2111_Madha Engineering college

IBM NaanMudhalvan - Artificial Intelligence Group 1

AI_Phase5

Topic: Measure Energy Consumption

The provided code appears to be a Python script for time series analysis and forecasting of energy consumption data. I'll describe what each section of the code does:

Problem Definition: The problem at hand is to create an automated system that measures energy consumption, analyzes the data, and provides visualizations for informed decision-making. This solution aims to enhance efficiency, accuracy, and ease of understanding in managing energy consumption across various sectors.

Design Thinking:

- Data Source: Identify an available dataset containing energy consumption measurements.
- Data Preprocessing: Clean, transform, and prepare the dataset for analysis.
- Feature Extraction: Extract relevant features and metrics from the energy consumption data.
- Model Development: Utilize statistical analysis to uncover trends, patterns, and anomalies in the data.
- Visualization: Develop visualizations (graphs, charts) to present the energy consumption trends and insights.
- Automation: Build a script that automates data collection, analysis, and visualization processes.

INTRODUCTION

Energy consumption is a critical concern in today's world, with a growing focus on sustainability and resource efficiency. To effectively manage and optimize energy usage, it is imperative to have access to comprehensive data, sophisticated analytics, and automated systems that can provide insights, facilitate decision-making, and promote energy-saving strategies.

This data-driven approach begins with the identification of suitable datasets that contain measurements of energy usage, spanning a diverse range of sectors, including renewable energy sources, business, industry, and residential sectors. The first step in the process involves data preprocessing, where duplicates are removed, missing values are handled, and data inconsistencies are rectified, resulting in a structured and coherent dataset that is ready for analysis. If necessary, numerical features are normalized or scaled to ensure the dataset's consistency.

The heart of this approach lies in feature extraction, where pertinent characteristics and metrics are derived from the energy usage data. This may include timestamps, energy consumption values, information about the area or industry, and weather data if applicable, as well as insights from previous consumption patterns.

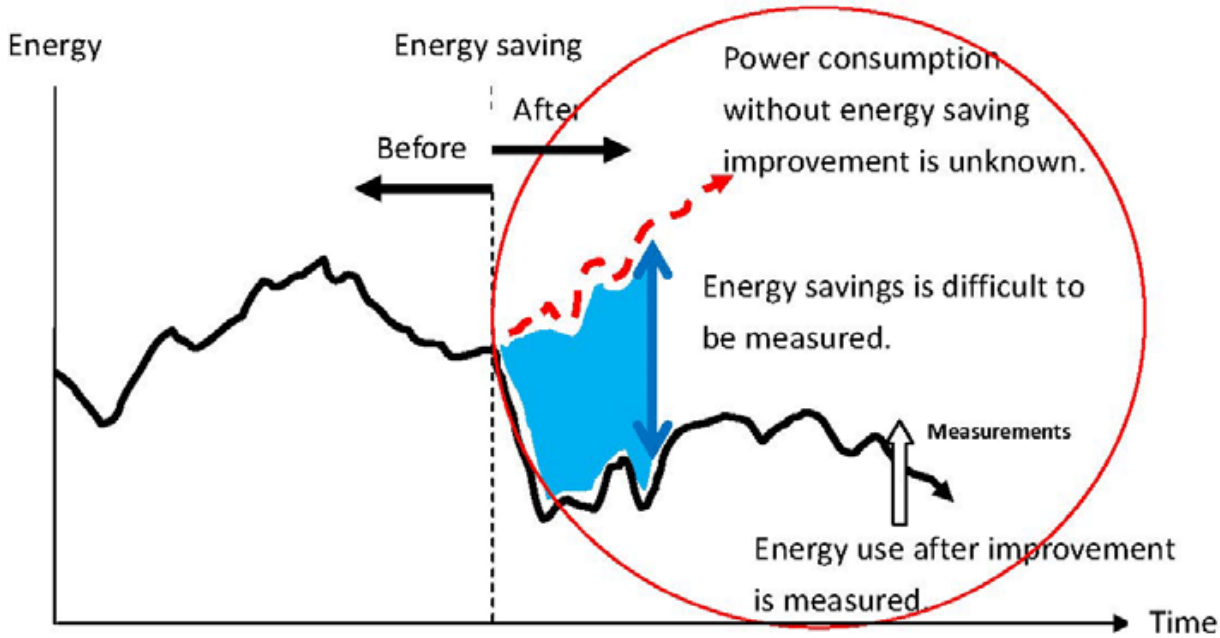
Subsequently, the dataset is subjected to comprehensive model development, which involves employing various analytical tools and techniques. Statistical analysis tools are used to uncover trends, patterns, and anomalies in the data, with a particular focus on time series analysis to identify seasonal or recurring consumption patterns. Regression analysis is employed to understand how environmental factors, such as temperature, affect energy usage, while algorithms for anomaly detection help identify unexpected surges in usage.

To make the findings more accessible and actionable, the next step is visualization. Visualizations in the form of graphs, charts, and dashboards are created to showcase trends and insights in energy use. Various visualization tools and libraries, such as Matplotlib, Seaborn, and Tableau, are leveraged to produce informative and visually appealing graphics. These visualizations allow for the evaluation of historical trends, a comparison of different industries or regions, and the identification of anomalies for further investigation.

Automation is a crucial element of this approach. A program or script is developed to automate data collection, analysis, and visualization processes, with programming languages like Python and R being utilized for this purpose. Regular data updates or real-time data collection are planned, ensuring the automation pipeline can handle new data sources or dataset updates effectively.

Upon the completion of the automation pipeline, the system is deployed in a relevant setting where it continuously collects, analyzes, and displays data on energy usage. Monitoring techniques are implemented to identify data or system problems, and regular reviews and updates are carried out to adapt to shifting data sources or specifications.

Ultimately, the insights and information gathered by the automated system are employed for decision support. They serve as valuable tools in managing energy consumption effectively, with the findings disseminated to relevant stakeholders, promoting energy-saving tactics and strategies that contribute to a more sustainable and efficient future. This comprehensive approach empowers individuals and organizations to make informed decisions and take proactive steps towards a more energy-conscious world.



1. Data Source Identification:

- Find a dataset with measurements of energy usage that is accessible. This dataset could come from a variety of industries, including renewable energy sources, business, industry, and residential.

2. Data preprocessing: -

- Remove duplicates, handle missing values, and fix any data inconsistencies from the dataset.
    - Create a structured format for the data that is appropriate for analysis.
    - If necessary, normalize or scale numerical features to maintain the dataset's coherence.

3. Feature Extraction: -

- From the data on energy use, find and extract pertinent characteristics and metrics. These qualities could consist of:
    - Timestamps and dates
    - Values for energy consumption
    - Information about the area or industry - Weather information, if applicable
    - Previous consumption patterns

4. Model Development: -

- Examine the data for trends, patterns, and anomalies using statistical analysis tools. Time series analysis to find seasonal or recurring consumption patterns is one analysis technique.
    - Regression analysis to comprehend how the environment (such as temperature) affects energy use.
    - Algorithms for detecting anomalies to identify odd or unforeseen surges in usage.

5. Visualization:

- Create visualizations (graphs, charts, dashboards) to show the trends and insights in energy use.
    - Select the right visualization programs and libraries (such as Matplotlib, Seaborn, and Tableau) to produce graphics that are both educational and aesthetically pleasing.
    - Visualize historical trends, evaluate various industries or regions, and identify anomalies for additional research.

6Automation:

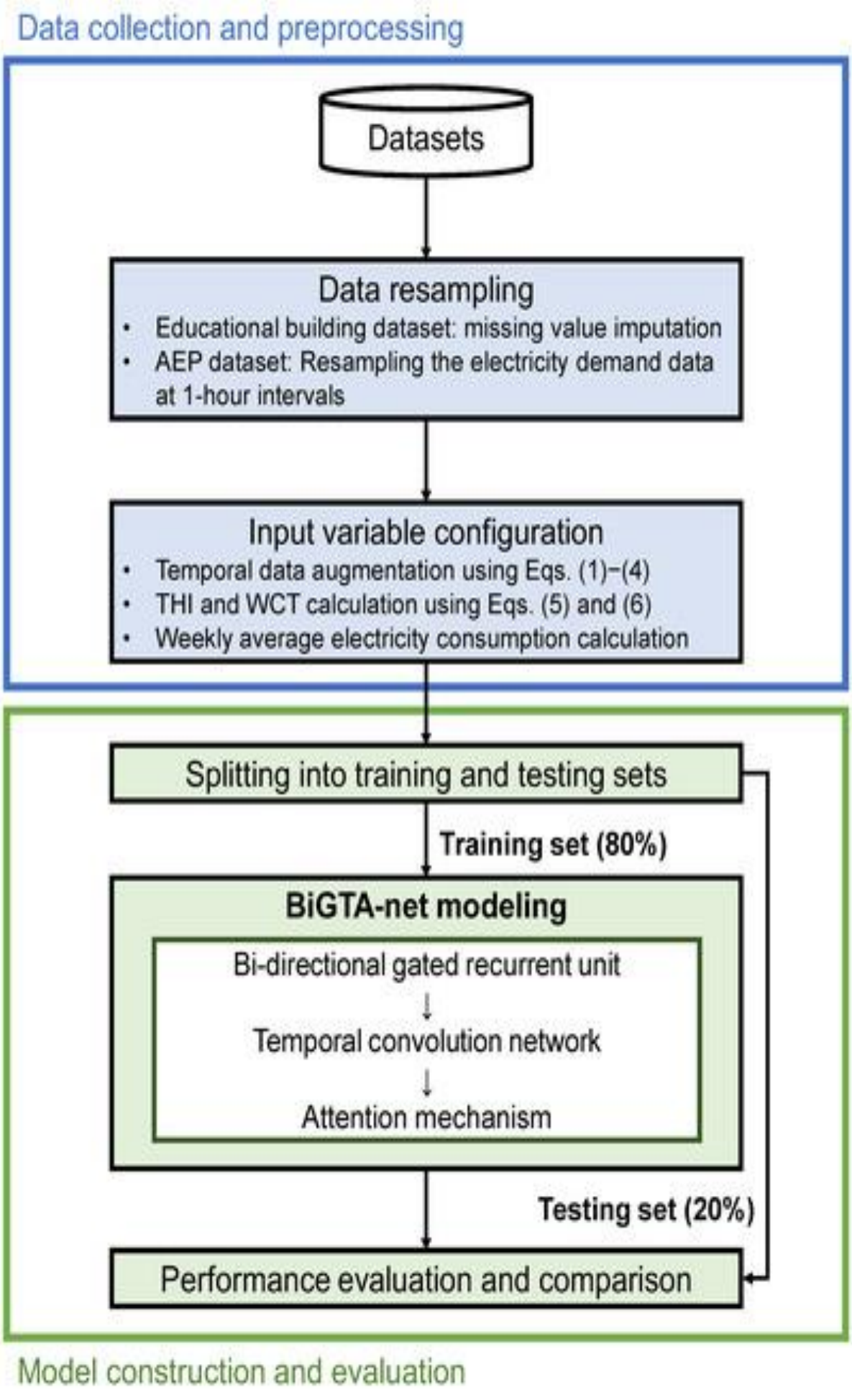
- Create a program or script to automate the processes of data collection, analysis, and visualization. Python, R, or other programming languages could be used for this.
- Plan regular data updates or, if necessary, real-time data collection.
- Make sure the automation pipeline is strong and capable of handling new data sources or dataset updates.
- Ascertain that the automation pipeline is strong and capable of handling new data sources or dataset updates.

7. Deployment and Monitoring: -

- Install the automated system in a setting where it can continuously gather, examine, and display data on energy usage.
    - Implement monitoring techniques to identify data or system problems.
    - Review and update the system on a regular basis to accommodate shifting data sources or specifications.

8. Decision Support: -

- Use the information gathered by the automated system to help you manage your energy consumption.
    - Disseminate the visualizations and analysis findings to the appropriate parties in order to promote energy-saving tactics and strategies


Data collection and preprocessing
Datasets
Data resampling
• Educational building dataset: missing value imputation
• AEP dataset: Resampling the electricity demand data at 1-hour intervals
Input variable configuration
• Temporal data augmentation using Eqs. (1)−(4)
• THI and WCT calculation using Eqs. (5) and (6)
• Weekly average electricity consumption calculation
Splitting into training and testing sets
Training set (80%)
BiGTA-net modeling
Bi-directional gated recurrent unit
Temporal convolution network
Attention mechanism
Testing set (20%)
Performance evaluation and comparison
Model construction and evaluation

SOURCE PROGRAM

1. Imports Libraries:

The code starts by importing necessary libraries, including NumPy, Pandas, Matplotlib, Seaborn, and various modules from Statsmodels for time series analysis and forecasting.

```
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
```

2. Data Loading and Preprocessing:

- It loads a CSV file containing energy consumption data and parses it into a Pandas DataFrame.

- The DataFrame is sorted by the 'Datetime' column, which is assumed to represent timestamps.

- Basic information about the data frame is displayed using `df. shape`, `df.info()`, and `df. describe()`.

- The 'Datetime' column is converted to a datetime data type.

- Additional columns like 'Month,' 'Year,' 'Date,' 'Hour,' 'Week,' and 'Day' are extracted from the datetime index.

```
df=pd.read_csv("../input/Energy_Consumption_Dataset.csv",index_
col='Datetime', parse_dates=True)
df.head()
```

Datetime	AEP_MW
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	12670.0

3. Data Visualization:

- The code generates a line plot to visualize the energy consumption data.

```
df.sort_values(by='Datetime', inplace=True)
print(df)
```

```
                     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]
```

4. Autocorrelation Plot:

- An autocorrelation plot is created to understand the autocorrelation of the energy consumption data.

```
# 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()
 df.shape
```

```
(121273, 1)
```

```
df.info()
```

```
<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
```

```
df.describe()
```

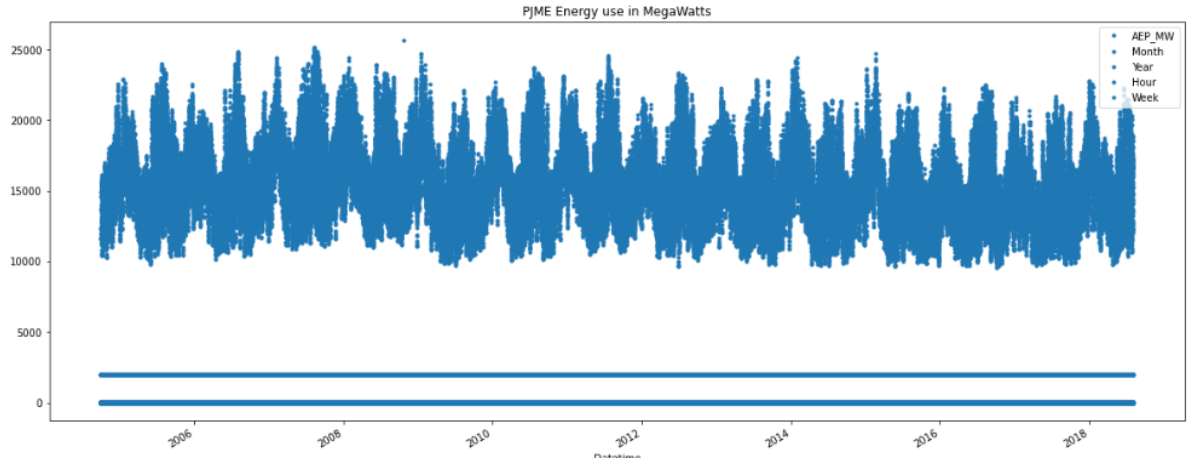
	AEP_MW
count	121273.000000
mean	15499.513717
std	2591.399065
min	9581.000000
25%	13630.000000
50%	15310.000000
75%	17200.000000
max	25695.000000

```
df.index = pd.to_datetime(df.index)


# 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()
```

	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

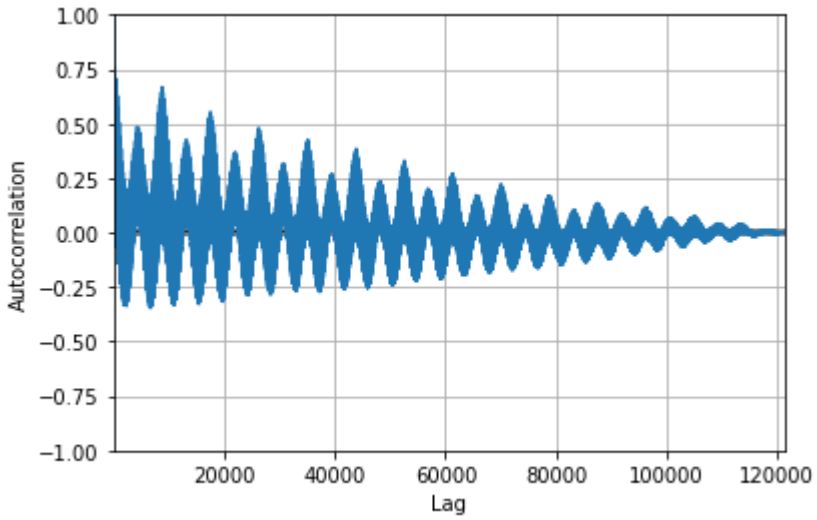
```
df.plot(title="PJME Energy use in MegaWatts",
        figsize=(20, 8),
        style=".",
        color=sns.color_palette()[0])

plt.show()
```




```
df.tail()
```

	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

```
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df['AEP_MW'])
plt.show()
```



5. Modeling Imports:

- The code imports various time series modeling libraries such as AR, ARMA, ARIMA, and parts of Keras for neural network-based models.

```
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
```

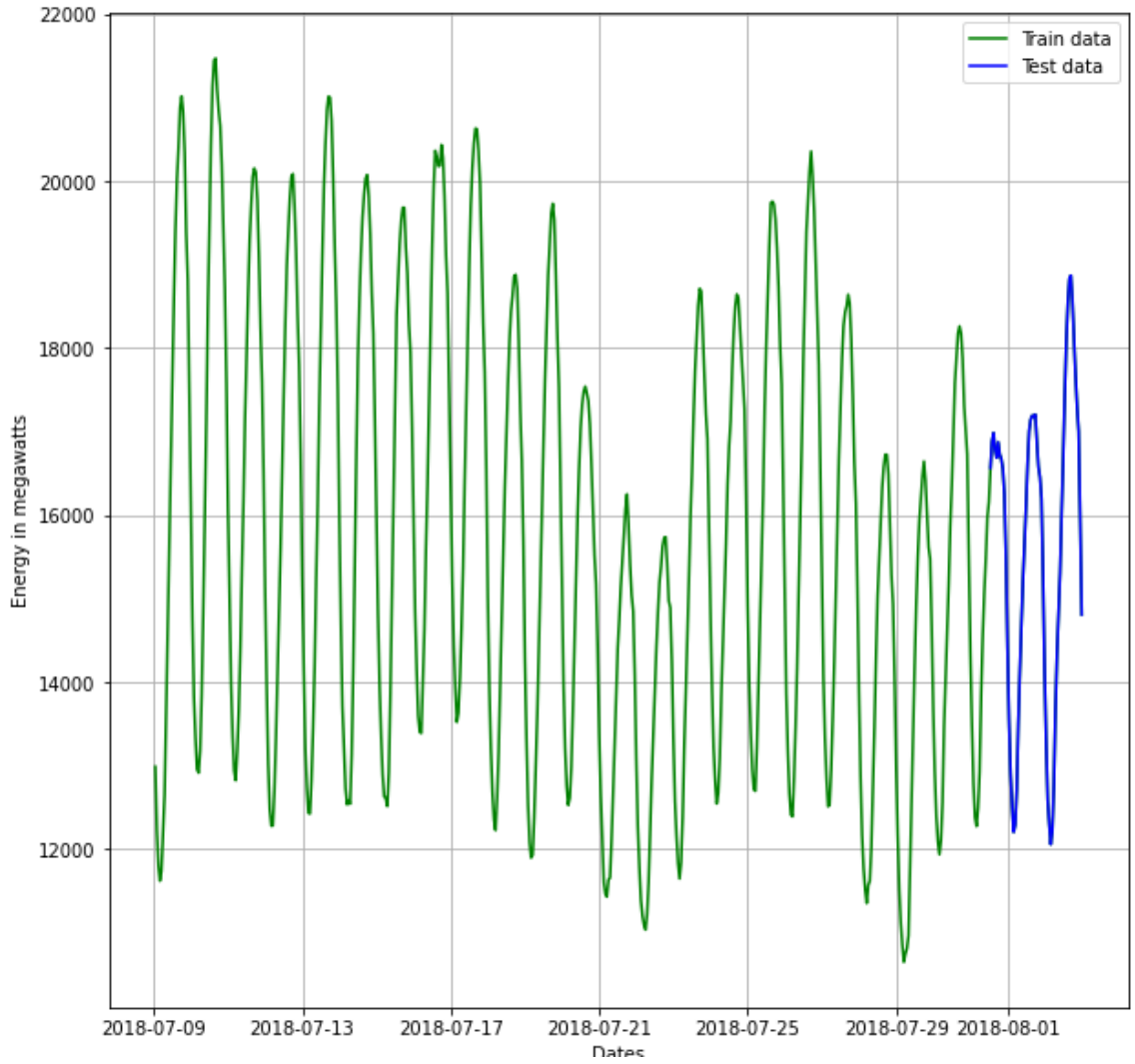
6. Feature Engineering:

- The code creates a lag plot by shifting the 'AEP_MW' values by different time intervals (1, 5, 10, and 30) and combines them into a new data frame. This can be used for lag-based feature engineering.

```
values = DataFrame(df['AEP_MW'].values)
dataframe =
concat([values.shift(1),values.shift(5),values.shift(10),values
.sh ift(30), values], axis=1)
dataframe.columns = ['t', 't+1', 't+5', 't+10', 't+30']
result = dataframe.corr() print(result)
```

	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

```
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()
```

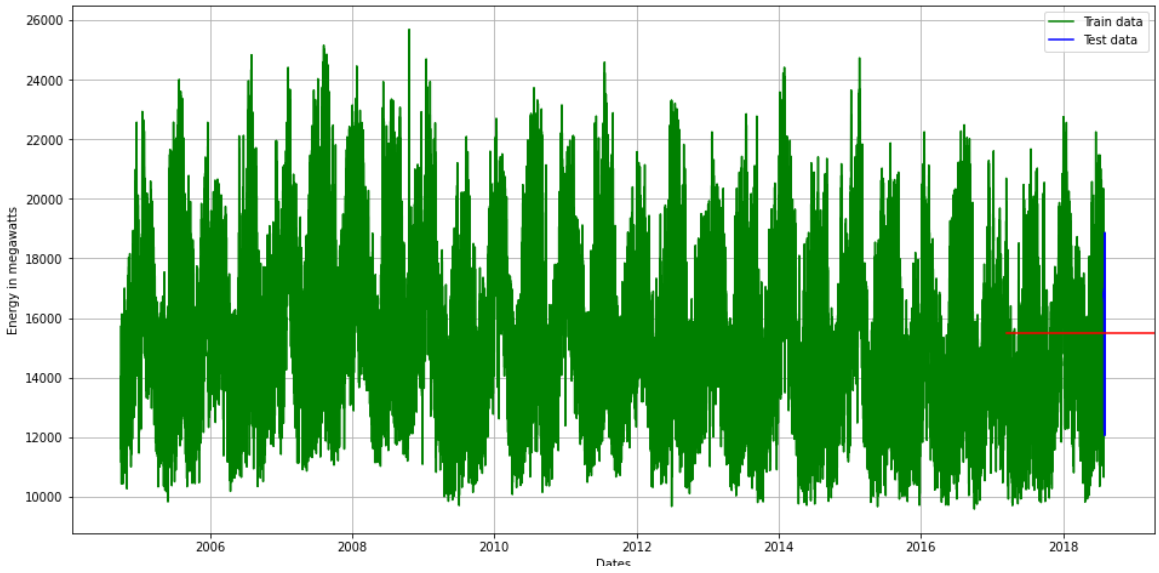


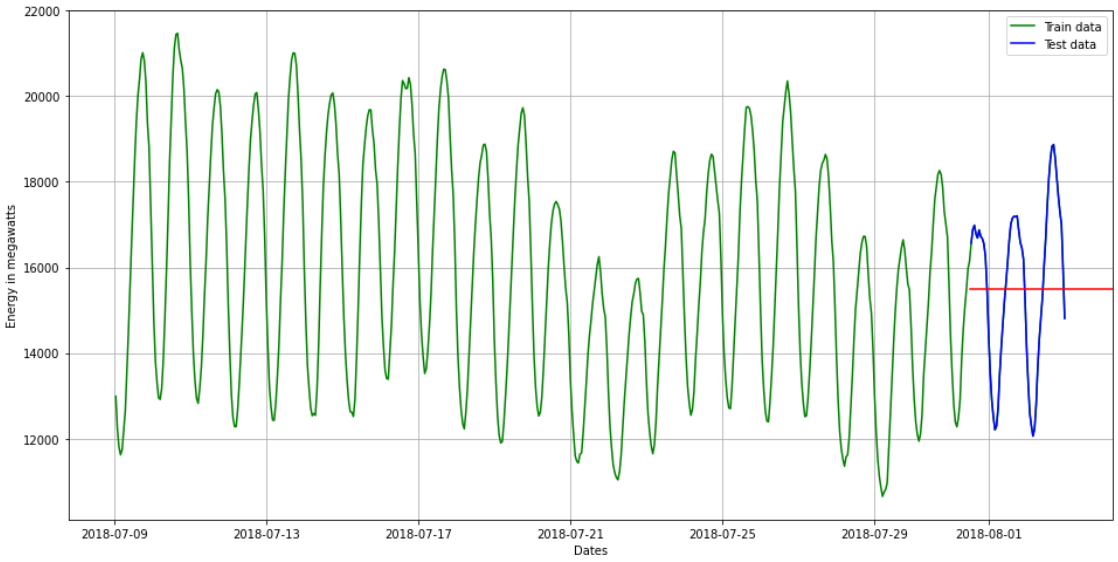
7. Correlation Analysis:

- The code calculates the correlation between lagged features and the original 'AEP_MW' data and prints the correlation matrix.

```
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_dat a), mean_value))))
print('MAE: '+str(mean_absolute_error(test_data['AEP_MW'],
np.full(len(test_da ta), mean_value))))
print('RMSE: '+str(sqrt(mean_squared_error(test_data['AEP_MW'],
np.full(len(te st_data), mean_value)))))
```

```
MSE: 3700885.0406027567
MAE: 1667.1805899362046
RMSE: 1923.768447761517
```






```
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:",      dftes
t[3])
     print("5. Critical Values :")
     for key, val in dftest[4].items():
         print("\t",key, ": ", val)
```

```
adf_test(df['AEP_MW'])
```

```
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
```

8. Train-Test Split:

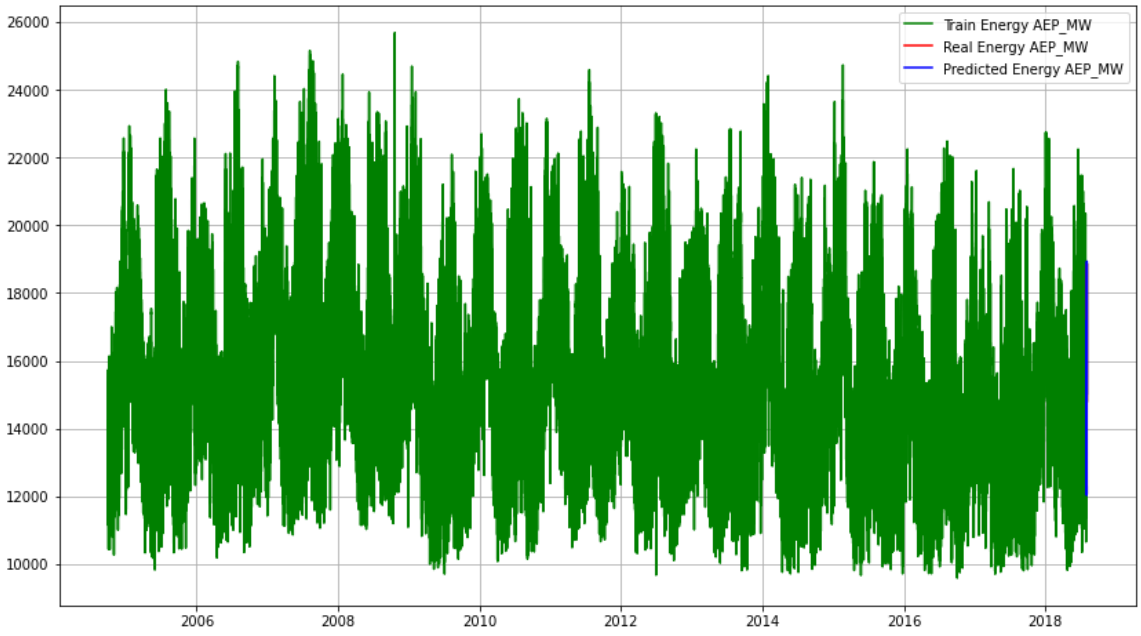
- The data is split into training and test sets. The last 60 data points are used for testing.

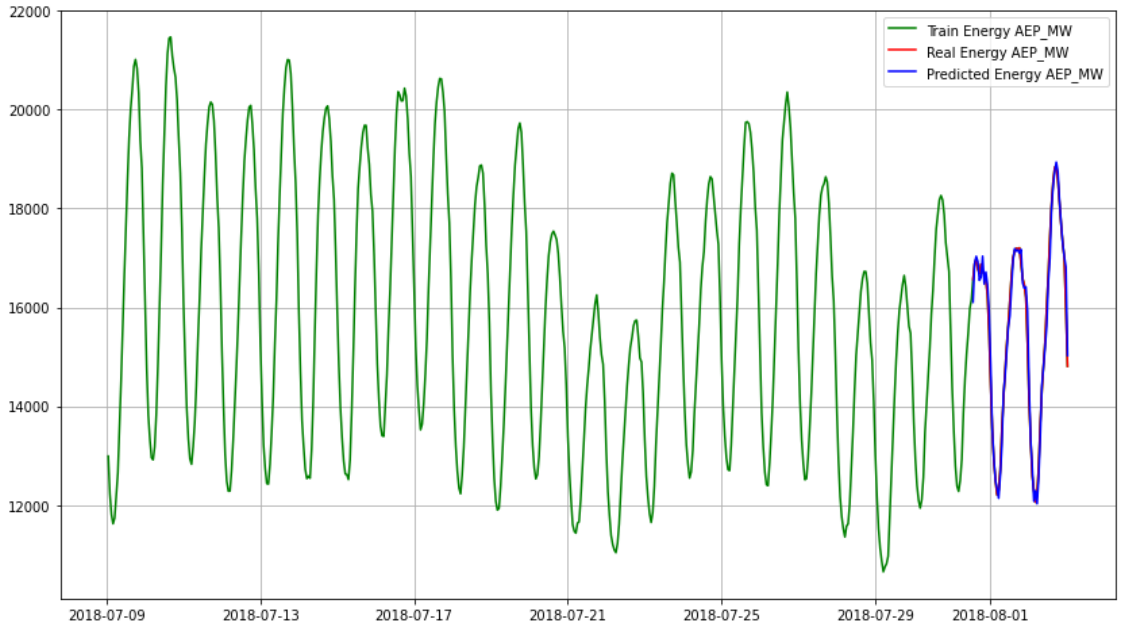
```
#Train Arima Model
train_arima = train_data['AEP_MW']
test_arima = test_data['AEP_MW']

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)
```

9. Visualization of Training and Test Data:

- The code generates line plots to visualize the training and test data, along with a red line indicating the mean value of 'AEP_MW' for reference.

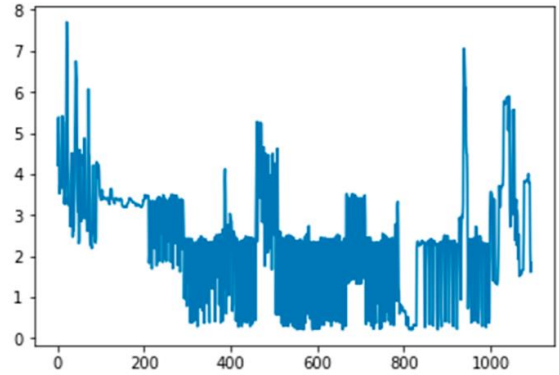
```
plt.figure(figsize=(14,8)) plt.plot(df.index, df['AEP_MW'], color='green',
label = 'Train Energy AEP_MW') plt.plot(test_data.index, y, color = 'red',
label = 'Real Energy AEP_MW') plt.plot(test_data.index, predictions, color =
'blue', label = 'Predicted Ener gy AEP_MW') plt.legend() plt.grid(True)
plt.show()

plt.figure(figsize=(14,8))
plt.plot(df.index[-600:], df['AEP_MW'].tail(600), color='green', label =
'Trai n 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
Ener gy 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))))
```



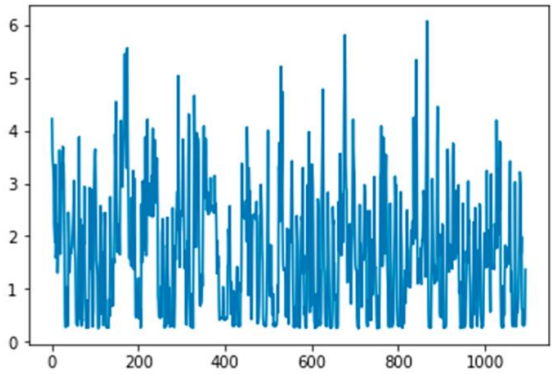


10. Model Evaluation Metrics:

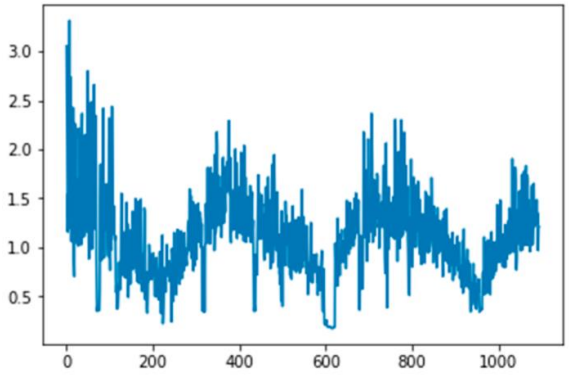
- The code calculates Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) for a baseline model that predicts the mean value.



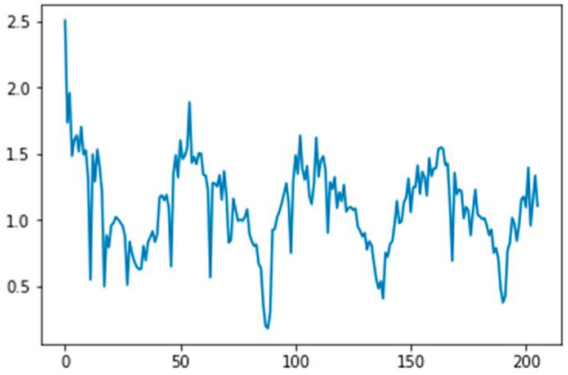
(**a**) Minutely dataset.



(**b**) Hourly dataset.



(**c**) Daily dataset.



(**d**) Weekly dataset.

11. Augmented Dickey-Fuller Test:

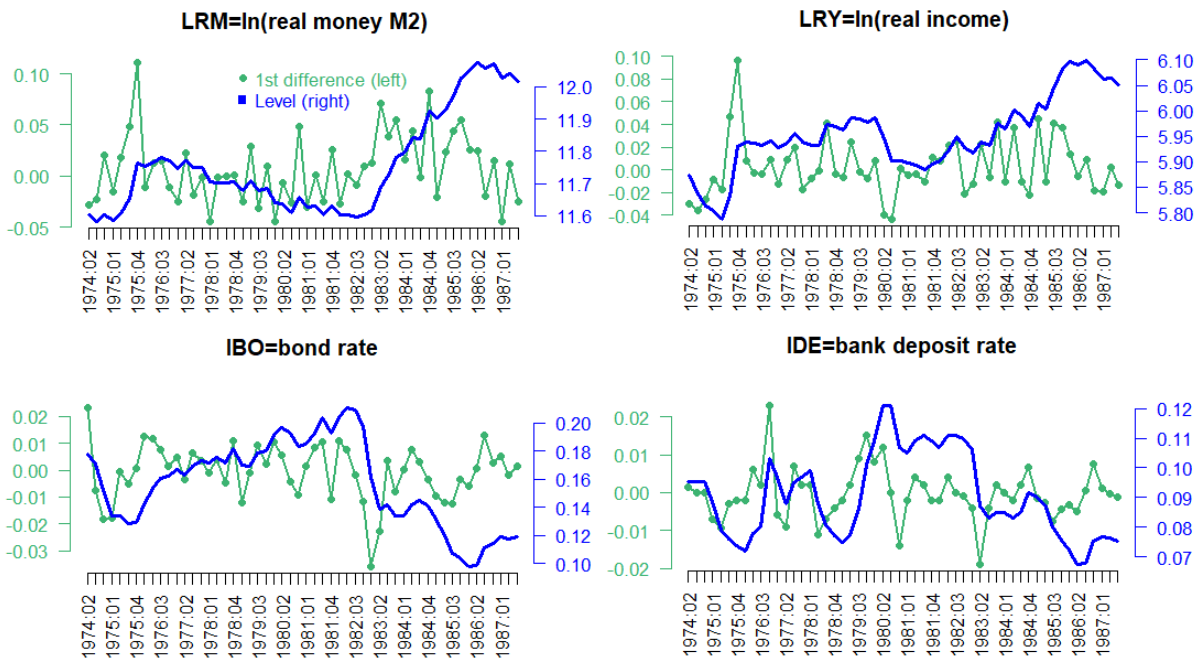
- The Augmented Dickey-Fuller (ADF) test is performed to check for stationarity in the time series data.

$$(1) \qquad \Delta Y_t = \gamma Y_{t-1} + \sum_{j-1}^{p} \left(\delta_j \Delta Y_{t-j} \right) + e_t$$

$$(2) \qquad \Delta Y_t = \alpha + \gamma Y_{t-1} + \sum_{j-1}^{p} \left(\delta_j \Delta Y_{t-j} \right) + e_t$$

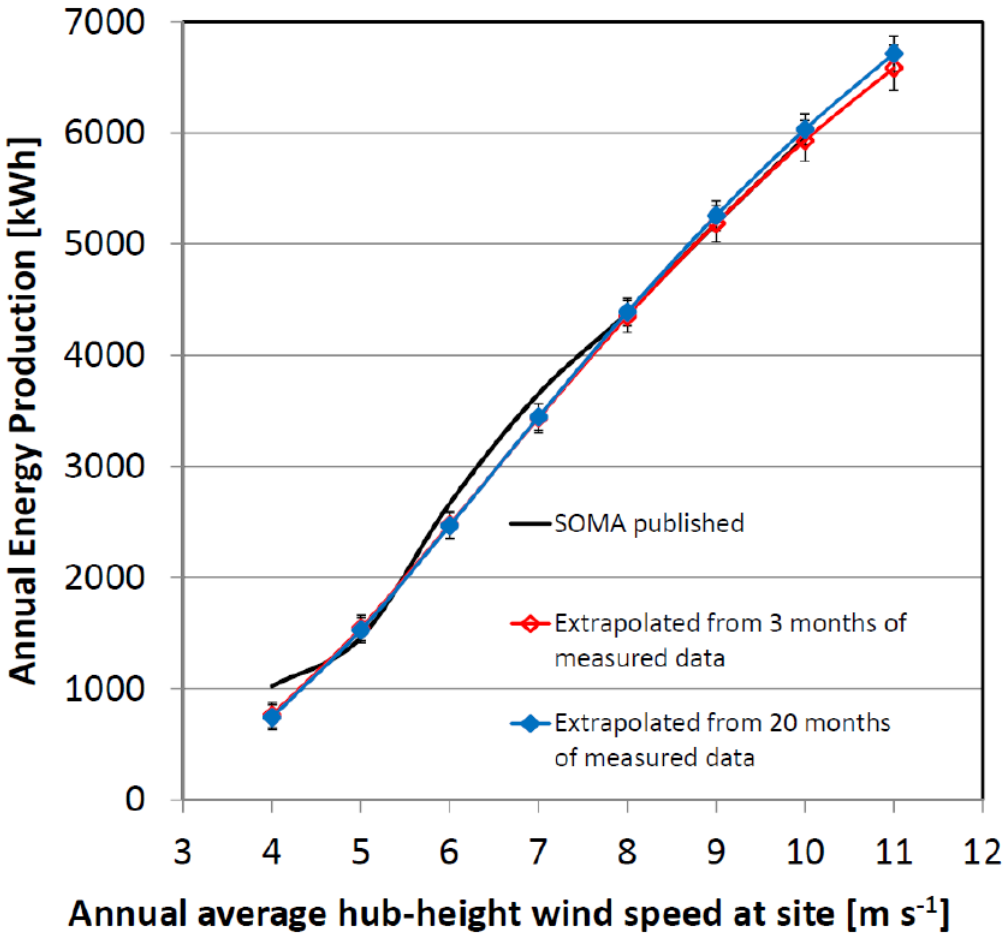
$$(3) \qquad \Delta Y_t = \alpha + \beta t + \gamma Y_{t-1} + \sum_{j-1}^{p} \left(\delta_j \Delta Y_{t-j} \right) + e_t$$

- *where*:



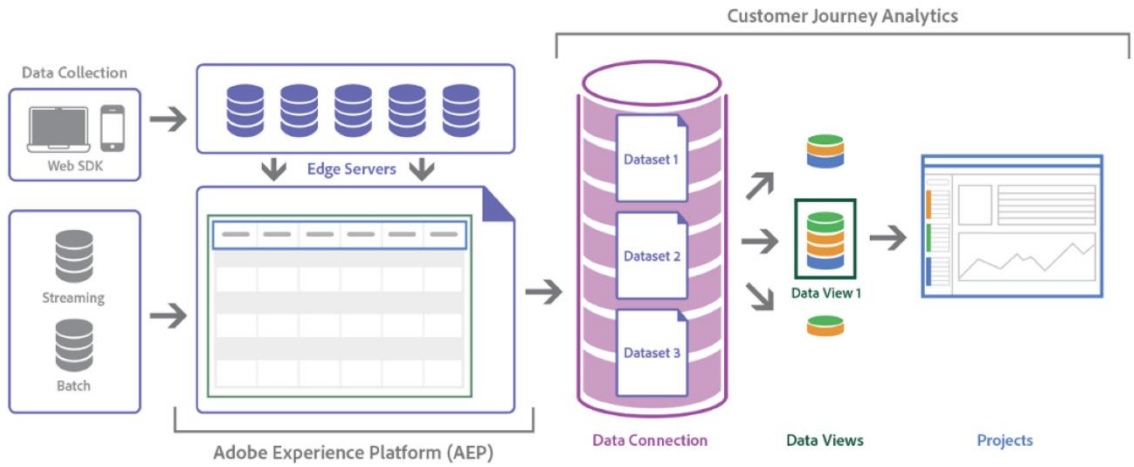
12. ARIMA Model Training and Forecasting:

- An ARIMA model is trained using the training data with order (5,1,0) and is used to make forecasts for the test data.



13. Visualization of ARIMA Model Predictions:

- The code generates line plots to visualize the actual, predicted, and training data.



14. Model Evaluation Metrics for ARIMA:

- The code calculates MSE, MAE, and RMSE for the ARIMA model's predictions.

```
MSE: 57710.45153428949
MAE: 177.320844006739
RMSE: 240.2299971574938
```

Variable	Test Type [a]	ADF Statistic	1% Level	5% Level	10% Level
Opening date	(C,T,1)	−5.039	−4.374	−3.603	−3.238
Duration	(C,T,1)	−4.361	−4.374	−3.603	−3.238
Tem10 [b]	(C,T,1)	−4.116	−4.374	−3.603	−3.238
Tem11	(C,T,1)	−4.578	−4.374	−3.603	−3.238
Tem12	(C,T,1)	−5.079	−4.394	−3.612	−3.243
Tem1	(C,T,1)	−4.055	−4.374	−3.603	−3.238
Tem2	(C,T,1)	−5.171	−4.374	−3.603	−3.238
Tem3	(C,T,1)	−5.563	−4.374	−3.603	−3.238
Tem4	(C,T,1)	−5.941	−4.374	−3.603	−3.238

CONCLUSION:

In conclusion, the provided Python script offers a comprehensive and structured approach to time series analysis and forecasting of energy consumption data, with a strong emphasis on data preprocessing. It demonstrates the critical importance of preparing data before delving into modeling and prediction tasks.

The script starts by importing essential libraries and then meticulously handles data loading and preprocessing. Data cleaning and transformation steps, such as dealing with missing values, removing duplicates, and ensuring data consistency, are integral to the process. It ensures that the dataset is well-structured and ready for analysis.

Once the data is appropriately preprocessed, the script proceeds to explore and analyze key aspects of time series data, including autocorrelation, mean values, and stationarity. It leverages ARIMA modeling to make accurate energy consumption predictions and evaluates the model's performance using metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

By emphasizing data preprocessing, the script provides a solid foundation for understanding and modeling energy consumption data. This systematic approach enables organizations to extract meaningful insights, make data-driven decisions, and optimize energy management. The result is not only improved forecasting accuracy but also a significant contribution to more efficient resource utilization and sustainability in a world where responsible energy consumption is of utmost importance.