MEASURE ENERGY CONSUMPTION

Phase-5:Project Documentation & Submission

Introduction:

Measuring energy consumption in Python is an essential practice for assessing and optimizing the efficiency of devices and systems. Python, as a versatile and powerful programming language, provides a flexible and capable platform for collecting, analyzing, and visualizing energy consumption data. Measuring energy consumption involves determining how much energy is used by a device, system, or an entire facility over a specific period. This is essential for various purposes, including tracking expenses, optimizing energy efficiency, and reducing environmental impact.

Content for Project Phase 5:

In this part you will document your project and prepare it for submission.

1. **Documentation:**
   - ❖ Clearly outline the problem statement, design thinking process, and the phases of development.
   - ❖ Describe the dataset used, data preprocessing steps, and visualization techniques.
   - ❖ Document any innovative techniques or approaches used during the development.

2. **Submission:**
   - ❖ Compile all the code files, including the data preprocessing, and visualization code.
   - ❖ Provide a well-structured README file that explains how to run the code and any dependencies.
   - ❖ Include the dataset source and a brief description.
   - ❖ Share the submission on platforms like GitHub or personal portfolio for others to access and review.

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.

Design thinking:

1. Data Source Identification:

- Identify a suitable dataset that contains energy consumption measurements. Some potential sources include government agencies, utility companies, research institutions, or open data portals. You can search for datasets on websites like data.gov, Kaggle, or the U.S. Energy Information Administration (EIA) website.

2. Data Preprocessing:

- Clean the dataset by addressing missing values, outliers, and data inconsistencies.
- Convert data types as needed (e.g., date/time conversion).
- Handle duplicates and remove irrelevant columns.
- Normalize or scale data if necessary.
- Ensure data quality and consistency.

3. Feature Extraction:

- Identify relevant features and metrics that can provide insights into energy consumption. These might include:
- Time-based features (hourly, daily, monthly, seasonal patterns).
- Weather data (temperature, humidity) if available, as it can impact energy usage.
- Demographic data (population, building types) if relevant.
- Calculate aggregate statistics, such as mean, median, or standard deviation, for different time periods.

4. Model Development:

- Utilize statistical analysis techniques to uncover trends, patterns, and anomalies in the data. This can involve time series analysis, regression, clustering, or anomaly detection methods.
- Develop predictive models if you want to forecast future energy consumption based on historical data.
- Use tools like Python's Pandas, NumPy, scikit-learn, and libraries specific to time series analysis for this step.

5. Visualization:

Create visualizations to present energy consumption trends and insights. This can include:

- Time series plots to visualize changes over time.
- Histograms or bar charts to show distribution of energy consumption.
- Heatmaps to display correlations between features.
- Box plots or violin plots for outlier detection.
- Geographic maps to visualize regional variations.
- Use visualization libraries like Matplotlib, Seaborn, Plotly, or geographic mapping tools like Folium (for maps).

6. Automation:

- Build a script or pipeline to automate the entire process, from data collection to visualization.
- Use scripting languages like Python to create a workflow that:
- Downloads and updates the dataset at regular intervals.
- Performs data preprocessing and feature extraction.
- Runs statistical analyses and generates insights.
- Creates and saves visualizations in a shareable format (e.g., PDFs or interactive web dashboards).
- Schedules the script to run automatically at specified intervals (e.g., daily, weekly).

Phases of Development:

- **Data loading:**

  Load the energy consumption dataset, which can be in various formats such as CSV, Excel, or a database. You can use libraries like pandas, numpy, or sqlite3, depending on the data format.

- **Preprocessing:**

  Preprocessing data in Python is the process of cleaning, transforming, and organizing raw data to make it suitable for analysis, visualization, or machine learning. Proper data preprocessing is essential to ensure the accuracy and reliability of your analysis.

- **Data Analyzing:**

  Analyzing data in Python refers to the process of examining and extracting meaningful insights, patterns, and information from a dataset using Python programming and data analysis libraries. This process can include various tasks such as data exploration, data cleaning, statistical analysis, visualization, and machine learning.

- **Data Visualizing:**

  Data visualization is the process of representing data in graphical or visual formats, such as charts, graphs, maps, and other visual elements. The goal of data visualization is to make complex data more understandable, accessible, and interpretable.

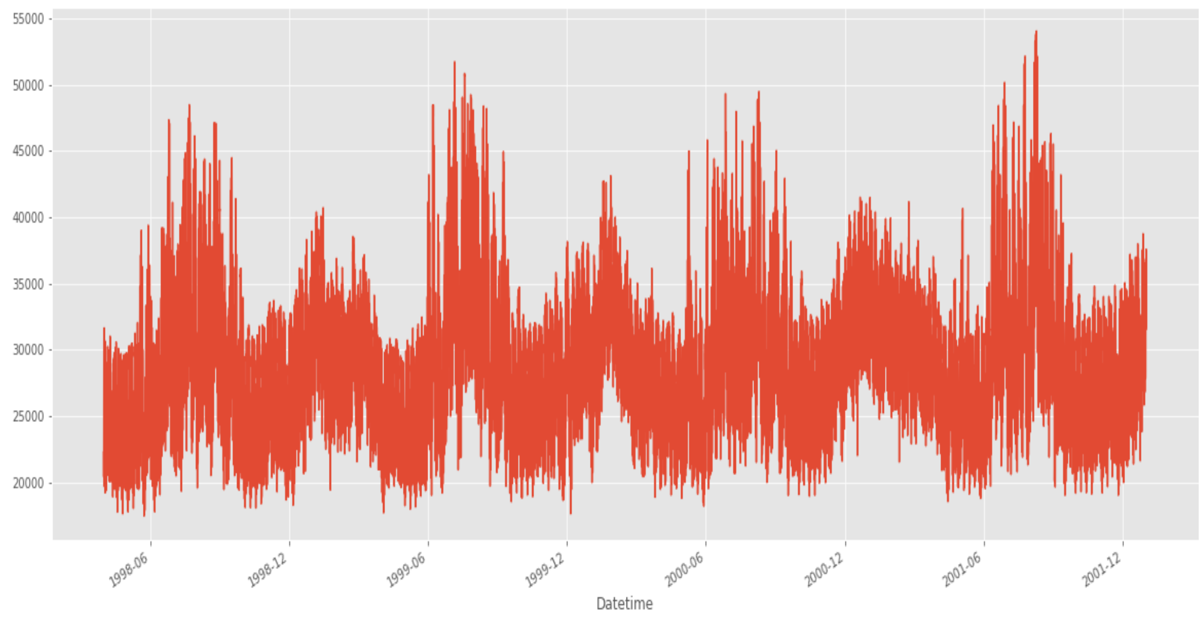
Data Source:

"PJM Hourly Energy Consumption Data"

PJM Interconnection LLC (PJM) is a regional transmission organization (RTO) in the United States. It is part of the Eastern Interconnection grid operating an electric transmission system serving all or parts of Delaware, Illinois, Indiana, Kentucky, Maryland, Michigan, New Jersey, North Carolina, Ohio, Pennsylvania, Tennessee, Virginia, West Virginia, and the District of Columbia.

The hourly power consumption data comes from PJM's website and are in megawatts (MW).

The regions have changed over the years so data may only appear for certain dates per region.

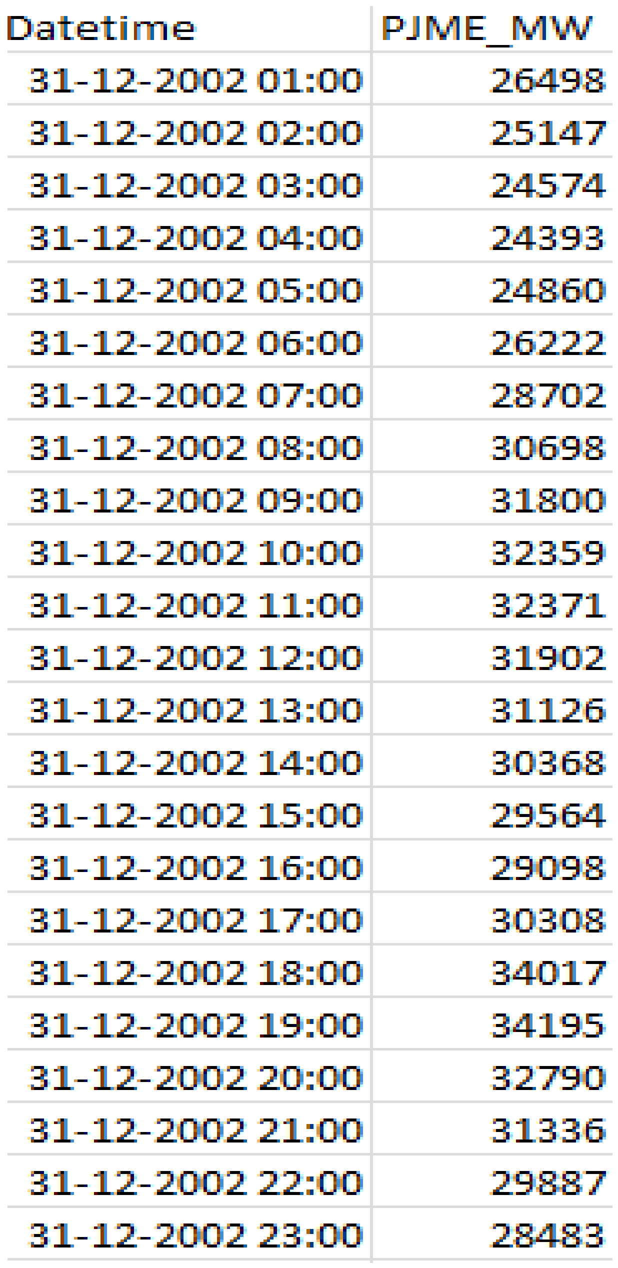


Ideas of what you could do with this dataset:

1. Split the last year into a test set- can you build a model to predict energy consumption?
2. Find trends in energy consumption around hours of the day, holidays, or long term trends?
3. Understand how daily trends change depending of the time of year. Summer trends are very different than winter trends.

Dataset Link:

https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption



Datetime	PJME_MW
31-12-2002 01:00	26498
31-12-2002 02:00	25147
31-12-2002 03:00	24574
31-12-2002 04:00	24393
31-12-2002 05:00	24860
31-12-2002 06:00	26222
31-12-2002 07:00	28702
31-12-2002 08:00	30698
31-12-2002 09:00	31800
31-12-2002 10:00	32359
31-12-2002 11:00	32371
31-12-2002 12:00	31902
31-12-2002 13:00	31126
31-12-2002 14:00	30368
31-12-2002 15:00	29564
31-12-2002 16:00	29098
31-12-2002 17:00	30308
31-12-2002 18:00	34017
31-12-2002 19:00	34195
31-12-2002 20:00	32790
31-12-2002 21:00	31336
31-12-2002 22:00	29887
31-12-2002 23:00	28483

Preprocessing:

- **Handling Missing Values:**

  Check for missing data, as missing values can affect the quality of your analysis. Depending on the extent of missing data, you can either drop rows with missing values or impute the missing values using techniques like mean, median, or interpolation.

- **Data Transformation:**

  Convert date and time columns to datetime objects if your dataset includes timestamps.

  Convert categorical variables into numerical representations if needed, using techniques like one-hot encoding or label encoding.

- **Feature Engineering:**

  Extract relevant features from the data, such as day of the week, hour of the day, or season, which can be important for energy consumption analysis.

  Create lag features, which involve using past values of energy consumption to predict future values.

Data Visualizing:

Data visualization is the process of representing data in graphical or visual formats, such as charts, graphs, maps, and other visual elements. The goal of data visualization is to make complex data more understandable, accessible, and interpretable.

It allows individuals to quickly grasp insights, patterns, and trends within the data, which might be challenging to discern from raw data or textual descriptions alone.

It is a fundamental part of data analysis and reporting, enabling analysts, data scientists, and decision-makers to better understand data, make informed decisions, and communicate their findings effectively.

There are numerous data visualization tools and libraries available, such as Matplotlib, Seaborn, Tableau, and D3.js, which cater to a wide range of visualization needs.

Create visualizations to present energy consumption trends and insights. This can include:

- Time series plots to visualize changes over time.
- Histograms or bar charts to show distribution of energy consumption.
- Heatmaps to display correlations between features.
- Box plots or violin plots for outlier detection.
- Geographic maps to visualize regional variations.
- Use visualization libraries like Matplotlib, Seaborn, Plotly, or geographic mapping tools like Folium (for maps).

Key aspects of data visualization include:

- Graphical Representations.
- Simplification.
- Pattern Recognition.
- Time Series Visualization.
- Exploratory Data Analysis (EDA).
- Big Data Visualization.
- Geospatial Visualization.
- Scientific Visualization.

Innovative Technique:

Time Series Forcasting:

Time series forecasting models are statistical and machine learning techniques used to make predictions based on time-ordered data. These models are widely applied in various fields, including finance, economics, weather forecasting, and demand forecasting. Here are some key aspects of time series forecasting models:

1. ARIMA (AutoRegressive Integrated Moving Average)
2. Exponential Smoothing
3. Prophet
4. XGBoost

5. Long Short-Term Memory (LSTM) Networks
6. GARCH (Generalized Autoregressive Conditional Heteroskedasticity)
7. VAR (Vector AutoRegressive)

Now we are using “XGBoost”

XGBoost, or Extreme Gradient Boosting, is a potent machine learning algorithm employed for time series forecasting, including the measurement of energy consumption in Python. While not specifically tailored for time series data, XGBoost can be effectively harnessed for this purpose with careful data preparation. By structuring the time series data chronologically and engineering relevant features such as lag values and potential influencers like weather patterns or holidays, XGBoost can be trained to predict future energy consumption values. Its ensemble of decision trees, optimized for speed and efficiency, excels at capturing complex relationships and patterns within the data. However, it's crucial to fine-tune hyperparameters and validate the model's performance using metrics like MAE and RMSE to ensure accurate forecasting. XGBoost's versatility, speed, and regularization capabilities make it a valuable tool in predicting energy consumption patterns, particularly when dealing with multifaceted datasets influenced by a combination of temporal and non-temporal factors.

PROGRAM:

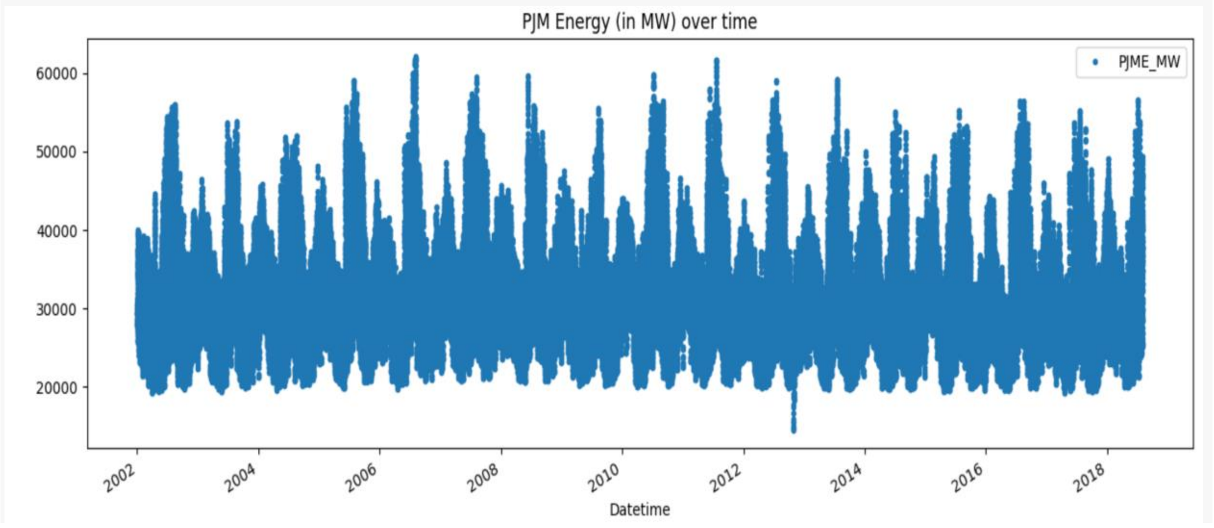
```
# import the libraries

import pandas as pd

 import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


# Customize the style

pd.options.display.float_format = '{:.5f}'.format
pd.options.display.max_rows = 12


# Load the data

df = pd.read_csv('PJME_hourly.csv')

print("Now, you're ready for step one")
```

```
Now, you're ready for step one
```

Explore the data:

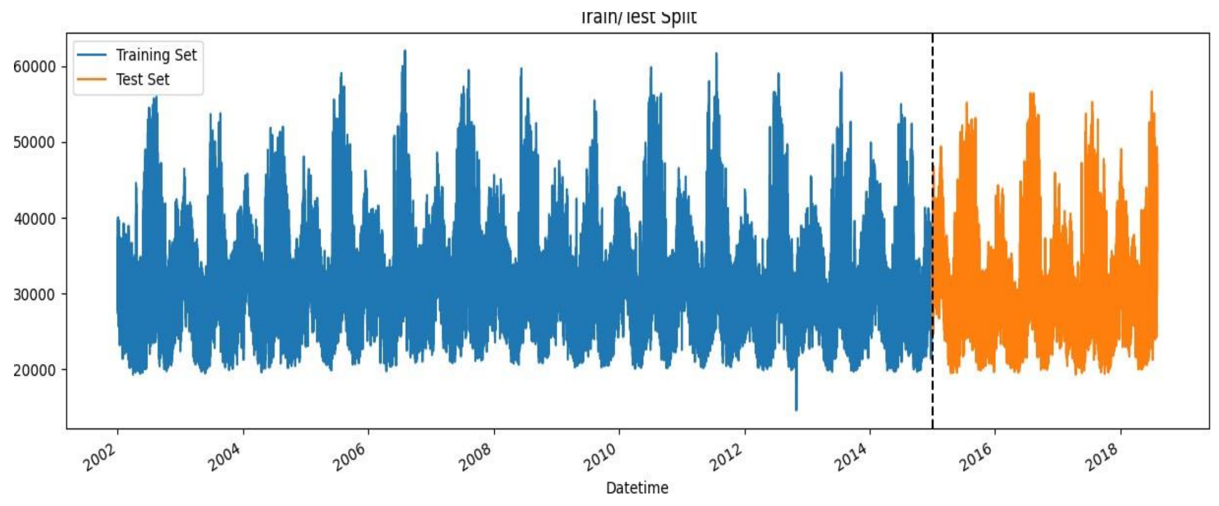
```
# turn data to datetime
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)
```

```
# create the plot
df.plot(style='.',
        figsize=(15, 5),
        title='PJM Energy (in MW) over time')
plt.show()
```



Split the data:

```
# train / test split

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='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()
```



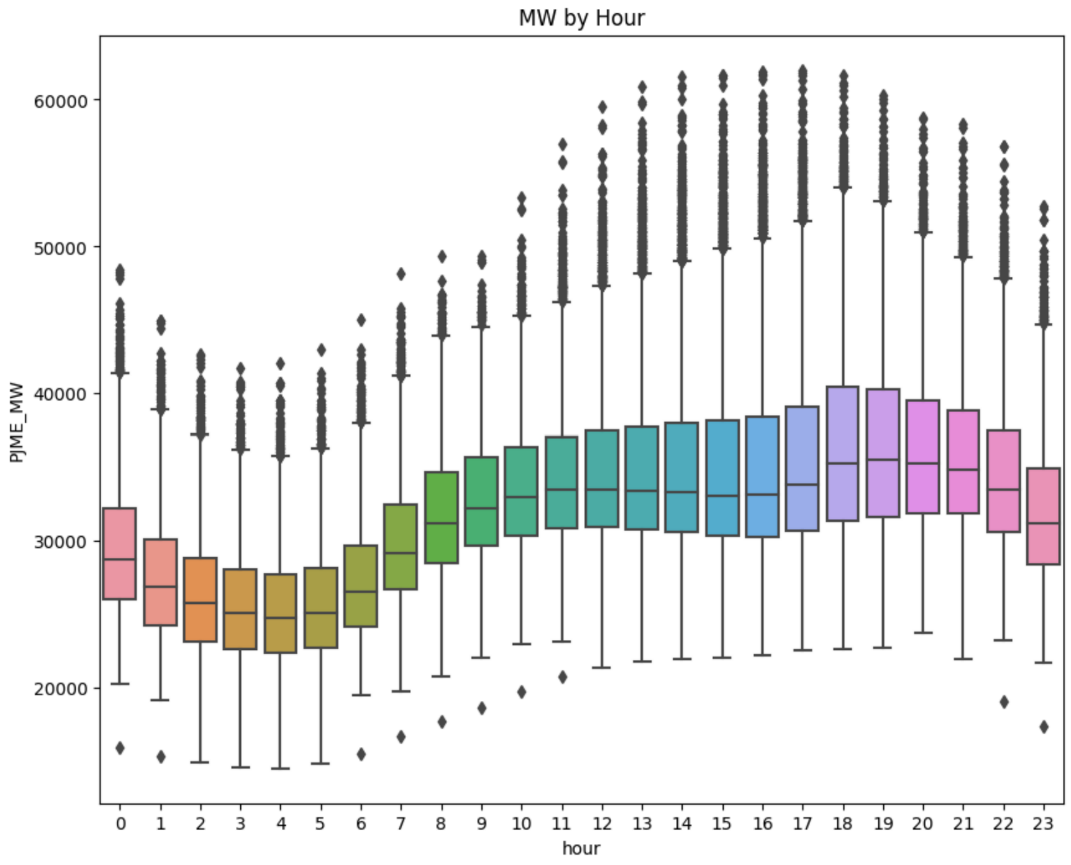
Feature Engineering:

```
# feature creation
def create_features(df):
    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['dayofmonth'] = df.index.day
    df['weekofyear'] = df.index.isocalendar().week
    return df
```

```
df = create_features(df)
```

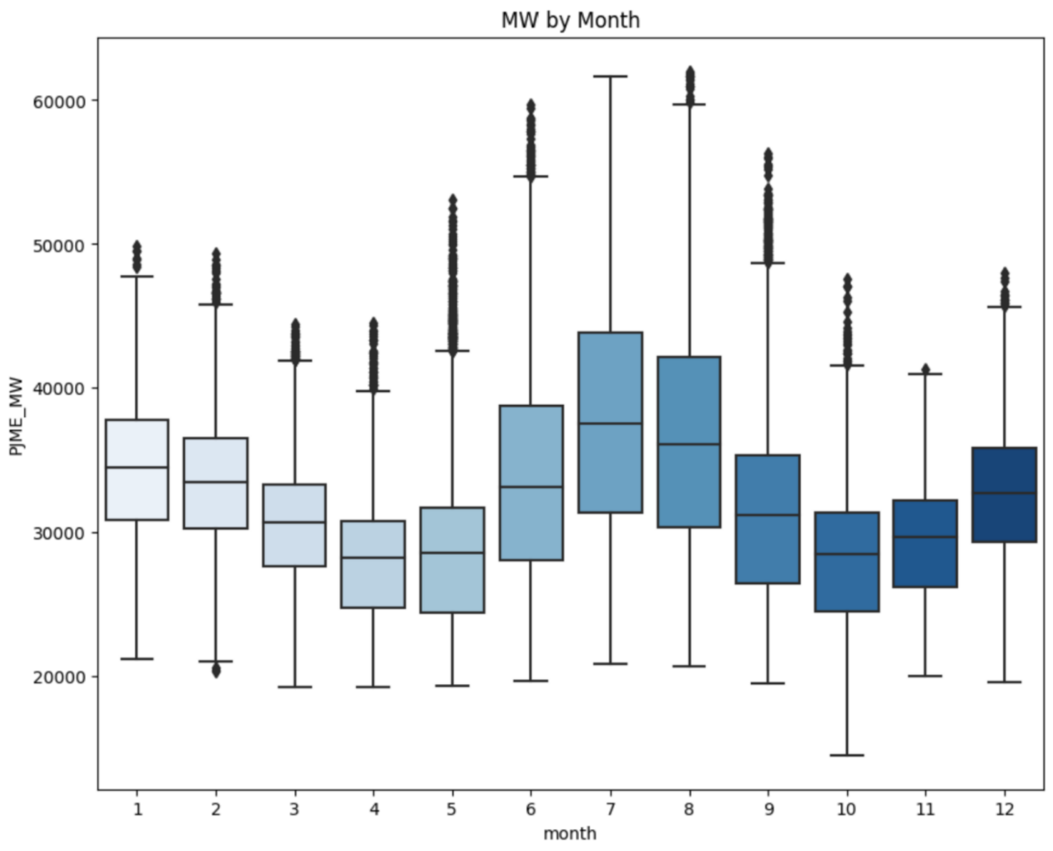
Data Visualizing:

```
# visualize the hourly Megawatt

fig, ax = plt.subplots(figsize=(10, 8))

sns.boxplot(data=df, x='hour', y='PJME_MW')

ax.set_title('MW by Hour')

plt.show()
```



We can see here that after midnight, the use of energy go down and it gets higher from around 6AM to 6PM and then go down again.

```
# viaualize the monthly Megawatt
fig, 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()
```



The monthly usage tends to peak here two times in the winter season, then in the fall and sprint it has lower and another peak in the middle of summer.

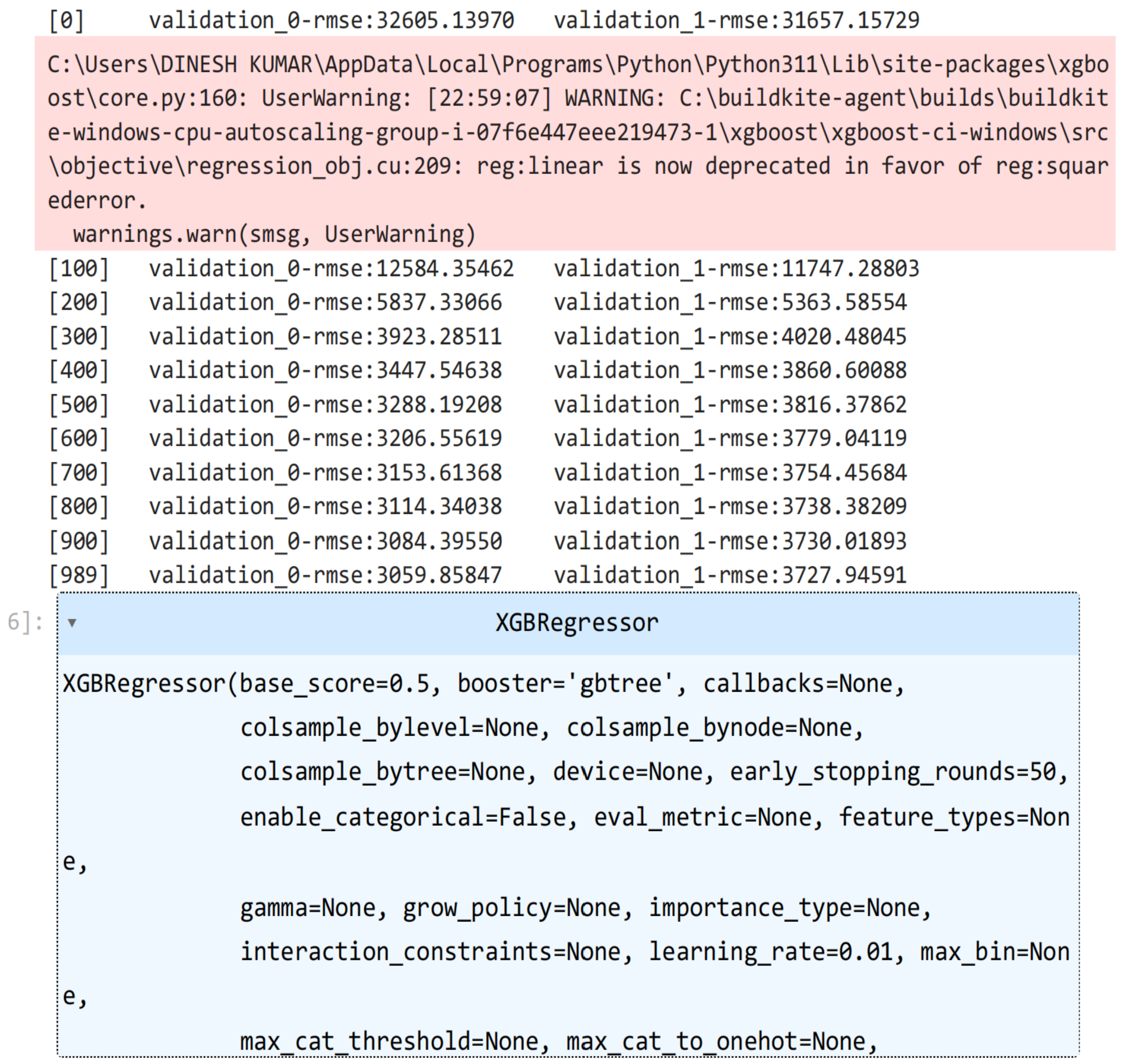
Modelling:

Preprocessing:

```
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]
```

Build the model:

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error

# build the regression model
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)
```

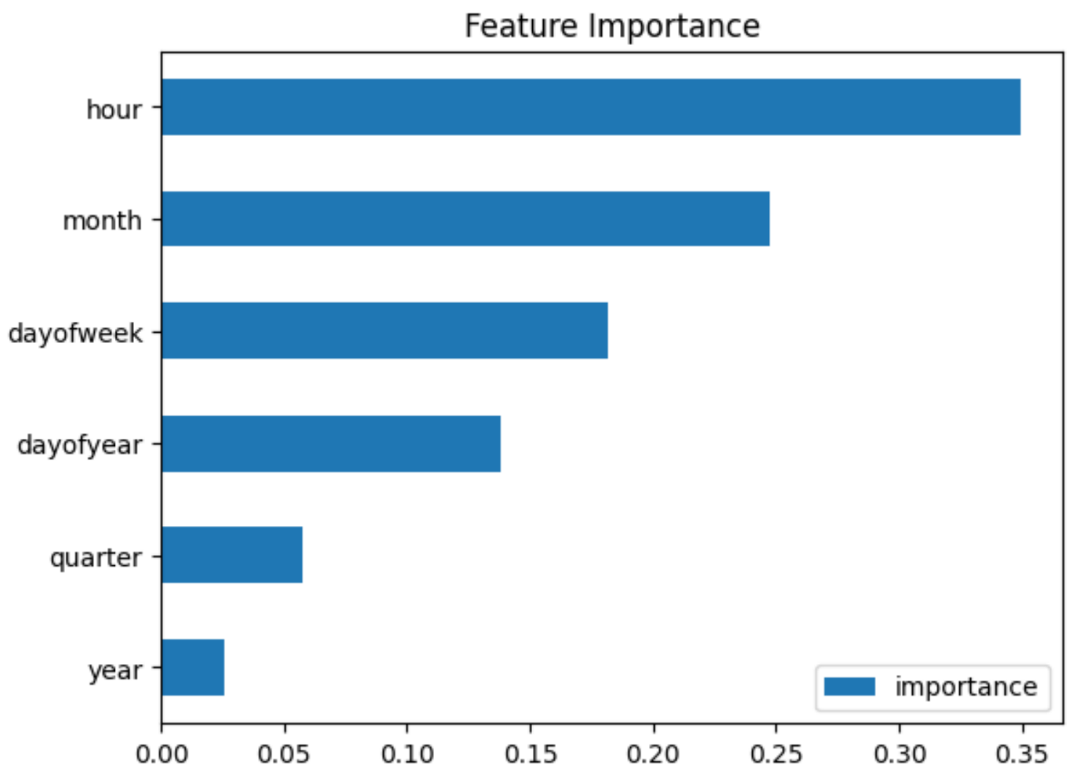


```
[0]     validation_0-rmse:32605.13970   validation_1-rmse:31657.15729
C:\Users\DINESH KUMAR\AppData\Local\Programs\Python\Python311\Lib\site-packages\xgbo
ost\core.py:160: UserWarning: [22:59:07] WARNING: C:\buildkite-agent\builds\buildkit
e-windows-cpu-autoscaling-group-i-07f6e447eee219473-1\xgboost\xgboost-ci-windows\src
\objective\regression_obj.cu:209: reg:linear is now deprecated in favor of reg:squar
ederror.
  warnings.warn(smsg, UserWarning)
[100]   validation_0-rmse:12584.35462   validation_1-rmse:11747.28803
[200]   validation_0-rmse:5837.33066    validation_1-rmse:5363.58554
[300]   validation_0-rmse:3923.28511    validation_1-rmse:4020.48045
[400]   validation_0-rmse:3447.54638    validation_1-rmse:3860.60088
[500]   validation_0-rmse:3288.19208    validation_1-rmse:3816.37862
[600]   validation_0-rmse:3206.55619    validation_1-rmse:3779.04119
[700]   validation_0-rmse:3153.61368    validation_1-rmse:3754.45684
[800]   validation_0-rmse:3114.34038    validation_1-rmse:3738.38209
[900]   validation_0-rmse:3084.39550    validation_1-rmse:3730.01893
[989]   validation_0-rmse:3059.85847    validation_1-rmse:3727.94591
```

```
6]:  ▾                          XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=50,
             enable_categorical=False, eval_metric=None, feature_types=Non
e,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=Non
e,
             max_cat_threshold=None, max_cat_to_onehot=None,
```

Features importance:

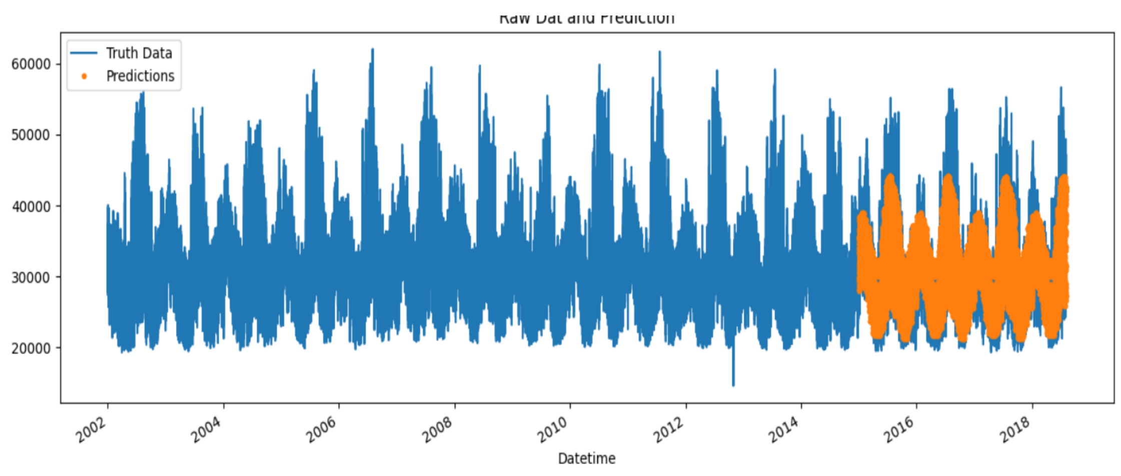
```
fi = pd.DataFrame(data=reg.feature_importances_,
             index=reg.feature_names_in_,
             columns=['importance'])
fi.sort_values('importance').plot(kind='barh', title='Feature Importance')
plt.show()
```



Forecasting on test data:

```
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', 'Predictions'])

ax.set_title('Raw Dat and Prediction')

plt.show()
```



```
# RMSE Score

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
```

```
# R2 Score

from sklearn.metrics import r2_score

r2 = r2_score(test['PJME_MW'], test['prediction'])

print("R-squared (R2) Score:", r2)
```

```
R-squared (R2) Score: 0.6670230260104328
```