First Name: ----------

Last Name: ----------

2024-2025

Time Series Analysis for Electricity Consumption

In this session, we will perform time series analysis on electricity consumption using the energy.csv dataset. We will use Python and various libraries such as pandas, numpy, matplotlib, seaborn, and xgboost to analyze and forecast electricity consumption patterns.

Understanding the Dataset

Let's start by understanding the dataset. The energy.csv dataset contains two columns: Datetime and AEP_MW. The Datetime column represents the timestamp of electricity consumption measurements, and the AEP_MW column represents the corresponding electricity consumption in megawatts.

```
import pandas as pd

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

df.head()
```

The df.head() command displays the first few rows of the dataset, giving us a glimpse of the data structure.

Visualizing the Data

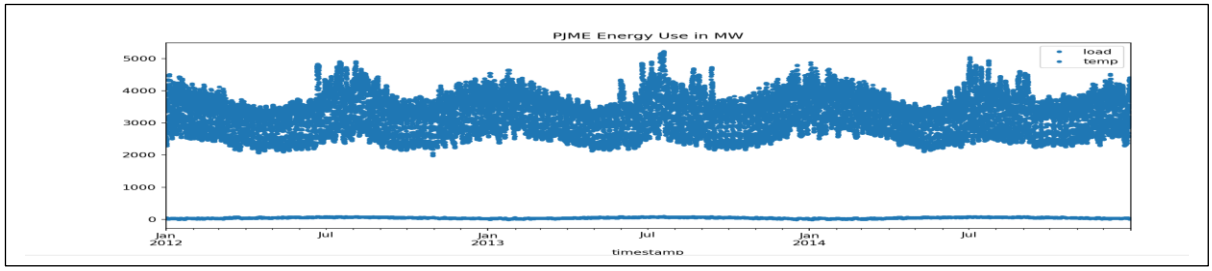
To get a better understanding of the electricity consumption patterns, let's visualize the data using line plots.

```
import matplotlib.pyplot as plt
import seaborn as sns

df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)

df.plot(style='.',
        figsize=(15, 5),
        color=sns.color_palette()[0],
        title='PJME Energy Use in MW')
plt.show()
```

The code above sets the Datetime column as the index and converts it to a datetime data type. Then, we plot the electricity consumption (AEP_MW) over time.

The resulting line plot provides an overview of the electricity consumption trends, highlighting any patterns or anomalies.

Insert your visualization:



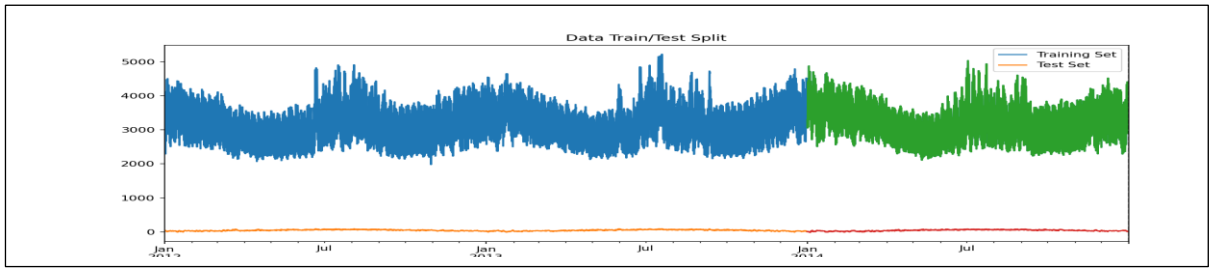
Train/Test Split

Before we proceed with the analysis and forecasting, we need to split the dataset into training and test sets. This allows us to train our model on historical data and evaluate its performance on unseen data.

```
train = df.loc[df.index < '01-01-2015']
test = df.loc[df.index >= '01-01-2014']

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()
```

In the code above, we split the dataset based on a specific date. The training set contains data before ‘01–01–2014’, while the test set contains data from ‘01–01–2014’ onwards. We visualize the split to visualize the separation of the two sets.

Insert your visualization:



Feature Creation

To extract more information from the timestamp, we can create additional time series features. These features can help capture different patterns and seasonality in the data.

```
def create_features(df):
    """
    Create time series features based on the 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['dayofmonth'] = df.index.day
    df['weekofyear'] = df.index.isocalendar().week
    return df

df = create_features(df)
```

Describe the above function:

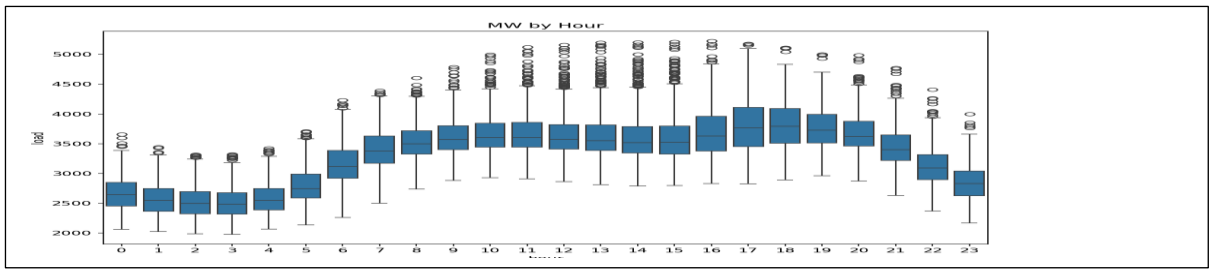
We are just separating the time stamps each hour. Like you said energy consumption per hour and As you have explained date time index module has special feature that could fetch time from the given data by itself. That is what we are doing here.

Visualizing Feature/Target Relationship

Now that we have created additional features, let's visualize the relationship between these features and the electricity consumption (AEP_MW) using box plots.

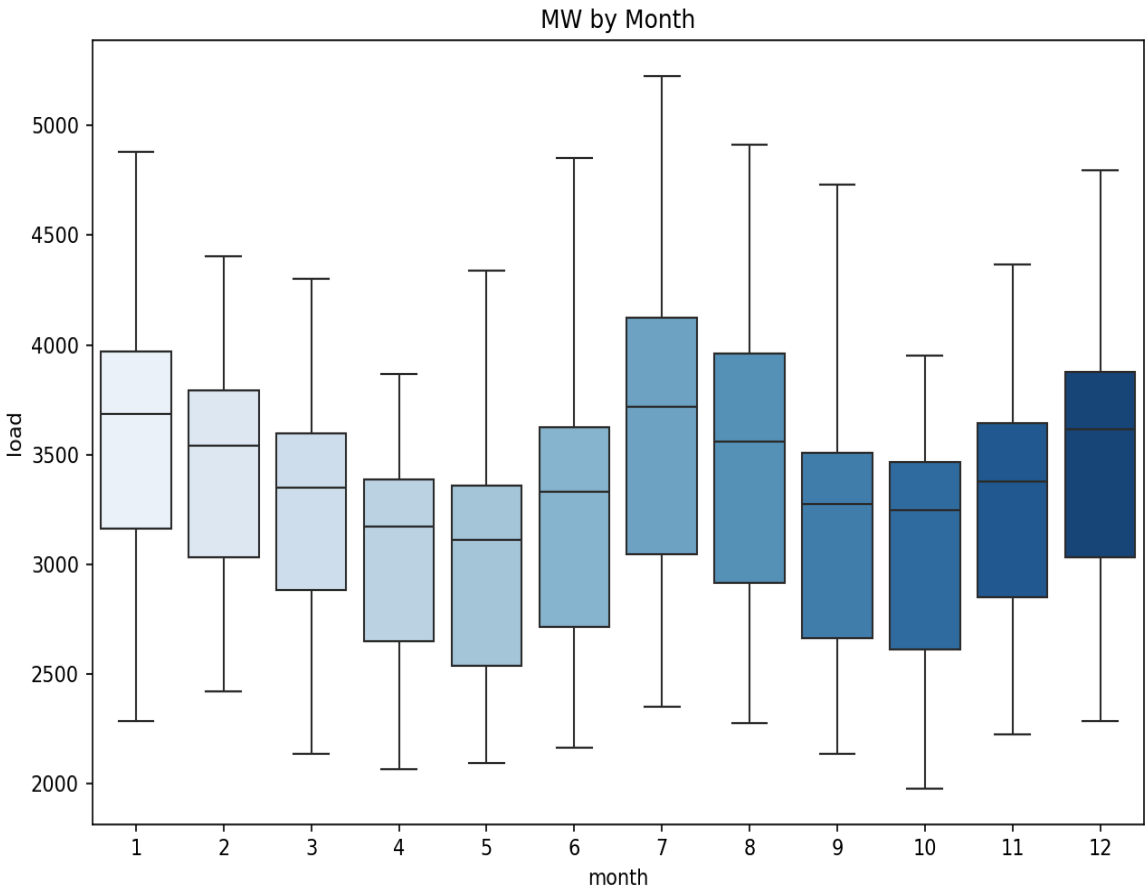
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='hour', y='AEP_MW')
ax.set_title('MW by Hour')
plt.show()
```

Insert your visualization:



The code above creates a box plot showing the distribution of electricity consumption (AEP_MW) across different hours of the day. This helps us understand if there are any hourly patterns in electricity consumption.

```
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='month', y='AEP_MW', palette='Blues')
ax.set_title('MW by Month')
plt.show()
```



Similarly, this code creates a box plot showing the distribution of electricity consumption (AEP_MW) across different months. This helps us identify any monthly patterns or seasonality in electricity consumption.

Model Creation

Now that we have explored the data and created relevant features, let's build a model to forecast electricity consumption.

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

FEATURES = ['dayofyear', 'hour', 'dayofweek', 'quarter', 'month', 'year']
TARGET = 'AEP_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)
```

In the code above, we define the features (X_train) and the target variable (y_train) for the training set. Similarly, we define the features (X_test) and the target variable (y_test) for the test set.

We use the XGBoost library to create an XGBoostRegressor model. XGBoost is a popular gradient boosting algorithm known for its performance in time series forecasting tasks. We fit the model to the training data and evaluate its performance on both the training and test sets using the root mean squared error (RMSE) metric.

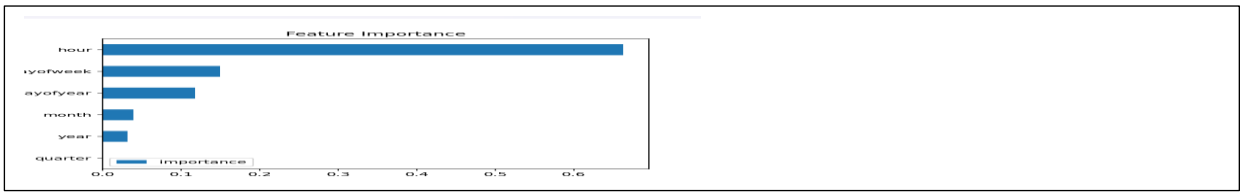
Feature Importance

To understand which features are most important in predicting electricity consumption, we can visualize the feature importance using a bar plot.

```
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()
```

Describe the feature created and interpret the feature score you obtained:

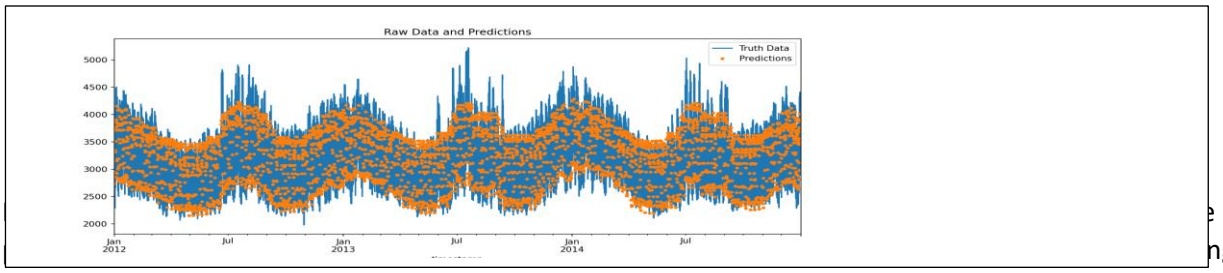


Forecast on Test Set

Now, let's use our trained model to make predictions on the test set and visualize the results.

```
test['prediction'] = reg.predict(X_test)
df = df.merge(test[['prediction']], how='left', left_index=True,
right_index=True)
ax = df[['AEP_MW']].plot(figsize=(15, 5))
test['prediction'].plot(ax=ax, style='.')
plt.legend(['Truth Data', 'Predictions'])
ax.set_title('Raw Data and Predictions')
plt.show()
```

Insert your visualization:



ng the timestamp as the common index. Finally, we plot the actual electricity consumption (AEP_MW) and the predicted values (prediction) on the same graph for visual comparison.

Evaluate Model Performance

To assess the performance of our model, we calculate the root mean squared error (RMSE) between the actual electricity consumption and the predicted values on the test set.

```
score = np.sqrt(mean_squared_error(test['AEP_MW'], test['prediction']))
print(f'RMSE Score on Test set: {score:0.2f}')
```

The code above calculates the RMSE using the mean_squared_error function from the scikit-learn library. The RMSE score provides an indication of how well our model predicts the electricity consumption values. A lower RMSE indicates better accuracy.

```
  return ax.plot(*args, **kwds)
RMSE Score on Test set: 188.36

Process finished with exit code 0
```

Rms value= $\frac{Rmse}{Max\ value - \min value}$ = 6% that is less than 10 so it is good.

Analyzing Prediction Errors

To gain further insights into our model's performance, let's analyze the prediction errors and identify the best and worst 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(5)
```

In the code above, we calculate the absolute error between the actual electricity consumption and the predicted values. We also extract the date from the timestamp and create a new column called 'date'. Finally, we group the data by date, calculate the mean error for each date, and sort the results in descending order. This helps us identify the dates with the highest prediction errors.

Conclusion

In this blog post, we performed time series analysis on electricity consumption using the energy.csv dataset. We explored the data, visualized the patterns, created time series features, built an XGBoost regression model, and evaluated its performance. The analysis provides insights into electricity consumption patterns and allows for accurate forecasting. By understanding electricity consumption trends, we can make informed decisions regarding energy management and resource allocation.

Reference:

This study project is based on an article publish by Kabila MD Musa on "Time Series Analysis for Electricity Consumption using energy.csv Dataset"