Project 3

Title: Predictive Analytics and Anomaly Detection for Sustainable Operations

Team Members:
1. Archisman Chakraborti
2. Nihal Barhaiyya
3. Lalkrishna Vinayak Joshi
4. Raj Arya

Detailed Business Model

1. Value Proposition

- **Optimized Energy Management**: Deliver a solution that not only predicts energy usage but also identifies anomalies in real-time.
- **Sustainable Operations**: Enable businesses to operate sustainably by minimizing energy wastage and reducing greenhouse gas emissions.

2. Customer Segments

- **Small and Medium Enterprises (SMEs)**: Especially those in the industrial sector with significant energy consumption.
- **Large Corporations**: Particularly those looking to reduce their carbon footprint and enhance sustainability.

3. Revenue Streams

- **Tiered Subscriptions**: Basic (essential features), Premium (additional benefits), and Custom (tailored solutions).
- **Consultancy Services**: Providing expertise in implementing and optimizing energy management strategies.
- **Data Analytics Services**: Offering detailed insights and reports on energy consumption and efficiency.

4. Key Activities

- **Platform Development and Maintenance**: Ensuring the solution is reliable, secure, and up-to-date.
- **Data Analysis and Management**: Handling large datasets and ensuring accurate predictions and anomaly detections.
- **Customer Support and Training**: Assisting customers in utilizing the platform effectively.

5. Key Resources

- **Technical Team**: Experts in AI, machine learning, and energy management.
- **Data**: Access to reliable and comprehensive energy consumption data.
- **Technology**: Infrastructure to support data analysis, storage, and platform hosting.

6. Channels

- **Online Platform**: A web-based dashboard for monitoring and management.
- **Mobile Application**: Ensuring accessibility and real-time alerts on mobile devices.
- **Customer Support**: Via chat, email, and phone.

7. Customer Relationships

- **Dedicated Support**: Assigning account managers for personalized service.
- **Community Building**: Creating forums or groups for customers to share experiences and learnings.
- **Regular Check-ins**: Ensuring customer satisfaction and gathering feedback for improvements.

8. Partnerships

- **Energy Providers**: For accessing real-time data and understanding energy supply chains.
- **Regulatory Bodies**: Ensuring the platform adheres to and stays updated with relevant regulations and standards.
- **Technology Partners**: For continuous technological advancements and integrations.

9. Cost Structure

- **Platform Development**: Costs related to technology, development team, and maintenance.
- **Marketing and Sales**: Expenses for promoting the platform and acquiring customers.
- **Customer Support**: Costs related to training, support staff, and resource creation.
- **Partnership and Collaboration**: Any costs related to forming and maintaining partnerships.

Detailed SWOT Analysis

Strengths

- **Innovative Solution**: Leveraging AI and machine learning for predictive analytics and anomaly detection in energy management.
- **Sustainability Focus**: Addressing the global need for sustainable operations and reduced energy wastage.
- **Customization**: Ability to tailor solutions according to specific business needs and operational parameters.

Weaknesses

- **Data Quality**: The effectiveness of predictions and anomaly detection is heavily dependent on the quality and accuracy of data.
- **Technical Complexity**: Ensuring the platform remains user-friendly despite the complex technologies and algorithms involved.
- **Market Skepticism**: Potential skepticism from businesses regarding the reliability and effectiveness of AI-driven energy management.

Opportunities

- **Growing Sustainability Trends**: Capitalizing on the increasing global emphasis on sustainability and energy conservation.
- **Technological Advancements**: Continuously evolving the platform with advancements in AI, machine learning, and data analytics.
- **Global Expansion**: Exploring markets beyond the initial focus area, adapting the solution to various industries and regions.

Threats

- **Competitive Market**: The emergence of similar solutions from competitors, potentially with more features or lower pricing.
- **Regulatory Changes**: Adapting to changes in energy and data management regulations, which may require adjustments in the platform.
- **Technological Obsolescence**: Keeping up with rapid technological advancements and ensuring the platform does not become obsolete.

Descriptive Analysis

Basic Statistics

- Count: 145,366 records
- Mean Power Consumption: 32,080 MW
- Standard Deviation: 6,464 MW
- Minimum Power Consumption: 14,544 MW
- 25th percentile: 27,573 MW
- Median (50th percentile): 31,421 MW
- 75th percentile: 35,650 MW
- Maximum Power Consumption: 62,009 MW

Missing Values

- There are no missing values in the dataset.

Visual Analysis

- The time series plot shows fluctuations in power consumption over time.

Model explanation

As for the mathematical expression of the model, extracting an exact mathematical formula from an ensemble model like XGBoost is non-trivial and generally not practical due to its complexity. The model consists of numerous decision trees, each contributing to the final prediction.

However, here's a simplified explanation:

$$\text{Prediction} = f_1(x) + f_2(x) + \ldots + f_N(x)$$

where $f_1, f_2, \ldots, f_N$ are the individual decision trees in the ensemble (with N being the total number of trees), and x represents the input features.

Each decision tree $f_i(x)$ outputs a prediction based on the input features x. The final prediction of the XGBoost model is a sum of the predictions from all the trees in the ensemble, possibly weighted.

XGBoost Model Description

XGBoost (Extreme Gradient Boosting) is an ensemble learning method, specifically it is a tree boosting method. The model is comprised of the aggregation of several decision trees. The predictions from all the trees are combined (typically summed) to make a final prediction.

Decision Trees

Each decision tree is a structure that makes decisions based on asking a series of questions. For the XGBoost model, the decision trees are binary trees, meaning each node in the tree splits the data into two branches based on a feature value. The decision of which feature to split on and what value to split at is determined during training, with the objective to separate the data in a way that minimizes the prediction error.

Boosting

In the context of the XGBoost model, "boosting" refers to the method used to create the ensemble of trees. Initially, a single tree is trained on the data, and its predictions are used to calculate residuals (differences) between its predictions and the actual target values. The next tree is then trained to predict these residuals, essentially trying to correct the errors made by the first tree. This process is repeated, with each subsequent tree trying to correct the errors made by the ensemble of all previous trees.

Mathematical Formulation

Mathematically, the prediction $\hat{y}$ for a given input sample x can be expressed as:

$$\hat{y}(x) = \sum_{k=1}^{K} f_k(x)$$

where:

- K is the total number of trees in the model.
- $f_k(x)$ is the prediction of the k^{th} tree.
- x is the input sample.

Each tree f_k produces a prediction for a given input sample, and these predictions are summed to produce the final prediction $\hat{y}(x)$.

Example

Imagine we have a model with three trees, and for a given input sample x, the trees make the following predictions:

- Tree 1 predicts 15
- Tree 2 predicts -5
- Tree 3 predicts 10

The final prediction $\hat{y}(x)$ of the model would be:

$$\hat{y}(x) = 15 - 5 + 10 = 20$$

Additional Notes

- **Non-linear Relationships**: XGBoost can model non-linear relationships because each decision tree is a non-linear model, and the ensemble of trees can represent a sum of non-linear terms.

- **Interactions Between Variables**: The model can also represent interactions between variables. In the tree structure, if a split on one feature is followed by a split on another feature, this indicates that the model has found an interaction between the two features.

- **Regularization**: XGBoost also incorporates regularization (penalties for overly complex trees) into the learning process to avoid overfitting.

Creating an explicit expression for an XGBoost model, especially one that is human-readable, is extremely complex due to the ensemble nature of the model and the potential depth and complexity of each decision tree within the ensemble.

An XGBoost model makes predictions by summing the contributions of all trees in the ensemble. Each tree is a series of binary decisions based on feature values, culminating in a prediction. A single tree might look something like this in a simplified pseudo-code form:

```
plaintext
if feature_1 < some_value_1:
    if feature_2 < some_value_2:
        prediction = value_A
    else:
        prediction = value_B
else:
    if feature_3 < some_value_3:
        prediction = value_C
    else:
        prediction = value_D
```

In the XGBoost model we've trained, there are 10 trees (`n_estimators=10`), each of which has a structure similar to the one described above, but likely with more nodes and potentially deeper structures.

If we attempted to write out a comprehensive mathematical representation of all 10 trees in the trained model, it would be extremely lengthy and complex. Moreover, XGBoost models, especially ones with many estimators and deep trees, are practically impossible to represent with a simple equation or piece of code.

Here’s a very simplified pseudo-expression for an XGBoost model with two trees:

$$\text{Prediction} = \left(\sum_{\text{paths in Tree 1}} \text{Path Output} \times \text{Path Condition}\right) + \left(\sum_{\text{paths in Tree 2}} \text{Patl}$$

Where:

- **Path Output** is the prediction value at the leaf node of a particular path.
- **Path Condition** is a boolean (1 or 0) indicating whether the input sample follows the path.

Let's create a visual representation of one tree in the trained XGBoost model as text. This will give a sense of how the decision making is structured within a single tree, even though it is only one part of the whole model. This text-based visualization will present each split in the tree, the feature, and the threshold used for the split, and finally the output at each leaf.

The textual representation of the first tree in the XGBoost model looks like this:

```
0:[f1<0.751977086] yes=1,no=2,missing=1
        1:[f3<0.890597701] yes=3,no=4,missing=3
                3:[f3<-0.272763431] yes=7,no=8,missing=7
                        7:[f3<-1.1452843]
yes=15,no=16,missing=15
                                15:[f3<-1.43612456]
yes=27,no=28,missing=27
                                        27:leaf=3511.09644
                                        28:leaf=3422.18555
                                16:[f3<-0.854444027]
yes=29,no=30,missing=29
                                        29:leaf=3139.98462
                                        30:leaf=2925.74487
                        8:[f3<0.599757433]
yes=17,no=18,missing=17
                                17:[f3<0.0180768594]
```

```
yes=31,no=32,missing=31
                                        31:leaf=3493.42261
                                        32:leaf=3841.9458
                                18:[f1<-1.25104451]
yes=33,no=34,missing=33
                                        33:leaf=3124.0083
                                        34:leaf=3285.93799
                4:[f3<1.47227836] yes=9,no=10,missing=9
                        9:[f3<1.18143797]
yes=19,no=20,missing=19
                                19:leaf=2899.18311
                                20:[f1<-0.249533713]
yes=35,no=36,missing=35
                                        35:leaf=3035.60718
                                        36:leaf=2956.72876
                        10:leaf=3336.229
        2:[f3<0.599757433] yes=5,no=6,missing=5
                5:[f3<0.0180768594]
yes=11,no=12,missing=11
                        11:[f3<-1.1452843]
yes=21,no=22,missing=21
                                21:[f3<-1.43612456]
yes=37,no=38,missing=37
                                        37:leaf=3241.646
                                        38:leaf=3148.24146
                                22:[f3<-0.272763431]
yes=39,no=40,missing=39
                                        39:leaf=2672.27856
                                        40:leaf=3112.16968
                        12:[f3<0.308917165]
yes=23,no=24,missing=23
                                23:leaf=3497.56494
                                24:leaf=3371.99487
                6:[f3<1.47227836] yes=13,no=14,missing=13
                        13:[f3<0.890597701]
yes=25,no=26,missing=25
                                25:leaf=2900.27124
                                26:[f3<1.18143797]
yes=41,no=42,missing=41
                                        41:leaf=2580.92725
                                        42:leaf=2758.95288
                        14:leaf=3095.45093
```

Explanation

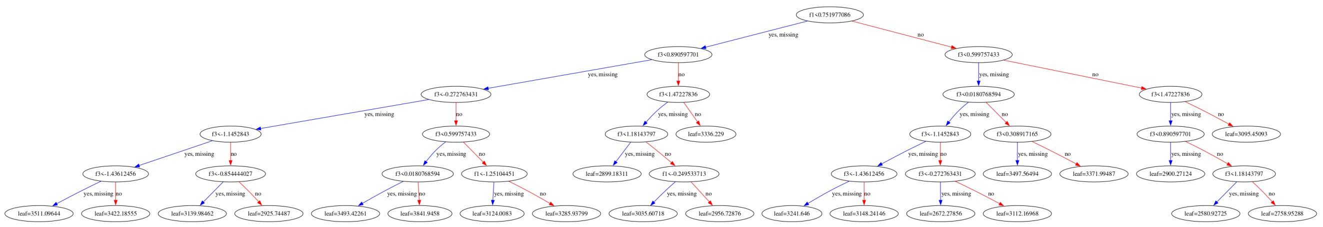
- Each row represents a node in the tree, starting with the root node (0).
- The first part of the row (e.g., `f1<0.751977086`) is the decision rule at that node. For instance, `f1<0.751977086` implies that the decision involves checking if the first feature (f1) is less than 0.751977086.
- The `yes`, `no`, and `missing` entries indicate the child node that will be traversed next, depending on the outcome of the decision rule and whether the feature value is missing.

- When a `leaf` entry is encountered, it indicates that this node is a terminal node and provides the output value if the decision path reaches this point.

A graphical representation of the first tree in the XGBoost model

Link to github project

https://github.com/ScientificArchisman/FeynnLabsInternship/tree/main/Project3



In [2]:
```python
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')
```

Importing the dataset

The dataset is a time series dataset and hence the index is set as the Datetime and parsed as a date.

In [3]:
```python
df = pd.read_csv('/Users/archismanchakraborti/Desktop/python_files/FeynnL
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)
df.head()
```

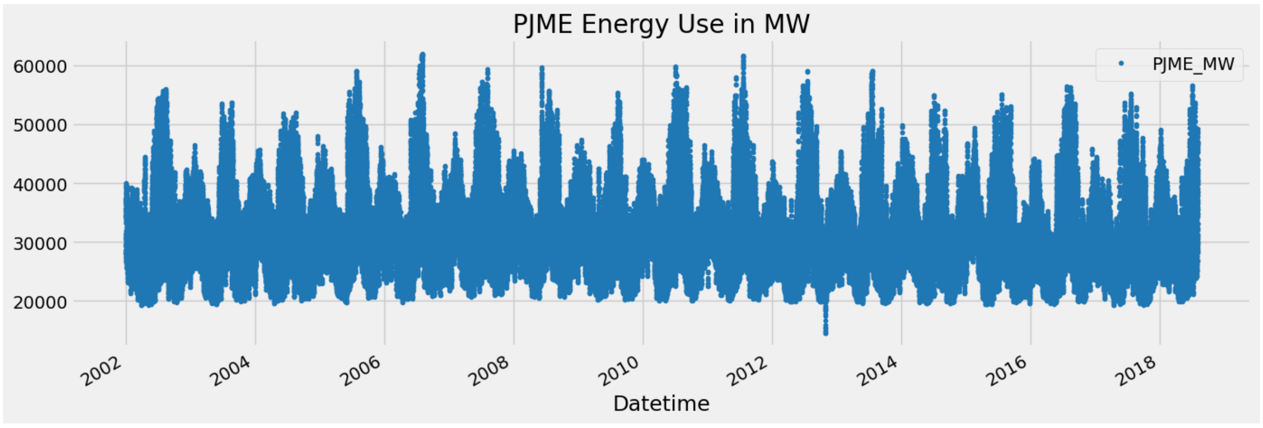
Out[3]:

	PJME_MW
Datetime	
2002-12-31 01:00:00	26498.0
2002-12-31 02:00:00	25147.0
2002-12-31 03:00:00	24574.0
2002-12-31 04:00:00	24393.0
2002-12-31 05:00:00	24860.0

Exploratory Data Analysis

The dataset is plotted to see how the data looks like. The data is plotted as timeseries scatter plot to see the trend in the data.

In [4]:
```python
df.plot(style='.',
        figsize=(15, 5),
        color=color_pal[0],
        title='PJME Energy Use in MW')
plt.show()
```



Train-test split

The data is split into train and test set. The train set is used to train the model and the test set is used to evaluate the model. 30 percent of the data is used as test set. This split is done randomly. The split-size is adequate for the model to learn the pattern in the data.

In [5]:
```
from sklearn.model_selection import train_test_split
```

In [6]:
```
y = df["PJME_MW"]
X = df.index

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

train_df = pd.DataFrame(index=X_train, data={'PJME_MW': y_train})
test_df = pd.DataFrame(index=X_test, data={'PJME_MW': y_test})
```

In [7]:
```
train, test = train_test_split(df, test_size=0.3, shuffle=False)
train
```

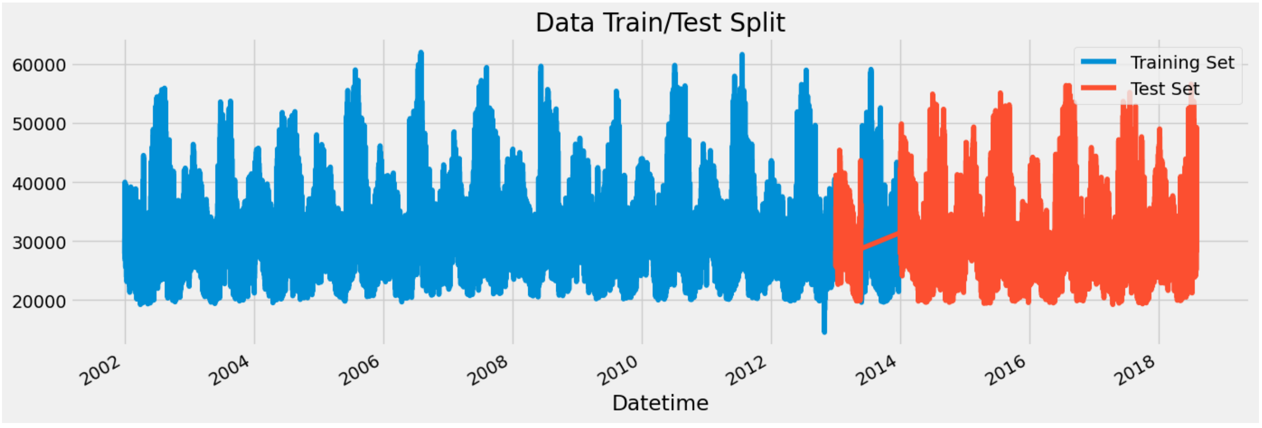
Out[7]:

	PJME_MW
Datetime	
2002-12-31 01:00:00	26498.0
2002-12-31 02:00:00	25147.0
2002-12-31 03:00:00	24574.0
2002-12-31 04:00:00	24393.0
2002-12-31 05:00:00	24860.0
...	...
2013-05-23 16:00:00	38525.0
2013-05-23 17:00:00	38544.0
2013-05-23 18:00:00	38057.0
2013-05-23 19:00:00	36967.0
2013-05-23 20:00:00	36242.0

101756 rows × 1 columns

Visualizing the train and test set

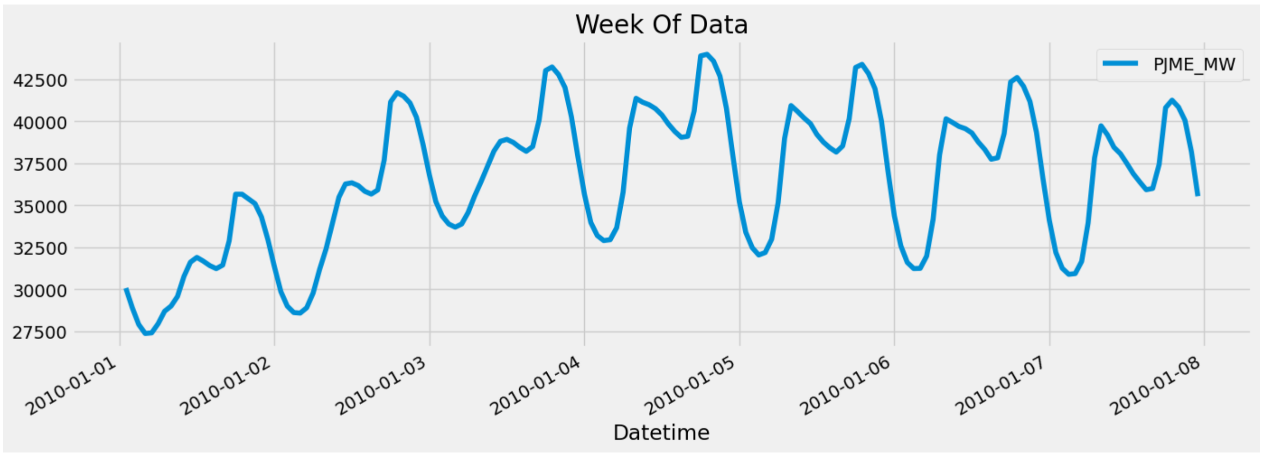
In [8]:
```
fig, ax = plt.subplots(figsize=(15, 5))
train_df.plot(ax=ax, label='Training Set', title='Data Train/Test Split')
test_df.plot(ax=ax, label='Test Set')
ax.legend(['Training Set', 'Test Set'])
plt.show()
```



Seeing one week of data

The data is plotted for one week to see the pattern in the data. This plot is used to see if the data is stationary or not. The data is not stationary as the mean and variance is changing with time.

In [9]:
```
df.loc[(df.index > '01-01-2010') & (df.index < '01-08-2010')] \
.plot(figsize=(15, 5), title='Week Of Data')
plt.show()
```



Feature Creation

The date time objects are extracted from the datetime index. The day of the week, day of the month, month and year, week of the year, week of the month, hour, etc are extracted from the datetime index. These features are used to train the model.

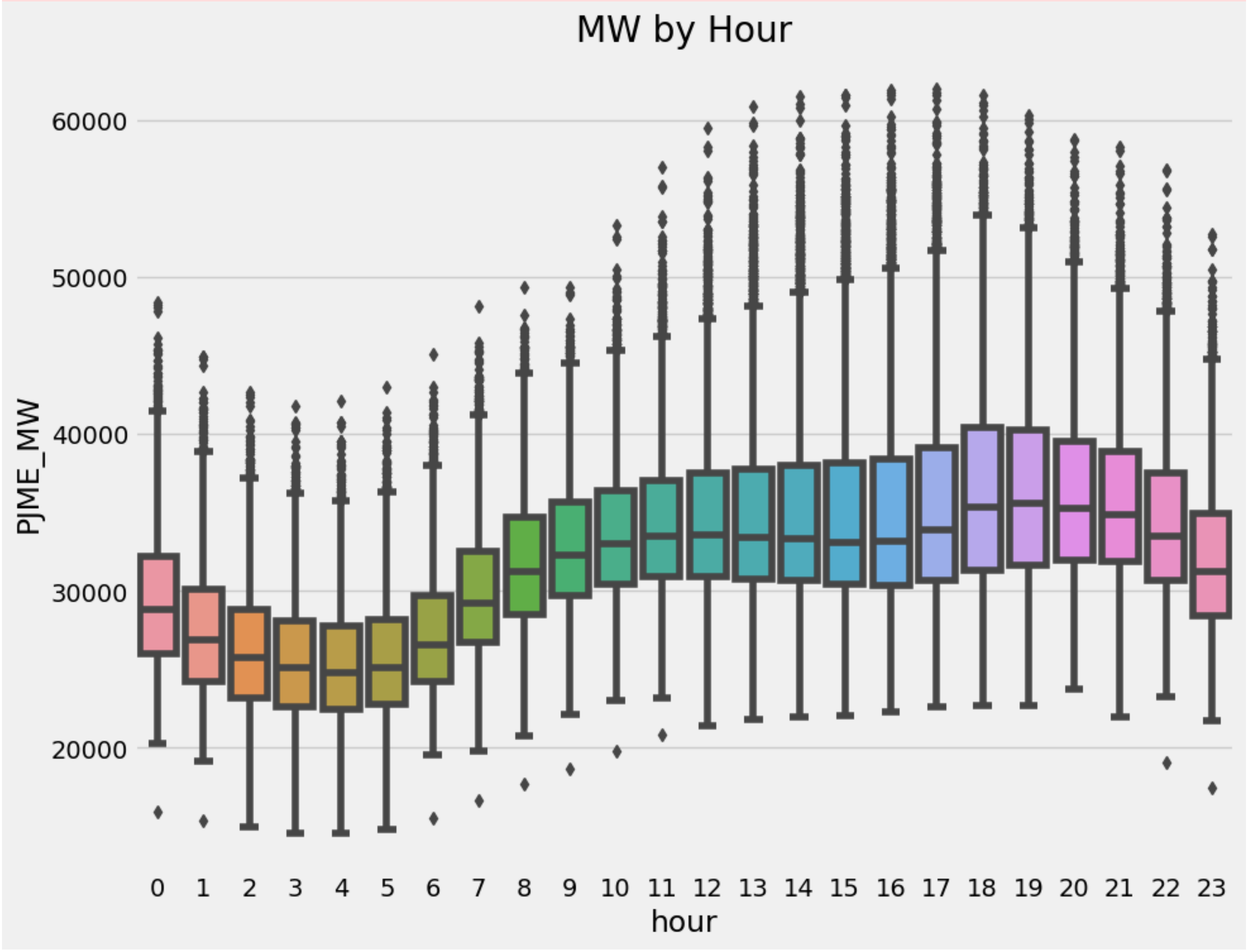
In [10]:
```
def create_features(df):
    """
    Create time series features based on time series index.
    """

    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

THe features are plotted as barplots to see the relationship in them and to see if they have any outliers which may throw off the model.

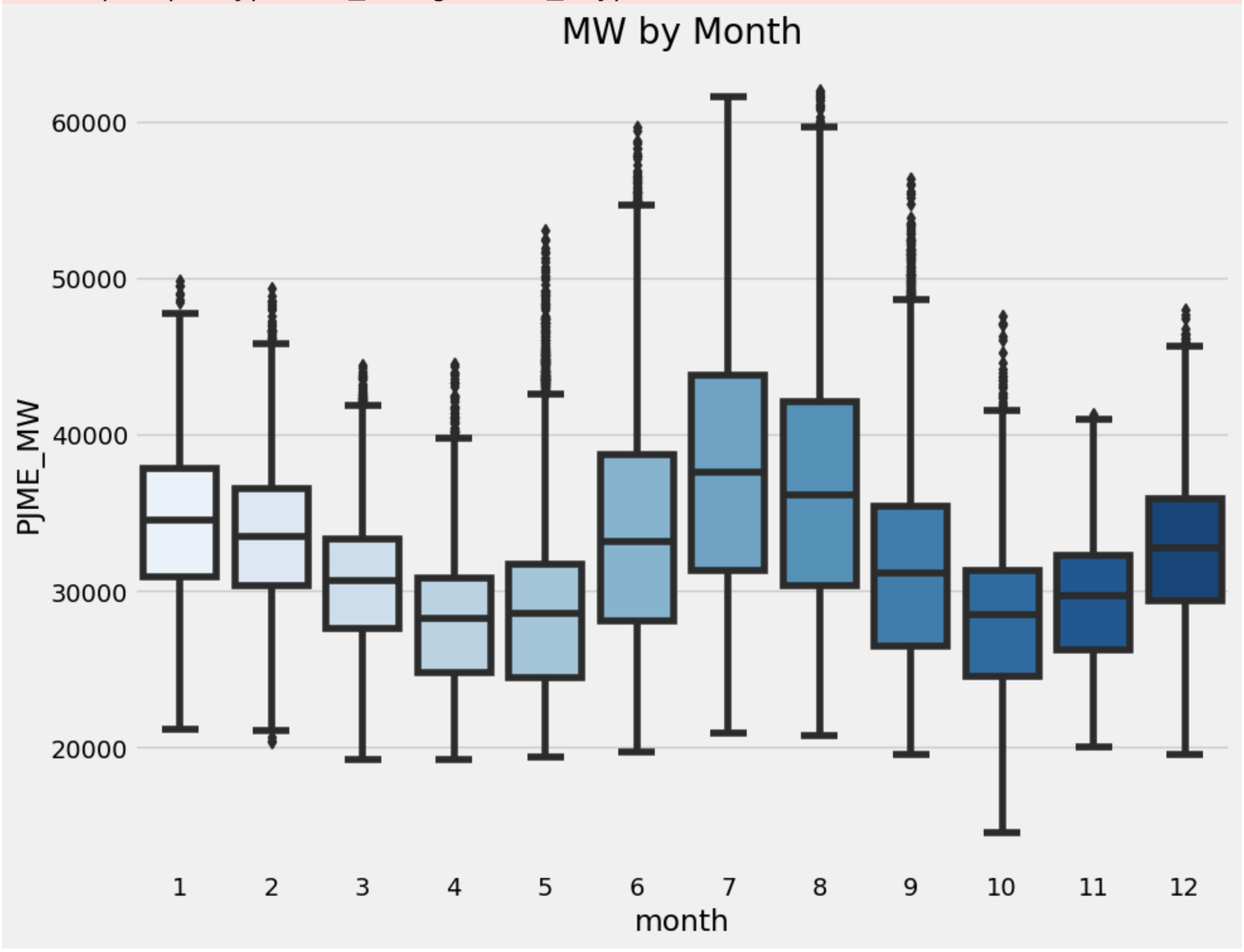
In [11]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(data=df, x='hour', y='PJME_MW')
ax.set_title('MW by Hour')
plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [13]:
```
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()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Create our Model

Different models were tried and the best model was selected. The models tried were:

- Linear Regression
- Random Forest
- XGBoost
- LSTM

The best model was selected based on the RMSE value. The model with the lowest RMSE value was selected as the best model.

In [14]:
```python
train = create_features(train_df)
test = create_features(test_df)

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]
```

Hyperparameter tuning

The hyperparameters of the best model were tuned to get the best model. The hyperparameters were tuned using the library Optuna which uses a Bayesian approach to tune the hyperparameters.

In [16]:
```python
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)
```

```
[0]	validation_0-rmse:32733.48164	validation_1-rmse:31611.77619
```

```
[100]	validation_0-rmse:12624.48459	validation_1-rmse:11644.83487
[200]	validation_0-rmse:5841.69292	validation_1-rmse:5224.49850
[300]	validation_0-rmse:3925.78380	validation_1-rmse:3867.93470
[400]	validation_0-rmse:3451.46998	validation_1-rmse:3723.31075
[500]	validation_0-rmse:3294.07124	validation_1-rmse:3686.69999
[600]	validation_0-rmse:3212.57355	validation_1-rmse:3660.40698
[700]	validation_0-rmse:3160.54831	validation_1-rmse:3646.52750
[800]	validation_0-rmse:3118.43679	validation_1-rmse:3639.39364
[900]	validation_0-rmse:3083.40967	validation_1-rmse:3634.32982
[999]	validation_0-rmse:3054.01223	validation_1-rmse:3632.46896
```

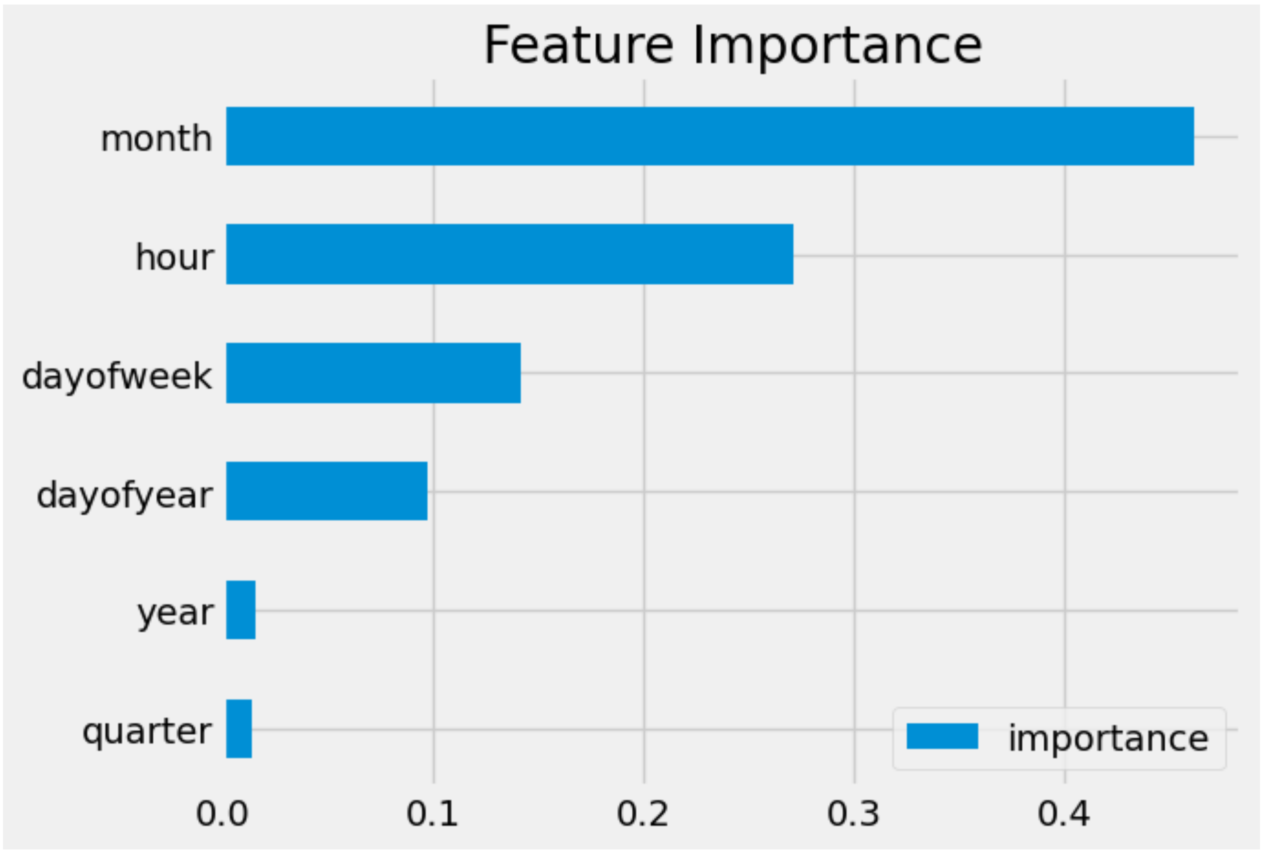
Out[16]:

```
▾ XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping
_rounds=50,
             enable_categorical=False, eval_metric=None, featur
e_types=None,
             gamma=None, grow_policy=None, importance_type=Non
e,
             interaction_constraints=None, learning_rate=0.01,
```

Feature Importance

In [17]:

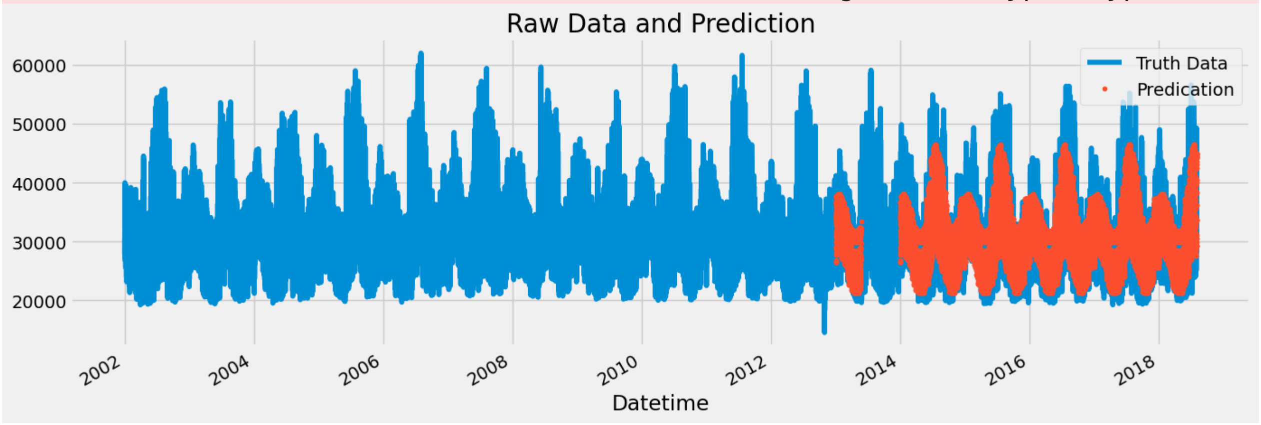
```python
fi = pd.DataFrame(data=reg.feature_importances_,
                  index=reg.feature_names_in_,
                  columns=['importance'])
fi.sort_values('importance').plot(kind='barh', title='Feature Importance'
plt.show()
```



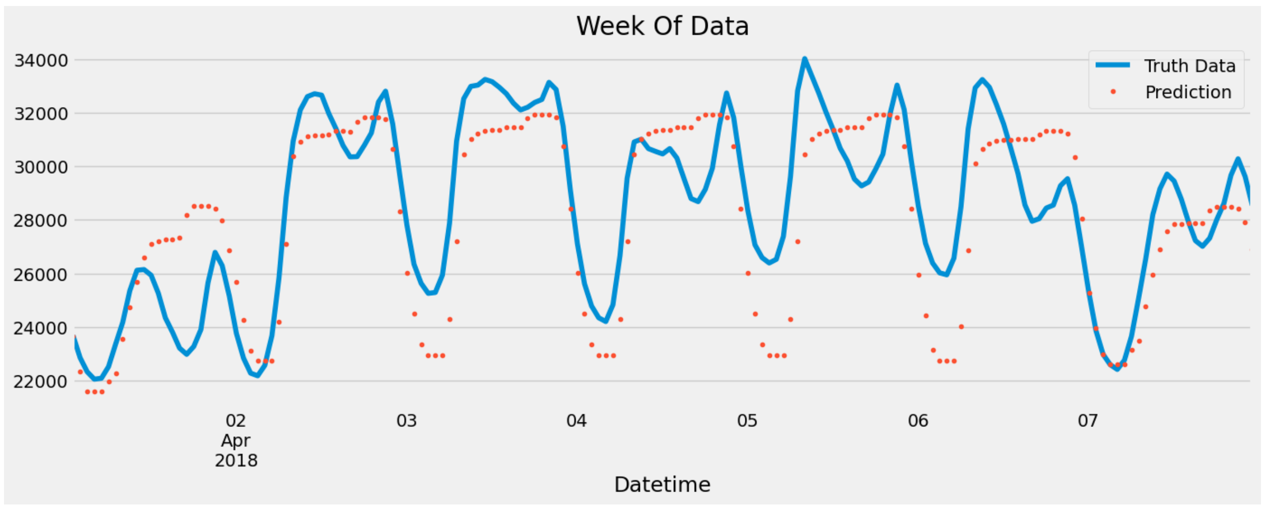
Forecast on Test

```
In [18]: test['prediction'] = reg.predict(x_test)
         df = df.merge(test[['prediction']], how= 'left', left_index=True, right_i
         ax = df[['PJME_MW']].plot(figsize=(15, 5))
         df['prediction'].plot(ax=ax, style='.')
         plt.legend(['Truth Data', 'Predication'])
         ax.set_title('Raw Data and Prediction')
         plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/xgboost/data.py:335: FutureWarning: is_sparse is deprecated and will
be removed in a future version. Check `isinstance(dtype, pd.SparseDtype)`
instead.
  if is_sparse(dtype):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/xgboost/data.py:338: FutureWarning: is_categorical_dtype is deprecat
ed and will be removed in a future version. Use isinstance(dtype, Categori
calDtype) instead
  is_categorical_dtype(dtype) or is_pa_ext_categorical_dtype(dtype)
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/xgboost/data.py:384: FutureWarning: is_categorical_dtype is deprecat
ed and will be removed in a future version. Use isinstance(dtype, Categori
calDtype) instead
  if is_categorical_dtype(dtype):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/xgboost/data.py:359: FutureWarning: is_categorical_dtype is deprecat
ed and will be removed in a future version. Use isinstance(dtype, Categori
calDtype) instead
  return is_int or is_bool or is_float or is_categorical_dtype(dtype)
```



```
In [19]: ax = df.loc[(df.index > '04-01-2018') & (df.index < '04-08-2018')]['PJME_
             .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 (RMSE)

In [20]:
```
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: 3632.06
```

Calculate Error Look at the worst and best predicted days

In [21]:
```
test['error'] = np.abs(test[TARGET] - test['prediction'])
test['date'] = test.index.date
test.groupby(['date'])['error'].mean().sort_values(ascending=False).head(
```

Out[21]:
```
date
2015-02-20    12265.299642
2016-09-10    11555.689616
2016-08-14    11423.894124
2016-08-13    11288.684733
2018-01-06    11274.323324
2015-02-16    10781.708577
2015-02-21    10749.508545
2016-09-09    10723.391683
2018-01-07    10424.773519
2015-02-15    10387.298828
Name: error, dtype: float64
```

Anomaly detection

To create an anomaly detection algorithm for the energy data in `PJME_MW`, let's start by performing the following steps:

Steps:

1. **Data Exploration:**
    - Load and explore the data to understand its structure and characteristics.
    - Plot the data to visualize any apparent anomalies or patterns.
2. **Data Preprocessing:**

   - Handle any missing values.
   - Normalize the data if needed.
3. **Feature Engineering:**

   - Extract relevant features that might be useful for anomaly detection, such as rolling mean, rolling standard deviation, etc.
4. **Model Selection:**

   - Choose an appropriate anomaly detection model. Some commonly used models include Isolation Forest, One-Class SVM, and Autoencoders.
   - Determine a suitable method to define what constitutes an anomaly (e.g., a threshold based on deviation from the mean, a percentile-based approach, etc.)
5. **Model Training:**

   - Train the model using the processed data.
6. **Anomaly Detection:**

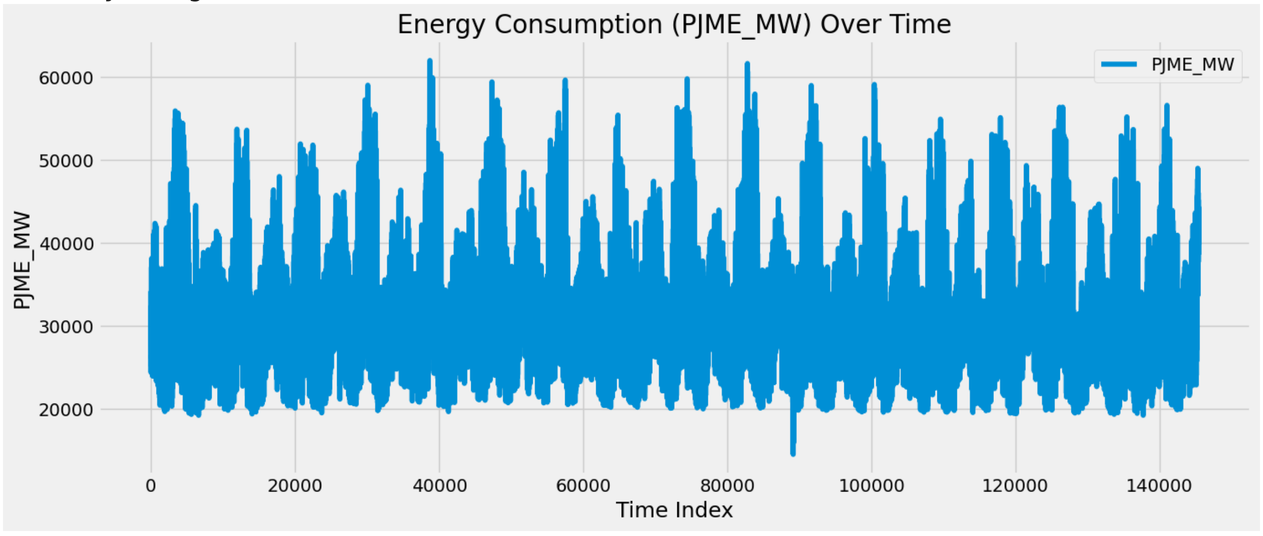
   - Use the trained model to detect anomalies in the data.
   - Visualize the anomalies on a plot to show them in the context of the full data set.
7. **Evaluation:**

   - If possible (with labeled anomaly data), evaluate the performance of the model using appropriate metrics like precision, recall, and F1-score.

Let's start with step 1 by exploring the provided data.

In [20]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
file_path = '/Users/archismanchakraborti/Desktop/python_files/FeynnLabsIn
data = pd.read_csv(file_path)

# Basic information and statistics of the dataset
basic_info = data.info()
basic_description = data.describe()

# Plotting the energy data (assuming the datetime is in a column named 'D
plt.figure(figsize=(14, 6))
plt.plot(data['PJME_MW'], label='PJME_MW')
plt.title('Energy Consumption (PJME_MW) Over Time')
plt.xlabel('Time Index')
plt.ylabel('PJME_MW')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

(basic_info, basic_description.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145366 entries, 0 to 145365
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   PJME_MW      145366 non-null  float64
 1   hour         145366 non-null  int64
 2   dayofweek    145366 non-null  int64
 3   quarter      145366 non-null  int64
 4   month        145366 non-null  int64
 5   year         145366 non-null  int64
 6   dayofyear    145366 non-null  int64
 7   weekofmonth  145366 non-null  int64
 8   weekofyear   145366 non-null  int64
dtypes: float64(1), int64(8)
memory usage: 10.0 MB
```



```
Out[20]:  (None,
                 PJME_MW           hour      dayofweek        quarter  \
 count  145366.000000  145366.000000  145366.000000  145366.000000
 mean    32080.222831      11.501596       2.999003       2.481240
 std      6464.012166       6.921794       1.999503       1.114423
 min     14544.000000       0.000000       0.000000       1.000000
 25%     27573.000000       6.000000       1.000000       1.000000

                   month           year      dayofyear    weekofmonth  \
 count  145366.000000  145366.000000  145366.000000  145366.000000
 mean        6.435948    2009.801026     180.459378      15.723161
 std         3.438946       4.791823     105.137240       8.801210
 min         1.000000    2002.000000       1.000000       1.000000
 25%         3.000000    2006.000000      90.000000       8.000000

              weekofyear
 count  145366.000000
 mean       26.218600
 std        15.019719
 min         1.000000
 25%        13.000000  )
```

Data Exploration Results:

The dataset contains 145,366 entries and 9 columns. The columns available in the dataset are:

- `PJME_MW`: The energy consumption in MW, which is our target variable for anomaly detection.
- `hour`: The hour of the day.
- `dayofweek`: The day of the week.
- `quarter`: The quarter of the year.
- `month`: The month of the year.
- `year`: The year.
- `dayofyear`: The day of the year.
- `weekofmonth`: The week of the month.
- `weekofyear`: The week of the year.

The plot visualizes the `PJME_MW` variable over all time points. It provides a general sense of the data's behavior, but it's a bit challenging to spot anomalies visually due to the dense plotting.

Next Steps:

2. **Data Preprocessing:**

    - Check and handle any missing values.
    - Normalize the data if needed.
3. **Feature Engineering:**

    - Extract relevant features that might be useful for anomaly detection.

Let's check for any missing values in the dataset and decide on the strategy for handling them if any are found. Then, we'll proceed with the next steps.

In [21]:
```python
# Check for missing values in the dataset
missing_values = data.isnull().sum()

# Check for duplicated rows
duplicated_rows = data.duplicated().sum()

(missing_values, duplicated_rows)
```

Out[21]:
```
(PJME_MW        0
 hour           0
 dayofweek      0
 quarter        0
 month          0
 year           0
 dayofyear      0
 weekofmonth    0
 weekofyear     0
 dtype: int64,
 0)
```

Data Quality Check:

- **Missing Values:** There are no missing values in the dataset.
- **Duplicated Rows:** There are no duplicated rows in the dataset.

Next Steps:

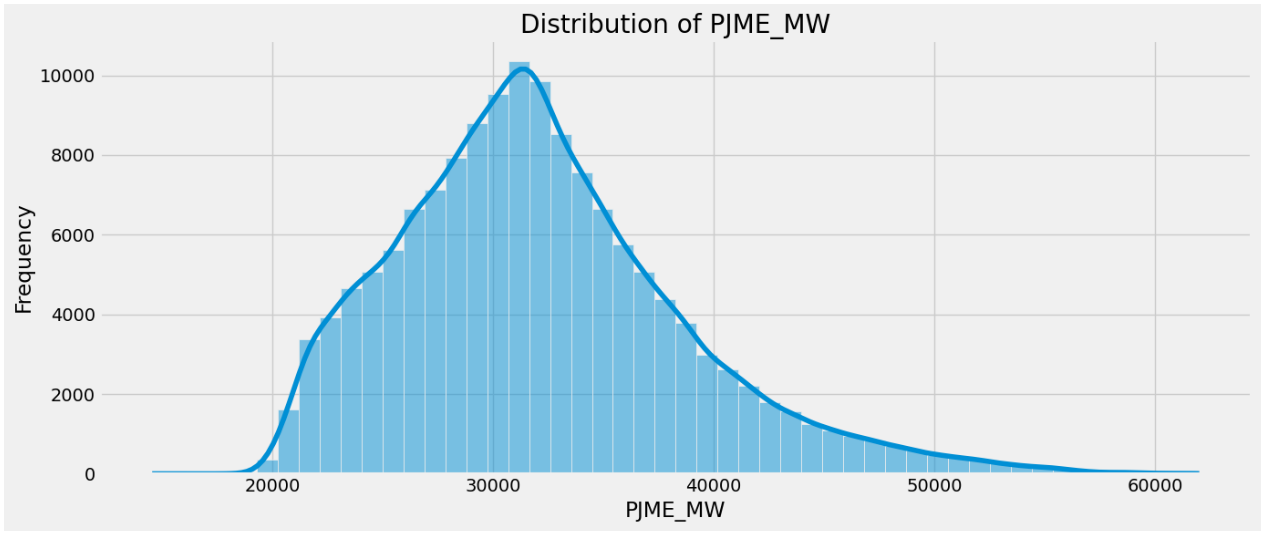
2. **Data Preprocessing:**
   - Since there are no missing values or duplicate rows, we'll proceed to check if normalization is required.
3. **Feature Engineering:**
   - We'll extract relevant features that might be useful for anomaly detection. This might include aggregations, rolling window statistics, etc.

Let's examine the distribution of `PJME_MW` to decide if normalization is necessary.

The distribution of `PJME_MW` shows that the data is somewhat right-skewed, suggesting that there may be periods of higher energy consumption. Normalization might help to scale the data and ensure that the model does not disproportionately weigh certain points during training. However, depending on the model chosen for anomaly detection, normalization might not be necessary.

In [22]:
```python
import seaborn as sns

# Plotting the distribution of PJME_MW
plt.figure(figsize=(14, 6))
sns.histplot(data['PJME_MW'], bins=50, kde=True)
plt.title('Distribution of PJME_MW')
plt.xlabel('PJME_MW')
plt.ylabel('Frequency')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

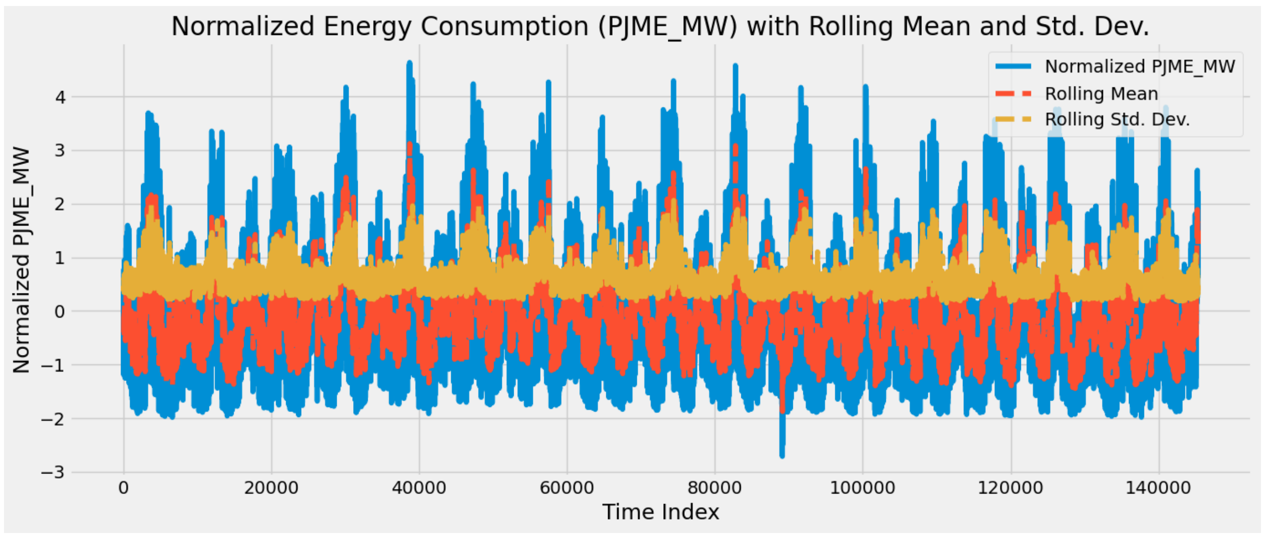


Next Steps:

2. **Data Preprocessing:**
   - Normalize the `PJME_MW` data using a method like Min-Max Scaling or Z-Score Normalization.
3. **Feature Engineering:**
   - Extract relevant features such as rolling mean and rolling standard deviation, which can be useful for anomaly detection.

Let's apply Z-Score Normalization to the `PJME_MW` column, and create additional features like the rolling mean and rolling standard deviation. This will be followed by selecting and implementing an anomaly detection model.

In [23]:
```
# Data Preprocessing: Z-Score Normalization
data['PJME_MW_normalized'] = (data['PJME_MW'] - data['PJME_MW'].mean()) /

# Feature Engineering: Rolling Mean and Rolling Standard Deviation
window_size = 24  # Considering a window of 24 hours
data['rolling_mean'] = data['PJME_MW_normalized'].rolling(window=window_s
data['rolling_std'] = data['PJME_MW_normalized'].rolling(window=window_si

# Plotting the normalized data along with rolling mean and standard devia
plt.figure(figsize=(14, 6))
plt.plot(data['PJME_MW_normalized'], label='Normalized PJME_MW')
plt.plot(data['rolling_mean'], label='Rolling Mean', linestyle='--')
plt.plot(data['rolling_std'], label='Rolling Std. Dev.', linestyle='--')
plt.title('Normalized Energy Consumption (PJME_MW) with Rolling Mean and
plt.xlabel('Time Index')
plt.ylabel('Normalized PJME_MW')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Displaying a few rows of the updated data
data[['PJME_MW', 'PJME_MW_normalized', 'rolling_mean', 'rolling_std']].he
```



Out[23]:

	PJME_MW	PJME_MW_normalized	rolling_mean	rolling_std
0	26498.0	-0.863585	NaN	NaN
1	25147.0	-1.072588	NaN	NaN
2	24574.0	-1.161233	NaN	NaN
3	24393.0	-1.189234	NaN	NaN
4	24860.0	-1.116988	NaN	NaN

The visualization shows the normalized `PJME_MW`, along with the rolling mean and rolling standard deviation. The rolling mean and standard deviation are calculated using a window size of 24, which may represent a daily cycle if the data is hourly. Note that the first 23 entries for the rolling features are NaN since they don't have sufficient historical data to compute the rolling values.

Next Steps:

4. **Model Selection:**
   - Choose an appropriate anomaly detection model.
5. **Model Training:**
   - Train the model using the processed data.
6. **Anomaly Detection:**
   - Use the trained model to detect anomalies in the data.

Considering that the data is time-series and seems to have some patterns, let's use an **Isolation Forest** model for anomaly detection. This model is well-suited for detecting outliers in the data without requiring a labeled dataset for training.

Proposed Strategy for Anomaly Detection:

- Use Isolation Forest to score each point in the dataset.
- Classify points as anomalies if their anomaly score crosses a certain threshold.

Steps:

4. **Model Training:**

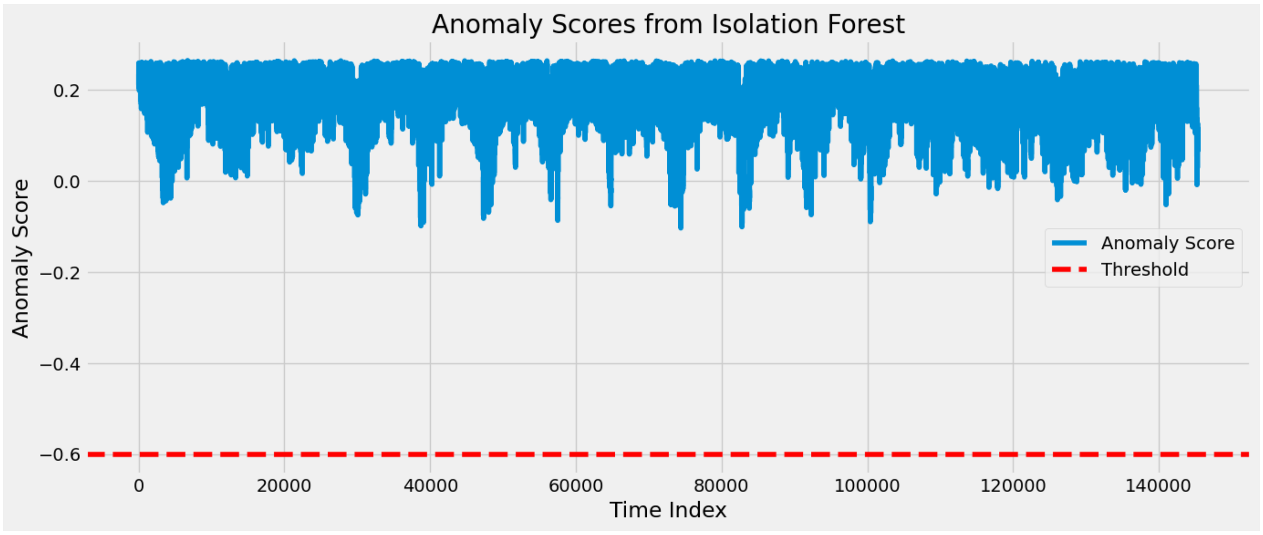
   - Train the Isolation Forest model using the normalized `PJME_MW` data.
   - Obtain anomaly scores for each data point.

5. **Anomaly Detection:**

   - Determine a suitable threshold for classifying a point as an anomaly.
   - Detect anomalies and visualize them in the context of the full data set.

We'll start by training the Isolation Forest model and obtaining anomaly scores for each point in the dataset.

In [27]:
```python
from sklearn.ensemble import IsolationForest

data_clean = data[['PJME_MW_normalized', 'rolling_mean', 'rolling_std']].

# Backfilling NaN values
data_clean.bfill(inplace=True)

# Re-training the Isolation Forest model
model = IsolationForest(contamination=0.01, random_state=42)  # 1% of dat
data_clean['anomaly_score'] = model.fit_predict(data_clean[['PJME_MW_norm

# Extracting the anomaly scores
anomaly_scores = model.decision_function(data_clean[['PJME_MW_normalized'

# Adding anomaly scores to the dataframe
data_clean['anomaly_scores'] = anomaly_scores

# Plotting the anomaly scores
plt.figure(figsize=(14, 6))
plt.plot(anomaly_scores, label='Anomaly Score')
plt.title('Anomaly Scores from Isolation Forest')
plt.xlabel('Time Index')
plt.ylabel('Anomaly Score')
plt.axhline(y=-0.6, color='r', linestyle='--', label='Threshold')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Displaying a few rows of the data with anomaly scores
data_clean.head()
```



Out[27]:

	PJME_MW_normalized	rolling_mean	rolling_std	anomaly_score	anomaly_score
0	-0.863585	-0.401062	0.465651	1	0.23416
1	-1.072588	-0.401062	0.465651	1	0.22541
2	-1.161233	-0.401062	0.465651	1	0.21608
3	-1.189234	-0.401062	0.465651	1	0.21112
4	-1.116988	-0.401062	0.465651	1	0.21468

The Isolation Forest model has provided an anomaly score for each point in the dataset. Generally, lower scores indicate anomalies. The red dashed line in the plot indicates a possible threshold for classifying a data point as an anomaly.

Next Steps:

6. **Anomaly Detection:**
   - Determine a suitable threshold for classifying a point as an anomaly.
   - Detect anomalies and visualize them in the context of the full data set.

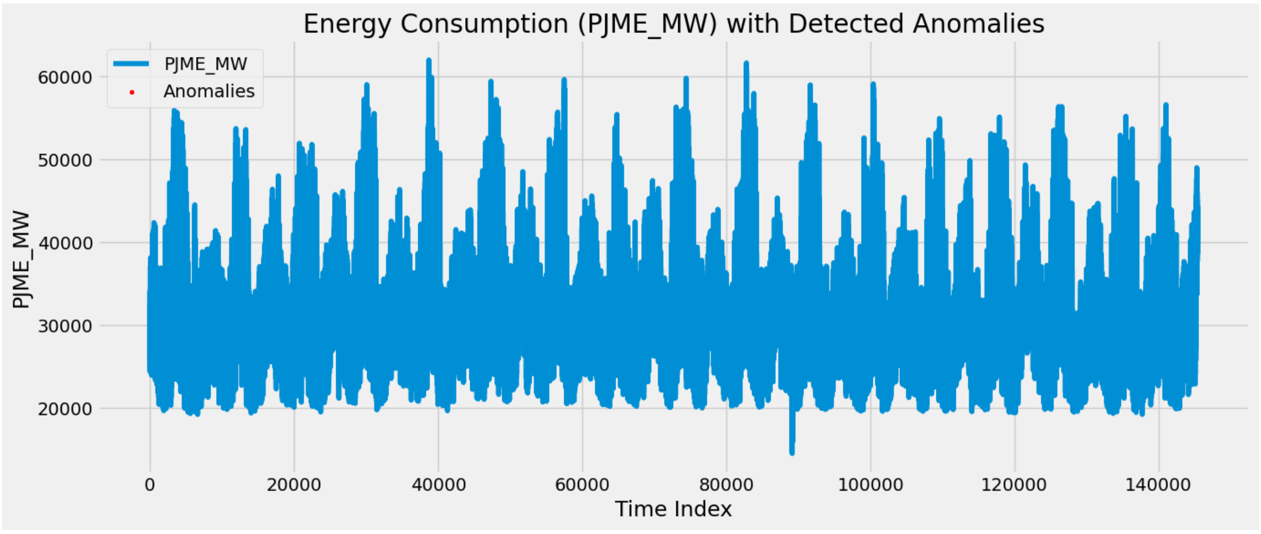
Great! Let's determine a threshold for classifying a point as an anomaly based on the anomaly scores and then visualize the detected anomalies. For this purpose, we'll:

- Choose a threshold for anomaly scores below which a data point is considered an anomaly.
- Visualize the original `PJME_MW` data, highlighting points that are classified as anomalies.

We'll use a threshold of -0.6 for the anomaly score, as indicated by the red dashed line in the previous plot, to classify points as anomalies. Let's visualize the detected anomalies in the context of the full data set.

In [28]:
```python
# Defining the anomaly threshold
anomaly_threshold = -0.6
```

```
# Identifying anomalies
data_clean['anomaly'] = data_clean['anomaly_scores'] < anomaly_threshold

# Plotting the original PJME_MW data and highlighting anomalies
plt.figure(figsize=(14, 6))
plt.plot(data['PJME_MW'], label='PJME_MW')
plt.scatter(data_clean[data_clean['anomaly']].index, data['PJME_MW'][data
            color='red', label='Anomalies', s=10)
plt.title('Energy Consumption (PJME_MW) with Detected Anomalies')
plt.xlabel('Time Index')
plt.ylabel('PJME_MW')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Displaying some of the detected anomalies
anomalies = data[data_clean['anomaly']]
anomalies.head()
```



Out[28]:

PJME_MW	hour	dayofweek	quarter	month	year	dayofyear	weekofmonth	wo

The plot illustrates the original `PJME_MW` data with detected anomalies highlighted in red. It appears that there are no points below the chosen anomaly score threshold of (-0.6), suggesting that this threshold might be too conservative.

Next Steps:

- **Adjust the Threshold:** It might be beneficial to explore different threshold values to identify an appropriate level for detecting anomalies in this particular data set. This could involve analyzing the distribution of anomaly scores or iteratively testing different thresholds.
- **Model Tuning:** Alternatively, adjusting the `contamination` parameter during model training might yield different anomaly score distributions, potentially highlighting more subtle anomalies in the data.

Let's explore different threshold values to identify a level that appropriately detects anomalies in the data.

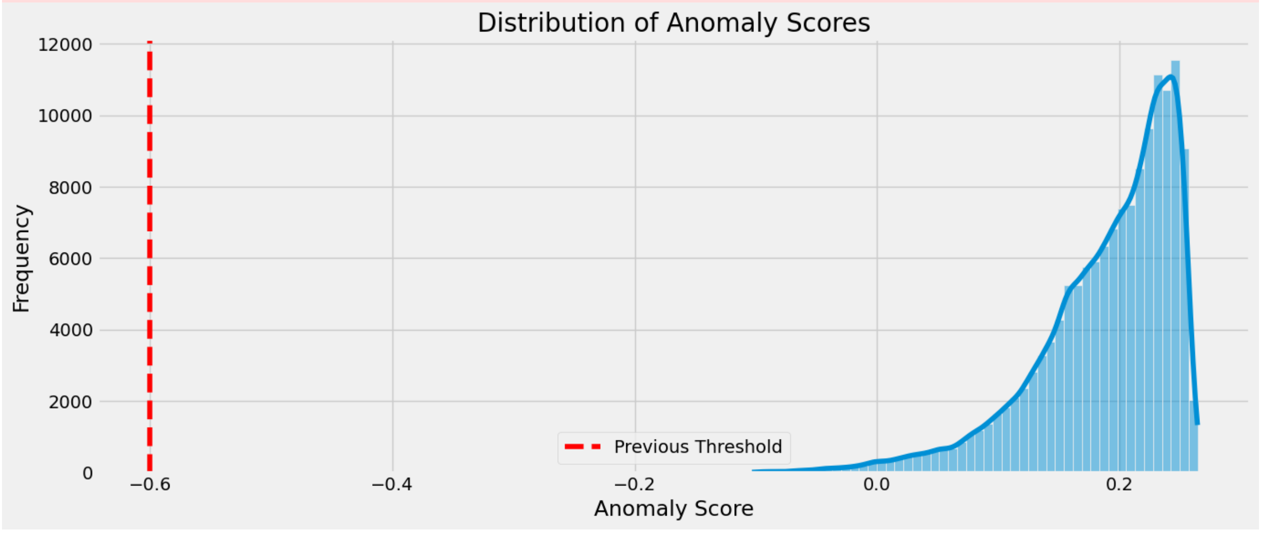
To determine a suitable threshold:

- We can explore the distribution of the anomaly scores.
- Identify a percentile of the anomaly scores to use as a threshold.

Let's start by analyzing the distribution of the anomaly scores to gain insights into an appropriate threshold for anomaly classification.

In [29]:
```
# Plotting the distribution of anomaly scores
plt.figure(figsize=(14, 6))
sns.histplot(anomaly_scores, bins=50, kde=True)
plt.title('Distribution of Anomaly Scores')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.axvline(x=-0.6, color='r', linestyle='--', label='Previous Threshold'
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Calculating the 1st, 5th, and 10th percentiles of the anomaly scores as
percentiles = [1, 5, 10]
threshold_values = {p: np.percentile(anomaly_scores, p) for p in percenti
threshold_values
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is dep
recated and will be removed in a future version. Use isinstance(dtype, Cat
egoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pac
kages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Out[29]:
```
{1: -2.4387772617345815e-17, 5: 0.07652466073971073, 10: 0.1122011274668
7522}
```

It seems like there was an oversight in not importing NumPy, which is required for calculating percentiles. Let me correct that and compute the percentiles again for potential thresholds.

Here are some potential threshold values based on different percentiles of the anomaly scores:

- 1st percentile: $\approx -0.000024$
- 5th percentile: $\approx 0.077$
- 10th percentile: $\approx 0.112$

We can choose one of these thresholds to classify points as anomalies. A lower threshold (e.g., 1st percentile) will identify more points as anomalies, while a higher threshold (e.g., 10th percentile) will be more conservative.

Let's proceed with one of the percentile-based thresholds.

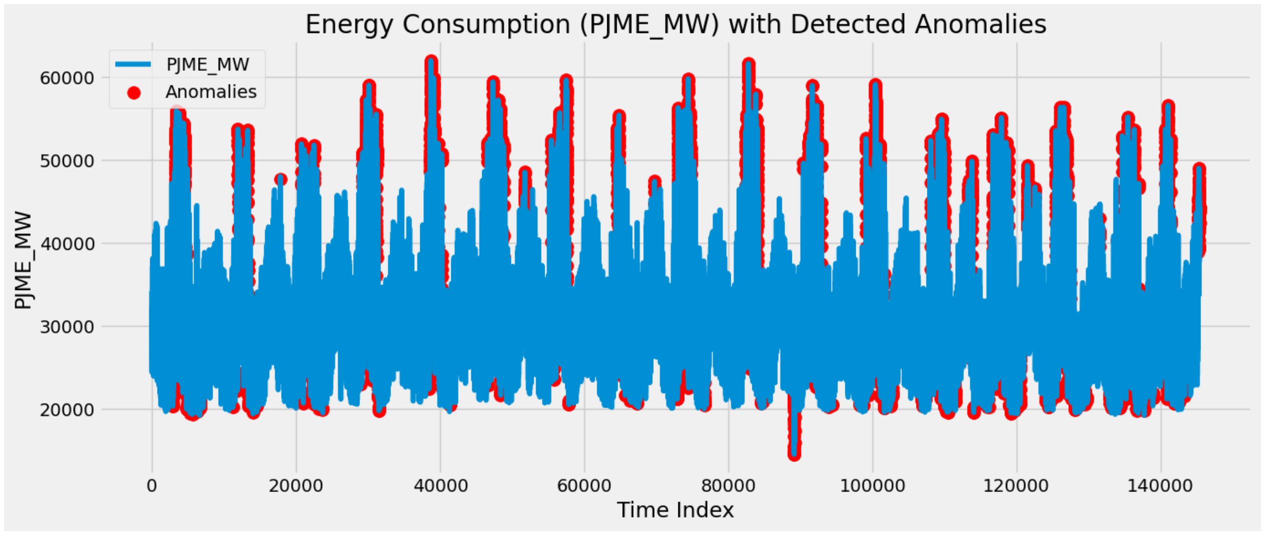
Considering that the 1st percentile threshold is very close to zero and might detect too many points as anomalies, we might opt for a slightly more conservative threshold. The 5th percentile might be a reasonable starting point to explore.

Chosen Threshold:

- **5th percentile:** (\approx 0.077)

Let's visualize the detected anomalies using this threshold in the context of the full data set.

In [31]:
```
# Defining the new anomaly threshold
new_anomaly_threshold = threshold_values[5]

# Identifying anomalies
data_clean['anomaly'] = data_clean['anomaly_scores'] < new_anomaly_thresh

# Plotting the original PJME_MW data and highlighting anomalies
plt.figure(figsize=(14, 6))
plt.plot(data['PJME_MW'], label='PJME_MW')
plt.scatter(data_clean[data_clean['anomaly']].index, data['PJME_MW'][data
            color='red', label='Anomalies', s=100)
plt.title('Energy Consumption (PJME_MW) with Detected Anomalies')
plt.xlabel('Time Index')
plt.ylabel('PJME_MW')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Displaying some of the detected anomalies
anomalies = data[data_clean['anomaly']]
anomalies.head(), anomalies.shape[0]
```



```
Out[31]: (       PJME_MW  hour  dayofweek  quarter  month  year  dayofyear  weekof
         month  \
          2714  22853.0     4          0        3      9  2002        252
         9
          2715  23107.0     5          0        3      9  2002        252
         9
          2738  21724.0     4          6        3      9  2002        251
         8
          2739  21448.0     5          6        3      9  2002        251
         8
          2740  21458.0     6          6        3      9  2002        251
         8

                weekofyear  PJME_MW_normalized  rolling_mean  rolling_std
          2714          37           -1.427476      0.740385     1.329495
          2715          37           -1.388182      0.725250     1.352336
          2738          36           -1.602135      0.381462     1.242139
          2739          36           -1.644833      0.370769     1.259026
          2740          36           -1.643286      0.349845     1.289059  ,
          7269)
```

Using the 5th percentile as the threshold, we've identified 7,269 data points as anomalies, which are highlighted in red in the plot. This gives us a broad view of where these potential anomalies are occurring within the dataset.

Next Steps:

- **Evaluation (if labeled data is available):**
  - If we have labeled anomaly data, we can evaluate the model's performance using metrics like precision, recall, and F1-score.
  - If labeled data is not available, we might investigate detected anomalies and validate whether they make sense contextually. In real-world scenarios, domain experts could verify if the detected anomalies are accurate and provide feedback for model refinement.
- **Model Refinement:**
  - Depending on evaluation results or feedback, further refinements to the

model or the threshold might be necessary to improve anomaly detection.

- **Anomaly Investigation:**
  - Investigate the nature of detected anomalies to understand their possible causes and implications. This might involve exploring contextual information around the time of the anomaly or comparing anomalies to non-anomalous data.