PHASE -2 PROJECT SUBMISSION

INNOVATION TECHNIQUES SUCH AS TIME SERIES ANALYSIS AND MACHINE LEARNING MODELS TO PREDICT FUTURE ENERGY CONSUMPTION PATTERNS

Time series analysis and machine learning models can be powerful tools for predicting future energy consumption. Here's how they can be applied:

Time Series Analysis:

Time series data involves observations collected or recorded at regular intervals over time (e.g., hourly, daily, monthly). Time series analysis techniques can be used to understand patterns and trends in historical energy consumption data.

Methods like moving averages, exponential smoothing, and autoregressive integrated moving average (ARIMA) models can help in modeling and forecasting energy consumption based on past data patterns.

Seasonal decomposition of time series (STL) can be used to separate the data into trend, seasonal, and residual components, making it easier to analyze and predict energy consumption.

Machine Learning Models:

Machine learning models can provide more sophisticated and accurate predictions by considering a broader range of factors and patterns. Some common techniques include:

Regression models: Linear regression, polynomial regression, or more complex variants like support vector regression (SVR) can be used to predict energy consumption based on various input features such as temperature, time of day, and historical data.

Neural networks: Deep learning models like recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) can capture complex temporal dependencies in energy consumption data.

Random forests, gradient boosting, and other ensemble methods can be effective in handling nonlinear relationships and feature interactions.

Feature Engineering:

Extracting relevant features from the data is crucial for machine learning models. Features can include weather data, holidays, economic indicators, and historical energy consumption patterns.

Feature selection and dimensionality reduction techniques can help in identifying the most informative features and reducing model complexity.

Model Evaluation:

It's important to evaluate the performance of the chosen technique using metrics like mean absolute error (MAE), mean squared error (MSE), or root mean squared error (RMSE) to assess prediction accuracy.

Cross-validation can be used to estimate how well the model will perform on unseen data.

Continuous Monitoring and Updating:

Energy consumption patterns may change over time due to factors like weather, technological advancements, or policy changes. It's essential to continuously monitor and update the models to maintain their accuracy.

By combining time series analysis and machine learning models, organizations can make more accurate predictions of future energy consumption, which can be valuable for optimizing energy generation, distribution, and consumption planning.

Certainly! To predict future energy consumption patterns using innovation techniques like time series analysis and machine learning models, you can follow these steps:

Data Collection:

Gather historical energy consumption data over a significant period. Include relevant variables like time/date stamps, weather conditions, economic indicators, and any other factors that may influence energy consumption.

Data Preprocessing:

Clean the data by handling missing values and outliers.

Normalize or scale the data if necessary to ensure that all variables have a similar range.

Time Series Analysis:

Perform exploratory data analysis (EDA) to understand the underlying patterns, trends, and seasonality in the time series data.

Use techniques like decomposition (e.g., seasonal decomposition of time series - STL) to separate the data into its components (trend, seasonality, and residual).

Feature Engineering:

Create additional features that can help improve predictions, such as lagged values (past energy consumption), rolling statistics, and categorical variables for special events or holidays.

Model Selection:

Choose appropriate machine learning models for time series forecasting. Common choices include:

Autoregressive Integrated Moving Average (ARIMA) models for univariate time series.

Exponential Smoothing methods like Holt-Winters for capturing seasonality.

Regression models like linear regression or decision trees for multivariate time series data.

Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks for deep learning-based forecasting.

Model Training:

Split the dataset into training and validation sets to train and evaluate the models.

Tune hyperparameters, select the best-performing model, and fine-tune it.

Model Evaluation:

Use evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) to assess the model's performance on the validation set.

Forecasting:

Use the trained model to make predictions for future energy consumption patterns.

Continuously update the model with new data to improve accuracy as more information becomes available.

Interpret Results:

Analyze the model's predictions and their implications for energy consumption patterns.

Use the insights to make informed decisions related to energy generation, distribution, and consumption optimization.

Deployment:

Deploy the predictive model in a production environment to provide real-time or periodic energy consumption forecasts.

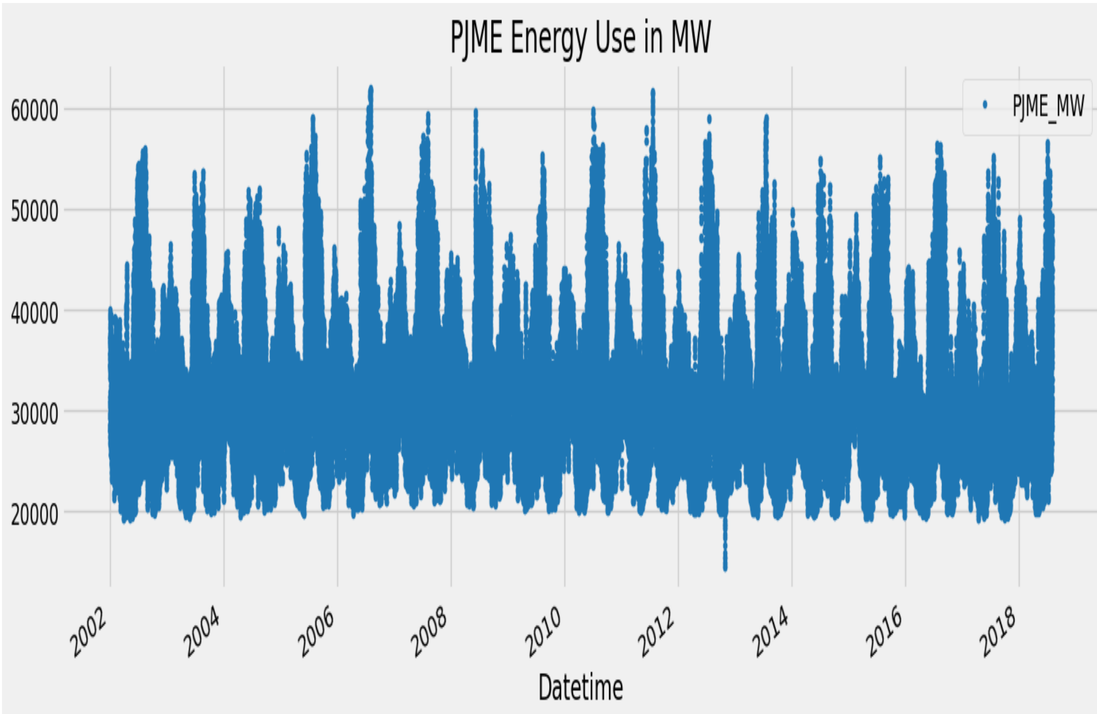
Implement monitoring to ensure the model's ongoing accuracy and reliability.

By applying these techniques, you can develop effective models to predict future energy consumption patterns, helping organizations make informed decisions and optimize their energy management strategies.

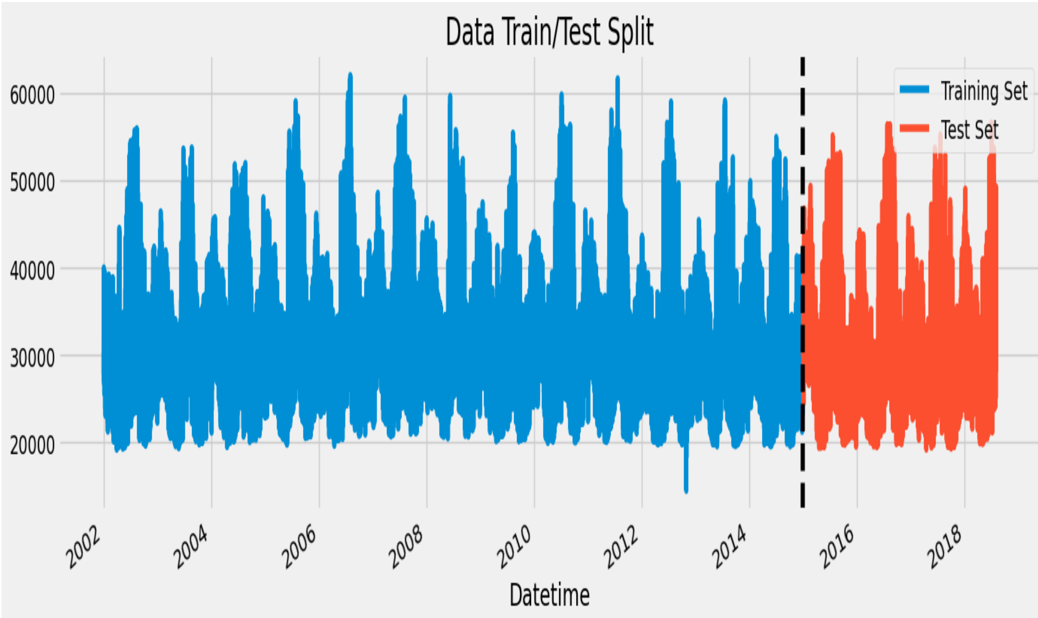
TIME SERIES FORECASTING ANALYSIS

PROGRAM:

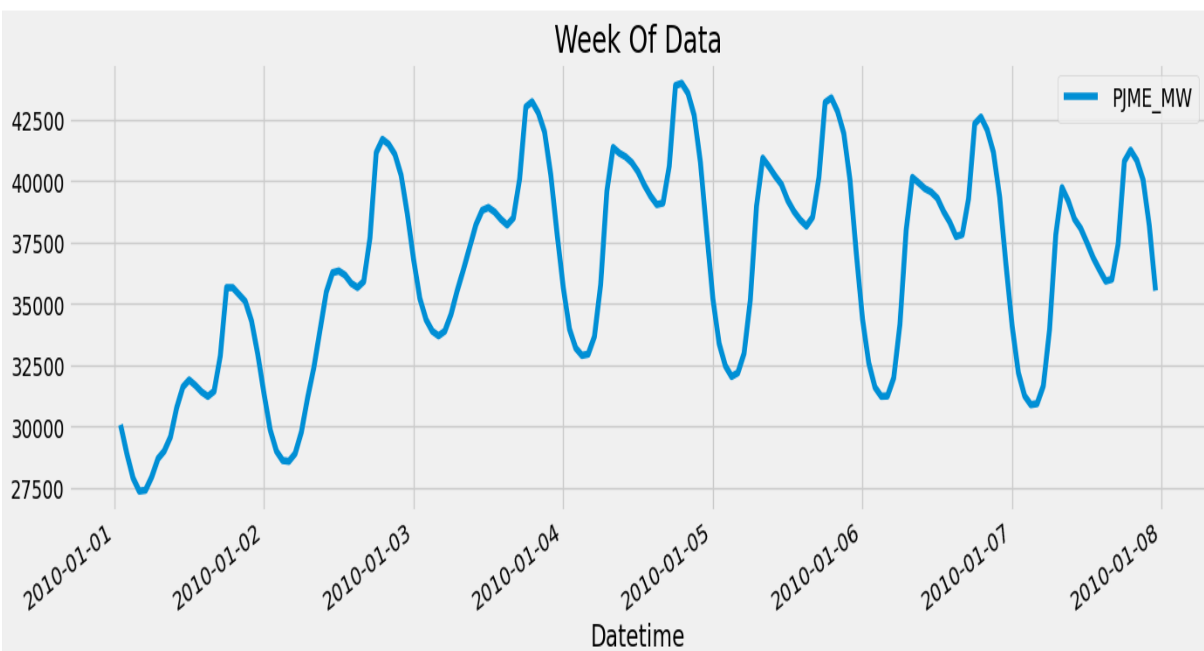
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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()
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()
```



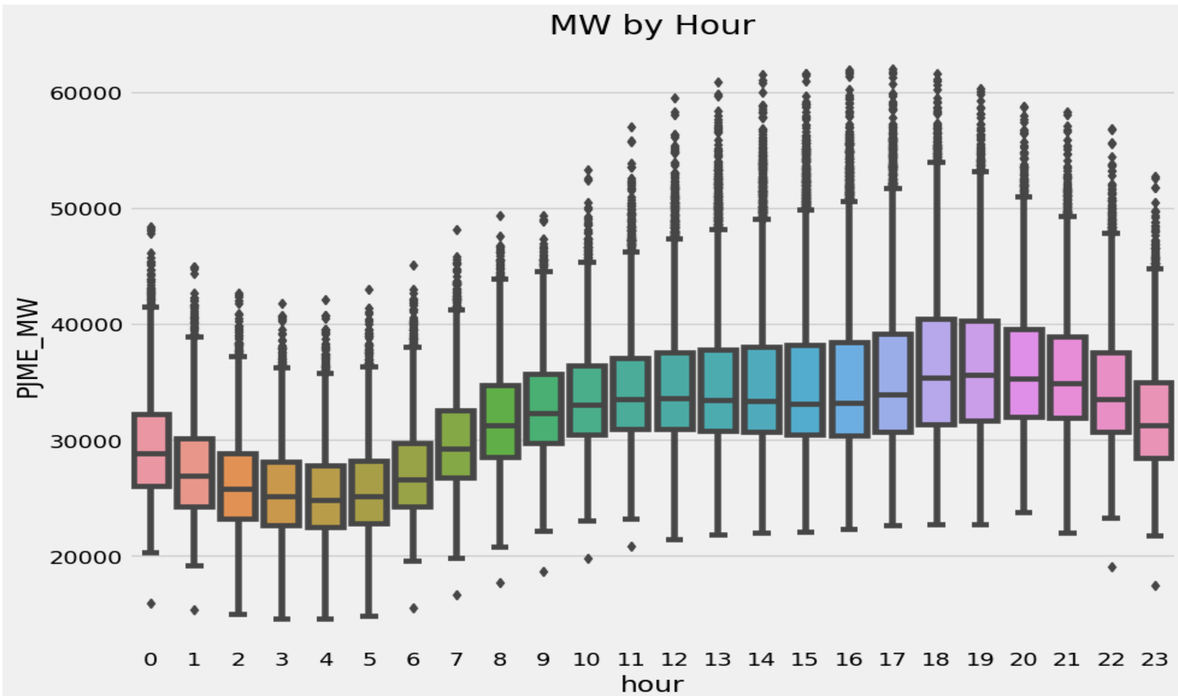
Feature Creation

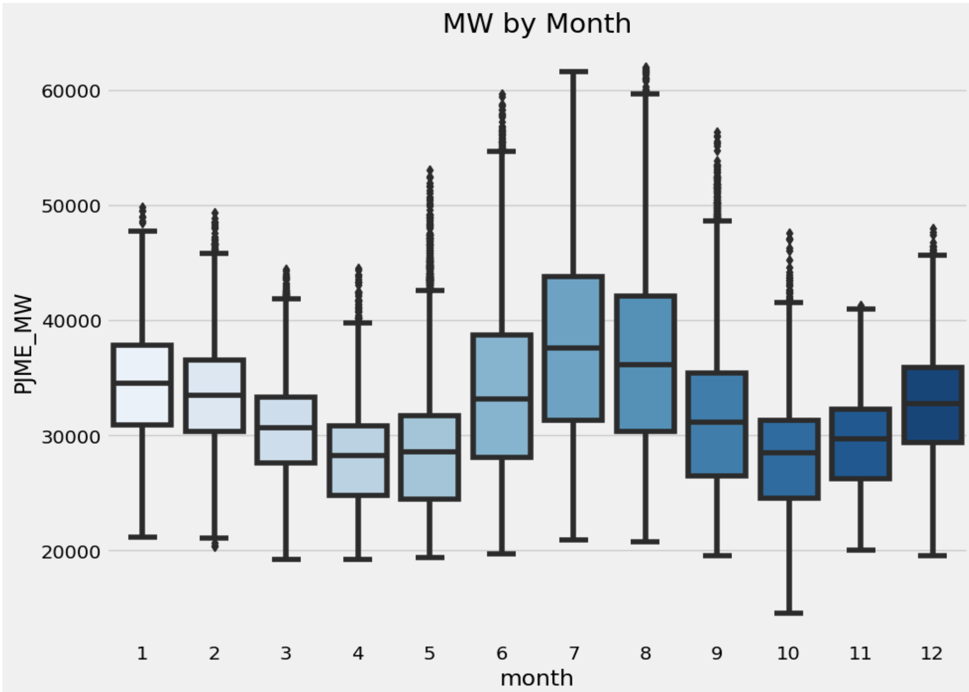
```python
def create_features(df):
    """
    Create time series features based on time series index.
    """

    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)
```

Visualize our Feature / Target Relationship

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

```
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()
```





Create our Model

```
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)
```

```
[20:46:20] WARNING: ../src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg: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
```

Output:

```
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,

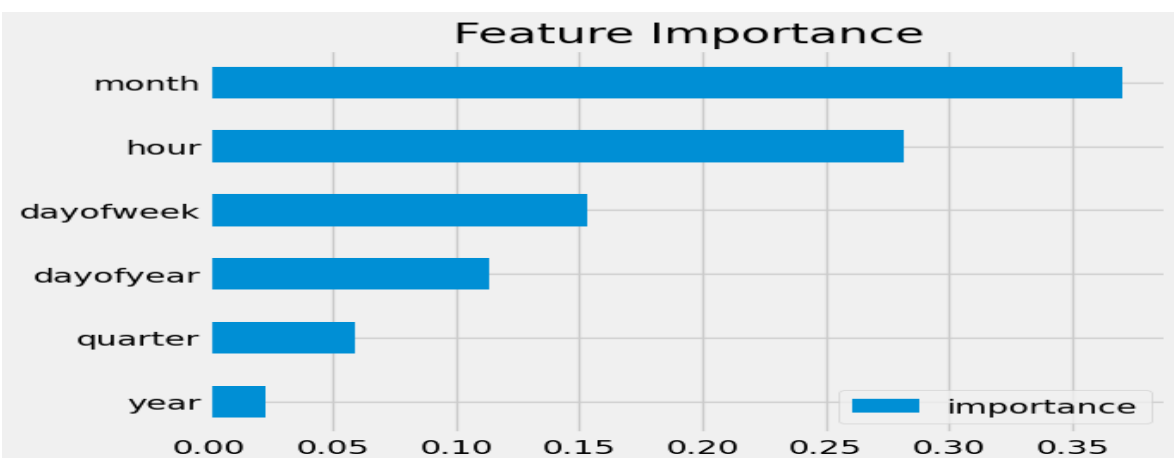
min_child_weight=None, missing=nan, monotone_constraints=None,

n_estimators=1000, n_jobs=None, num_parallel_tree=None,

objective='reg:linear', predictor=None, ...)

Feature Importance:

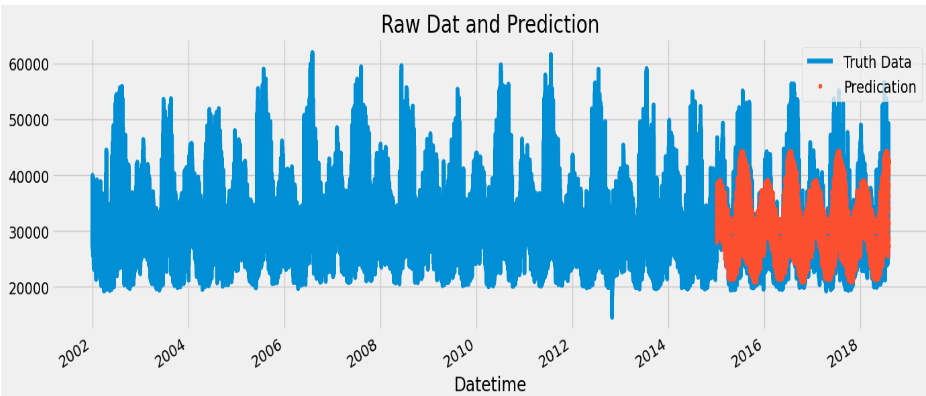
```
fi = pd.DataFrame(data=reg.feature_importances_,
        index=reg.feature_names_in_,
        columns=['importance'])
fi.sort_values('importance').plot(kind='barh', title='Feature Importance')
plt.show()
```



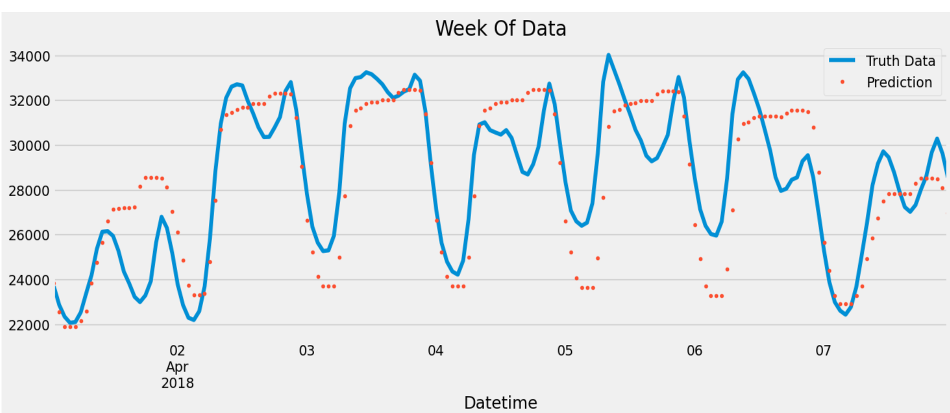
Forecast on test:

```
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(ascending=False).head(10)
```

Output:

```
date
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
```