AI_PHASE 2

# MEASURE ENERGY CONSUMPTION

| | |
|---|---|
| **DATE** | **10 October 2023** |
| **TEAM MEMBER** | **PRADEEP V.** |
| **REG NO** | **950621104066** |
| **TEAM ID** | **PROJ_212174_TEAM_4** |
| **PROJECT NAME** | **MEASURE ENERGY CONSUMPTION** |
| **MAXIMUM MARKS** | |

#Importing the packages needed for the above

given problem import numpy as np  import

pandas as pd

#importing the necessary packages and libraries for the above

given problems import numpy as np from numpy import

concatenate

import urllib.request as urllib

from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OneHotEncoder

from sklearn.model_selection import

train_test_split from sklearn.metrics

import mean_squared_error   from

keras.models import Sequential from

keras.layers import Dense

#importing seaborn and

matplot libraries import

seaborn as sns import

matplotlib.pyplot as plt from

math import sqrt

#importing the required dataset libraries from

sklearn.metrics import

mean_squared_error,mean_absolute_error from

keras.models import Sequential

from keras.layers import

Dense,Dropout from

keras.layers import LSTM

color_pal =

sns.color_palette()

#importing the dataset “PJME_hourly.csv “ file to create a table for

the datetime of the dataset data =

pd.read_csv('PJME_hourly.csv',index_col=[0], parse_dates=[0])

data.head();

OUTPUT:

| | **PJME_MW** |
|---|---|
| **Datetime** | |
| **2002-12-31 01:00:00** | 26498.0 |
| **2002-12-31 02:00:00** | 25147.0 |
| **2002-12-31 03:00:00** | 24574.0 |
| **2002-12-31 04:00:00** | 24393.0 |
| **2002-12-31 05:00:00** | 24860.0 |

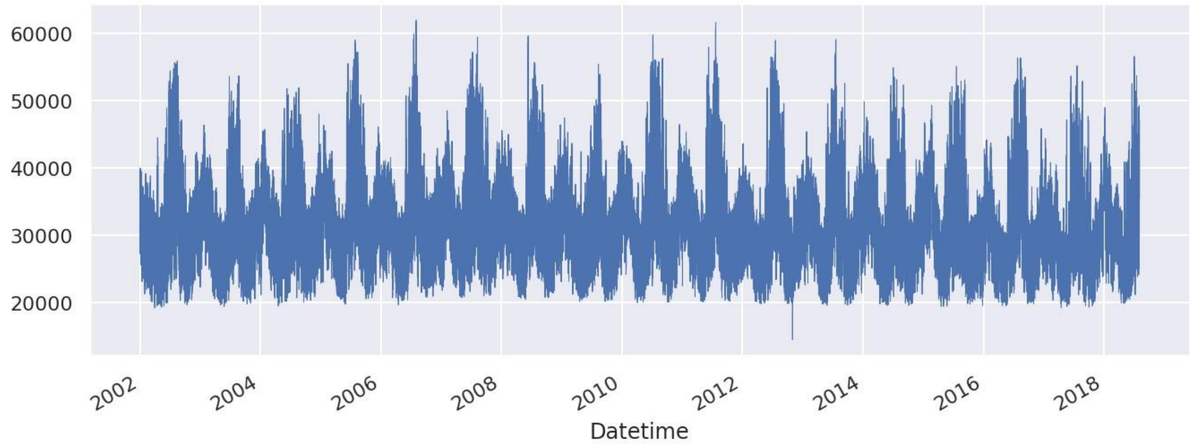
#Ploting a graph for the data set

file  import seaborn as sns

sns.set(rc={'figure.figsize':(11, 4)})

data['PJME_MW'].plot(linewidth=0.

5); OUTPUT:




From the above graph we can analyse that datetime of the energy consumed from the years 2002 to 2018 from varies from each year throughout the energy consumed . We can also see that the between the years of 2006 and 2008 has the highest point of energy consumed.

#Finding if the dataset has any

null values data.isnull().sum()

OUTPUT:

PJME_MW 0

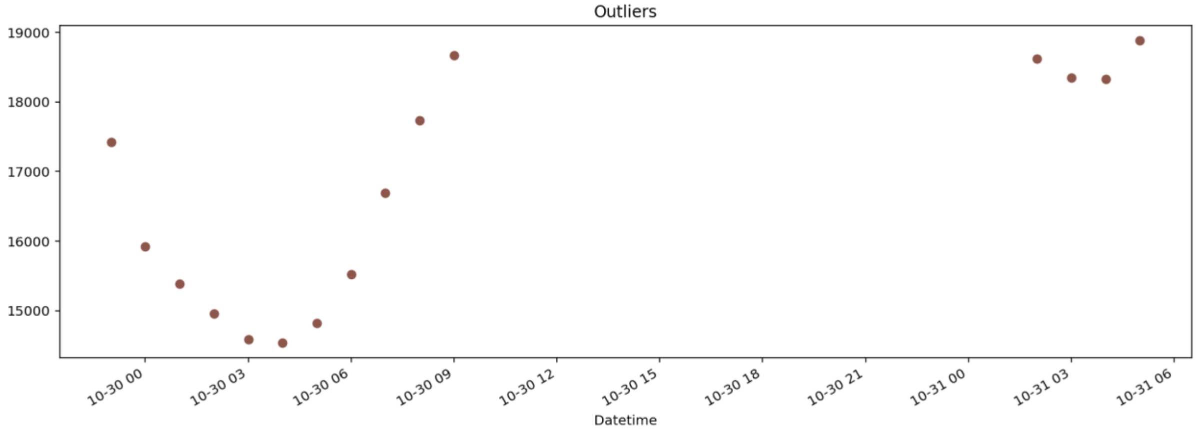
dtype: int64

#Ploting a graph to outline the outliers in the given dataset

```
data.query('PJME_MW < 19_000')['PJME_MW'] \
    .plot(style='o',
figsize=(15, 5),
color=color_pal[5],
title='Outliers')
```

OUTPUT:

<Axes: title={'center': 'Outliers'}, xlabel='Datetime'>




From the above graph we can see that there are outliers present in the dataset. There are four outliers present in the above graph.

```
#Datacleaning the dataset import pandas as pd data =
pd.read_csv('PJME_hourly.csv',index_col=[0],
parse_dates=[0]) data.head() print("\nDataset after
Data Cleaning:") print(data.head())
```

OUTPUT:

Dataset after Data Cleaning: PJME_MW Datetime 2002-12-31 01:00:00 26498.0 2002-12-31 02:00:00 25147.0 2002-12-31 03:00:00 24574.0 2002-12-31 04:00:00 24393.0 2002-12-31 05:00:00 24860.0

```
#Plotting a table to differentiate between the years of calculations made
from the dataset def create_features(df, label=None):
    """
    Creates time series features from datetime index.
    """
    df = df.copy()     df['date'] =
df.index     df['hour'] =
df['date'].dt.hour
df['dayofweek'] =
df['date'].dt.dayofweek
df['quarter'] = df['date'].dt.quarter
df['month'] = df['date'].dt.month
df['year'] = df['date'].dt.year
df['dayofyear'] =
df['date'].dt.dayofyear
df['dayofmonth'] =
df['date'].dt.day
df['weekofyear'] =
df['date'].dt.weekofyear
    X = df[['hour','dayofweek','quarter','month','year',
'dayofyear','dayofmonth','weekofyear']]
    if label:
        y =
df[label]
return X, y
return X
    X, y = create_features(data, label='PJME_MW')
```

```
df = pd.concat([X, y], axis=1)    df.head()
```

OUTPUT:

| | hour | dayofweek | quarter | month | year | dayofyear | dayofmonth | weekofyear | PJME_MW |
|---|---|---|---|---|---|---|---|---|---|
| **Datetime** | | | | | | | | | |
| **2002-12-31 01:00:00** | 1 | 1 | 4 | 12 | 2002 | 365 | 31 | 1 | 26498.0 |
| **2002-12-31 02:00:00** | 2 | 1 | 4 | 12 | 2002 | 365 | 31 | 1 | 25147.0 |
| **2002-12-31 03:00:00** | 3 | 1 | 4 | 12 | 2002 | 365 | 31 | 1 | 24574.0 |
| **2002-12-31 04:00:00** | 4 | 1 | 4 | 12 | 2002 | 365 | 31 | 1 | 24393.0 |
| **2002-12-31 05:00:00** | 5 | 1 | 4 | 12 | 2002 | 365 | 31 | 1 | 24860.0 |

```
#Finding the mean for
the dataset mean =
data.mean()
print("Mean:")
print(mean)
```

OUTPUT:

```
Mean:
PJME_MW 32080.222831
dtype: float64
```

```
#Finding the median for
the dataset median =
data.median()
```

print("\nMedian:")

print(median) OUTPUT:

Median:

PJME_MW

31421.0

dtype: float64

#Finding the mode for

the dataset mode =

data.mode().iloc[0]

print("\nMode:")

print(mode)

OUTPUT:

Mode:

PJME_MW 30051.0

Name: 0, dtype: float64

The outliers and the null values in the dataset can be overcome by the mean, median, mode models which analyse the dataset for the null values and outliers present inside the data. These in terms help the dataset to remove unnecessary data values present in it . It may lead to removing of false values present in the dataset.

# Load the dataset dataset_path =

"path/to/hourly_energy_consumption.csv"

data = pd.read_csv(dataset_path)

# Explore the first few rows of

the dataset print("Initial

Dataset:") print(data.head())

# Data Cleaning: Handling missing

values (if any) data = data.dropna()

# Data Cleaning: Handling duplicate

entries (if any) data =

data.drop_duplicates()

# Data Cleaning: Handling other errors (specific to your dataset)

05555555555555553

-+88888888+8/2# After cleaning

print("\nDataset 9aft.0-9er Data

Cleaning:") print(data.head())

# Further data preprocessing steps can be added based on project requirements

In the above code, replace "path/to/hourly_energy_consumption.csv" with the actual path where you have saved the downloaded dataset. This code snippet loads the dataset, removes any rows with missing values, and drops duplicate entries. You can add more specific cleaning operations based on the characteristics of your dataset, such as handling outliers, correcting inconsistent values, or dealing with formatting errors.

# Preprocess data
labelEncoder =
LabelEncoder()

```
oneHotEncoder =
OneHotEncoder(categorical_features=[0]) ss =
StandardScaler() values = df.values

# integer encode direction

#encoder = LabelEncoder()

#values[:,8] = encoder.fit_transform(values[:,8])

# ensure all data is float
values = values.astype('float32')

# normalize features scaler =
MinMaxScaler(feature_range=(0,
1)) scaled =
scaler.fit_transform(values)

# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)

# drop columns we don't want to predict
reframed.drop(reframed.columns[[9,10,11,12,13,14,15,16]], axis=1,
inplace=True) print(reframed.shape)
print(reframed.head())
```

OUTPUT:

```
(145366, 10)
var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
0       NaN       NaN       NaN       NaN       NaN       NaN
1       0.043478   0.166667       1.0       1.0       0.0    0.99726
2       0.086957   0.166667       1.0       1.0       0.0    0.99726
3       0.130435   0.166667       1.0       1.0       0.0    0.99726
4       0.173913   0.166667       1.0       1.0       0.0
0.99726
var7(t-1)  var8(t-1)  var9(t-1)   var9(t)   0       NaN       NaN       NaN
0.251849
1                 1.0       0.0   0.251849  0.223386
2                 1.0       0.0   0.223386  0.211314
3                 1.0       0.0   0.211314  0.207500
4                 1.0       0.0   0.207500  0.217339
```

```
# make a prediction
yhat =
model.predict(X_te
st)
X_test = X_test.reshape((X_test.shape[0], X_test.shape[2]))

# invert scaling for forecast inv_yhat =
concatenate((X_test[:,:-1],yhat),
axis=1) inv_yhat =
scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,-1] # invert
scaling for actual
y_test = y_test.reshape((len(y_test),
1)) inv_y = concatenate((X_test[:,:-
1],y_test), axis=1) inv_y =
scaler.inverse_transform(inv_y)
inv_y = inv_y[:,-1]
# calculate RMSE
MSE=mean_squared_error(inv_y,inv_yhat)
MAE=mean_absolute_error(inv_y,inv_yhat)
RMSE = sqrt(mean_squared_error(inv_y, inv_yhat))
print('MSE: %.3f' % MSE + '   MAE: %.3f' % MAE + '
RMSE: %.3f' % RMSE) OUTPUT:
```

MSE: 1522100.750   MAE: 933.959   RMSE: 1233.734

```
#Calculates the MAPE for the dataset def
mean_absolute_percentage_error(y_true,
y_pred):     """Calculates MAPE given y_true
and y_pred"""     y_true, y_pred =
np.array(y_true), np.array(y_pred)     return
np.mean(np.abs((y_true - y_pred) / y_true)) *
100

print(mean_absolute_percentage_error(inv_y,inv_yh
at)) OUTPUT:
```

3.113434463739395

#Plotting a graph to differentiate the actual value and the predicted value from the datasets file and plots the difference

```
aa=[x for x in range(500)]
plt.figure(figsize=(8,4)) plt.plot(aa,
inv_y[:500], marker='.',
label="actual") plt.plot(aa,
inv_yhat[:500], 'r', label="prediction")

plt.tight_layout()
sns.despine(top=True)
plt.subplots_adjust(left=
0.07)
plt.ylabel('PJME_MW',
size=15) plt.xlabel('Time
step', size=15)
plt.legend(fontsize=15)
plt.show();
```
OUTPUT: