MEASURE OF ENRGY CONSUMTION

PHASE 3: LOADING AND PREPROCESSING THE DATASET

INTRODUCTION:

Electric energy consumption is energy consumption in the form of electrical energy. About a fifth of global energy is consumed as electricity: for residential, industrial, commercial, transportation and other purposes. Quickly increasing this share by further electrification is extremely important to limit climate change because most other energy is consumed by burning fossil fuels thus emitting greenhouse gases which trap heat. The global electricity consumption in 2022 was 24,398 terawatt-hour (TWh), almost exactly three times the amount of consumption in 1981 (8,132 TWh). China, the United States, India and Japan accounted for more than half of the global share of electricity consumption.

DATASET:

The dataset can be taken from the following below link.

Dataset link:

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

DATA COLLECTION AND PREPROCESSING:

Data collection is one major prerequisite for energy efficient production facilities, enabling further analyses by assessing KPIs, condition or status monitoring applications and direct feedback or evaluation of efficiency measures. Data-driven models are widely used for energy consumption prediction Our goal in this project is to use the hourly power consumption data to predict the PJME in the future. This is done by developing a model that can be used to predict future values based on the past values of the time series.

DATA SOURCE:

Energy consumption (primary energy, energy mix and energy intensity): this data is sourced from a combination of two sources—the BP Statistical Review of World Energy and SHIFT Data Portal.

PROGRAM:

Step 0: Set it UP

In [1]:

```
# 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
filepath = '../input/hourly-energy-consumption/PJME_hourly.csv'
df = pd.read_csv(filepath)

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

Out [1]:

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

Step 1: Explore the data

To better understand the data, I need to create a graph to see the change in PJM Energy over time.

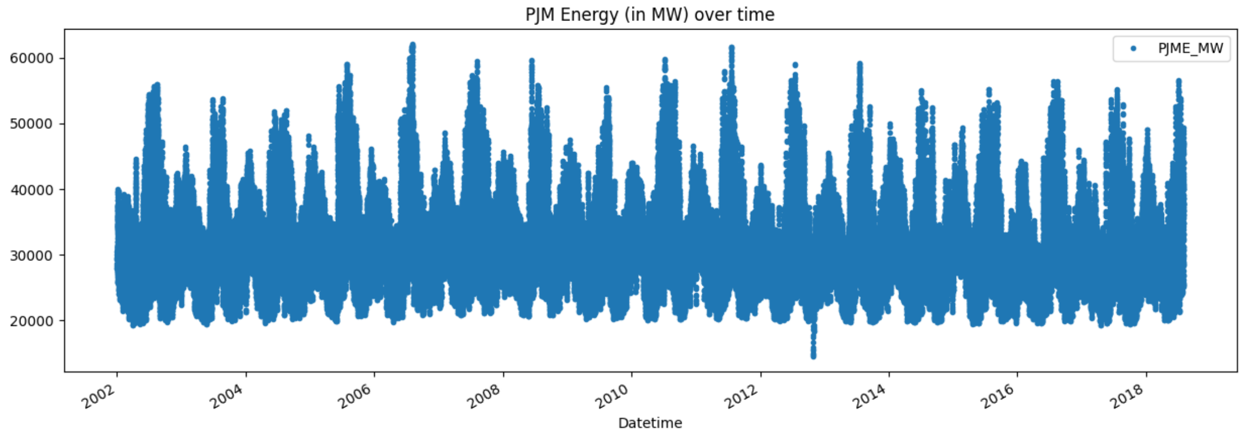
In [2]:

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

In [3]:

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

Out[3]:



Step 2: Split the data

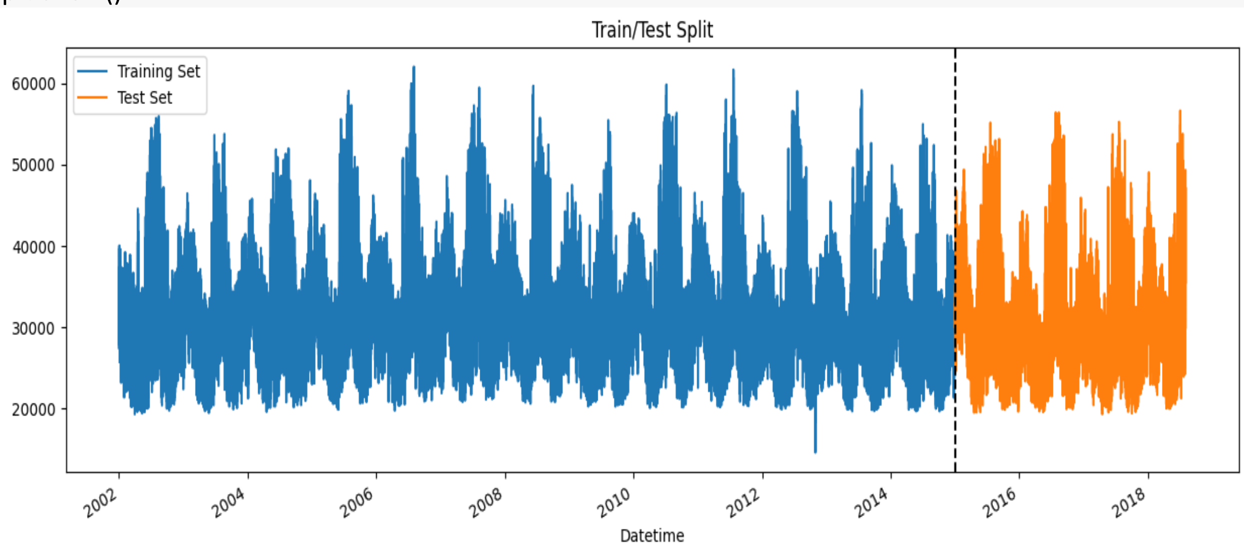
Everything prior to January 2015 will be our training data and keep our test data as the following dates.

In [4]:

```
# train / test split
train = df.loc[df.index < '01-01-2015']
test = df.loc[df.index >= '01-01-2015']
```

In [5]:

```
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()
```



Step 3: Feature Engineering

We're going to create some time features using the Datetime index. After that, we'll explore the distributions of Hourly and Monthly megawatt usage.

In [6]:

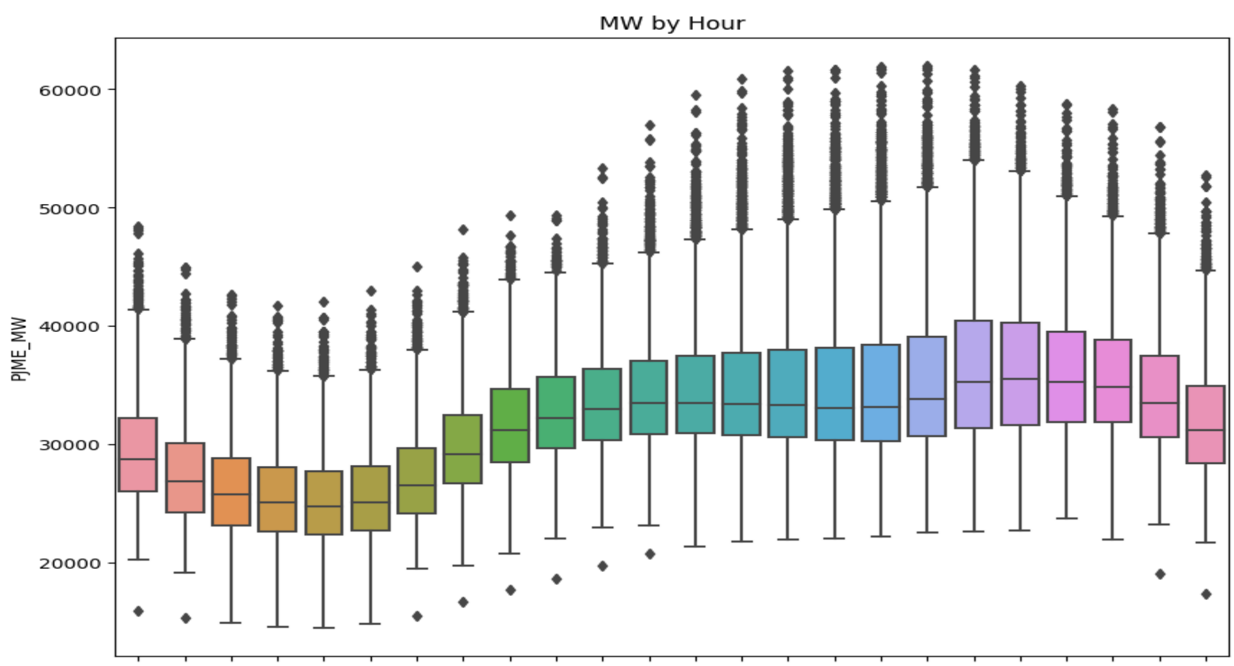
```
# feature creation
def create_features(df):
    df = df.copy()
    df['hour'] = df.index.hour
    df['dayofweek'] = df.index.dayofweek
    df['quarter'] = df.index.quarter
    df['month'] = df.index.month
    df['year'] = df.index.year
    df['dayofyear'] = df.index.dayofyear
    df['dayofmonth'] = df.index.day
    df['weekofyear'] = df.index.isocalendar().week
    return df

df = create_features(df)
```

In [7]:

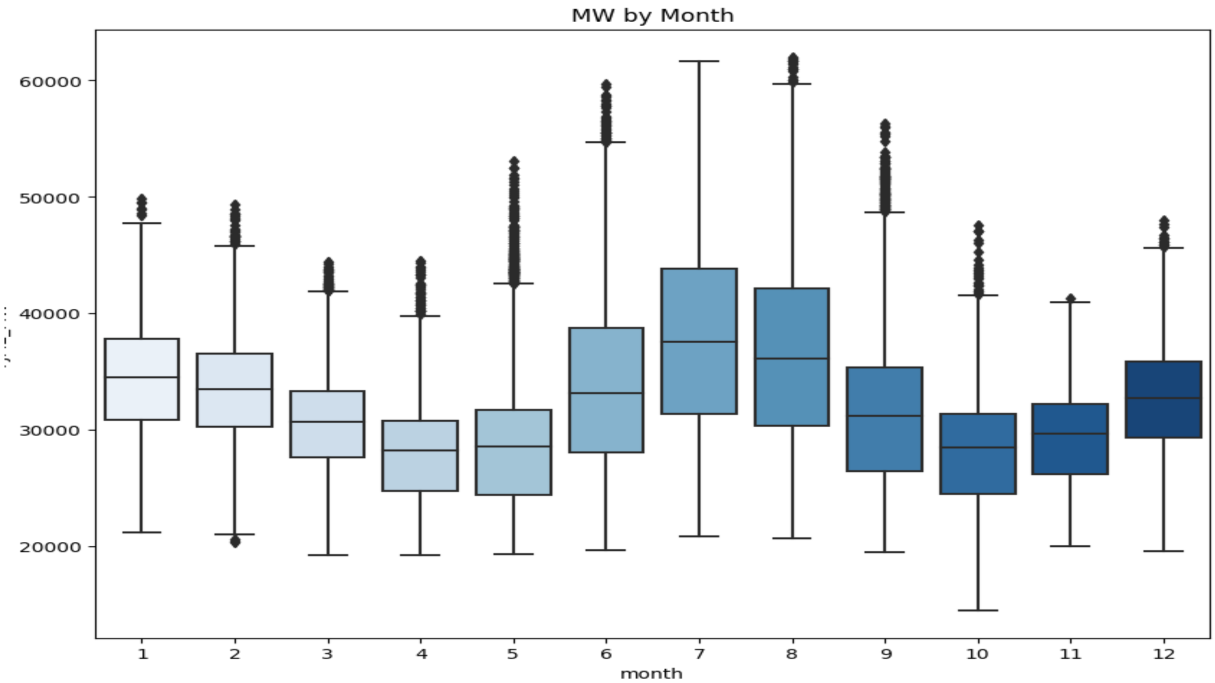
```
# visualize the hourly Megawatt
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='hour', y='PJME_MW')
ax.set_title('MW by Hour')
plt.show()
```

out [7]:



In [8]:
```
# 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()
```

out [8]:



Step 4: Modelling

XGBoost is good and reliable model for regression and time series analysis as well. Also, for the metrics, we'll use mean squared error.

Prepare the data

In [9]:
```
# 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]
```

out [9]:

```
[19:42:36] WARNING: ../src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of re
g:squarederror.
[0]	validation_0-rmse:32605.13860	validation_1-rmse:31657.15907
[100]	validation_0-rmse:12581.21569	validation_1-rmse:11743.75114
[200]	validation_0-rmse:5835.12466	validation_1-rmse:5365.67709
[300]	validation_0-rmse:3915.75557	validation_1-rmse:4020.67023
[400]	validation_0-rmse:3443.16468	validation_1-rmse:3853.40423
[500]	validation_0-rmse:3285.33804	validation_1-rmse:3805.30176
[600]	validation_0-rmse:3201.92936	validation_1-rmse:3772.44933
[700]	validation_0-rmse:3148.14225	validation_1-rmse:3750.91108
[800]	validation_0-rmse:3109.24248	validation_1-rmse:3733.89713
[900]	validation_0-rmse:3079.40079	validation_1-rmse:3725.61224
[999]	validation_0-rmse:3052.73503	validation_1-rmse:3722.92257
```

Build the model

In [10]:

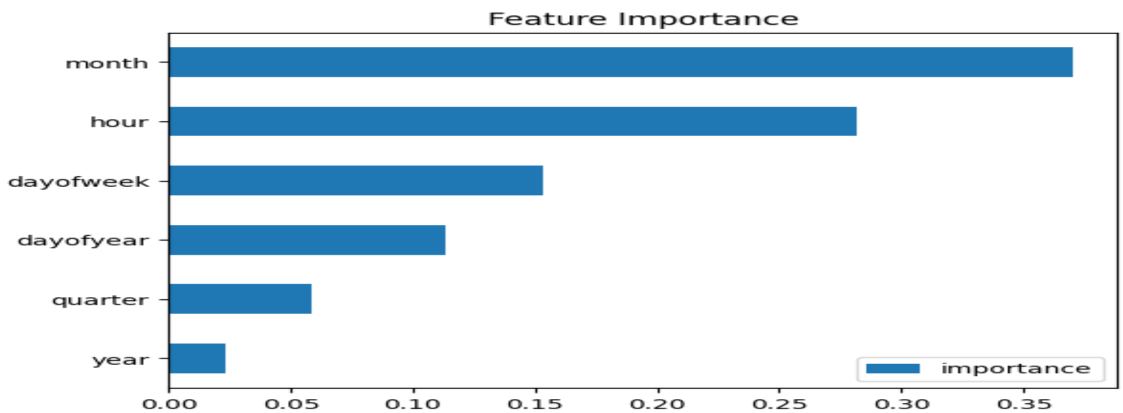
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error

# 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)
```

In [11]:

```
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()
```

out [11]:



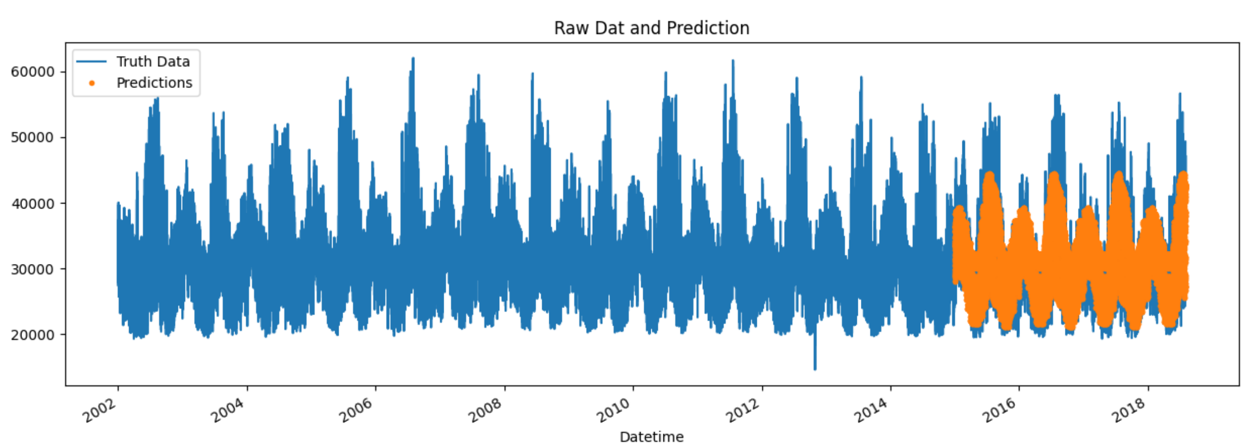
Step 5: Forecasting on test data

compare the prediction with the actual values.

In [12]:

```
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()
```

out [12]:



In [13]:

```
# 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

In [14]:

```
# R2 Score
from sklearn.metrics import r2_score

r2 = r2_score(test['PJME_MW'], test['prediction'])
print("R-squared (R2) Score:", r2)
```

out [14]:

R-squared (R2) Score: 0.6670230260104328