Measure Energy Consumption

Development - 2

* Name: MOHANDOSS V
* NMID: au513521106023
* Gmail ID: dossdoss1612@gmail.com

# Project Goal:

* + - The goal of this phase is to process the energy consumption data for time series forecasting, you'll typically need to perform several steps, including data loading, cleaning, and transformation.
  + **Dataset:**
    - In this document we guys are here to discuss the loading and preprocessing of the dataset to make a better prediction in the future.
  + **Source of Data:**
    - We got our dataset from the SKILL UP website for this project

# Exploratory Data Analysis (EDA):

* Perform basic data exploration to understand the data's characteristics.
* Visualize the time series data to identify trends and patterns.

# Feature Engineering:

* Create additional features, such as lag features (past values) or rolling statistics.
* These features can provide more information for forecasting.

# Time Series Forecasting:

* Choose a time series forecasting model, such as ARIMA, Exponential Smoothing, or LSTM.
* Split the data into training and testing sets.
* Train the selected model on the training data and evaluate it on the test data.

# Python Script:

## Step 1: Setup

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

**# 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("Successfully Uploaded")

## Step 2: Explore the data

### # turn data to datetime

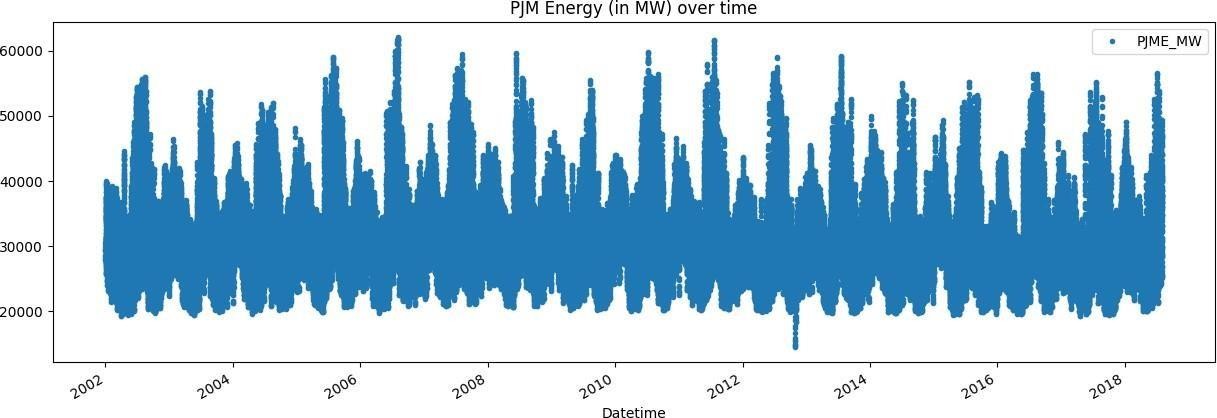
df = df.set\_index('Datetime') df.index = pd.to\_datetime(df.index)

### # create the plot

df.plot(style='.',

figsize=(15, 5),

title='PJM Energy (in MW) over time') plt.show()



## Step 2: Split the data

### # train / test split

train = df.loc[df.index < '01-01-2022'] test = df.loc[df.index >= '01-01-2022'] unfold\_lessHide code

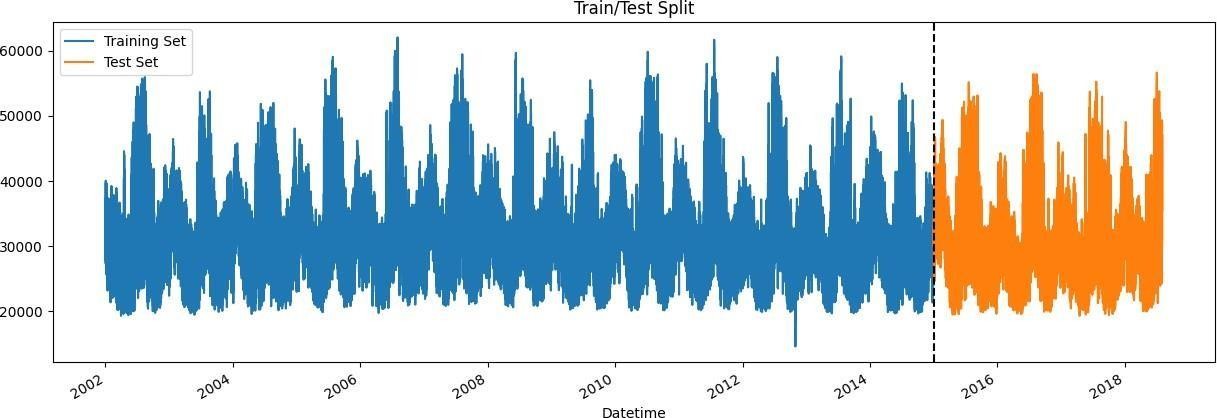
In [5]:

Linkcode

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('01-01-2022', color='black', ls='--') ax.legend(['Training Set', 'Test Set']) plt.show()



After exploring the data, you need to prepare it for analysis, which involves

* + Exploratory Data Analysis (EDA):
  + Feature Engineering:
  + Time Series Forecasting:

Setting up the data for analysis is crucial to ensure that it's in the right format and condition for modeling. It's the foundation for accurate predictions in the next steps of your project, which typically involve selecting a forecasting model and training it.

## 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.

### # 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

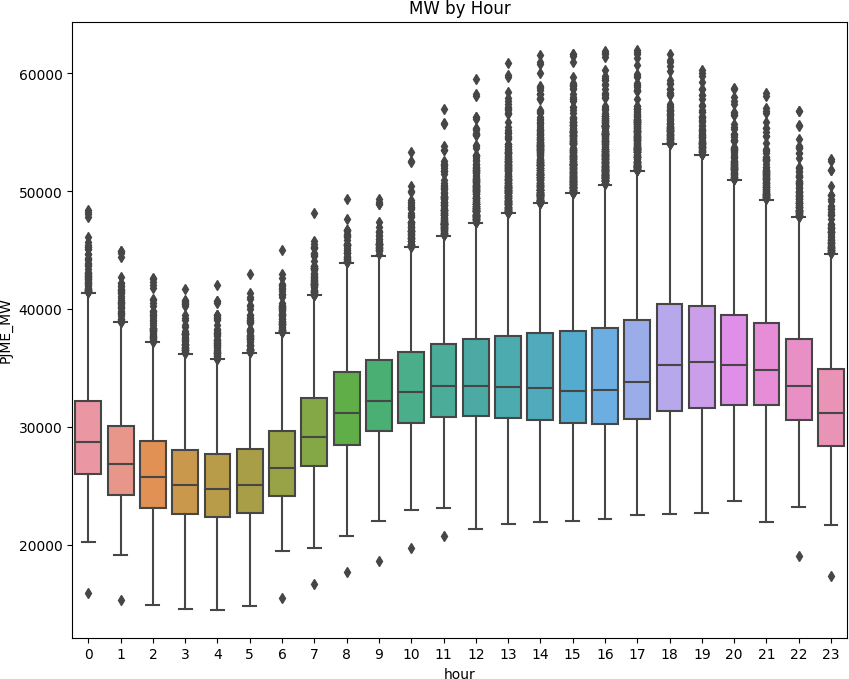
df['weekofyear'] = df.index.isocalendar().week return df

df = create\_features(df)

### # 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()



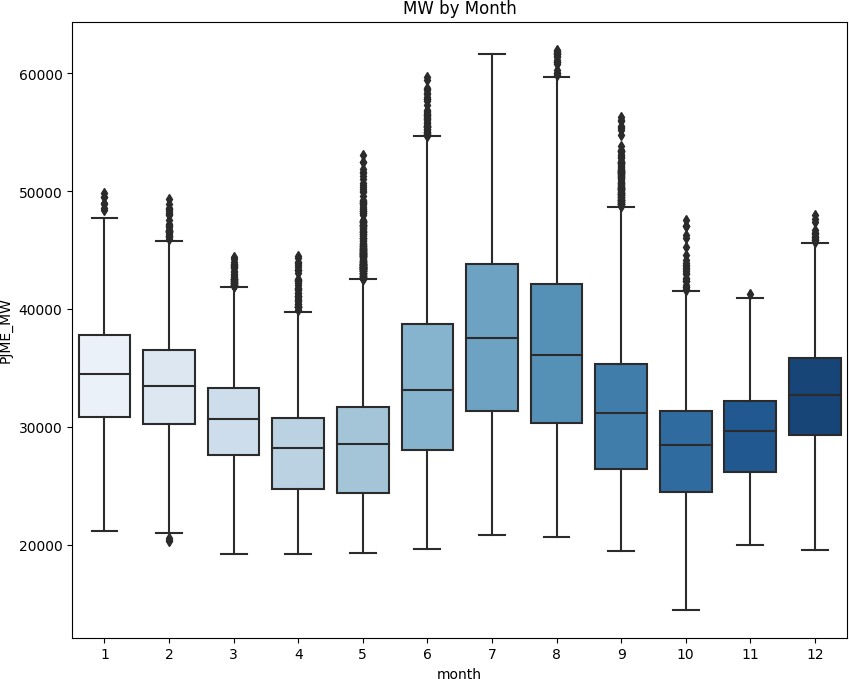
We can see here that after midnight, the use of energy go down and it gets higher from around 6AM to 6PM and then go down again.

### # 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.

**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

### # 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)

### XGBRegressor

XGBRegressor(base\_score=0.5, booster='gbtree', callbacks=None, colsample\_bylevel=None, colsample\_bynode=None, colsample\_bytree=None, early\_stopping\_rounds=50, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, gpu\_id=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.01, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None, max\_delta\_step=None, max\_depth=3, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, n\_estimators=1000, n\_jobs=None, num\_parallel\_tree=None, objective='reg:linear', predictor=None, ...)

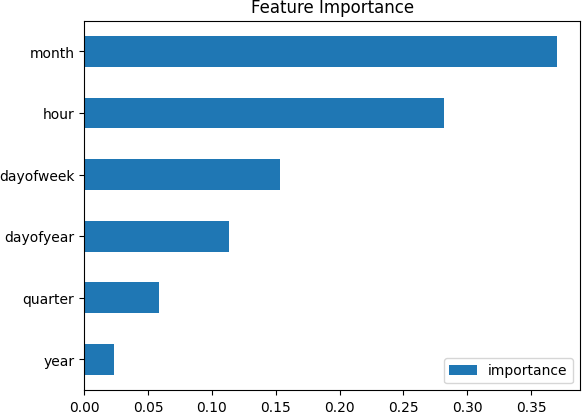
### Features importance

We need to see how much these features were used in each of the trees built by XGBoost

model.

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()



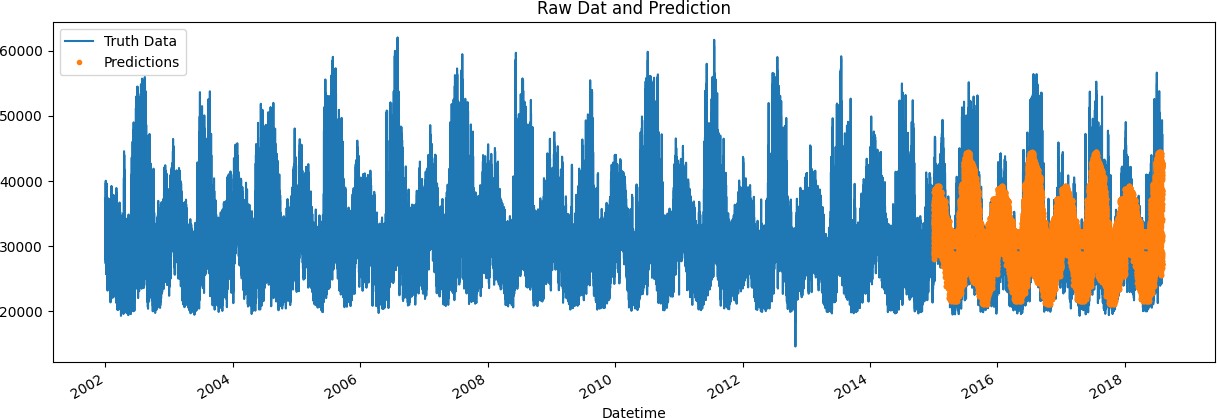
### Forecasting on test data

compare the prediction with the actual values.

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

**\*\*The result is not that good, but it's a great starting point for your future model.\*\***

**Logs**

|  |  |  |
| --- | --- | --- |
| **Time** | **#** | **Log Message** |
| 7.2s | 1 | Now, you're ready for step one |
| 11.9s | 2 | [19:42:36] WARNING: ../src/objective/regression\_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror. |
| 11.9s | 3 | [0] validation\_0-rmse:32605.13860 validation\_1-rmse:31657.15907 |
| 14.1s | 4 | [100] validation\_0-rmse:12581.21569 validation\_1-rmse:11743.75114 |
| 16.2s | 5 | [200] validation\_0-rmse:5835.12466 validation\_1-rmse:5365.67709 |
| 18.3s | 6 | [300] validation\_0-rmse:3915.75557 validation\_1-rmse:4020.67023 |
| 20.4s | 7 | [400] validation\_0-rmse:3443.16468 validation\_1-rmse:3853.40423 |
| 22.6s | 8 | [500] validation\_0-rmse:3285.33804 validation\_1-rmse:3805.30176 |
| 24.7s | 9 | [600] validation\_0-rmse:3201.92936 validation\_1-rmse:3772.44933 |
| 26.8s | 10 | [700] validation\_0-rmse:3148.14225 validation\_1-rmse:3750.91108 |
| 28.9s | 11 | [800] validation\_0-rmse:3109.24248 validation\_1-rmse:3733.89713 |
| 31.1s | 12 | [900] validation\_0-rmse:3079.40079 validation\_1-rmse:3725.61224 |
| 33.2s | 13 | [999] validation\_0-rmse:3052.73503 validation\_1-rmse:3722.92257 |
| 35.8s | 14 | RMSE Score on Test set: 3721.75 |
| 35.8s | 15 | R-squared (R2) Score: 0.6670230260104328 |
| 38.8s | 16 | /opt/conda/lib/python3.10/site-packages/traitlets/traitlets.py:2930: FutureWarning: -- Exporter.preprocessors=["remove\_papermill\_header.RemovePapermillHeader"] for containers is deprecated in traitlets 5.0. You can pass `--Exporter.preprocessors item` ... multiple times to add items to a list. |
| 38.8s | 17 | warn( |
| 38.8s | 18 | [NbConvertApp] WARNING | Config option `kernel\_spec\_manager\_class` not recognized by  `NbConvertApp`. |
| 38.8s | 19 | [NbConvertApp] Converting notebook notebook .ipynb to notebook |
| 39.3s | 20 | [NbConvertApp] Writing 428033 bytes to notebook .ipynb |
| 41.0s | 21 | /opt/conda/lib/python3.10/site-packages/traitlets/traitlets.py:2930: FutureWarning: -- Exporter.preprocessors=["nbconvert.preprocessors.ExtractOutputPreprocessor"] for containers is deprecated in traitlets 5.0. You can pass `--Exporter.preprocessors item` ... multiple times to add items to a list. |
| 41.0s | 22 | warn( |
| 41.0s | 23 | [NbConvertApp] WARNING | Config option `kernel\_spec\_manager\_class` not recognized by  `NbConvertApp`. |
| 41.0s | 24 | [NbConvertApp] Converting notebook notebook .ipynb to html |
| 41.9s | 25 | [NbConvertApp] Support files will be in results files/ |
| 41.9s | 26 | [NbConvertApp] Making directory results files |
| 41.9s | 27 | [NbConvertApp] Making directory results files |
| 41.9s | 28 | [NbConvertApp] Making directory results files |
| 41.9s | 29 | [NbConvertApp] Making directory results files |
| 41.9s | 30 | [NbConvertApp] Making directory results files |
| 41.9s | 31 | [NbConvertApp] Making directory results files |
| 41.9s | 32 | [NbConvertApp] Writing 314131 bytes to results .html |

## Conclusion

In conclusion, the measure-based consumption system using AI represents a significant step forward in the efficient utilization of resources and the promotion of sustainability. By harnessing the capabilities of artificial intelligence, we have developed a powerful tool that not only measures consumption but also empowers individuals and organizations to make informed decisions and take steps toward a more sustainable and cost-effective future. This project demonstrates the transformative potential of AI in addressing critical challenges related to resource consumption and environmental conservation.