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_test[0],␣
 ↪test_predict[:,0])))
```

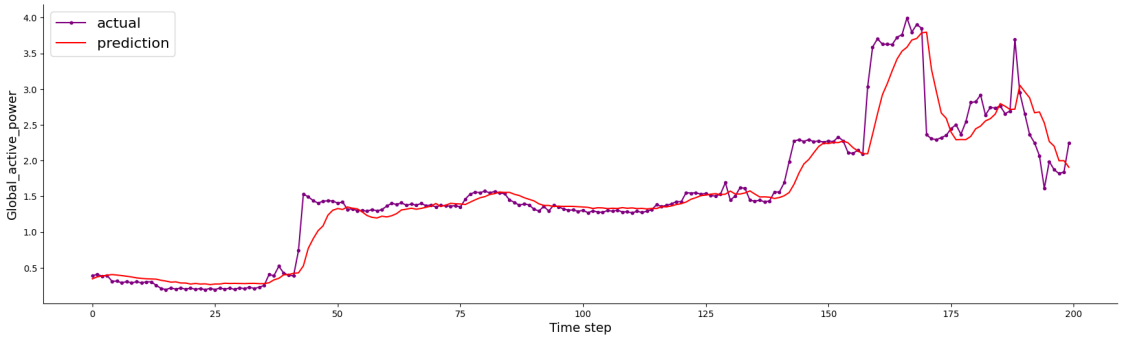
```
Train Mean Absolute Error: 0.31377592386897496
Train Root Mean Squared Error: 0.5394772785251718
Test Mean Absolute Error: 0.17077471124254273
Test Root Mean Squared Error: 0.3247452886525539
```

[28]:
```
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show();
```



[29]:
```
aa=[x for x in range(200)]
# Creating a figure object with desired figure size
```

```
plt.figure(figsize=(20,6))

# Plotting the actual values in blue with a dot marker
plt.plot(aa, Y_test[0][:200], marker='.', label="actual", color='purple')

# Plotting the predicted values in green with a solid line
plt.plot(aa, test_predict[:,0][:200], '-', label="prediction", color='red')

# Removing the top spines
sns.despine(top=True)

# Adjusting the subplot location
plt.subplots_adjust(left=0.07)

# Labeling the y-axis
plt.ylabel('Global_active_power', size=14)

# Labeling the x-axis
plt.xlabel('Time step', size=14)

# Adding a legend with font size of 15
plt.legend(fontsize=16)

# Display the plot
plt.show()
```



[ ]: