Development part 1

MEASURE ENERGY CONSUMPTION :

Abstract:

- The "Home Energy Consumption Monitoring System" is a vital endeavor aimed at promoting sustainability and energy efficiency in residential settings. In today's world, where environmental concerns and the cost of energy are paramount, a system that empowers homeowners to monitor and manage their energy consumption becomes increasingly essential.

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 innovative project designed to address this need by providing homeowners with a practical tool to track, analyze, and optimize their eneObjectives

Objectives:

Real-time Monitoring:

- The project aims to enable homeowners to monitor their Energy consumption in real time, allowing for a clear and Immediate understanding of electricity usage Patterns

Data analysis:

- By processing and analyzing the data collected from energy Sensors, the system provides insights into energy usage Patterns, helping users identify opportunities for conservation.

User-Friendly Interface:

- A user-friendly interface is created to present data in an easilyComprehensible manner, empowering homeowners to make Informed decisions about energy efficiency.

Promoting Sustainability:

- The project aligns with broader sustainability goals, Encouraging responsible energy consumption andEnvironmental awareness.

Components:

Hardware:

- The system integrates hardware components, including current sensors and microcontrollers (e.g., Raspberry Pi or Arduino), to collect energy consumption data.

Software:

- Software applications are developed to process, analyze, and visualize energy consumption data, ensuring a seamless user experience.

Data Analysis:

- Advanced data analysis techniques are employed to identify usage patterns, anomalies, and trends in energy consumption.

Implementation:

Real-Time Data:

- The system collects and displays energy consumption data in real time, allowing users to see how their actions impact energy usage.

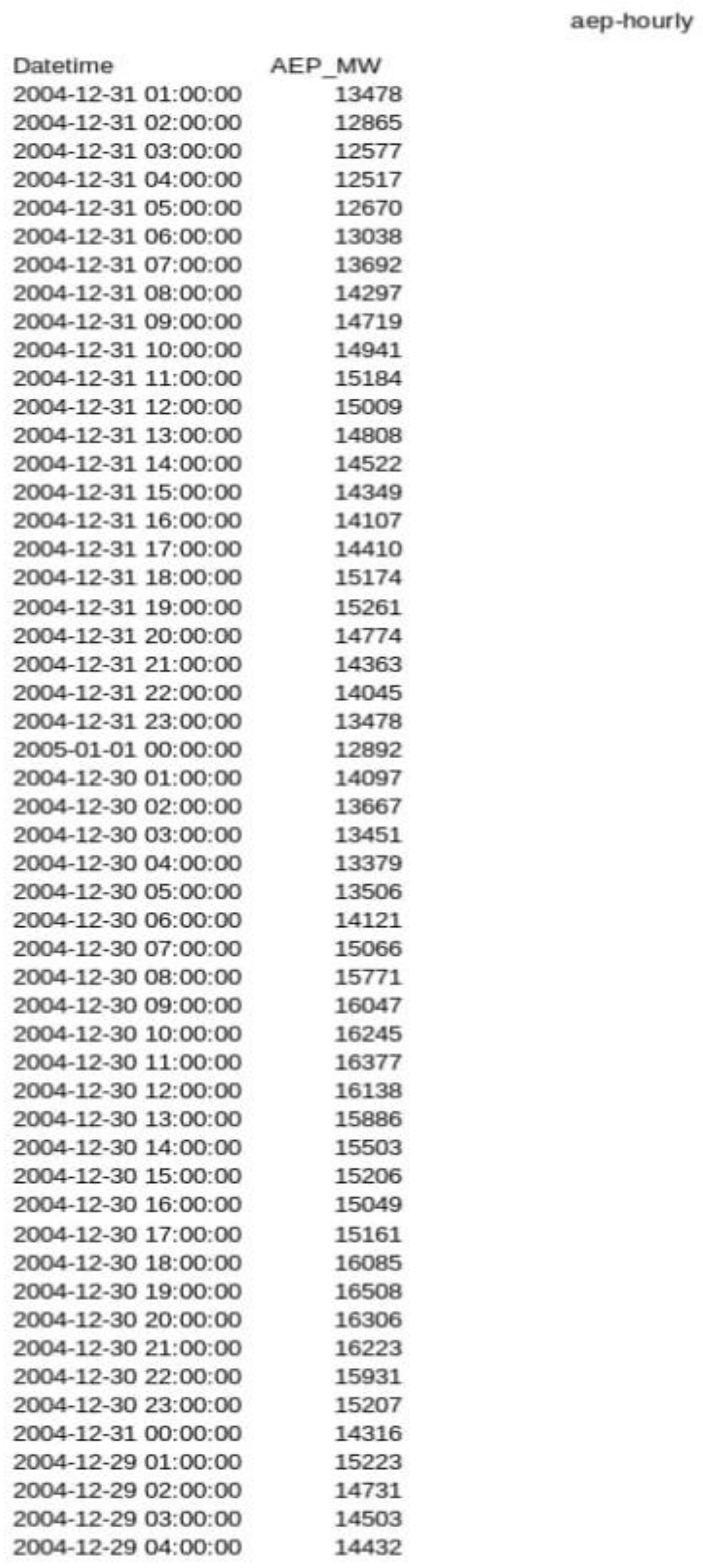
User Interaction:

- Users can interact with the system through a user-friendly interface, enabling them to access real-time and historical data.

Insights:

- The project generates insights into energy consumption patterns and offers recommendations to optimize energy use.

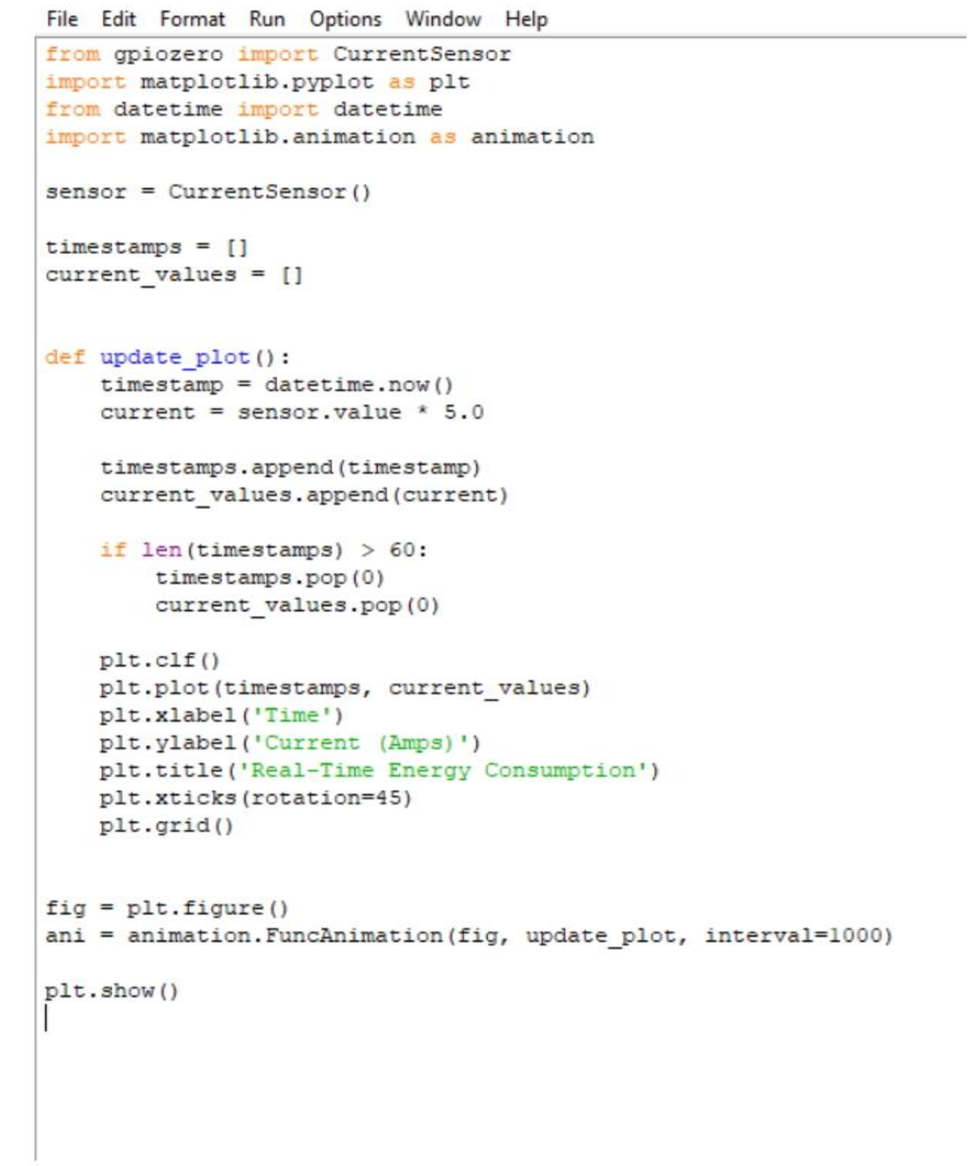
Dataset:.



aep-hourly

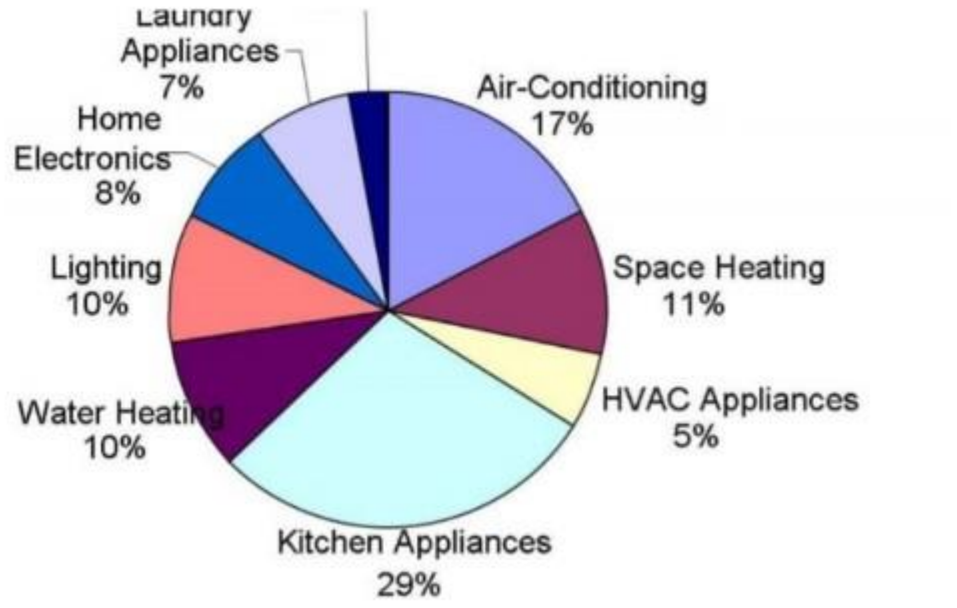
Datetime	AEP_MW
2004-12-31 01:00:00	13478
2004-12-31 02:00:00	12865
2004-12-31 03:00:00	12577
2004-12-31 04:00:00	12517
2004-12-31 05:00:00	12670
2004-12-31 06:00:00	13038
2004-12-31 07:00:00	13692
2004-12-31 08:00:00	14297
2004-12-31 09:00:00	14719
2004-12-31 10:00:00	14941
2004-12-31 11:00:00	15184
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

Program :



```python
from gpiozero import CurrentSensor
import matplotlib.pyplot as plt
from datetime import datetime
import matplotlib.animation as animation

sensor = CurrentSensor()

timestamps = []
current_values = []


def update_plot():
    timestamp = datetime.now()
    current = sensor.value * 5.0

    timestamps.append(timestamp)
    current_values.append(current)

    if len(timestamps) > 60:
        timestamps.pop(0)
        current_values.pop(0)

    plt.clf()
    plt.plot(timestamps, current_values)
    plt.xlabel('Time')
    plt.ylabel('Current (Amps)')
    plt.title('Real-Time Energy Consumption')
    plt.xticks(rotation=45)
    plt.grid()


fig = plt.figure()
ani = animation.FuncAnimation(fig, update_plot, interval=1000)

plt.show()
```

Output:



Impact:

- The "Home Energy Consumption Monitoring System" has the potential to significantly impact the way homeowners manage their energy consumption. By providing real-time data and insights, the system empowers individuals to take control of their energy use, reduce costs, and contribute to a more sustainable future.

Conclusion:

- This project represents a critical step towards fostering energy-conscious households, environmental sustainability, and personal financial savings. It exemplifies the role of technology in addressing contemporary challenges, highlighting the potential for individual actions to collectively impact global energy sustainability.

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 behavior only 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

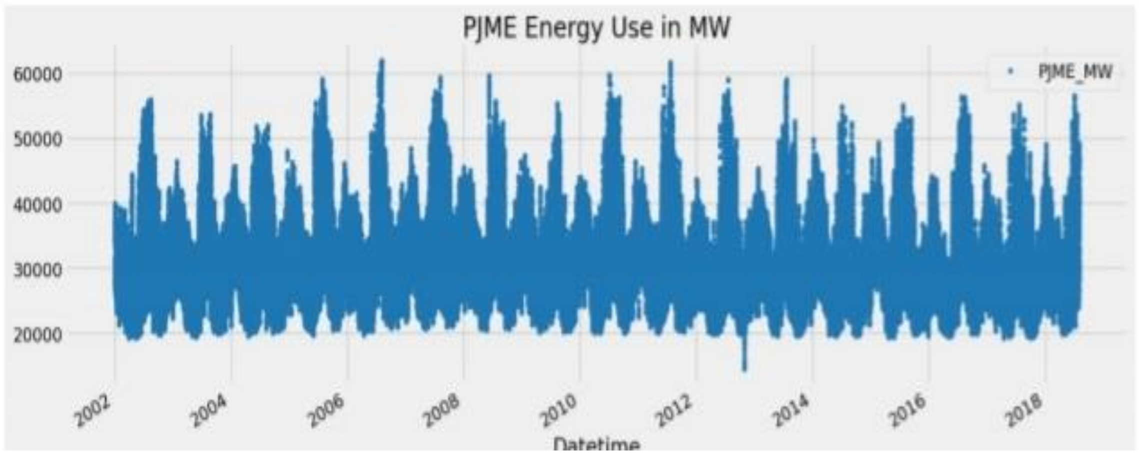
- Once we have our optimal model parameters, we can fit an ARIMA model to learn the pattern of the series. Always remember that time series algorithms work on stationary data only hence making a series stationary is important aspect.

5.predictions:

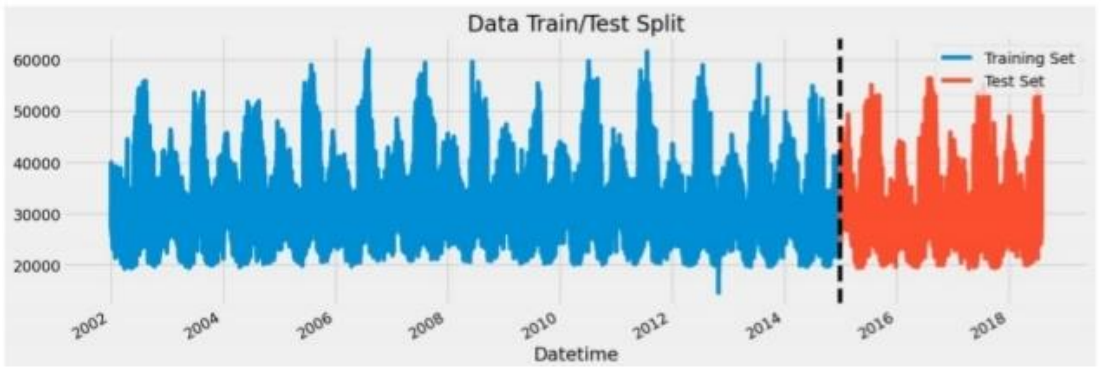
- After fitting our model, we will be predicting the future in this stage. Since we are now familiar with a basic flow of solving a time series problem, let us get to the implementation.

Problem Statement

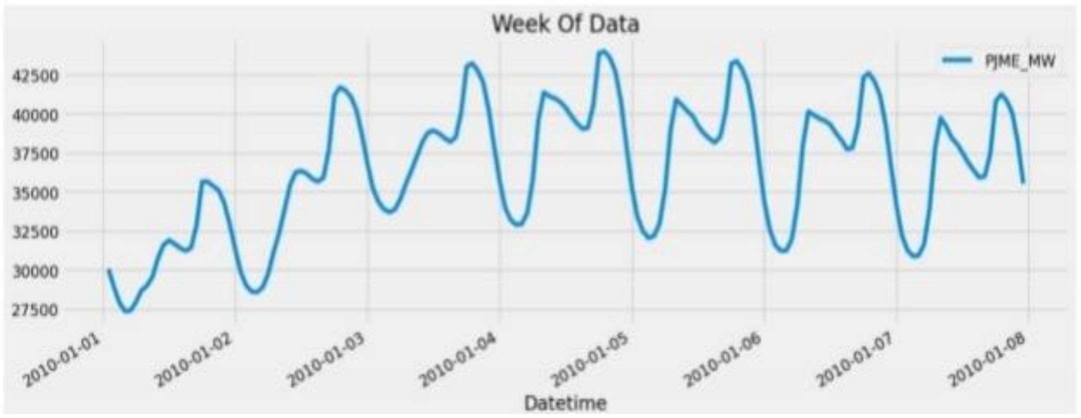
```
 import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.metrics import mean_squared_error
color_pal = sns.color_palette()
plt.style.use('fivethirtyeight')
df = pd.read_csv('../input/hourly-energy-
consumption/PJME_hourly.csv')
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()
```



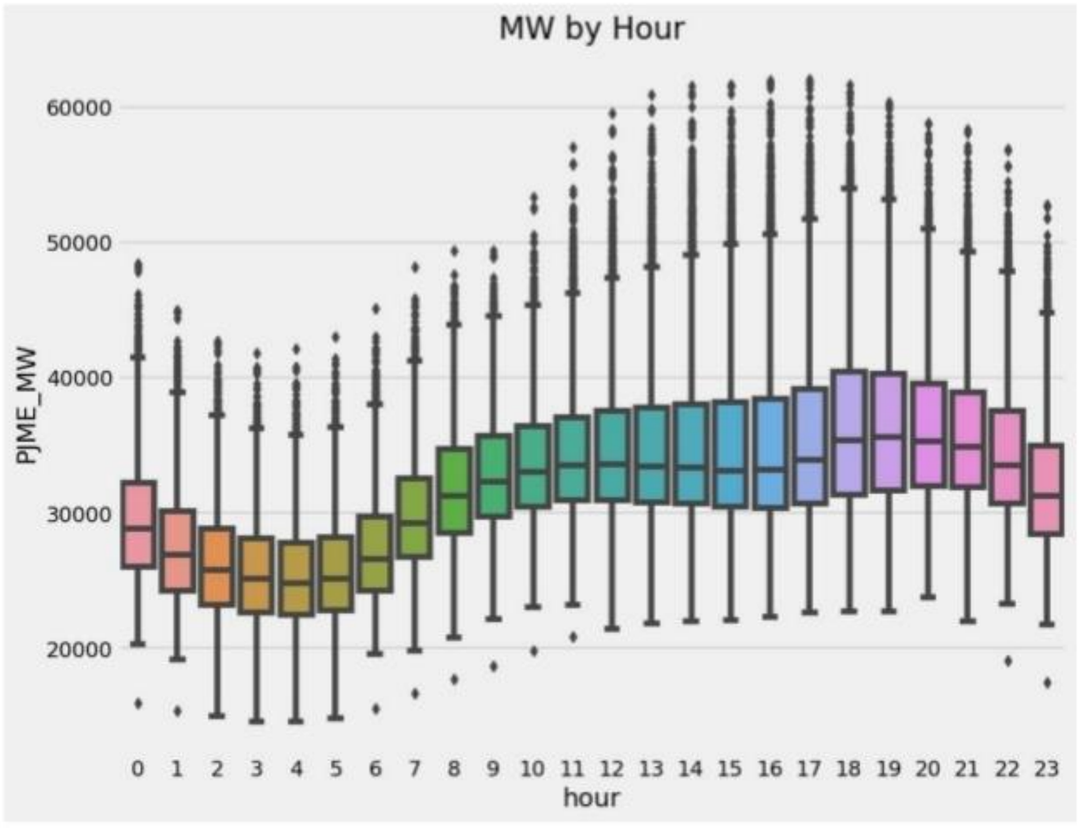
```
train = df.loc[df.index < '01-01-2015']
test = df.loc[df.index >= '01-01-2015']
fig, ax = plt.subplots(figsize=(15, 5))
train.plot(ax=ax, label='Training Set', title='Data
Train/Test Split')
test.plot(ax=ax, label='Test Set')
ax.axvline('01-01-2015', color='black', ls='--')
ax.legend(['Training Set',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



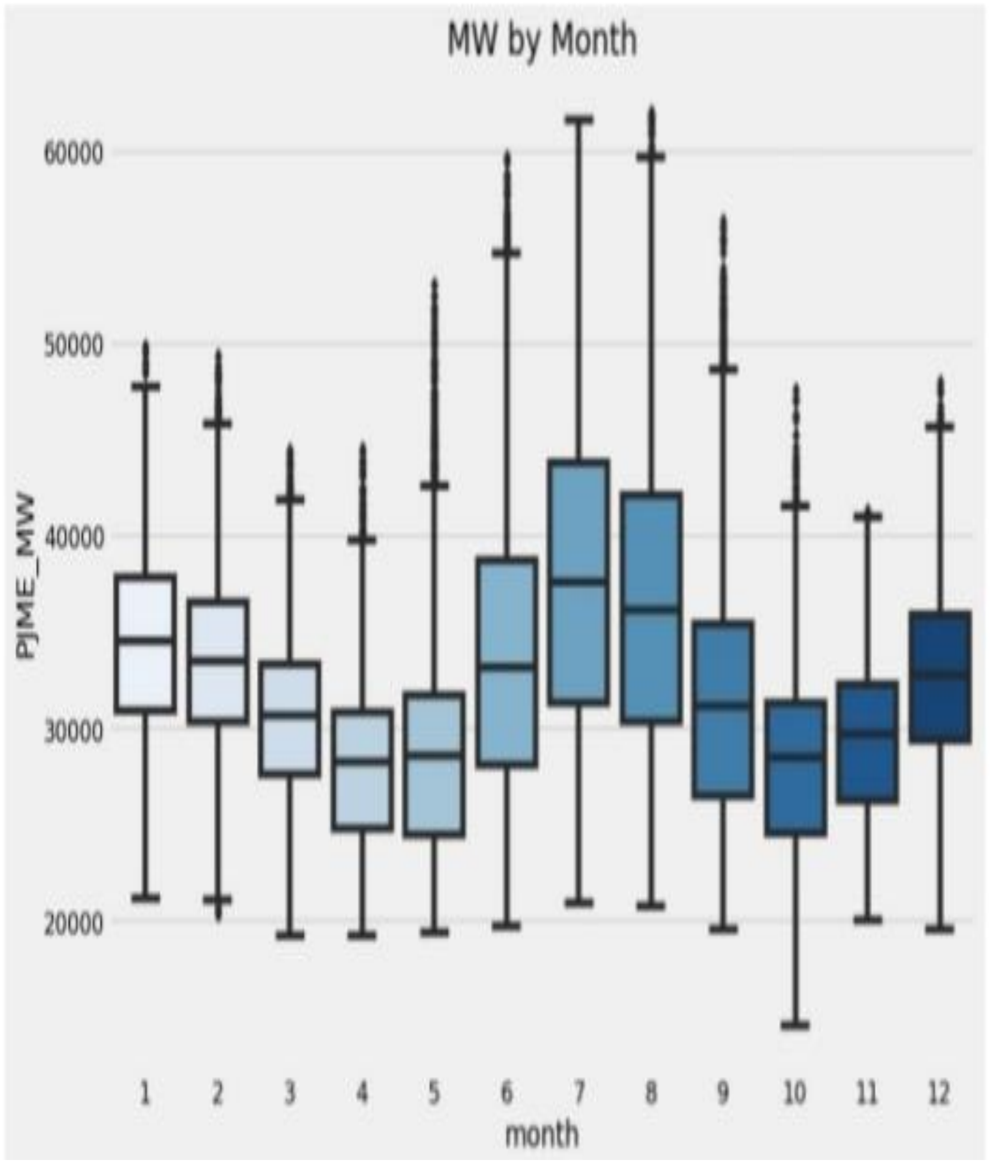
```
df.loc[(df.index > '01-01-2010') & (df.index < '01-08-
2010')] \
.plot(figsize=(15, 5), title='Week Of Data')
plt.show()
```



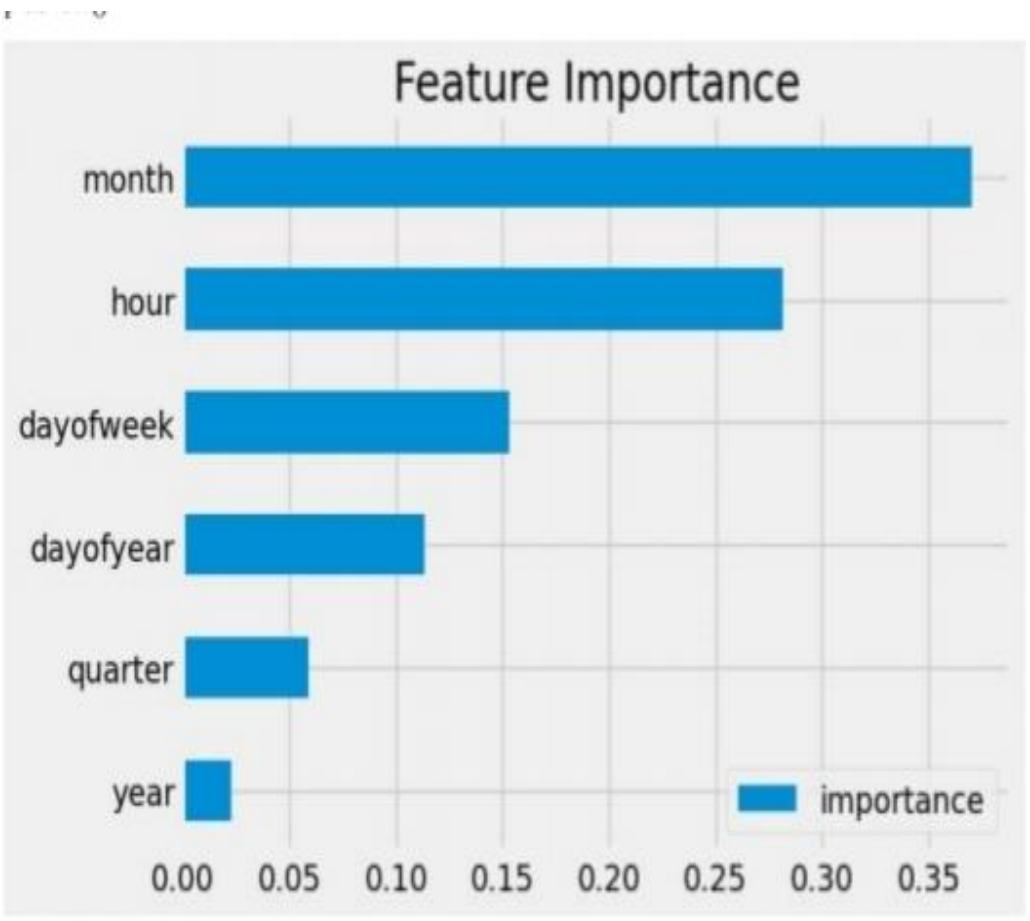
```
def create_features(df):
 """

 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()
```



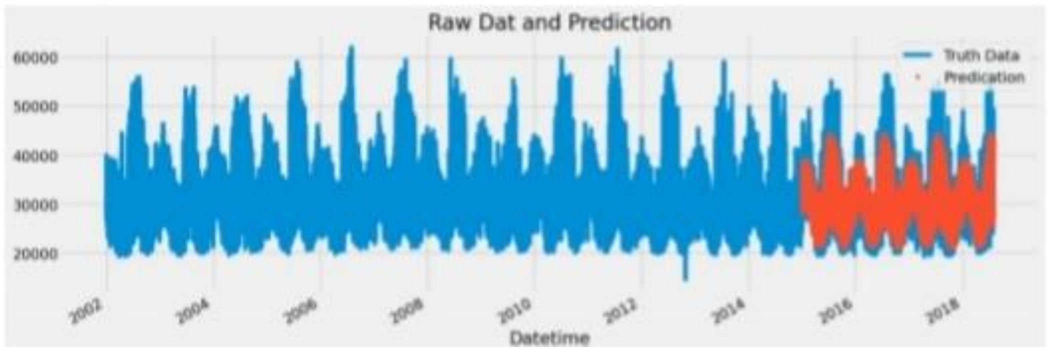
```
 ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='month', y='PJME_MW',
palette='Blues')
ax.set_title('MW by Month')
plt.show()
```



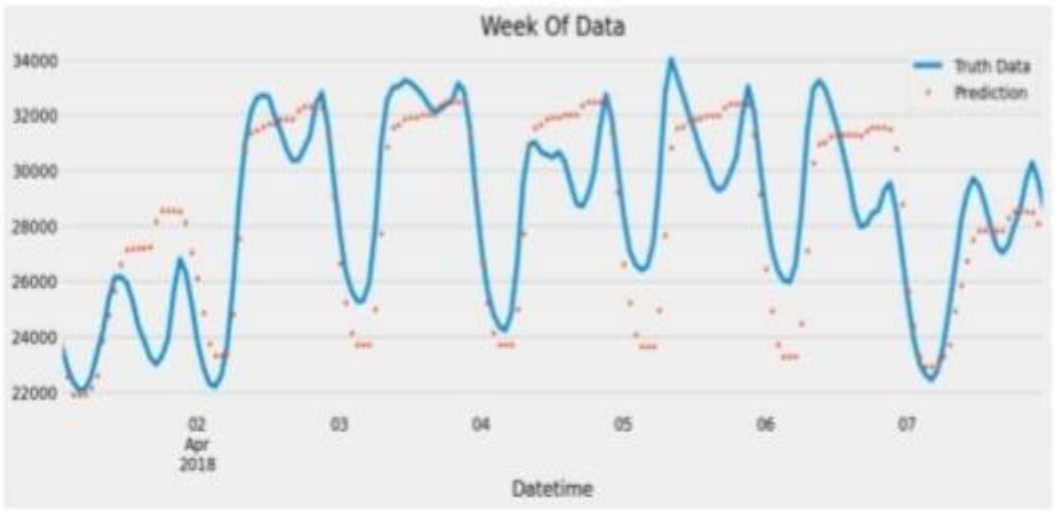
```
train = create_features(train)
test = create_features(test)
FEATURES = ['dayofyear', 'hour', 'dayofweek',
'quarter', 'month', 'year']
TARGET = 'PJME_MW'
x_train = train[FEATURES]
y_train = train[TARGET]
x_test = test[FEATURES]
y_test = test[TARGET]
reg =xgb.XGBRegressor(base_score=0.5,
booster='gbtree', n_estimators=1000,
 early_stopping_rounds=50,
 objective='reg:linear',
 max_depth=3,
 learning_rate=0.01)
reg.fit(x_train, y_train,
 eval_set= [(x_train, y_train), (x_test, y_test)],
 verbose=100)
fi = pd.DataFrame(data=reg.feature_importances_,
 index=reg.feature_names_in_,
columns=['importance'])
fi.sort_values('importance').plot(kind='barh',
title='Feature Importance')
plt.show()
```



```
test['prediction'] = reg.predict(x_test)
df = df.merge(test[['prediction']], how= 'left',
left_index=True, right_index=True)
ax = df[['PJME_MW']].plot(figsize=(15, 5))
df['prediction'].plot(ax=ax, style='.')
plt.legend(['Truth Data', 'Predication'])
ax.set_title('Raw Dat and Prediction')
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: float64
```