PROJECT TITLE: MEASURE ENERGY CONSUMPTION

PHASE 3: DEVELOPMENT PART 1

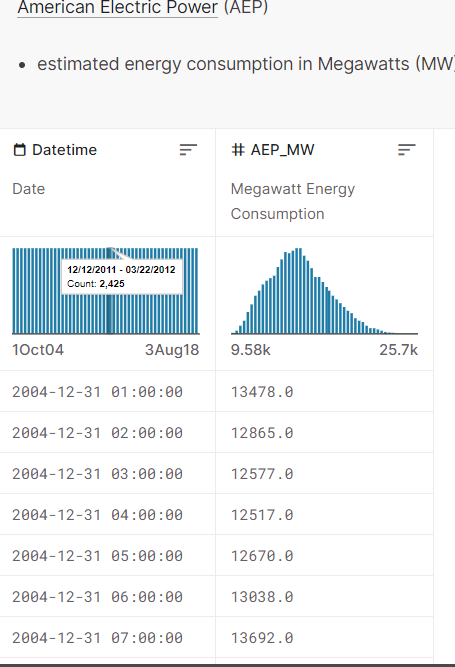
LOADING AND PREPROCESSING THE DATASET

DATA SOURCE:

A good data source for measure energy consumption for machine learning should be accurate, complete and accessible.

DATASET LINK: <https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption>

GIVEN DATASET:



1. Loading the dataset:

 Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.

 The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used.However, there are some general steps that are common to most machine learning frameworks:

a.Identify the dataset:

The first step is to identify the dataset that you want to load. This may be stored in a local file, in a database, or in a cloud storage service.

b.Load the dataset:

Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

c.Preprocess the dataset:

Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.

Here, how to load a dataset using machine learning in Python

LOAD THE DATASET:

In [1]:

df=pd.read\_csv("../input/hourly-energy-consumption/AEP\_hourly.csv",index\_col='Datetime',parse\_dates=True)

df.head()

Out [1]:

AEP\_MW

Datetime

2004-12-31 01:00:00 13478.0

2004-12-31 02:00:00 12865.0

2004-12-31 03:00:00 12577.0

2004-12-31 04:00:00 12517.0

2004-12-31 05:00:00 12670.0M:

PROGRAM:

MEASURE ENERGY CONSUMPTION

In [2]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

%matplotlib inline

import seaborn as sns

import warnings

warnings.filterwarnings("ignore")

from pandas.plotting import lag\_plot

from pylab import rcParams

from statsmodels.tsa.seasonal import seasonal\_decompose

from pandas import DataFrame

from pandas import concat

In [3]:

df.sort\_values(by='Datetime', inplace=True)

print(df)

Out [3]:

AEP\_MW

Datetime

2004-10-01 01:00:00 12379.0

2004-10-01 02:00:00 11935.0

2004-10-01 03:00:00 11692.0

2004-10-01 04:00:00 11597.0

2004-10-01 05:00:00 11681.0

... ...

2018-08-02 20:00:00 17673.0

2018-08-02 21:00:00 17303.0

2018-08-02 22:00:00 17001.0

2018-08-02 23:00:00 15964.0

2018-08-03 00:00:00 14809.0

[121273 rows x 1 columns]

In [4]:

df.shape()

Out [4]:

(121273,1)

In [5]:

df.info()

Out [5]:

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

DatetimeIndex: 121273 entries, 2004-10-01 01:00:00 to 2018-08-03 00:00:00

Data columns (total 1 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 AEP\_MW 121273 non-null float64

dtypes: float64(1)

memory usage: 1.9 MB

In [6]:

df.describe()

Out [6]:

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

In [7]:

df.index = pd.to\_datetime(df.index)

In [8]:

# Extract all Data Like Year MOnth Day Time etc

df["Month"] = df.index.month

df["Year"] = df.index.year

df["Date"] = df.index.date

df["Hour"] = df.index.hour

df["Week"] = df.index.week

df["Day"] = df.index.day\_name()

df.head()

Out [8]:

AEP\_MW Month Year Date Hour Week Day

Datetime

2004-10-01 01:00:00 12379.0 10 2004 2004-10-01 1 40 Friday

2004-10-01 02:00:00 11935.0 10 2004 2004-10-01 2 40 Friday

2004-10-01 03:00:00 11692.0 10 2004 2004-10-01 3 40 Friday

2004-10-01 04:00:00 11597.0 10 2004 2004-10-01 4 40 Friday

2004-10-01 05:00:00 11681.0 10 2004 2004-10-01 5 40 Friday

2. Preprocessing the dataset:

 Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.

 This may involve removing errors and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable range.

VISUALISATION AND PREPROCESSING OF DATASETS:

In [9]:

df.plot(title="PJME Energy use in MegaWatts",

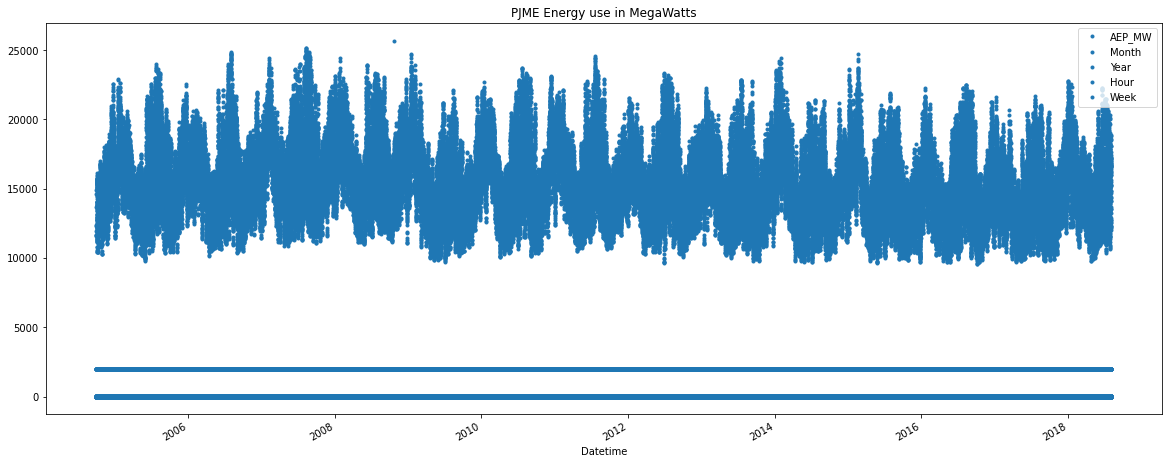
        figsize=(20, 8),

        style=".",

        color=sns.color\_palette()[0])

plt.show()

Out [9]:



In [10]:

df.tail()

Out [10]:

AEP\_MW Month Year Date Hour Week Day

Datetime

2018-08-02 20:00:00 17673.0 8 2018 2018-08-02 20 31 Thursday

2018-08-02 21:00:00 17303.0 8 2018 2018-08-02 21 31 Thursday

2018-08-02 22:00:00 17001.0 8 2018 2018-08-02 22 31 Thursday

2018-08-02 23:00:00 15964.0 8 2018 2018-08-02 23 31 Thursday

2018-08-03 00:00:00 14809.0 8 2018 2018-08-03 0 31 Friday

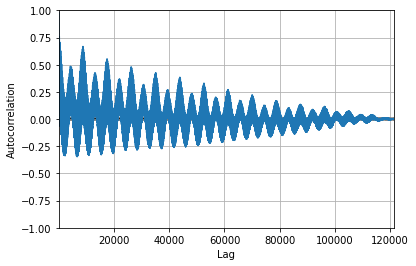
In [11]:

from pandas.plotting import autocorrelation\_plot

autocorrelation\_plot(df['AEP\_MW'])

plt.show()

Out [11]:



In [12]:

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from math import sqrt

from sklearn.preprocessing import MinMaxScaler

# Analysis imports

from pandas.plotting import lag\_plot

from pylab import rcParams

from statsmodels.tsa.seasonal import seasonal\_decompose

from pandas import DataFrame

from pandas import concat

# Modelling imports

from statsmodels.tsa.ar\_model import AR

from statsmodels.tsa.arima\_model import ARMA

from statsmodels.tsa.arima\_model import ARIMA

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM, GRU, RNN

from keras.layers import Dropout

In [13]:

values = DataFrame(df['AEP\_MW'].values)

dataframe = concat([values.shift(1),values.shift(5),values.shift(10),values.shift(30), values], axis=1)

dataframe.columns = ['t', 't+1', 't+5', 't+10', 't+30']

result = dataframe.corr()

print(result)

Out [13]:

t t+1 t+5 t+10 t+30

t 1.000000 0.731161 0.345667 0.501972 0.976223

t+1 0.731161 1.000000 0.630009 0.847210 0.630007

t+5 0.345667 0.630009 1.000000 0.644479 0.317277

t+10 0.501972 0.847210 0.644479 1.000000 0.408315

t+30 0.976223 0.630007 0.317277 0.408315 1.000000

In [14]:

train\_data, test\_data = df[0:-60], df[-60:]

plt.figure(figsize=(10,10))

plt.grid(True)

plt.xlabel('Dates')

plt.ylabel('Energy in megawatts')

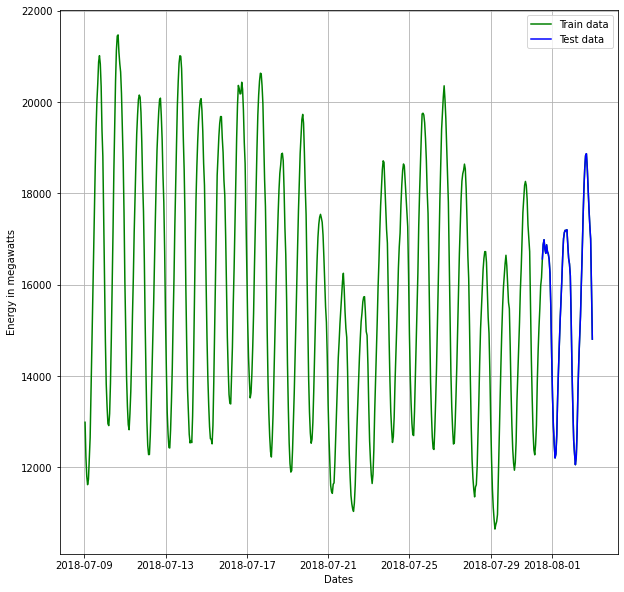
plt.plot(df['AEP\_MW'].tail(600), 'green', label='Train data')

plt.plot(test\_data['AEP\_MW'], 'blue', label='Test data')

plt.legend()

Out [14]:

<matplotlib.legend.Legend at 0x78af47a94e50>



In [15]:

mean\_value = df['AEP\_MW'].mean() # calculation of mean price

plt.figure(figsize=(16,8))

plt.grid(True)

plt.xlabel('Dates')

plt.ylabel('Energy in megawatts')

plt.plot(df['AEP\_MW'], 'green', label='Train data')

plt.plot(test\_data['AEP\_MW'], 'blue', label='Test data')

plt.axhline(y=mean\_value, xmin=0.864, xmax=1, color='red')

plt.legend()

plt.figure(figsize=(16,8))

plt.grid(True)

plt.xlabel('Dates')

plt.ylabel('Energy in megawatts')

plt.plot(df['AEP\_MW'].tail(600), 'green', label='Train data')

plt.plot(test\_data['AEP\_MW'], 'blue', label='Test data')

plt.axhline(y=mean\_value, xmin=0.864, xmax=1, color='red')

plt.legend()

print('MSE: '+str(mean\_squared\_error(test\_data['AEP\_MW'], np.full(len(test\_data), mean\_value))))

print('MAE: '+str(mean\_absolute\_error(test\_data['AEP\_MW'], np.full(len(test\_data), mean\_value))))

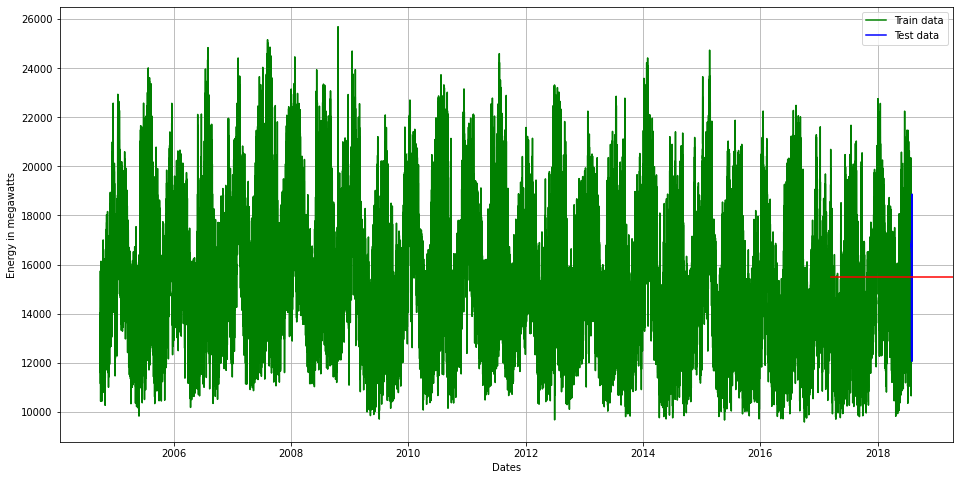
print('RMSE: '+str(sqrt(mean\_squared\_error(test\_data['AEP\_MW'], np.full(len(test\_data), mean\_value)))))

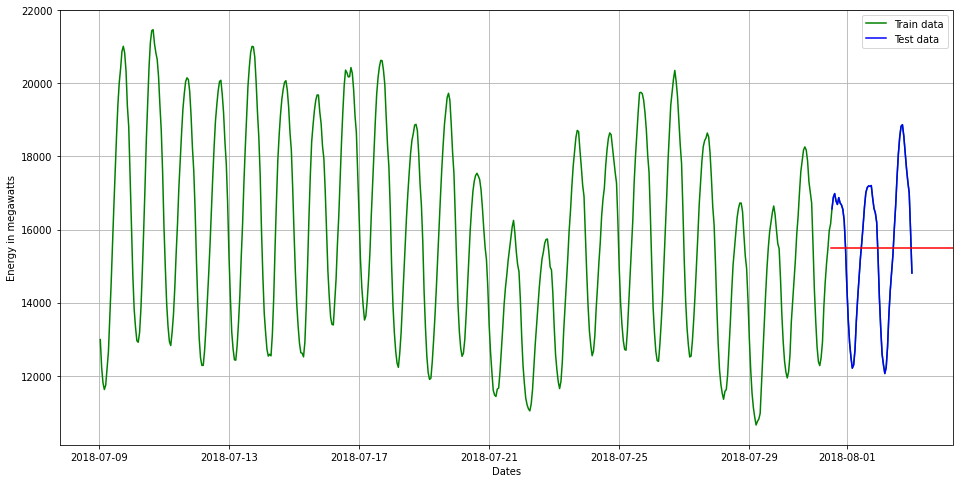
Out [15]:

MSE: 3700885.0406027567

MAE: 1667.1805899362046

RMSE: 1923.768447761517





In [16]:

from statsmodels.tsa.stattools import adfuller

def adf\_test(dataset):

     dftest = adfuller(dataset, autolag = 'AIC')

     print("1. ADF : ",dftest[0])

     print("2. P-Value : ", dftest[1])

     print("3. Num Of Lags : ", dftest[2])

     print("4. Num Of Observations Used For ADF Regression:",      dftest[3])

     print("5. Critical Values :")

     for key, val in dftest[4].items():

         print("\t",key, ": ", val)

In [17]:

adf\_test(df['AEP\_MW'])

Out [17]:

1. ADF : -18.285883882257217

2. P-Value : 2.3029539101747796e-30

3. Num Of Lags : 71

4. Num Of Observations Used For ADF Regression: 121201

5. Critical Values :

1% : -3.430403955318047

5% : -2.8615638474512295

10% : -2.566782693155802

In [18]:

import statsmodels.api as sm

In [19]:

*#Train Arima Model*

train\_arima = train\_data['AEP\_MW']

test\_arima = test\_data['AEP\_MW']

history = [x for x **in** train\_arima]

y = test\_arima

*# make first prediction*

predictions = list()

model = sm.tsa.arima.ARIMA(history, order=(5,1,0))

model\_fit = model.fit()

yhat = model\_fit.forecast()[0]

predictions.append(yhat)

history.append(y[0])

*# rolling forecasts*

for i **in** range(1, len(y)):

*# predict*

model = sm.tsa.arima.ARIMA(history, order=(5,1,0))

model\_fit = model.fit()

yhat = model\_fit.forecast()[0]

*# invert transformed prediction*

predictions.append(yhat)

*# observation*

obs = y[i]

history.append(obs)

plt.figure(figsize=(14,8))

plt.plot(df.index, df['AEP\_MW'], color='green', label = 'Train Energy AEP\_MW')

plt.plot(test\_data.index, y, color = 'red', label = 'Real Energy AEP\_MW')

plt.plot(test\_data.index, predictions, color = 'blue', label = 'Predicted Energy AEP\_MW')

plt.legend()

plt.grid(True)

plt.show()

plt.figure(figsize=(14,8))

plt.plot(df.index[-600:], df['AEP\_MW'].tail(600), color='green', label = 'Train Energy AEP\_MW')

plt.plot(test\_data.index, y, color = 'red', label = 'Real Energy AEP\_MW')

plt.plot(test\_data.index, predictions, color = 'blue', label = 'Predicted Energy AEP\_MW')

plt.legend()

plt.grid(True)

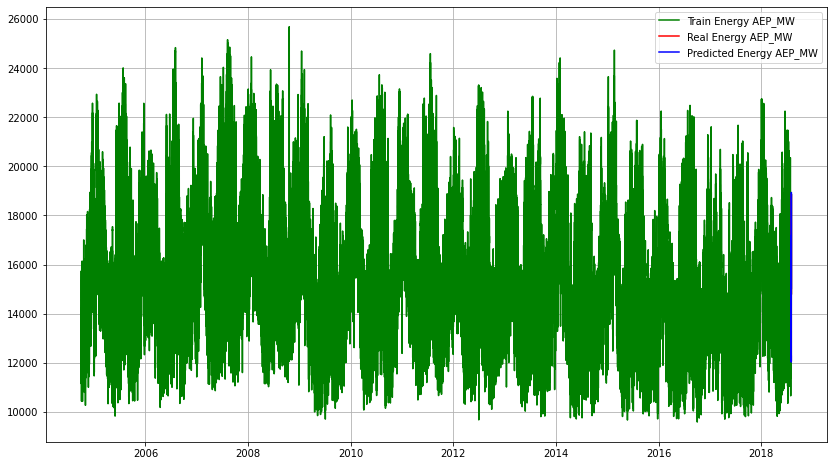
plt.show()

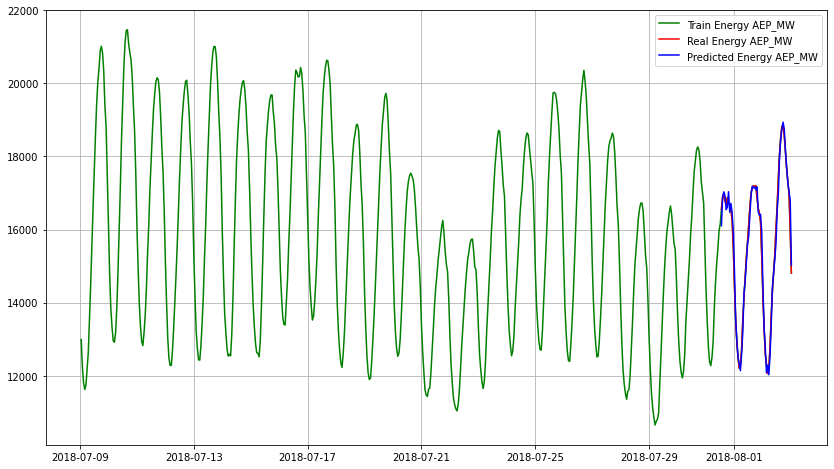
print('MSE: '+str(mean\_squared\_error(y, predictions)))

print('MAE: '+str(mean\_absolute\_error(y, predictions)))

print('RMSE: '+str(sqrt(mean\_squared\_error(y, predictions))))

Out [19]:





MSE: 57710.45153428949

MAE: 177.320844006739

RMSE: 240.2299971574938