PROJECT TITLE: MEASURE ENERGY CONSUMPTION

PHASE 4: DEVELOPMENT PART 2

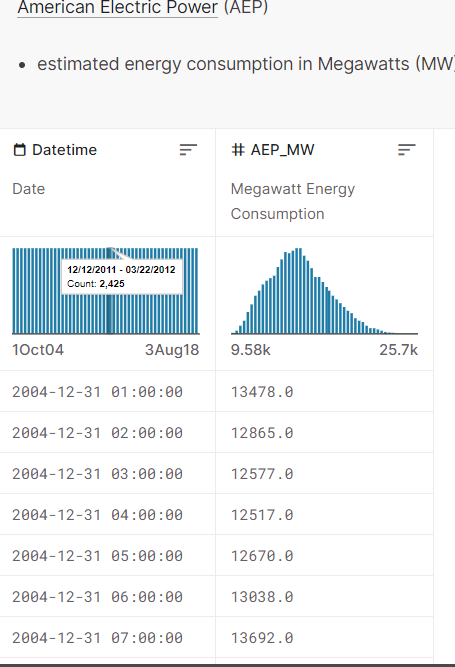
CONTINUE BUILDING THE MEASURE ENERGY CONSUMPTION BY FEATURE ENGINEERING,MODEL TRAINING AND EVALUATION

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:



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:

PROCEDURE:

FEATURE SELECTION:

Feature selection is important to reduce dimensionality and improve the efficiency and accuracy of models for measuring energy consumption. Here are some methods and considerations for feature selection:

1. \*Correlation Analysis:\*

- Calculate the correlation between each feature and the target variable (energy consumption). Select features with the highest correlations.

2. \*Mutual Information:\*

- Use mutual information scores to assess the information shared between features and the target variable.

3. \*Recursive Feature Elimination (RFE):\*

- Train models (e.g., regression or tree-based models) and iteratively remove the least important features based on model performance.

4. \*L1 Regularization (Lasso):\*

- Apply L1 regularization to linear models, encouraging sparse feature selection by driving some feature coefficients to zero.

5. \*Tree-Based Methods:\*

- Decision trees and ensemble methods like Random Forest and Gradient Boosting can provide feature importances, which can guide feature selection.

6. \*Principal Component Analysis (PCA):\*

- Use PCA to reduce dimensionality while preserving variance. Be cautious as PCA may make it harder to interpret feature importance.

7. \*Univariate Feature Selection:\*

- Select features based on statistical tests like ANOVA or chi-squared tests, which evaluate the relationship between each feature and the target.

8. \*Feature Importance from Machine Learning Models:\*

- Some machine learning models provide feature importance scores. Use these scores to rank and select features.

9. \*Domain Knowledge:\*

- Consult domain experts to identify the most relevant features for energy consumption in a specific context.

10. \*Forward and Backward Selection:\*

- Start with no features and add them one by one (forward selection) or start with all features and remove them one by one (backward selection) based on performance.

11. \*Regularization Techniques:\*

- Techniques like Ridge regression can help reduce the impact of less important features by penalizing their coefficients.

12. \*Feature Importance Stability:\*

- Evaluate feature importance stability across different models and datasets to ensure robust selections.

13. \*Cross-Validation:\*

- Perform feature selection within a cross-validation framework to avoid overfitting and select features that generalize well.

It's important to strike a balance between feature reduction and maintaining model interpretability and performance. The choice of feature selection method depends on the dataset, the modeling approach, and the specific goals of your energy consumption measurement task

PROGRAM:

import numpy as np

import pandas as pd

from sklearn.linear\_model import LassoCV

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Load your dataset

data = pd.read\_csv('energy\_data.csv') # Replace with your dataset file

# Define features and target variable

X = data.drop('energy\_consumption', axis=1)

y = data['energy\_consumption']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Create a Lasso model with cross-validation to select features

lasso = LassoCV(alphas=np.logspace(-6, 6, 13), cv=5)

lasso.fit(X\_train, y\_train)

# Get the selected features

selected\_features = X.columns[lasso.coef\_ != 0]

# Apply feature selection to your dataset

X\_train\_selected = X\_train[:, lasso.coef\_ != 0]

X\_test\_selected = X\_test[:, lasso.coef\_ != 0]

# Train a model using the selected features (e.g., Linear Regression)

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X\_train\_selected, y\_train)

# Make predictions and evaluate the model

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

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

# Now 'selected\_features' contains the names of the selected features for your model.

MODEL TRAINING:

1. \*Data Collection:\*

- Gather historical energy consumption data, along with relevant features, as discussed in previous responses.

2. \*Data Preprocessing:\*

- Clean the data by handling missing values, outliers, and data quality issues.

- Normalize or scale the features to ensure they have similar scales.

- Split the data into training, validation, and test sets.

3. \*Feature Engineering:\*

- Create or select relevant features as discussed earlier to improve model accuracy.

4. \*Model Selection:\*

- Choose an appropriate machine learning model for regression tasks. Common choices include:

- Linear Regression

- Decision Trees

- Random Forest

- Gradient Boosting (e.g., XGBoost, LightGBM)

- Neural Networks

5. \*Model Training:\*

- Train the selected model on the training dataset using the features and energy consumption as the target variable.

6. \*Model Evaluation:\*

- Assess the model's performance using appropriate evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or R-squared (R²).

- Use the validation dataset to fine-tune hyperparameters and prevent overfitting.

7. \*Feature Importance Analysis:\*

- Analyze feature importances to understand which features have the most significant impact on energy consumption.

8. \*Model Interpretability:\*

- Consider using techniques to interpret the model's predictions, especially if the model is complex (e.g., SHAP values for tree-based models).

9. \*Hyperparameter Tuning:\*

- Optimize the model's hyperparameters using techniques like grid search or random search.

10. \*Cross-Validation:\*

- Perform cross-validation to ensure the model's generalizability and robustness.

11. \*Model Deployment:\*

- Deploy the trained model in a production environment for real-time or batch predictions.

12. \*Monitoring and Maintenance:\*

- Continuously monitor the model's performance in the production environment and retrain as needed to account for changes in energy consumption patterns.

13. \*Explainability and Reporting:\*

- Create reports and dashboards to communicate model predictions and insights to stakeholders.

MACHINE LEARNING MODELS:

Machine Learning Models:

models = pd.DataFrame(columns=["Model","MAE","MSE","RMSE","R2 S

core","RMSE (Cross-Validation)"])

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

MODEL EVALUATION:

1. \*Mean Absolute Error (MAE):\*

- MAE measures the average absolute difference between the model's predictions and the actual energy consumption values. It provides an easily interpretable measure of prediction accuracy.

2. \*Mean Squared Error (MSE):\*

- MSE measures the average of the squared differences between predictions and actual values. It penalizes larger errors more heavily than MAE.

3. \*Root Mean Squared Error (RMSE):\*

- RMSE is the square root of the MSE and provides an interpretable metric in the same units as the target variable (energy consumption). It is sensitive to outliers.

4. \*R-squared (R²) or Coefficient of Determination:\*

- R² measures the proportion of the variance in the energy consumption that is explained by the model. A higher R² indicates a better fit, but be cautious about overfitting.

5. \*Coefficient of Variation (CV):\*

- CV calculates the ratio of the standard deviation to the mean of the residuals. It can help assess the relative error.

6. \*Percentage Error or Relative Error:\*

- Calculate the percentage difference between predicted and actual energy consumption to gauge the relative error.

7. \*Residual Analysis:\*

- Examine the distribution of residuals (the differences between predictions and actual values) to check for patterns or biases.

8. \*Cross-Validation:\*

- Perform cross-validation, such as k-fold cross-validation, to assess the model's generalization performance and reduce overfitting.

9. \*Feature Importance Analysis:\*

- Analyze feature importances to understand which features contribute the most to energy consumption predictions.

10. \*Domain Expert Review:\*

- Consult domain experts to ensure that the model's predictions align with physical, economic, and behavioral realities.

11. \*Business Impact Assessment:\*

- Evaluate the practical impact of model predictions on energy management and decision-making within the specific application.

12. \*Benchmarking:\*

- Compare the model's performance against existing benchmarks or other models to assess its relative effectiveness.

13. \*Interpretability and Explainability:\*

- Use methods like SHAP (SHapley Additive exPlanations) to provide interpretability and insights into how the model is making predictions

In [9]:

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

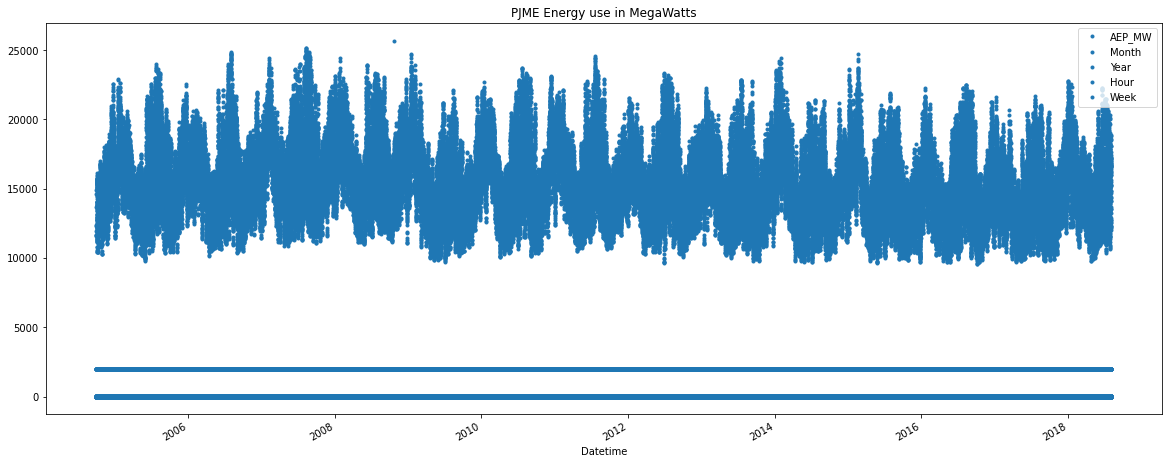
        figsize=(20, 8),

        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

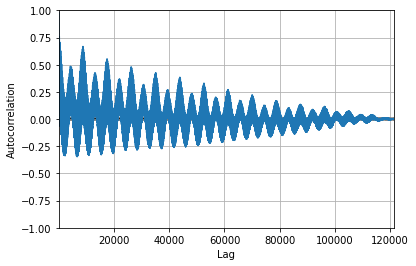
In [11]:

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')

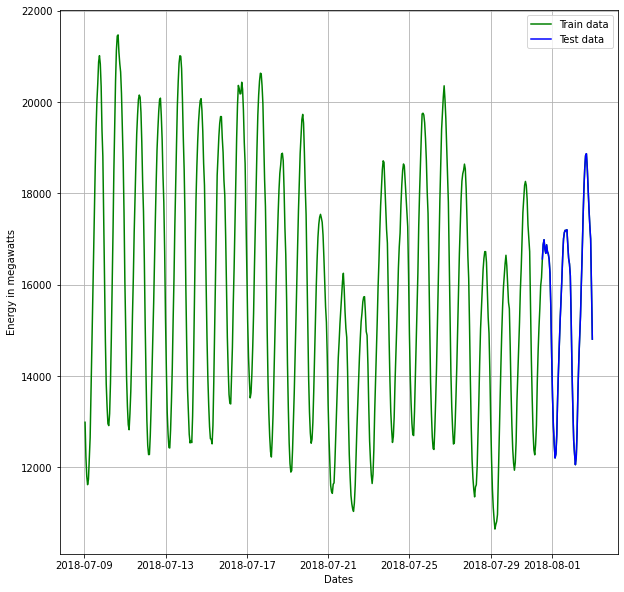
plt.plot(df['AEP\_MW'].tail(600), 'green', label='Train data')

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))))

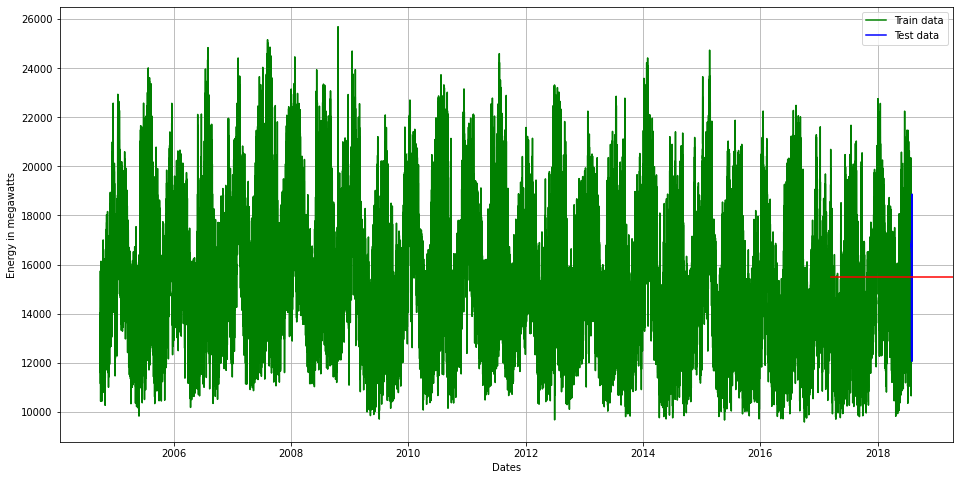
print('RMSE: '+str(sqrt(mean\_squared\_error(test\_data['AEP\_MW'], np.full(len(test\_data), mean\_value)))))

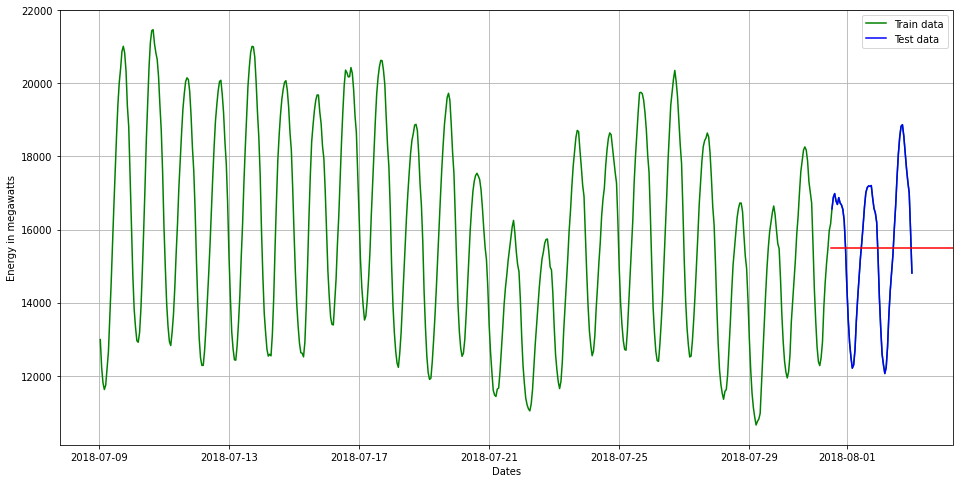
Out [15]:

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)

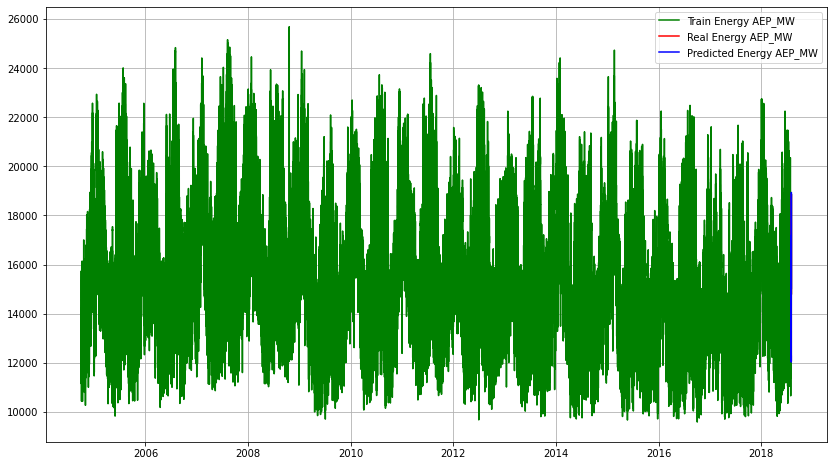
plt.show()

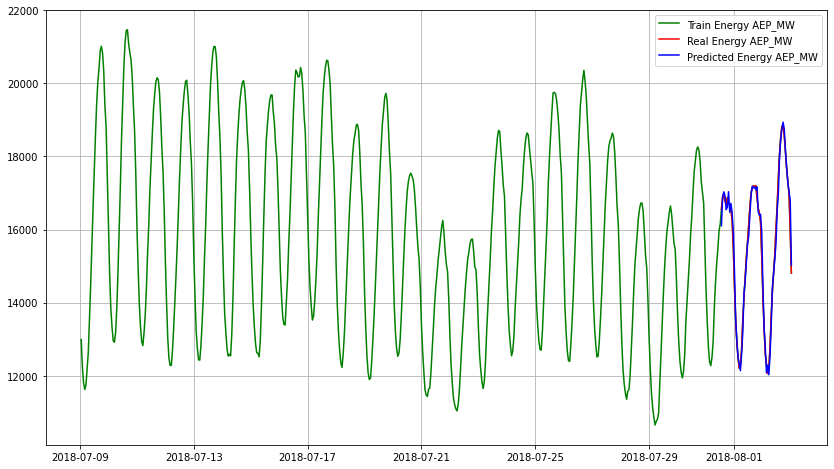
print('MSE: '+str(mean\_squared\_error(y, predictions)))

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

FEATURE ENGINEERING:

Feature engineering is crucial when measuring energy consumption. Here are some key features to consider:

1. Time-Based Features:

- Timestamps: Extract features like hour of the day, day of the week, and season.

- Time intervals: Calculate the time between readings.

2. Weather Data:

- Temperature, humidity, and weather conditions can impact energy consumption.

3. Building Characteristics:

- Square footage, number of occupants, insulation, and building age can influence consumption.

4. Appliance Information:

- Identify major appliances and their energy ratings to attribute consumption.

5. Previous Usage:

- Lag features to account for historical energy consumption patterns.

6. Demographics:

- Location-specific data, such as population density, can be relevant.

7. Occupancy Data:

- Use occupancy sensors to detect when a building or room is in use.

8. Energy Price:

- If applicable, include the cost of energy, which can influence consumption.

9. Holidays and Events:

- Special occasions or holidays can affect energy usage.

10. Behavioral Data:

- User behaviors, routines, and preferences can provide insights.