Ai phase 5

Measure energy consumption documentation

By,

R.vikram

S.karmegam

M.gokul

R.vinayaprasath

G.dhanalakshmi

Introduction:

• In a world increasingly conscious of environmental Sustainability and cost-effectiveness, monitoring and Managing energy consumption in residential settings Has become an imperative. The “Home Energy Consumption Monitoring System”is an innovativeProject designed to address this need by providing Homeowners with a practical tool to track, analyze, And optimize their eneObjectives

Problem statement :

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.

To measure energy consumption, you’ll need to consider the following steps:

1. **Identify the Source:**

- Determine the energy source you want to measure (electricity, gas, water, etc.).

2. **Select a Measurement Tool:**

- For electricity, use a smart meter, energy monitor, or utility bills.

- For gas or water, specific meters or utility bills are typically used.

3. Read and Record Data:

- Regularly check the meter or monitoring device and record the measurements.

4. **Analyze and Interpret Data:**

- Track patterns and changes in consumption over time.

- Look for high consumption periods or identify where energy is being used.

5. **Use Energy-Efficient Practices**

Implement energy-efficient devices and practices to reduce consumption.

Design thinking:

Design thinking involves a human-centric approach to problem-solving. Applying this to energy consumption using machine learning could follow these steps:

- **Empathize**:

Understand users and their energy usage behavior.

Gather data on current energy consumption patterns and user needs.

- **Define**:

Define the problem area—where energy is being excessively consumed or where optimization is needed.

Create personas or user profiles to understand different consumer needs.

- **Ideate**:

Brainstorm potential solutions that incorporate machine learning to analyze energy usage data.

Consider predictive models, anomaly detection, or optimization algorithms.

- **Prototype**:

Develop a prototype model or system that can analyze and predict energy consumption.

Use historical data to train machine learning models for consumption prediction.

- **Test**:

Validate the machine learning model with real-time or simulated data.

Check the model's accuracy and efficacy in predicting and optimizing energy usage.

- **Implement**:

Integrate the machine learning system into the energy infrastructure for continuous monitoring and analysis.

Update the system based on new data or changing consumption patterns.

1. This design thinking process combines human understanding with machine learning to create a system that learns and adapts to optimize energy consumption over time

Dataset:

aep-hourly

Datetime	AEP_MW
2004-12-31 01:00:00	13478
2004-12-31 02:00:00	12865
2004-12-31 03:00:00	12577
2004-12-31 04:00:00	12517
2004-12-31 05:00:00	12670
2004-12-31 06:00:00	13038
2004-12-31 07:00:00	13692
2004-12-31 08:00:00	14297
2004-12-31 09:00:00	14719
2004-12-31 10:00:00	14941
2004-12-31 11:00:00	15184
2004-12-31 12:00:00	15009
2004-12-31 13:00:00	14808
2004-12-31 14:00:00	14522
2004-12-31 15:00:00	14349
2004-12-31 16:00:00	14107
2004-12-31 17:00:00	14410
2004-12-31 18:00:00	15174
2004-12-31 19:00:00	15261
2004-12-31 20:00:00	14774
2004-12-31 21:00:00	14363
2004-12-31 22:00:00	14045
2004-12-31 23:00:00	13478
2005-01-01 00:00:00	12892
2004-12-30 01:00:00	14097
2004-12-30 02:00:00	13667
2004-12-30 03:00:00	13451
2004-12-30 04:00:00	13379
2004-12-30 05:00:00	13506
2004-12-30 06:00:00	14121
2004-12-30 07:00:00	15066
2004-12-30 08:00:00	15771
2004-12-30 09:00:00	16047
2004-12-30 10:00:00	16245
2004-12-30 11:00:00	16377
2004-12-30 12:00:00	16138
2004-12-30 13:00:00	15886
2004-12-30 14:00:00	15503
2004-12-30 15:00:00	15206
2004-12-30 16:00:00	15049
2004-12-30 17:00:00	15161
2004-12-30 18:00:00	16085
2004-12-30 19:00:00	16508
2004-12-30 20:00:00	16306
2004-12-30 21:00:00	16223
2004-12-30 22:00:00	15931
2004-12-30 23:00:00	15207
2004-12-31 00:00:00	14316
2004-12-29 01:00:00	15223
2004-12-29 02:00:00	14731
2004-12-29 03:00:00	14503
2004-12-29 04:00:00	14432

Phase of development:

Feature engineering:

Introduction:

• Feature engineering is a crucial aspect of measuring energy Consumption effectively. It involves selecting, creating, and transforming Variables to extract meaningful information from energy data. By Carefully crafting features, we can uncover patterns, trends, and insights That help optimize energy usage, identify inefficiencies, and make Informed decisions. In this context, feature engineering plays a pivotal Role in enhancing the accuracy and efficiency of energy consumption Analysis and management systems.

Feature Engineering for Time Series :

That is something that looks like:

Time 1, value 1

Time 2, value 2

Time 3, value 3

To something that looks like:

Input 1, output 1

Input 2, output 2

Input 3, output 3

• Input variables are also called features in the field of machine learning, and the task before us is to create or invent new input features from our time series dataset. Ideally, we only want input features that best help the learning methods model the relationship between the inputs (X) and

the outputs (y) that we would like to predict.

Features:

• **Date Time Features**: these are components of the time step itself for each observation.

• **Lag Features: t**hese are values at prior time steps.

• **Window Features: t**hese are a summary of values over a fixed window of prior time steps.

"Date","Temperature"

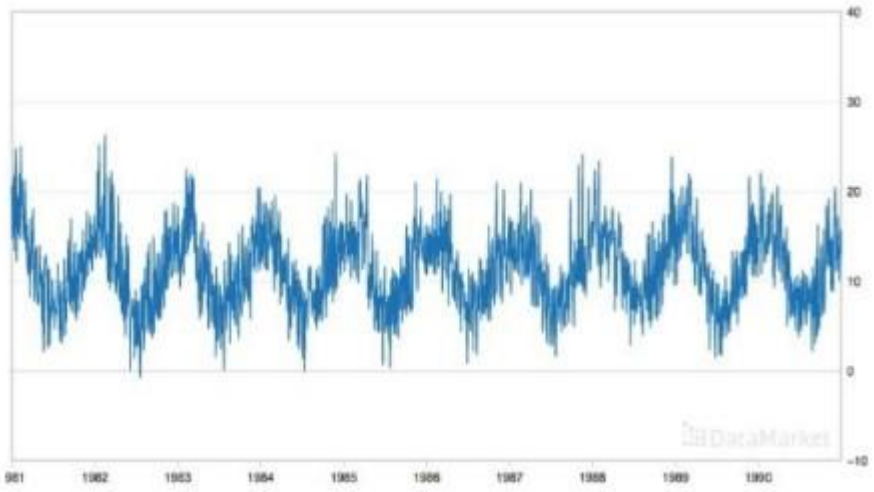
"1981-01-01",20.7

"1981-01-02",17.9

"1981-01-03",18.8

"1981-01-04",14.6

"1981-01-05",15.8



Program:

```
Import pandas as pd
Import numpy as np
Import matplotlib.pyplot as plt
From statsmodels.tsa.stattools import adfuller
From statsmodels.graphics.tsaplots import plot_acf
From statsmodels.tsa.statespace.sarimax import SARIMAX
From sklearn.metrics import mean_squared_error
From sklearn.metrics import mean_absolute_error
Read the data. In this case we are using just a single smart meter.
Df = pd.read_csv("../../data/ladpu_smart_meter_data_01.csv")
Print(df.info())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 105012 entries, 0 to 105011
Data columns (total 5 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 INTERVAL_TIME 105012 non-null object
1 1 METER_FID 105012 non-null int64
2 START_READ 105012 non-null float64
3 END_READ 105012 non-null float64
4 INTERVAL_READ 105012 non-null float64
Dtypes: float64(3), int64(1), object(1)
Memory usage: 4.0+ MB
```

None

Set the datetime index and resample to a daily frequency.

Df.set_index(pd.to_datetime(df[“INTERVAL_TIME”]), inplace=True)

Df.sort_index(inplace=True)

Print(df.info())

OUTPUT:

<class ‘pandas.core.frame.DataFrame’>

DatetimeIndex: 105012 entries, 2017-01-01 00:00:00 to 2019-12-31 23:45:00

Data columns (total 5 columns):

Column Non-Null Count Dtype

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

0 INTERVAL_TIME 105012 non-null object

1 1 METER_FID 105012 non-null int64

2 START_READ 105012 non-null float64

3 END_READ 105012 non-null float64

4 INTERVAL_READ 105012 non-null float64

Dtypes: float64(3), int64(1), object(1)

Memory usage: 4.8+ MB

None

Python:

Daily_data = pd.DataFrame(df.resample(“D”)[“INTERVAL_READ”].sum())

Print(daily_data.head())

OUTPUT:

INTERVAL_READ

INTERVAL_TIME

2017-01-01 11.7546

2017-01-02 15.0690

2017-01-03 11.6406

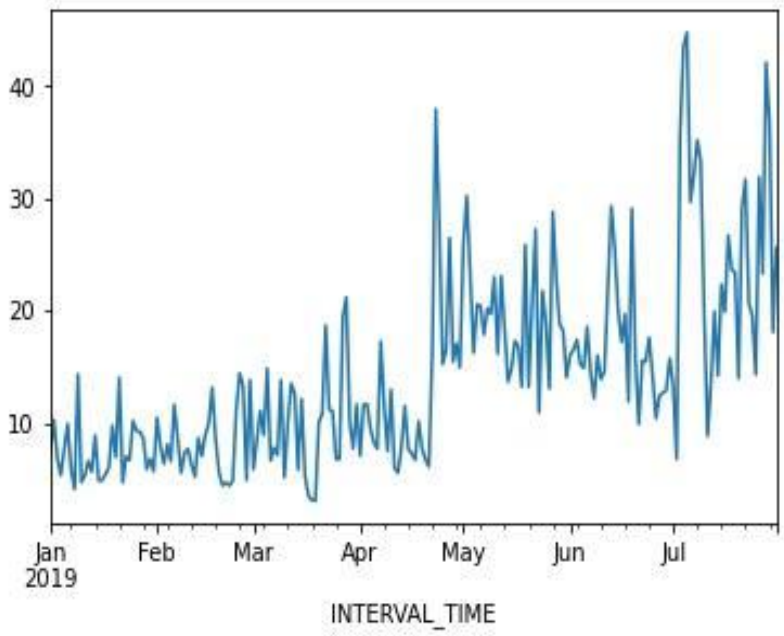
2017-01-04 22.0788

2017-01-05 12.8070

Subset to January – July, 2019 and plot the data.

Jan_july_2019 = daily_data.loc[“2019-01”: “2019-07”].copy()

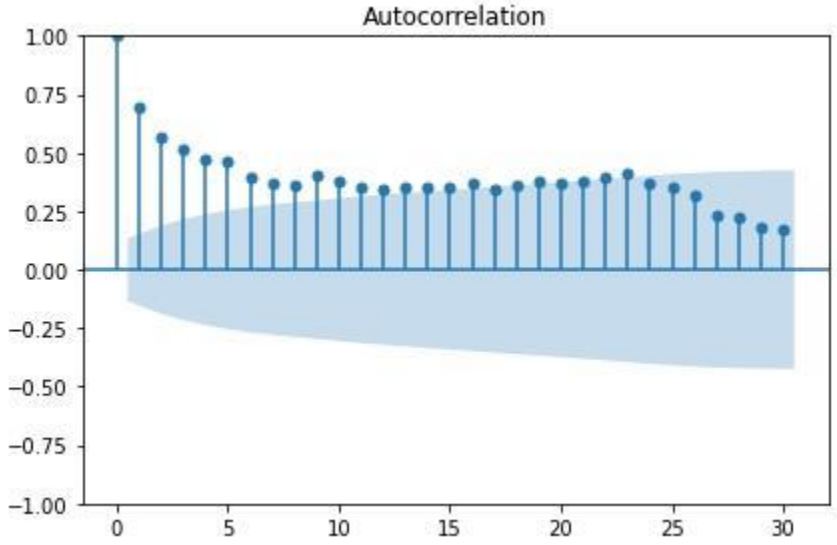
Jan_july_2019[“INTERVAL_READ”].plot()


40
30
20
10
Jan
2019
Feb
Mar
Apr
May
Jun
Jul
INTERVAL_TIME

test for autocorrelation:

PYTHON:

```
Plot_acf(jan_july_2019[“INTERVAL_READ”], lags=30)

Plt.tight_layout()
```



PYTHON:

```
Jan_july_2019_differenced = np.diff(jan_july_2019[“INTERVAL_READ”], n=1)

Print(“Head of dataframe:”, jan_july_2019.head())

Print(“\nDifferenced values:”, jan_july_2019_differenced[:5])
```

OUTPUT:

Head of dataframe: INTERVAL_READ

INTERVAL_TIME

2019-01-01 7.5324

2019-01-02 10.2534

2019-01-03 6.8544

2019-01-04 5.3250

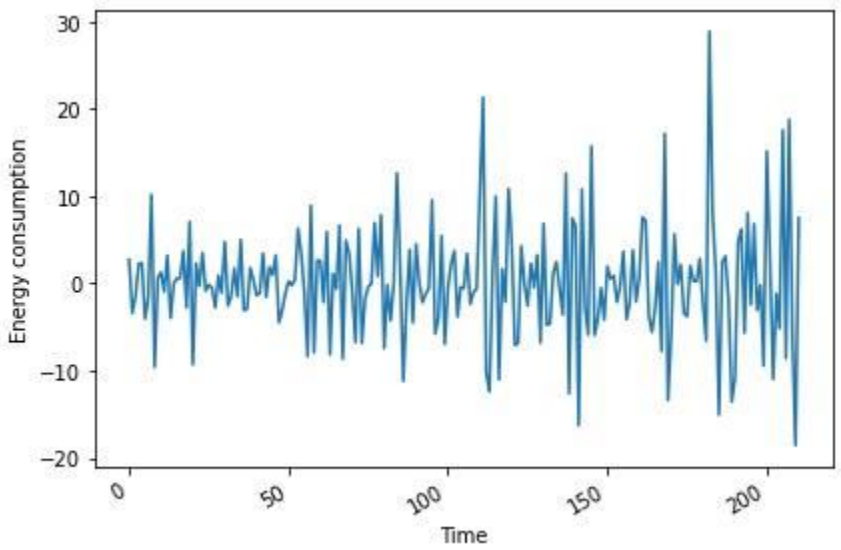
2019-01-05 7.5480

Differenced values: [2.721 -3.399 -1.5294 2.223 2.3466]

Plotting the result shows that there are no obvious trends in the differenced data.

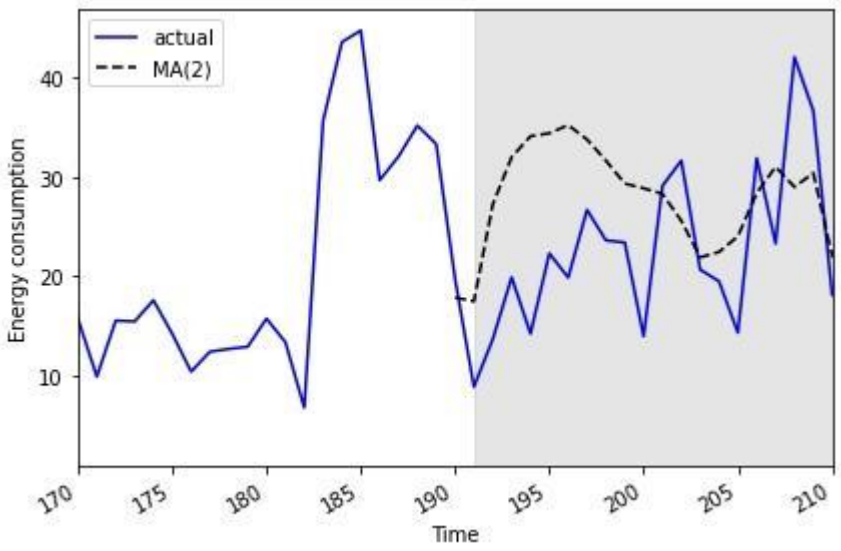
PYTHON:

```
Fig, ax = plt.subplots()
Ax.plot(jan_july_2019_differenced)
Ax.set_xlabel('Time')
Ax.set_ylabel('Energy consumption')
Fig.autofmt_xdate()
Plt.tight_layout()
```



PYTHON:

```
Fig, ax = plt.subplots()
Ax.plot(jan_july_2019['INTERVAL_READ'].values, 'b-', label='actual')
Ax.plot(jan_july_2019['pred_usage'].values, 'k—', label='MA(2)')
Ax.legend(loc=2)
Ax.set_xlabel('Time')
Ax.set_ylabel('Energy consumption')
Ax.axvspan(191, 210, color='#808080', alpha=0.2)
Ax.set_xlim(170, 210)
Fig.autofmt_xdate()
Plt.tight_layout()
```



- to evaluate the performance of the moving average forecast against

the actual values in the undifferenced data, we use the

mean_absolute_error from the sklearn library.

PYTHON:

Mae_MA_undiff =

mean_absolute_error(jan_july_2019['INTERVAL_READ'].iloc[191:],

Jan_july_2019['pred_usage'].iloc[191:])

Print("Mean absolute error of moving average forecast", mae_MA_undiff)

OUTPUT:

Mean absolute error of moving average forecast 8.457690692889582

Visualization:

• visualize the actual vs. predicted values for further analysis:

Python:

Import matplotlib.pyplot as plt

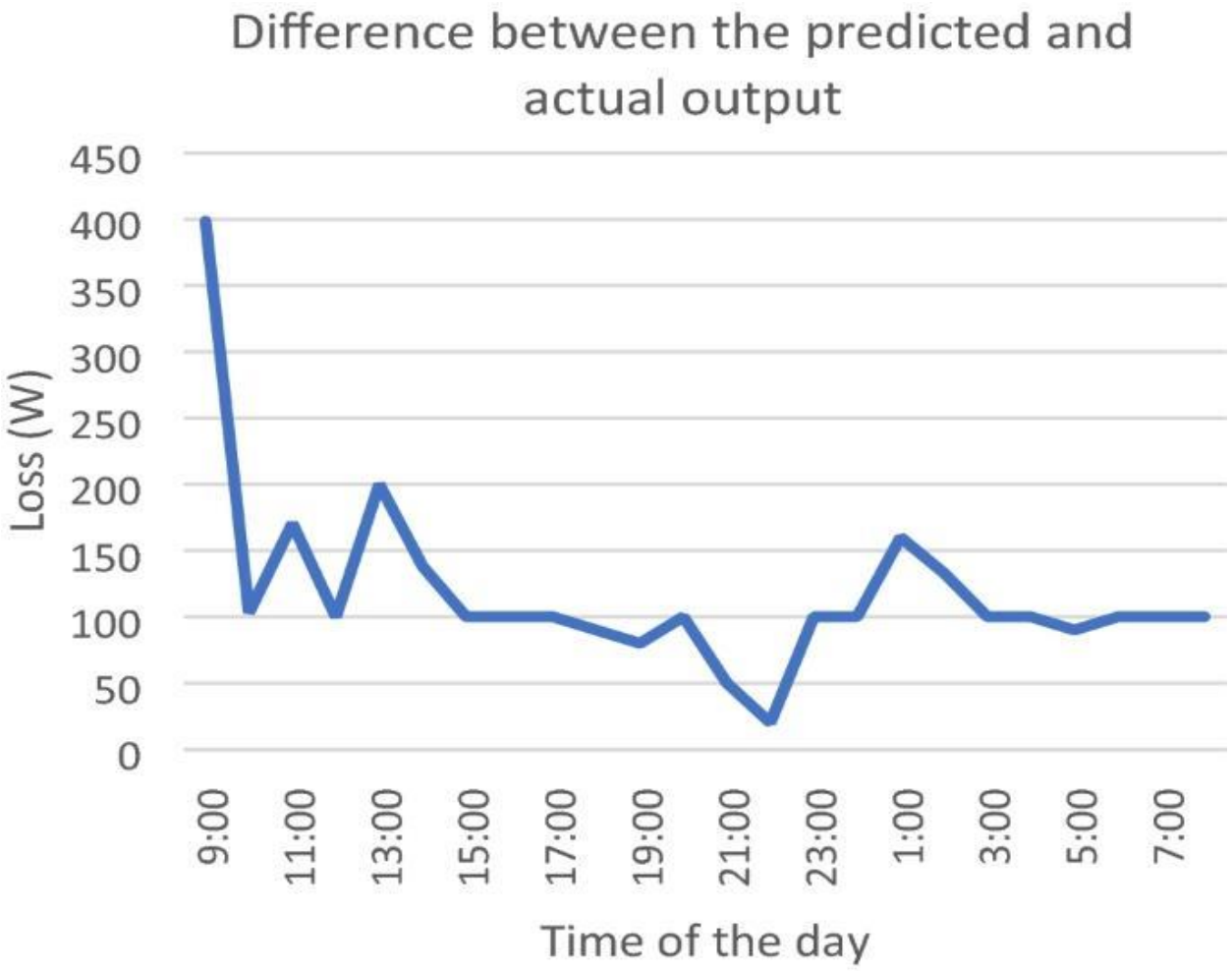
Plt.scatter(y_test, y_pred)

Plt.xlabel("Actual Energy Consumption")

Plt.ylabel("Predicted Energy Consumption")

Plt.title("Actual vs. Predicted Energy Consumption")

Plt.show()



Innovation technique:

Time series forecasting:

Stages in Time Series Forecasting:

• Solving a time series problem is a little different as compared to a regular modeling task. A simple/basic journey of solving a time series problem can be demonstrated through the following processes. We will understand about tasks which one needs to perform in every stage. We will also look at the python implementation of each stage of our problem-solving journey.

Steps are –

1. Visualizing time series:

• In this step, we try to visualize the series. We try to identify all the underlying patterns related to theseries like trend and seasonality. Do not worry about these terms right now, as we will discuss them during implementation. You can say that this is more a type of exploratory analysis of time series data.

2. Stationarising time series:

• A stationary time series is one whose statistical properties such as mean, variance, autocorrelation, etc. are all constant over time. Most statistical forecasting methods are based on the assumption thatthe time series can be rendered approximately

stationary (i.e., "stationarised") through the use of mathematical transformations. A stationarised series is relatively easy to predict: you simply predict that its statistical properties will be the same in the future as they have been in the past! Another reason for trying to stationarise a time series is to be able to obtain meaningful sample statistics such as means, variances, and correlations with other variables. Such statistics are useful as descriptors of future behavioronly if the series is stationary. For example, if the series is consistently increasing over time, the sample mean and variance will grow with the size of the sample, and they will always underestimate the mean and variance in future periods. And if the mean and variance of a series are not well-defined, then neither are its correlations with other variables.

3. Finding the best parameters for our model:

• We need to find optimal parameters for forecasting models one's we have a stationary series. These parameters come from the ACF and PACF plots. Hence, this stage is more about plotting above two graphs and extracting optimal model parameters based on them. Do not worry, we will cover on how to determine these parameters during the implementation part below!

4. Fitting model:

• Once we have our optimal model parameters, we can fit an ARIMA model to learn the pattern of the series. Always remember that time series algorithms work on stationary data only hence making a series stationary is important aspect.

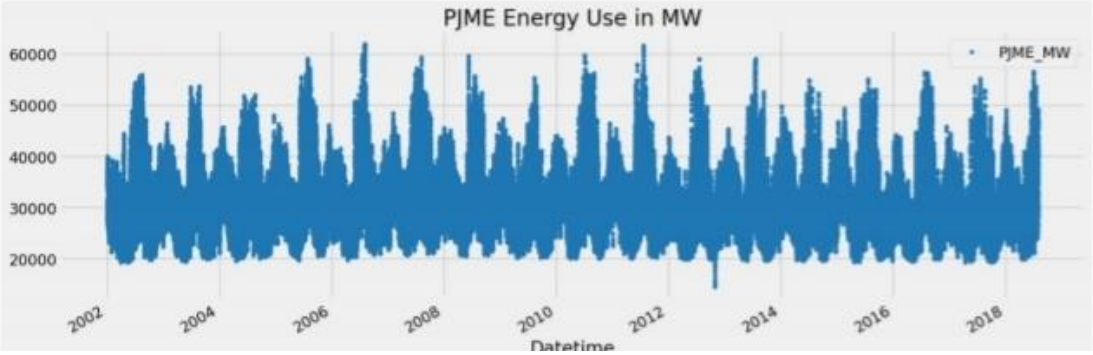
5.predictions:

• After fitting our model, we will be predicting the future in this stage. Since we are now familiar with a basic flow of solving a time series problem, let us get to the implementation.

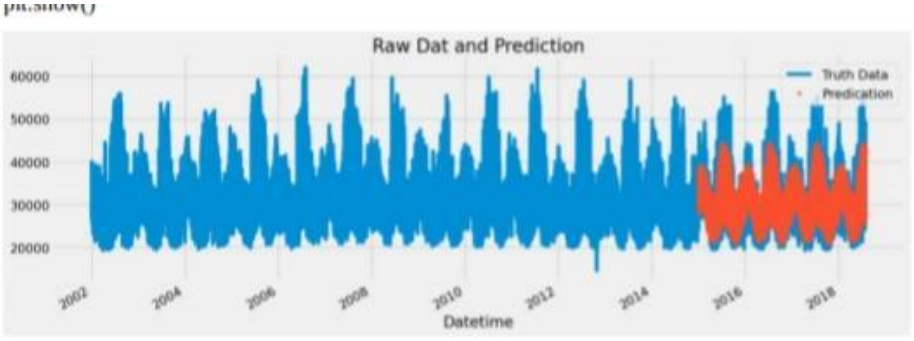
Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
 import seaborn as sns
import xgboost as xgb
from sklearn.metrics import mean_squared_error
color_pal = sns.color_palette()
plt.style.use('fivethirtyeight')
df = pd.read_csv('../input/hourly-energy-
consumption/PJME_hourly.csv')
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)
df.plot(style='.',
figsize=(15, 5),
color=color_pal[0],
title='PJME Energy Use in MW')
plt.show()
```

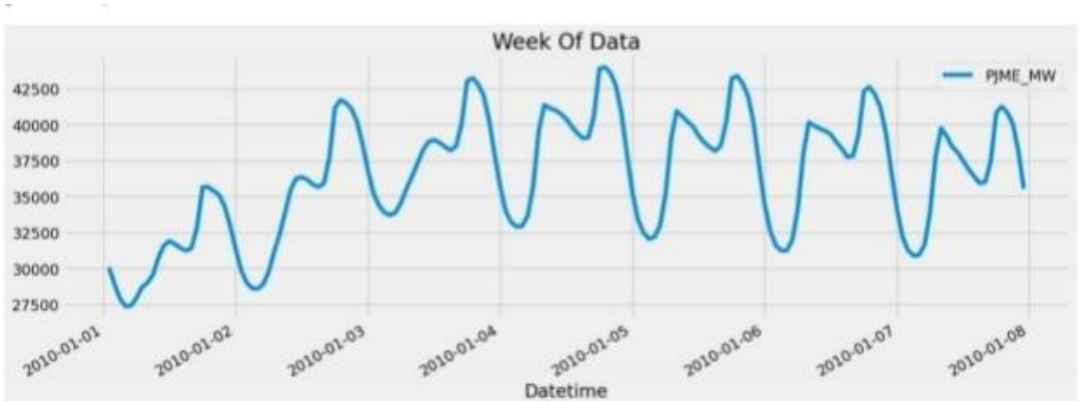
```
train = df.loc[df.index < '01-01-2015']
```




```
test = df.loc[df.index >= '01-01-2015']
fig, ax = plt.subplots(figsize=(15, 5))
train.plot(ax=ax, label='Training Set', title='Data
Train/Test Split')
 test.plot(ax=ax, label='Test Set')
ax.axvline('01-01-2015', color='black', ls='--')
ax.legend(['Training Set', 'Test Set'])
plt.show()
```

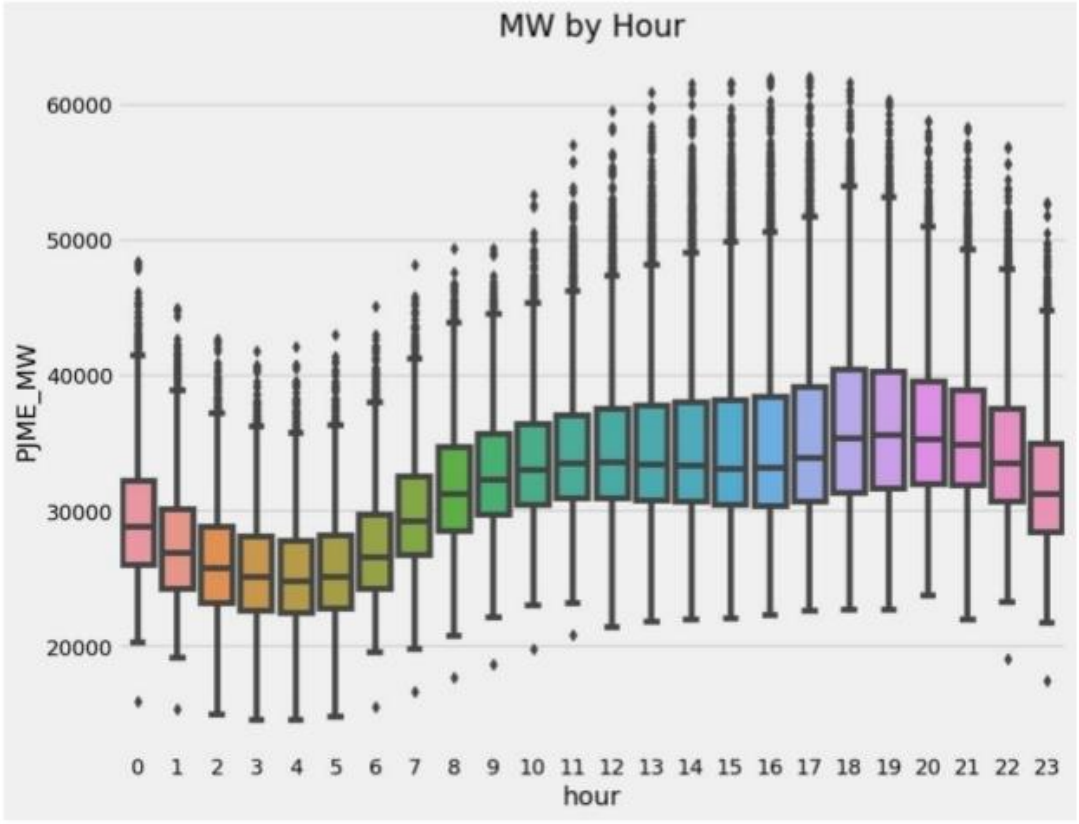


```
df.loc[(df.index > '01-01-2010') & (df.index < '01-08-
2010')] \
.plot(figsize=(15, 5), title='Week Of Data')
plt.show()
```




```
def create_features(df):
"""
Create time series features based on time series
index.
"""
df = df.copy()
df['hour'] = df.index.hour
df['dayofweek'] = df.index.dayofweek
df['quarter'] = df.index.quarter
df['month'] = df.index.month
df['year'] = df.index.year
df['dayofyear'] = df.index.dayofyear
```

```
df['weekofmonth'] = df.index.day
df['weekofyear'] = df.index.isocalendar().week
return df
df = create_features(df)
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='hour', y='PJME_MW')
ax.set_title('MW by Hour')
plt.show(
```



```
ax = df.loc[(df.index > '04-01-2018') & (df.index < '04-
08-2018')]['PJME_MW'] \
.plot(figsize=(15, 5), title='Week Of Data')
df.loc[(df.index > '04-01-2018') & (df.index < '04-08-
2018')]['prediction'] \ .plot(style='.')
plt.legend(['Truth Data','Prediction'])
plt.show()
```


```
score = np.sqrt(mean_squared_error(test['PJME_MW'],
test['prediction']))
print(f'RMSE Score on Test set: {score:0.2f}')
```

RMSE Score on Test set: 3721.75

Calculate Error Look at the worst and best predicted days

```
test['error'] = np.abs(test[TARGET] - test['prediction'])
test['date'] = test.index.date
test.groupby(['date'])['error'].mean().sort_values(asce
nding=False).head(10)
```

date

Output:

2016-08-13 12839.597087

2016-08-14 12780.209961

2016-09-10 11356.302979

2015-02-20 10965.982259

2016-09-09 10864.954834

2018-01-06 10506.845622

2016-08-12 10124.051595

2015-02-21 9881.803711

2015-02-16 9781.552246

2018-01-07 9739.144206

Name: error, dtype: floa

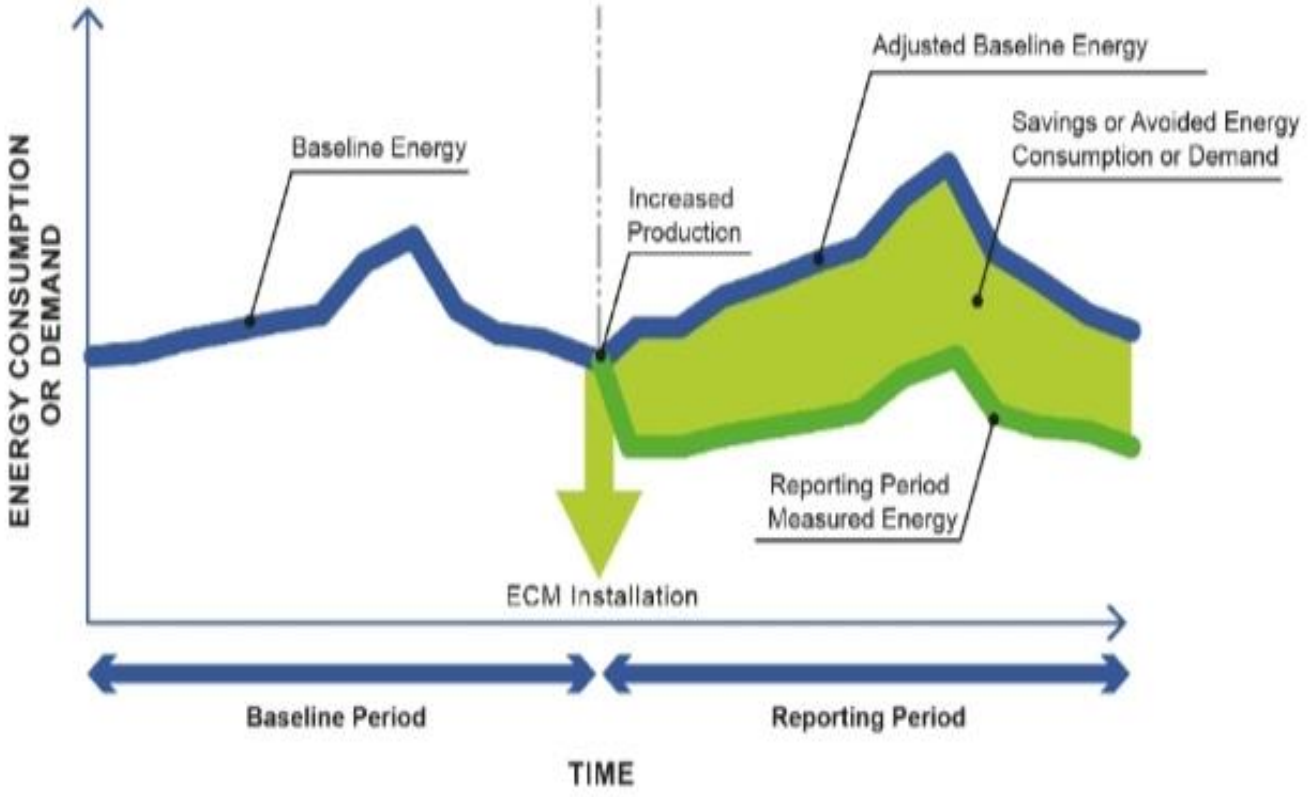
Invocation technique: (documentation):

Electricity Prediction for Measurement and Verification:

- Prediction is a common machine learning (ML) technique used on building energy consumption data. This process is valuable for anomaly detection, load profile-based building control and measurement and verification procedures.

- The graphic below comes from the IPMVP to show how prediction can be used for M&V to calculate how much energy would have been consumed if an energy savings intervention had not been implemented.

Prediction for Measurement and Verification:

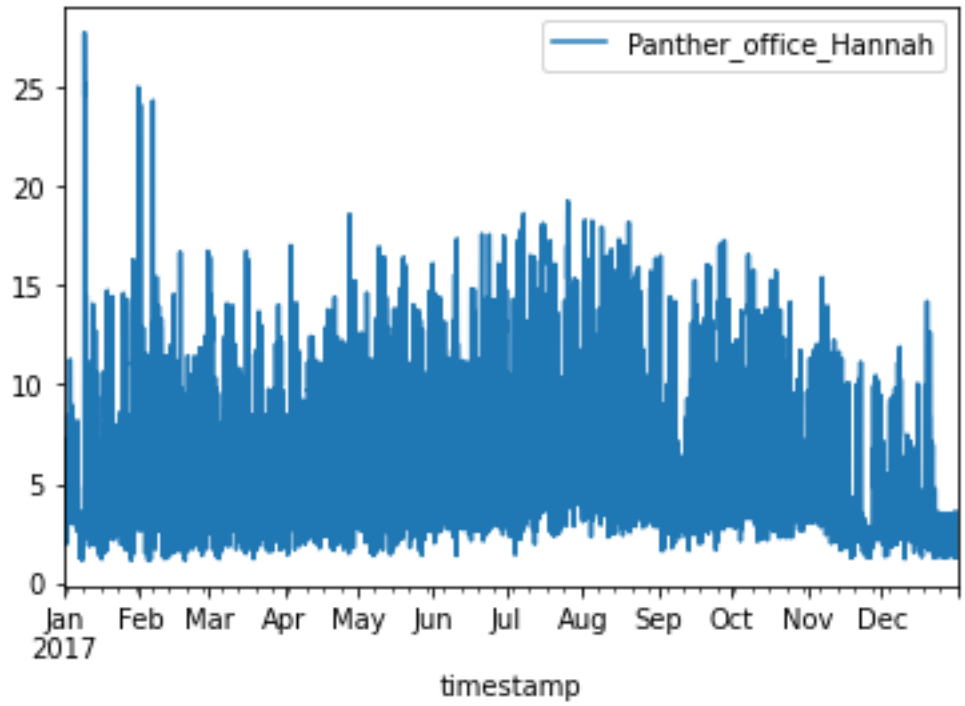


Load electricity data and weather data:

- First we can load the data from the BDG in the same as our previous weather analysis influence notebook from the Construction Phase videos

```
Elec_all_data = pd.read_csv("../input/buildingdatagenomeproject2/electricity_cleaned.csv", index_col='timestamp', parse_dates=True)


Elec_all_data.info()


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

DatetimeIndex: 17544 entries, 2016-01-01 00:00:00 to 2017-12-31 23:00:00

Columns: 1578 entries, Panther_parking_Lorriane to Mouse_science_Micheal

Dtypes: float64(1578)

Memory usage: 211.3 MB


Buildingname = 'Panther_office_Hannah'


Office_example_prediction_data = pd.DataFrame(elec_all_data[buildingname].truncate(before='2017-01-01')).fillna(method='ffill')


Office_example_prediction_data.info()


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

DatetimeIndex: 8760 entries, 2017-01-01 00:00:00 to 2017-12-31 23:00:00

Data columns (total 1 columns):

 #   Column              Non-Null Count  Dtype

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

 0   Panther_office_Hannah  8760 non-null   float64

Dtypes: float64(1)

Memory usage: 136.9 KB


Office_example_prediction_data.plot()
```

<AxesSubplot:xlabel='timestamp'>



Weather_data = pd.read_csv("../input/buildingdatagenomeproject2/weather.csv", index_col='timestamp', parse_dates=True)

Weather_data_site = weather_data[weather_data.site_id == 'Panther'].truncate(before='2017-01-01')

Weather_data_site.info()

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

DatetimeIndex: 8760 entries, 2017-01-01 00:00:00 to 2017-12-31 23:00:00

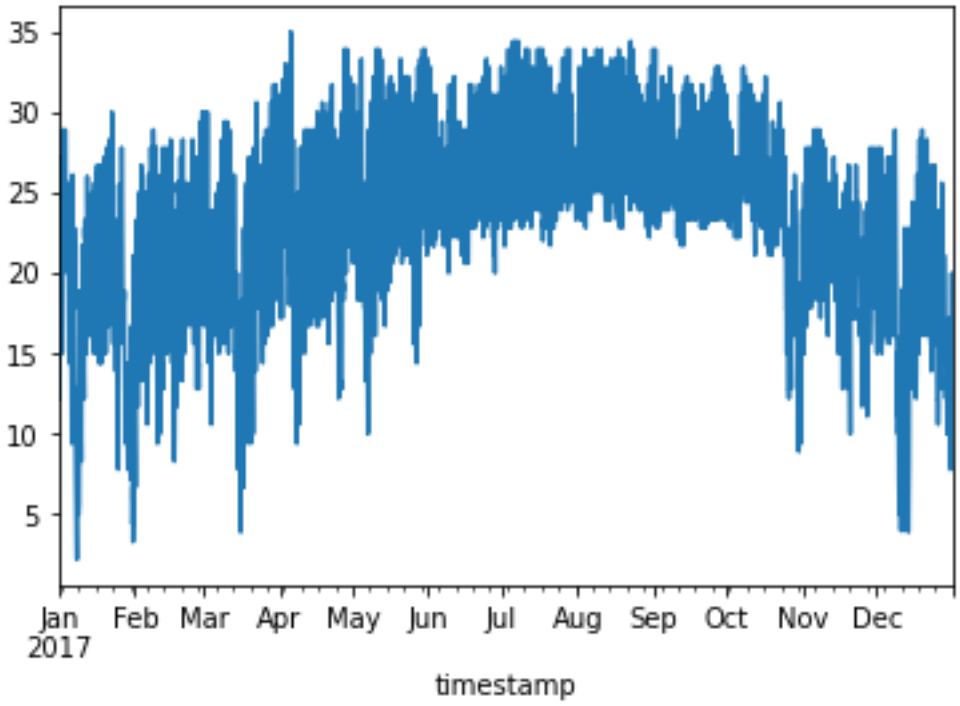
Data columns (total 9 columns):

#	Column	Non-Null Count	Dtype

```
 0   site_id         8760 non-null   object
 1   airTemperature  8760 non-null   float64
 2   cloudCoverage   5047 non-null   float64
 3   dewTemperature  8760 non-null   float64
 4   precipDepth1HR  8752 non-null   float64
 5   precipDepth6HR  329 non-null    float64
 6   seaLvlPressure  8522 non-null   float64
 7   windDirection   8511 non-null   float64
 8   windSpeed       8760 non-null   float64
Dtypes: float64(8), object(1)
Memory usage: 684.4+ KB
```

```
Weather_hourly = weather_data_site.resample("H").mean()
Weather_hourly_nooutlier = weather_hourly[weather_hourly > -40]
Weather_hourly_nooutlier_nogaps = weather_hourly_nooutlier.fillna(method='ffill')

Temperature = weather_hourly_nooutlier_nogaps["airTemperature"]

Temperature.plot()
```

```
<AxesSubplot:xlabel='timestamp'>
```



Create Train and Test Datasets:

- The model is given a set of data that will be used to train the model to predict a specific objectice. In this case, we will use a few simple time series features as well as outdoor air temperature to predict how much energy a building uses.For this demonstration, we will use three months of data from April, May, and June to prediction July.

Training_months = [4,5,6]

Test_months = [7]

- divide the data set by using the datetime index of the data frame and a function known as .isin to extract the months for the model

Trainingdataoffice_example_prediction_data[office_example_prediction_data.index.month.isin(training_months)]

```
Testdata=office_example_prediction_data[office_example_prediction_data.index.month.isin(test_months)]


Trainingdata.info()


<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2184 entries, 2017-04-01 00:00:00 to 2017-06-30 23:00:00
Data columns (total 1 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Panther_office_Hannah  2184 non-null   float64
Dtypes: float64(1)
Memory usage: 34.1 KB


Testdata.info()


<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 744 entries, 2017-07-01 00:00:00 to 2017-07-31 23:00:00
Data columns (total 1 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Panther_office_Hannah  744 non-null    float64
Dtypes: float64(1)
Memory usage: 11.6 KB
```

- extract the training input data features that will go into the model and the training label data which is what are are targeting to predict.

Encoding Categorical Variables:

- We use the pandas .get_dummies() function to change the temporal variables of time of day and day of week into categories that the model can use more effectively. This process is known as enconding

```
Train_features = pd.concat([pd.get_dummies(trainingdata.index.hour),
                            Pd.get_dummies(trainingdata.index.dayofweek),
Pd.DataFrame(temperature[temperature.index.month.isin(training_months)].values)],
axis=1).dropna()
```

```
Train_features.head()
```

0	1	2	3	4	5	6	7	8	9	…	22	23
	0	1	2	3	4	5	6	0				
0	1	0	0	0	0	0	0	0	0	…	0	
	0	0	0	0	0	0	1	0	21.7			
1	0	1	0	0	0	0	0	0	0	…	0	
	0	0	0	0	0	0	1	0	21.0			
2	0	0	1	0	0	0	0	0	0	…	0	
	0	0	0	0	0	0	1	0	18.9			
3	0	0	0	1	0	0	0	0	0	…	0	
	0	0	0	0	0	0	1	0	20.6			
4	0	0	0	0	1	0	0	0	0	…	0	
	0	0	0	0	0	0	1	0	21.0			

5 rows × 32 columns

Train a K-Neighbor Model:

- This model was chosen after following the process in the cheat sheet until a model that worked and provided good results was found.

```
Model = KNeighborsRegressor().fit(np.array(train_features), np.array(trainingdata.values));
```

```
Test_features = np.array(pd.concat([pd.get_dummies(testdata.index.hour),
                                    Pd.get_dummies(testdata.index.dayofweek),
Pd.DataFrame(temperature[temperature.index.month.isin(test_months)].values)],
axis=1).dropna())
```

Use the Model to predict for the Test period:

- Then the model is given the test_features from the period which we want to predict. We can then merge those results and see how the model did

```
Predictions = model.predict(test_features)
```

```
Predicted_vs_actual = pd.concat([testdata, pd.DataFrame(predictions, index=testdata.index)],
axis=1)
```

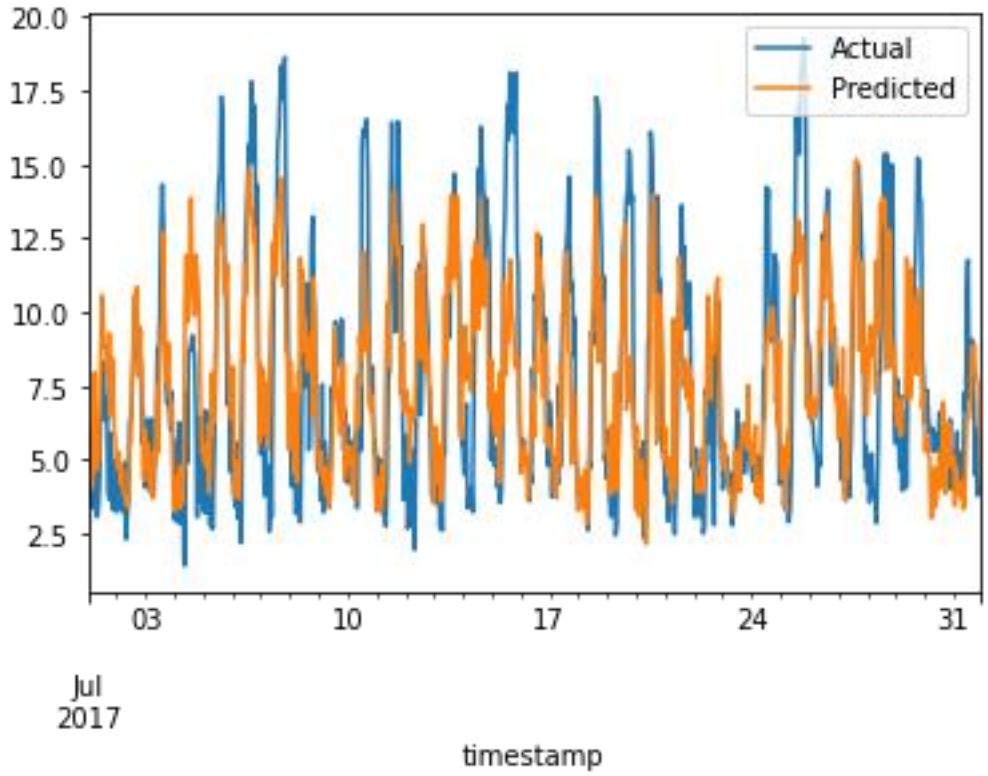
```
Predicted_vs_actual.columns = [“Actual”, “Predicted”]
```

```
Predicted_vs_actual.head()
```

	Actual	Predicted
Timestamp		
2017-07-01 00:00:00	5.3370	5.75910
2017-07-01 01:00:00	3.8547	6.02898
2017-07-01 02:00:00	5.5751	4.39686
2017-07-01 03:00:00.	4.1248	4.23180
2017-07-01 04:00:00	3.3497	4.03858

Predicted_vs_actual.plot()

<AxesSubplot:xlabel='timestamp'>

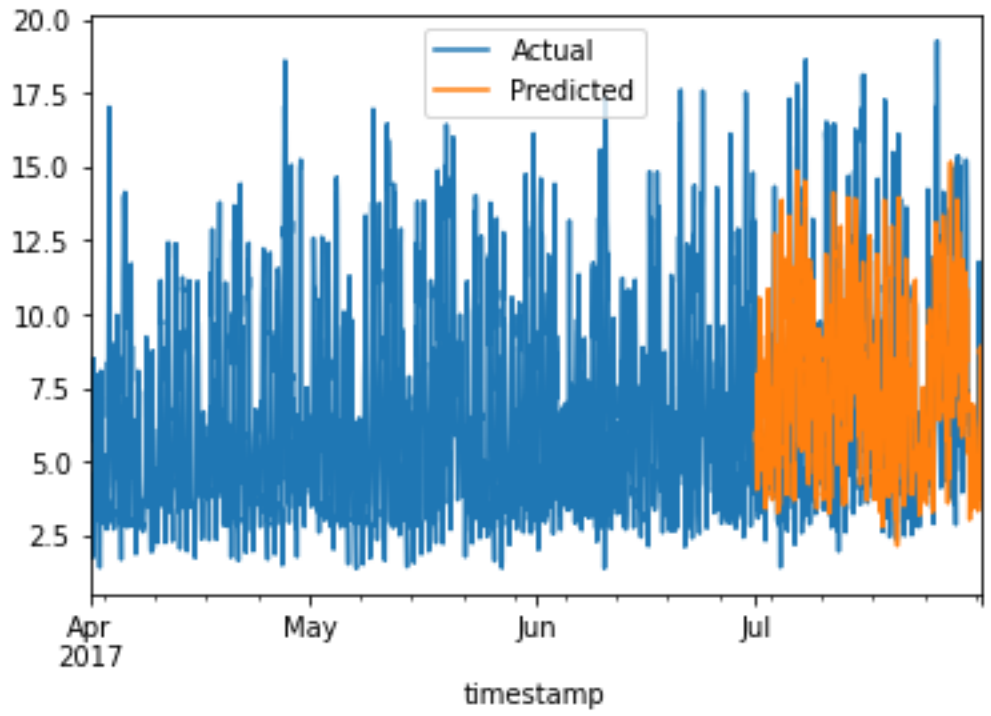


Trainingdata.columns = [“Actual”]

Predicted_vs_actual_plus_training = pd.concat([trainingdata, predicted_vs_actual], sort=True)

Predicted_vs_actual_plus_training.plot()

<AxesSubplot:xlabel=’timestamp’>



Evaluation metrics:

- In order to understand quanitatively how the model performed, we can use various evaluation metrics to understand how well the model compared to reality.In this situation, let's use the error metric Mean Absolute Percentage Error (MAPE)

```
# Calculate the absolute errors

Errors = abs(predicted_vs_actual['Predicted'] – predicted_vs_actual['Actual'])

# Calculate mean absolute percentage error (MAPE) and add to list

MAPE = 100 * np.mean((errors / predicted_vs_actual['Actual']))
```

MAPE

34.22379683897996

Conclusion:

- Measuring energy consumption is crucial for understanding usage patterns, identifying inefficiencies, and implementing strategies for conservation. It helps in cost savings, environmental impact reduction, and making informed decisions about energy usage. Whether for households or industries, monitoring energy consumption is fundamental in driving sustainability and efficient resource management.