Measure Energy Consumption

Development - 2

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.

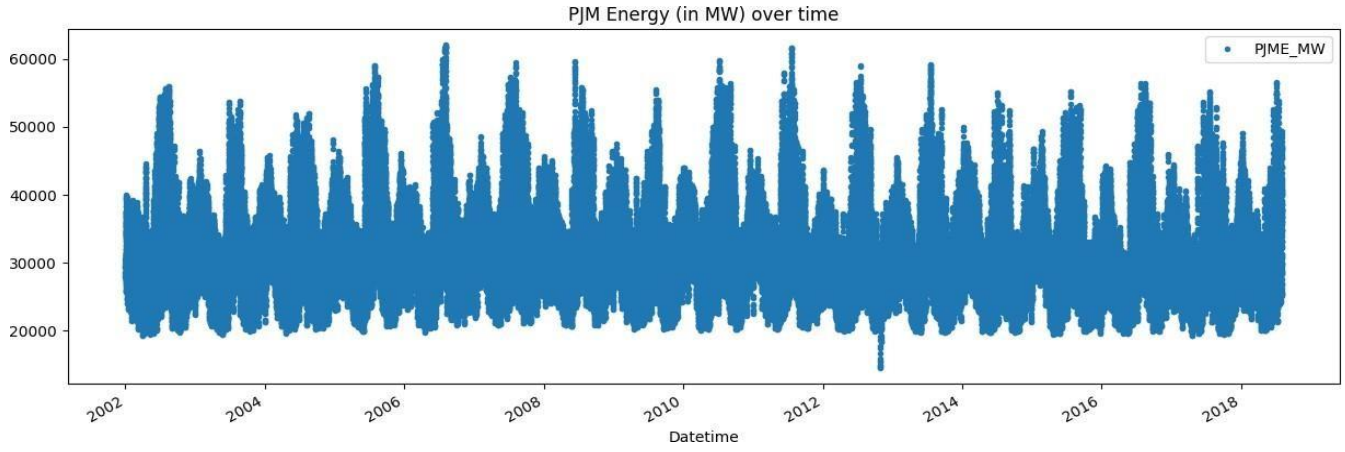
Python Script:

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

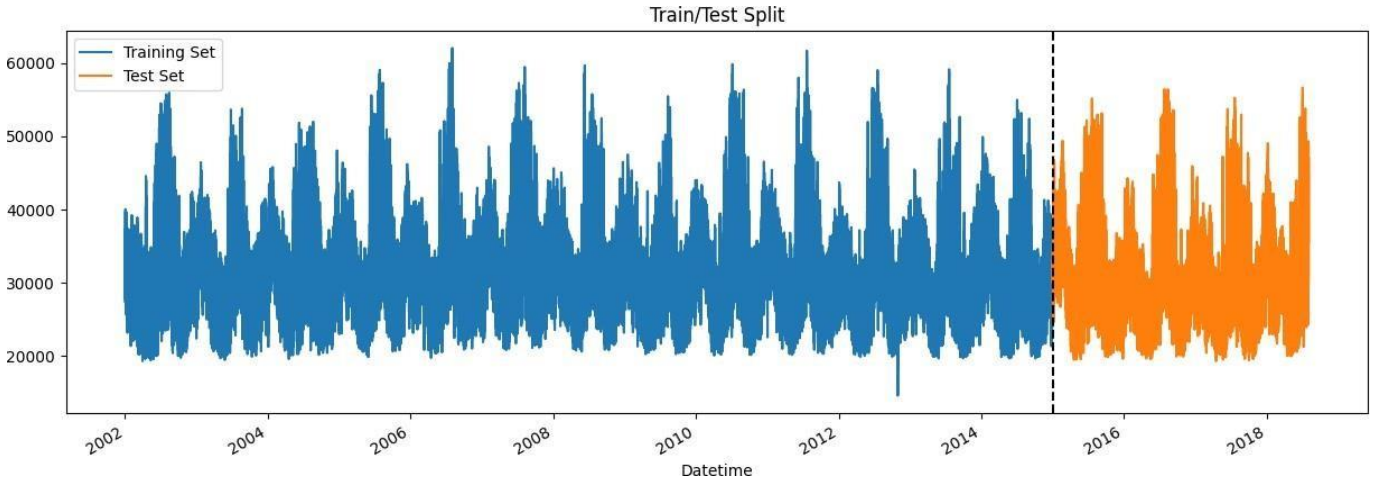
```
# turn data to datetime
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)

# 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

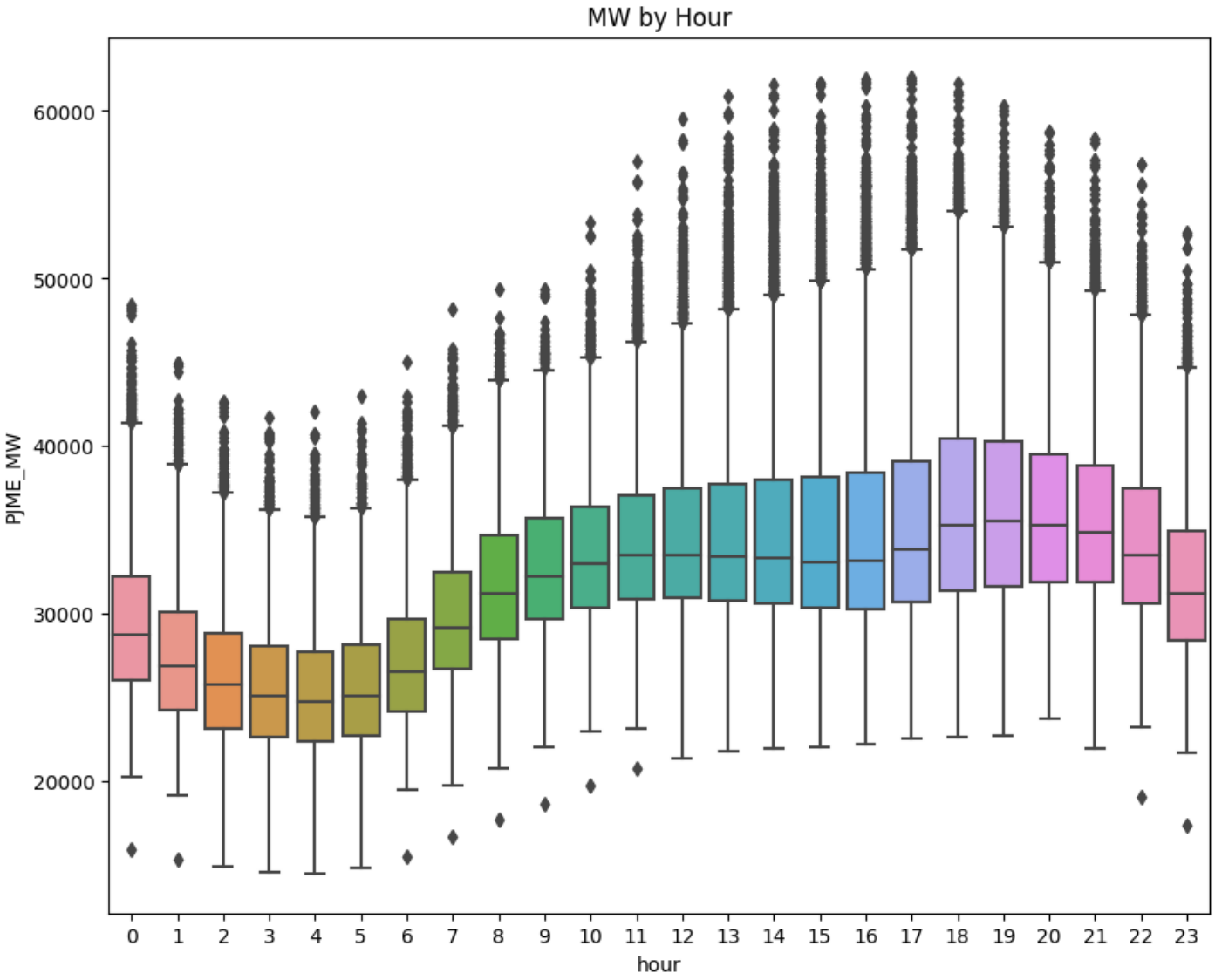
- ❖ Exploratory Data Analysis (EDA):
- ❖ Feature Engineering:
- ❖ Time Series Forecasting:

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.

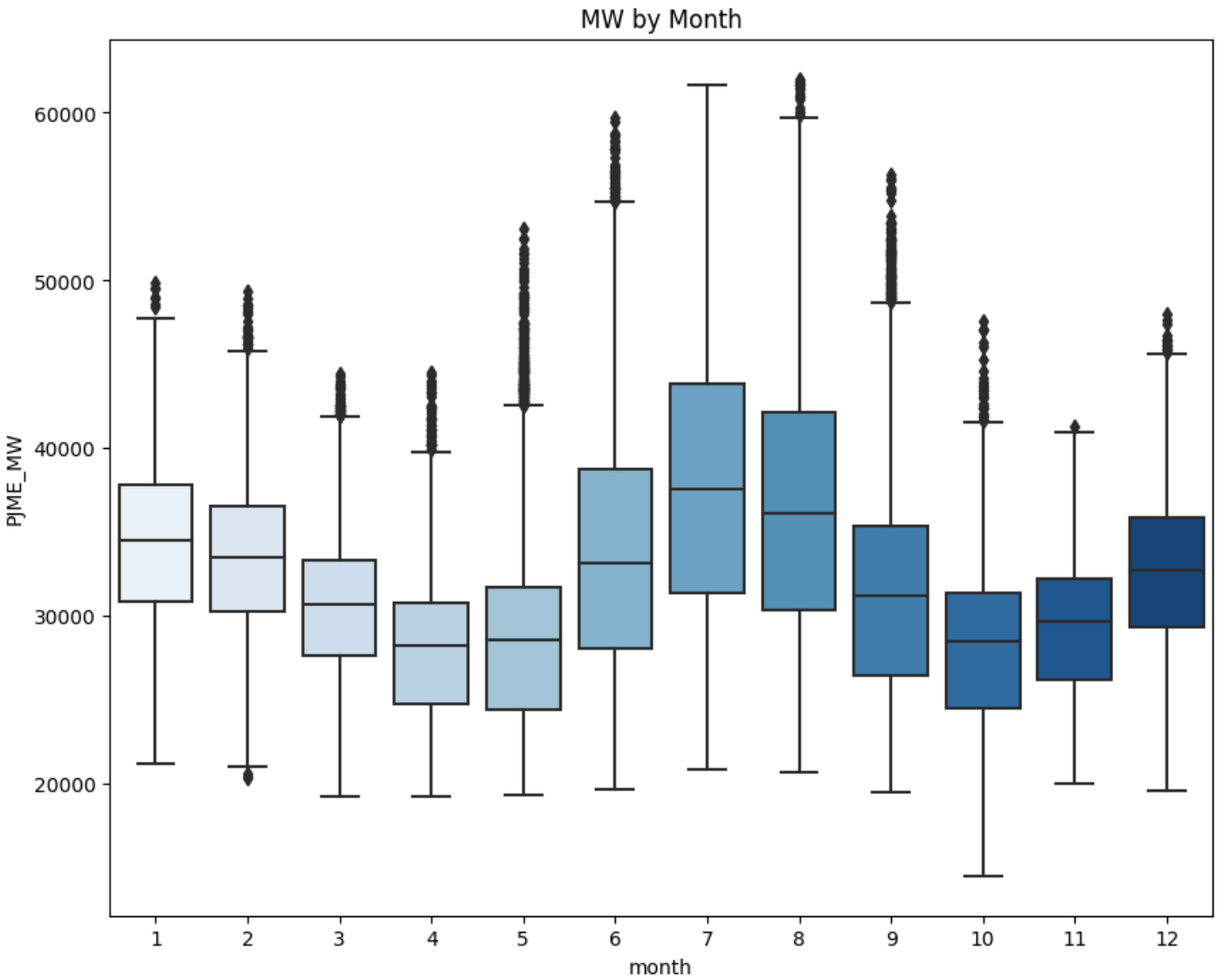
```
# feature creation
def create_features(df):
    df = df.copy()
    df['hour'] = df.index.hour
    df['dayofweek'] = df.index.dayofweek
    df['quarter'] = df.index.quarter
    df['month'] = df.index.month
    df['year'] = df.index.year
    df['dayofyear'] = df.index.dayofyear
    df['dayofmonth']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

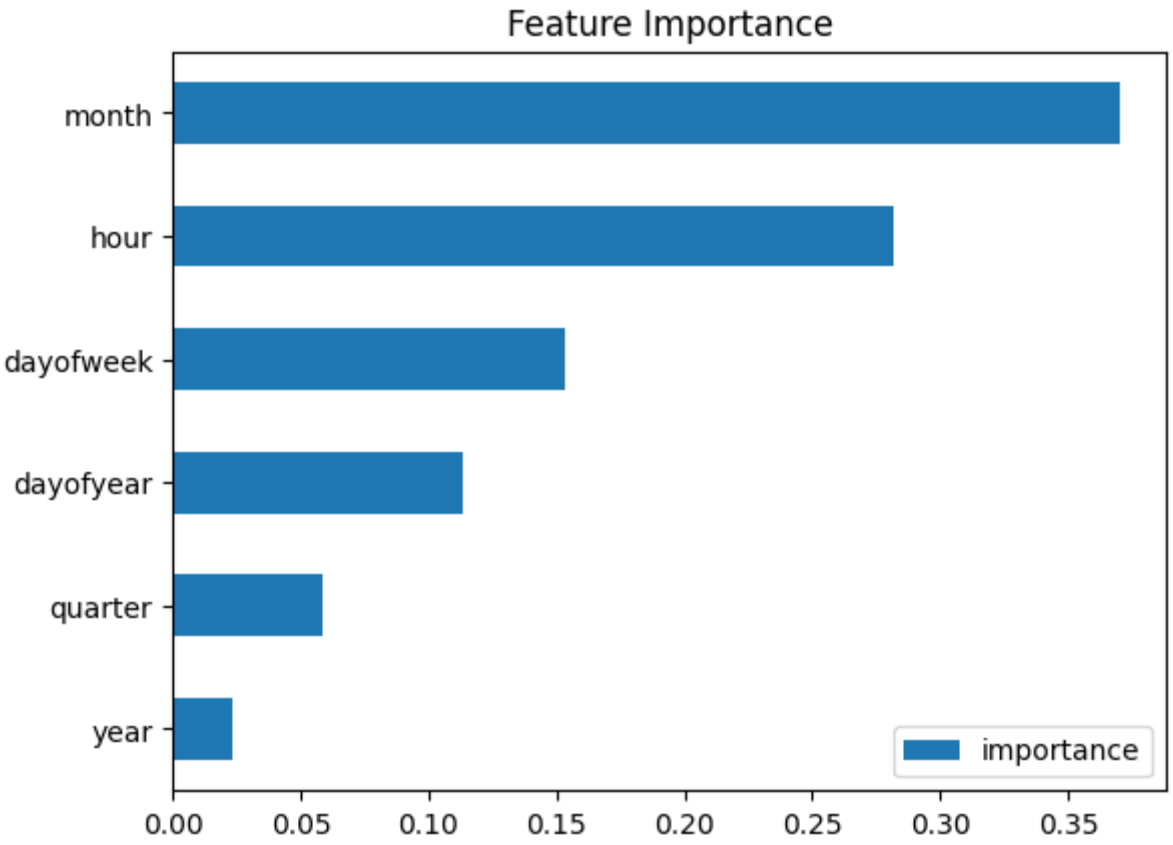
XGBRegressor

```
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.

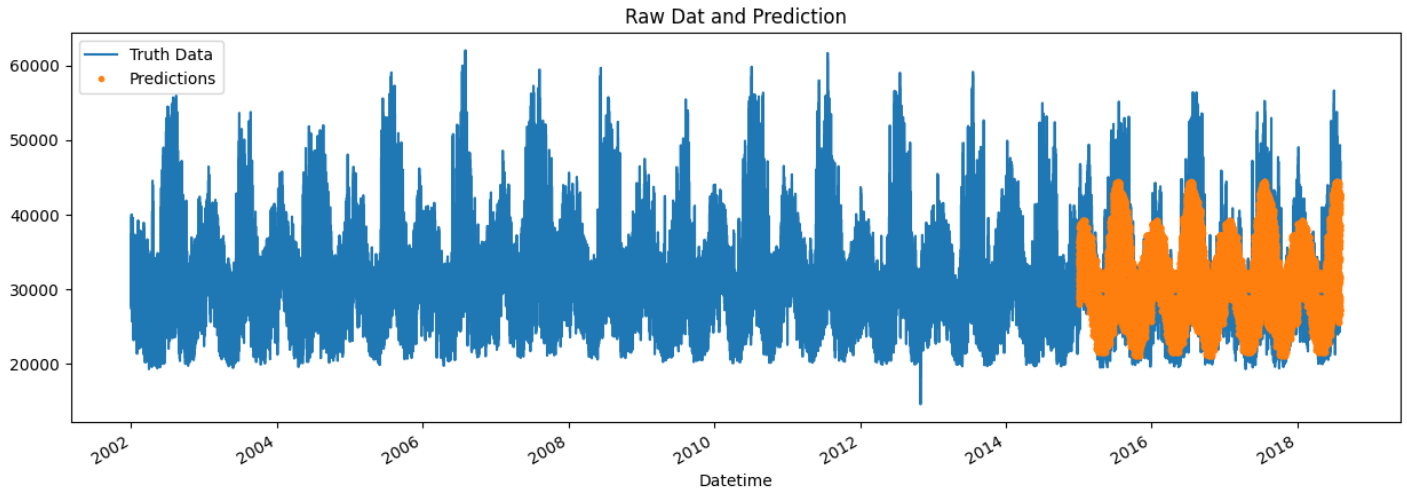
```
fi = pd.DataFrame(data=reg.feature_importances_,
         index=reg.feature_names_in_,
         columns=['importance'])
fi.sort_values('importance').plot(kind='barh', title='Feature Importance')
plt.show()
```



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.