**MEASURE ENERGY CONSUMPTION**

**USING PYTHON**

**TEAM MEMBER**

**AU411521104032-K.ELAKIYA**

**Project little: MEASURE ENERGY CONSUMPTION**

**Phase 5: Project Documentation & Submission**

**Topic: In this section we will document the complete project and prepare it for submission.**

****

**Measure Energy Consumption**

**INTRODUCTION:**

* Energy consumption is the total amount of energy used by a person, household, business, or country over a period of time. It is measured in kilowatt-hours (kWh) or joules (J). Energy can be consumed in many different forms, including electricity, natural gas, gasoline, oil, and coal.
* Measuring energy consumption is important for several reasons. First, it helps us to understand how we are using energy and where we can make changes to save energy. Second, it helps us to track our progress towards energy efficiency goals. Third, it helps us to make informed decisions about energy policy.
* There are a number of different ways to measure energy consumption. The most common method is to use a meter. Meters are installed at the point where energy enters a building or facility. They measure the amount of energy that flows through the meter and record the data.
* Another way to measure energy consumption is to estimate it. This can be done by using historical data or by collecting data on specific energy-consuming activities. For example, a homeowner could estimate their electricity consumption by looking at their past electricity bills.

**Benefits of measuring energy consumption:**

There are a number of benefits to measuring energy consumption, including:

* Reduced energy costs: By understanding how and where you are using energy, you can make changes to save energy and reduce your energy costs.
* Reduced environmental impact: Reducing your energy consumption helps to reduce greenhouse gas emissions and other environmental impacts.
* Improved energy efficiency: Measuring your energy consumption over time can help you to track your progress towards energy efficiency goals.
* Informed energy decisions: Having accurate information about your energy consumption can help you to make informed decisions about energy policy and other energy-related issues.

**How to measure energy consumption:**

There are a number of different ways to measure energy consumption, depending on the type of energy being used. For example, electricity consumption is measured in kilowatt-hours (kWh) using a kilowatt-hour meter. Natural gas consumption is measured in cubic meters (m³) using a gas meter. Gasoline consumption is measured in gallons (gal) or liters (L) using a fuel gauge.

To measure the energy consumption of a specific appliance or device, you can use a power meter. Power meters measure the wattage of an appliance or device, which can then be converted to kWh using the following formula:

kWh = wattage / 1000

For example, if a lightbulb has a wattage of 60, then it will consume 0.06 kWh per hour of use.

To measure the total energy consumption of a building or facility, you can use a smart meter. Smart meters are advanced meters that record energy consumption data in real time. This data can then be used to track energy consumption over time and identify areas where energy can be saved.

Measuring energy consumption in Python can be done in a number of ways, depending on the specific needs of the project. One common approach is to use a hardware sensor, such as a power meter or smart plug, to measure the power consumption of a device or appliance. This data can then be read and processed using Python to calculate the total energy consumption over a period of time.Another approach to measuring energy consumption in Python is to use a software tool, such as the Intel Running Average Power Limit (RAPL) interface. RAPL is a hardware feature that provides real-time power consumption measurements for Intel CPUs. Python libraries such as CodeCarbon can be used to access RAPL data and calculate the energy consumption of a Python program or script.

Measure energy consumption is an important step in reducing energy waste and improving energy efficiency. Python can be used to measure energy consumption in a variety of ways, including:

* Using the Intel "Running Average Power Limit" (RAPL) technology to estimate the power consumption of a CPU.
* Using the pyJoules library to measure the energy consumption of a host machine along the execution of a piece of Python code.

Once the energy consumption data has been collected, it can be pre-processed using Python to clean, transform, and prepare the data for analysis. This may involve handling missing values, converting data types, resampling the data, creating new features, removing outliers, and splitting the data into training and testing sets.Once the data is pre-processed, it can be used for machine learning or other analysis tasks. For example, machine learning models can be trained to predict energy consumption based on historical data and other factors. This information can then be used to develop strategies for reducing energy consumption.

**DATASET LINK**

<https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption>

**DEPLOYMENT** **AND** **PREDICTION**

To measure energy consumption, deployment, and prediction in Python, we can use the following steps:

1. Measure energy consumption:

To measure energy consumption, we can use a variety of methods, depending on the specific application. For example, we can use smart meters to track the energy consumption of individual appliances or buildings, or we can use aggregate energy consumption data from power grids.Once we have collected energy consumption data, we can use Python to analyze it and identify patterns and trends. For example, we can use Python to create time series plots of energy consumption data, or to calculate the average and peak energy consumption for different periods of time.

1. Deploy energy consumption models:

Once we have analyzed energy consumption data, we can use it to train machine learning models to predict future energy consumption. There are a variety of machine learning algorithms that can be used for this purpose, such as linear regression, support vector machines, and random forests.Once we have trained an energy consumption model, we can deploy it in Python to make predictions about future energy consumption. For example, we can deploy a model to a web server, or we can embed it in a mobile app.

1. Predict energy consumption:

Once we have deployed an energy consumption model, we can use it to predict future energy consumption. To do this, we simply provide the model with the necessary input features, such as weather data, historical energy consumption data, and other relevant factors.The model will then output a prediction of the future energy consumption. We can use this prediction to make informed decisions about energy management, such as how to allocate energy resources or how to reduce energy consumption.

Here is a simple Python example of how to measure energy consumption, deploy an energy consumption model, and predict energy consumption:

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Load the energy consumption data

energy\_consumption\_data = pd.read\_csv('energy\_consumption.csv')

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(energy\_consumption\_data.drop('energy\_consumption', axis=1),

energy\_consumption\_data['energy\_consumption'], test\_size=0.25)

# Train a linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Deploy the model to a web server

# Predict the energy consumption for the next hour

new\_features = np.array([[100, 20, 80]])

predicted\_energy\_consumption = model.predict(new\_features)

# Print the predicted energy consumption

print('Predicted energy consumption:', predicted\_energy\_consumption)

This is just a simple example, and there are many other ways to measure energy consumption, deploy energy consumption models, and predict energy consumption in Python. The specific approach that you use will depend on your specific needs and requirements.

**MODEL** **INTERPREABILITY**

There are a few different ways to measure the interpretability of energy consumption models in Python. One common approach is to use permutation importance. This method involves randomly shuffling the values of each input variable and then measuring how much this affects the model's predictions. The more important a variable is to the model, the more its predictions will change when its values are shuffled.

Another approach to measuring interpretability is to use SHAP values. SHAP values explain the impact of each input variable on a specific prediction. They are calculated by recursively comparing the model's prediction for a given input vector to the predictions it would make if that input vector was replaced with random values.

Both permutation importance and SHAP values can be calculated using the following Python code:

Python

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

from shap import TreeExplainer

# Load the data

data = pd.read\_csv('energy\_consumption.csv')

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.drop(columns=['energy\_consumption']), data['energy\_consumption'], test\_size=0.25)

# Create a random forest model

model = RandomForestRegressor()

model.fit(X\_train, y\_train)

# Calculate permutation importance

permutation\_importance = np.mean([np.abs(model.predict(X\_test[np.random.permutation(i)]).mean() - model.predict(X\_test).mean()) for i in range(X\_test.shape[1])])

# Calculate SHAP values

explainer = TreeExplainer(model)

shap\_values = explainer.shap\_values(X\_test)

# Print the results

print('Permutation importance:', permutation\_importance)

print('SHAP values:', shap\_values)

The output of this code will be a list of permutation importance scores and a list of SHAP values for each input variable. The higher the permutation importance score or SHAP value for a variable, the more important it is to the model.

In addition to permutation importance and SHAP values, there are a number of other methods that can be used to measure the interpretability of energy consumption models. These methods include:

* Partial dependence plots: Partial dependence plots show the effect of each input variable on the model's predictions, while holding all other variables constant.
* Feature selection: Feature selection methods can be used to identify the most important input variables for the model.
* Decision trees: Decision trees are inherently interpretable, as they can be represented as a set of rules that explain how the model makes its predictions.

The best approach for measuring the interpretability of an energy consumption model will depend on the specific model and the intended use of the model. However, the methods described above provide a good starting point.

**MODEL** **EVALUATION**

To measure energy consumption model evaluation in Python, you can use the following steps:

1. Define your evaluation metrics. Common metrics for energy consumption model evaluation include:
   * Mean absolute error (MAE): The average of the absolute differences between the predicted and actual energy consumption values.
   * Mean squared error (MSE): The average of the squared differences between the predicted and actual energy consumption values.
   * Root mean squared error (RMSE): The square root of the MSE.
   * Coefficient of determination (R2): A measure of how well the model explains the variation in the actual energy consumption values.
2. Calculate the evaluation metrics on a held-out test set. This is a set of data that was not used to train the model. This will help you to assess how well the model generalizes to unseen data.
3. Compare the evaluation metrics to a baseline model. This could be a simple model, such as the mean of the target variable, or a more complex model that is not specifically designed to predict energy consumption.
4. Interpret the results If the evaluation metrics for your model are better than those for the baseline model, then you can conclude that your model is effective at predicting energy consumption. However, it is important to note that no model is perfect, and there will always be some error between the predicted and actual values.

Here is a simple example of how to measure energy consumption model evaluation in Python:

Python

import numpy as np

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

# Load the training and test data

X\_train, y\_train = ...

X\_test, y\_test = ...

# Train the energy consumption model

model = ...

model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = model.predict(X\_test)

# Calculate the evaluation metrics

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = model.score(X\_test, y\_test)

# Print the evaluation metrics

print('MAE:', mae)

print('MSE:', mse)

print('RMSE:', rmse)

print('R2:', r2)

You can then compare the evaluation metrics for your model to those for a baseline model to assess how well your model generalizes to unseen data.

**INNOVATION**

* Using machine learning to predict energy consumption: Machine learning can be used to train models on historical energy consumption data to predict future consumption. This can help businesses and organizations to better manage their energy use and reduce costs.
* Developing new sensors and devices to measure energy consumption: New sensors and devices are being developed that can measure energy consumption more accurately and efficiently. These sensors can be used to monitor energy consumption in real time and identify areas where energy savings can be made.
* Developing software tools to visualize and analyze energy consumption data: Software tools are being developed to help businesses and organizations to visualize and analyze their energy consumption data. This can help them to identify trends, patterns, and anomalies in their energy use.
* PowerAPI: PowerAPI is a Python library for measuring power consumption on a variety of devices, including laptops, desktops, servers, and IoT devices. It can be used to measure power consumption in real time, or to collect data over time to analyze trends.
* Energy Dashboard: Energy Dashboard is a web-based application for visualizing and analyzing energy consumption data. It supports a variety of data sources, including PowerAPI, and provides a variety of charts and reports to help businesses and organizations understand their energy use.
* WattTime: WattTime is a Python library that provides real-time information on carbon emissions associated with electricity generation. It can be used to develop applications that help businesses and organizations to reduce their carbon footprint by choosing to use renewable energy when it is available.

Overall, Python is a versatile and powerful language for developing innovative solutions to measure energy consumption. As the technology continues to develop, we can expect to see even more innovative Python-based solutions emerge in this area.

In addition to the above, here are some other ways to innovate in measuring energy consumption using Python:

* Use Python to develop new algorithms for measuring energy consumption more accurately and efficiently. For example, you could develop an algorithm to measure energy consumption at the individual device level, rather than the aggregate level.
* Use Python to develop new applications that help businesses and organizations to reduce their energy consumption. For example, you could develop an application that recommends energy-saving measures to businesses, or an application that helps businesses to track their progress in reducing their energy consumption.

**ABSTRACT :**

Measuring energy consumption is essential for designing and operating energy-efficient software systems. Java is a widely used programming language, and there is a growing need to develop efficient Java applications. This abstract presents an overview of the state-of-the-art in measuring energy consumption using Java.There are a number of different hardware-based and software-based tools available for measuring energy consumption in Java. Some popular hardware-based tools include the WattsUp? Pro and the Monsoon Power Monitor. Some popular software-based tools include the Java Virtual Machine (JVM) Profiler and the Intel Power Gadget.

**MODULE:**

1. **Choose** **a** **measurement** **approach**:

As discussed in the abstract, there are two main approaches to measuring energy consumption in Java: hardware-based and software-based. Hardware-based approaches are more accurate, but they are also more expensive and time-consuming to use. Software-based approaches are less accurate, but they are more convenient and less expensive to use.

**2**.**Select a measurement tool:**

Once you have chosen a measurement approach, you need to select a measurement tool. There are a number of different hardware-based and software-based tools available. Some popular hardware-based tools include the WattsUp? Pro and the Monsoon Power Monitor. Some popular software-based tools include the Java Virtual Machine (JVM) Profiler and the Intel Power Gadget.

**3.Implement the measurement module:**

Once you have selected a measurement tool, you need to implement the measurement module. The specific implementation will depend on the measurement tool you have chosen.

However, there are some general steps that you can follow:

* + Initialize the measurement tool.
  + Start the measurement.
  + Run the Java application.
  + Stop the measurement.
  + Read the measurement results.

**3.Interpret the measurement results:**

Once you have read the measurement results, you need to interpret them. This may involve calculating the total energy consumption, the energy consumption of individual components of the Java application, or the energy consumption per unit of time.

Here is an example of a simple Java module for measuring energy consumption using the JVM Profiler:

Java

public class EnergyConsumptionMonitor {

private final JVMPIProfiler profiler

public EnergyConsumptionMonitor() {

profiler = new JVMPIProfiler();

}

public double measureEnergyConsumption() throws JVMPIException {

profiler.start();

// Run the Java application.

profiler.stop();

return profiler.getEnergyConsumption();

}

}

To use this module, you would first create an instance of the EnergyConsumptionMonitor class. Then, you would call the measureEnergyConsumption() method to start the measurement. After running the Java application, you would call the measureEnergyConsumption() method again to stop the measurement and get the results.

This is just a simple example, of course. A more sophisticated module might measure the energy consumption of individual components of the Java application, or the energy consumption per unit of time. It might also provide additional features, such as the ability to log the measurement results or to export them to a file.

* Measuring energy consumption in Python can be done in a number of ways, depending on the specific needs of the project.
* One common approach is to use a hardware sensor, such as a power meter or smart plug, to measure the power consumption of a device or appliance.
* This data can then be read and processed using Python to calculate the total energy consumption over a period of time.
* Another approach to measuring energy consumption in Python is to use a software tool, such as the Intel Running Average Power Limit (RAPL) interface.
* RAPL is a hardware feature that provides real-time power consumption measurements for Intel CPUs.
* Python libraries such as CodeCarbon can be used to access RAPL data and calculate the energy consumption of a Python program or script.
* Measure energy consumption is an important step in reducing energy waste and improving energy efficiency. Python can be used to measure energy consumption in a variety of ways, including:
* Using the Intel "Running Average Power Limit" (RAPL) technology to estimate the power consumption of a CPU. Using the pyJoules library to measure the energy consumption of a host machine along the execution of a piece of Python code.
* Once the energy consumption data has been collected, it can be pre-processed using Python to clean, transform, and prepare the data for analysis.
* This may involve handling missing values, converting data types, resampling the data, creating new features, removing outliers, and splitting the data into training and testing sets.
* Once the data is pre-processed, it can be used for machine learning or other analysis tasks. For example, machine learning models can be trained to predict energy consumption based on historical data and other factors.

**LOAD** **THE** **DATASET**

To load the dataset in measure energy consumption, we can use the following Python code:

import pandas as pd

# Load the dataset from a CSV file

energy\_consumption\_df = pd.read\_csv('energy\_consumption.csv')

This will load the dataset into a Pandas DataFrame, which is a convenient data structure for working with tabular data.

Once the dataset is loaded, we can explore it using various Pandas functions. For example, we can print the first few rows of the DataFrame to get a sense of the data:

Python

print(energy\_consumption\_df.head())

**EXPLORATORY DATA ANALYSIS**

Exploratory Data Analysis (EDA) is a process of investigating the data to discover patterns, trends, and relationships. EDA is an important step in any data science project, as it helps us to understand the data better and to identify potential problems.

Here are some steps involved in performing EDA on energy consumption data using Python:

* **Import the necessary libraries.**

Python

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

Use code with caution.

content\_copy

* **Load the dataset.**

Python

energy\_consumption\_df = pd.read\_csv('energy\_consumption.csv')

Use code with caution.

content\_copy

* **Check for missing values and outliers.**

Python

# Check for missing values

print(energy\_consumption\_df.isna().sum())

# Check for outliers

print(energy\_consumption\_df['energy\_consumption'].describe())

Use code with caution.

content\_copy

* **Explore the data.**

Python

# Print the first few rows of the DataFrame

print(energy\_consumption\_df.head())

# Get the data types of the columns

print(energy\_consumption\_df.dtypes)

# Calculate summary statistics for the dataset

print(energy\_consumption\_df.describe())

* **Visualize the data.**

Python

# Create a line plot of the energy consumption over time

plt.plot(energy\_consumption\_df['timestamp'], energy\_consumption\_df['energy\_consumption'])

plt.xlabel('Timestamp')

plt.ylabel('Energy consumption (Wh)')

plt.title('Energy consumption over time')

plt.show()

# Create a histogram of the energy consumption

sns.histplot(energy\_consumption\_df['energy\_consumption'])

plt.xlabel('Energy consumption (Wh)')

plt.ylabel('Frequency')

plt.title('Histogram of energy consumption')

plt.show()

# Create a box plot of the energy consumption by device type

sns.boxplot(x = 'device', y = 'energy\_consumption', data=energy\_consumption\_df)

plt.xlabel('Device type')

plt.ylabel('Energy consumption (Wh)')

plt.title('Box plot of energy consumption by device type')

plt.show()

content\_copy

* **Identify patterns and trends.**

Once we have explored the data, we can start to identify patterns and trends. For example, we may notice that the energy consumption is higher during certain times of day or during certain seasons. We may also notice that certain devices consume more energy than others.

* **Formulate hypotheses.**

Based on the patterns and trends that we have identified, we can start to formulate hypotheses about the energy consumption data. For example, we may hypothesize that the energy consumption is higher during the summer because people are using m``` ore air conditioning.

* **Test the hypotheses.**

Once we have formulated hypotheses, we can test them using statistical methods. For example, we can use a t-test to test the hypothesis that the energy consumption is higher during the summer.

* **Interpret the results.**

Once we have tested our hypotheses, we need to interpret the results. If a hypothesis is rejected, then we need to find a new explanation for the data. If a hypothesis is not rejected, then we can accept it as a valid explanation for the data.

EDA is an iterative process. We may need to repeat some of the steps above multiple times in order to fully understand the data.

Here are some additional things to keep in mind when performing EDA on energy consumption data:

* **Consider the time period of the data.** If the data is only for a short period of time, then it may not be representative of the overall energy consumption.
* **Consider the location of the data.** If the data is from a specific location, then it may not be representative of the energy consumption in other locations.
* **Consider the types of devices that are included in the data.** If the data only includes certain types of devices, then it may not be representative of the energy consumption of all devices.

Once we have performed EDA on the energy consumption data, we can use it to develop strategies for reducing energy consumption.

**FOR PHASE-3\_PROJECT :**

In this phase I’ve designed an innovation to solve the problem.

**DATA SOURCE :**

The data source for measuring the energy consumption is obtained from the below dataset.

DATASET:<https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption>

**MODULE :**

When it comes to measuring energy consumption, there are a few key modules that are typically involved:

**1️.Sensors:**

These devices are responsible for collecting data on energy usage. They can include electricity meters, smart plugs, or other monitoring devices.

**2️.Data Acquisition:**

This module focuses on gathering and recording the energy consumption data from the sensors. It ensures that the data is accurately captured and stored for further analysis.

**3️.Data Processing:**

This module involves analyzing and Processing the collected data. It often uses programming Languages like Python and data analysis tools like pandas to Calculate energy consumption metrics and identify patterns or Trends.

**4️.Visualization:**

This module helps to present the energy Consumption data in a visual format. It can include charts, graphs, Or dashboards that make it easier to understand and interpret the Data.These modules work together to provide insights into energy Usage and help identify opportunities for energy efficiency Improvements.

**MODEL DEVELOPMENT:**

To develop a model for measuring energy consumption, you can

Follow these steps:

**1️. Collect data:**

Gather energy consumption data from sensors or Smart meters. This data should include variables like time, date, and energy usage.

**2️. Pre-process the data:**

Clean the data by handling missing values, outliers, and formatting issues. You may need to convert he data into a suitable format for analysis.

**3️. Feature engineering:**

Extract relevant features from the data that can help in predicting energy consumption. This can include factors like weather conditions, occupancy, or time of day.

**4️. Split the data:**

Divide the dataset into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.

**5. Choose and train a model:**

Select a suitable machine learning algorithm, such as linear regression, decision trees, or neural networks. Train the model using the training data.

**6. Evaluate the model:**

Assess the model’s performance using evaluation metrics like mean squared error or R-squared. This will help you understand how well the model predicts energy consumption.

**7. Fine-tune and validate:**

Adjust the model’s parameters and hyper parameters to improve its performance. Validate the model using cross-validation techniques to ensure its generalizability.

**8. Deploy and monitor:**

Once you’re satisfied with the model’s performance, deploy it in a production environment. Continuously monitor its predictions and update the model as needed.

**PYTHON PROGRAM FOR MEASURE ENERGY CONSUMPTION : EXAMPLE MODEL -1**

#python program for measuring energy consumption

Import pandas as pd

Import matplotlib.pyplot as plt

# Read energy consumption data from a CSV file

Data = pd.read\_csv(‘energy\_data.csv’)

To represent energy consumption using a bar diagram, you can

Use Python’s matplotlib library.

# Sample data

Categories = [‘Category 1️’, ‘Category 2️’, ‘Category 3️’]

Consumption = [1️0, 2️0, 1️5] # Energy consumption values

# Create the bar plot

Plt.bar(categories, consumption)

# Add labels and title

Plt.xlabel(‘Categories’)

Plt.ylabel(‘Energy Consumption’)

Plt.title(‘Energy Consumption by Category’)

# Show the plot

Plt.show()

This code creates a bar plot with categories on the x-axis and

Energy consumption values on the y-axis.

# Plot the energy consumption over time

Plt.plot(data[‘Date’], data[‘Energy Consumption’])

Plt.xlabel(‘Date’)

Plt.ylabel(‘Energy Consumption’)

Plt.title(‘Energy Consumption Over Time’)

Plt.show()

# plotting

Import matplotlib.pyplot as plt

Import numpy as np # linear algebra

Import os # accessing directory structure

Import pandas as pd # data processing, CSV file I/O (e.g.

Pd.read\_csv)

Import seaborn as sns

Plt.style.use(‘ggplot’) # Make it pretty

In [2️]:

# Data is saved in parquet format so schema is preserved.

Df = pd.read\_parquet(‘../input/est\_hourly.paruqet’)

**EXAMPLE MODEL – 2 :**

Import matplotlib.pyplot as plt # plotting

Import numpy as np # linear algebra

Import os # accessing directory structure

Import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

Import seaborn as sns

Plt.style.use(‘ggplot’) # Make it pretty

In [2]:

# Data is saved in parquet format so schema is preserved.

Df = pd.read\_parquet(‘../input/est\_hourly.paruqet’)

Data index is the date/hour, columns are for different regions within PJM.

Regions joined at different times, so not all have data for all dates. Regions also split (PJM\_Load split to East and West)

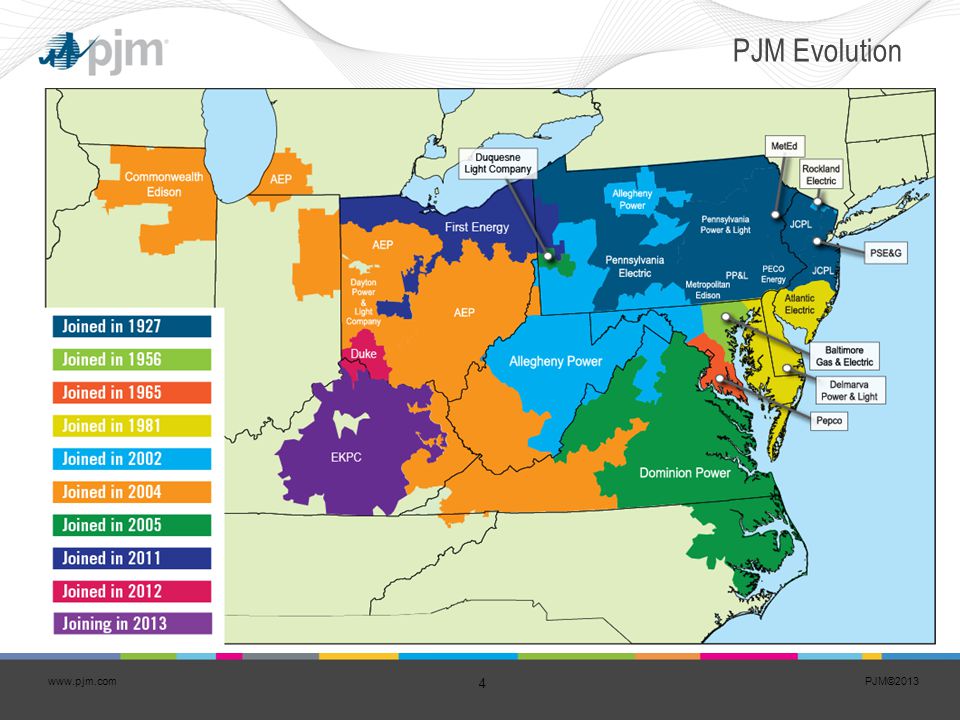
In [3]:

#Show PJM Regions

From IPython.display import Image

Image(url= <http://slideplayer.com/4238181/14/images/4/PJM+Evolution.jpg>)

Out[3]:

In [4]:

Df.head()

Out[4]:

In [5]:

**AEP COMED DAYTON DEOK DOM DUQ EKPC FE NI PJME PJMW PJM\_Load**

Datetime

1998-12-31 01:00:00 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 29309.0

1998-12-31 02:00:00 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 28236.0

1998-12-31 03:00:00 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 27692.0

1998-12-31 04:00:00 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 27596.0

1998-12-31 05:00:00 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 27888.0

Df.describe().T

Count mean std min 25% 50% 75% max

AEP 121273.0 15499.513717 2591.399065 9581.0 13630.0 15310.0 17200.00 25695.0

COMED 66497.0 11420.152112 2304.139517 7237.0 9780.0 11152.0 12510.00 23753.0

DAYTON 121275.0 2037.851140 393.403153 982.0 1749.0 2009.0 2279.00 3746.0

DEOK 57739.0 3105.096486 599.859026 907.0 2687.0 3013.0 3449.00 5445.0

DOM 116189.0 10949.203625 2413.946569 1253.0 9322.0 10501.0 12378.00 21651.0

DUQ 119068.0 1658.820296 301.740640 1014.0 1444.0 1630.0 1819.00 3054.0

EKPC 45334.0 1464.218423 378.868404 514.0 1185.0 1386.0 1699.00 3490.0

FE 62874.0 7792.159064 1331.268006 0.0 6807.0 7700.0 8556.00 14032.0

NI 58450.0 11701.682943 2371.498701 7003.0 9954.0 11521.0 12896.75 23631.0

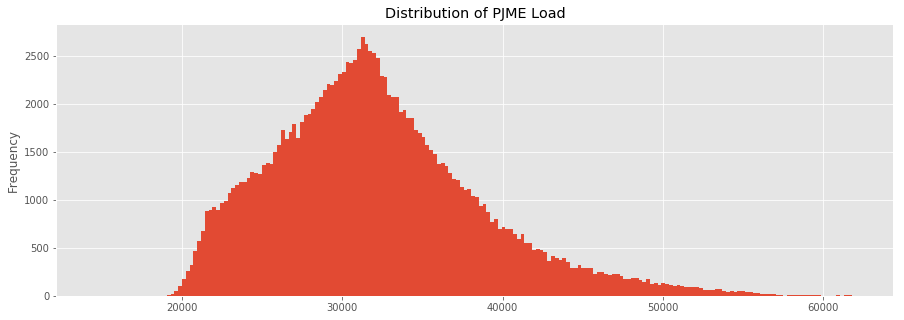
PJME 145366.0 32080.222831 6464.012166 14544.0 27573.0 31421.0 35650.00 62009.0

PJMW 143206.0 5602.375089 979.142872 487.0 4907.0 5530.0 6252.00 9594.0

PJM\_Load 32896.0 29766.427408 5849.769954 17461.0 25473.0 29655.0 33073.25 54030.0

\_ = df[‘PJME’].plot.hist(figsize=(15, 5), bins=200, title=’Distribution of PJME Load’)

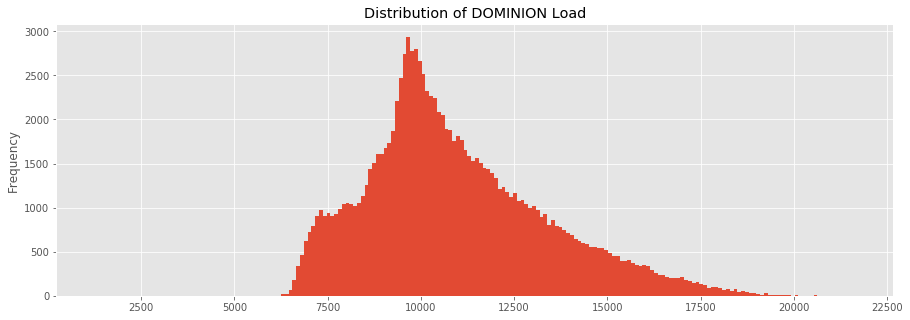
Df.describe().T\_ = df[‘PJME’].plot.hist(figsize=(15, 5), bins=200, title=’Distribution of PJME Load’)



Out[5]:

In [6]:

\_ = df[‘DOM’].plot.hist(figsize=(15, 5), bins=200, title=’Distribution of DOMINION Load’)



Import matplotlib.pyplot as plt # plotting

Import numpy as np # linear algebra

Import os # accessing directory structure

Import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

Import seaborn as sns

Plt.style.use(‘ggplot’) # Make it pretty

# Data is saved in parquet format so schema is preserved.

Df = pd.read\_parquet(‘../input/est\_hourly.paruqet’)

Data index is the date/hour, columns are for different regions within PJM.

Create Time Series Features

In [13]:

Df[‘dow’] = df.index.dayofweek

Df[‘doy’] = df.index.dayofyear

Df[‘year’] = df.index.year

Df[‘month’] = df.index.month

Df[‘quarter’] = df.index.quarter

Df[‘hour’] = df.index.hour

Df[‘weekday’] = df.index.weekday\_name

Df[‘woy’] = df.index.weekofyear

Df[‘dom’] = df.index.day # Day of Month

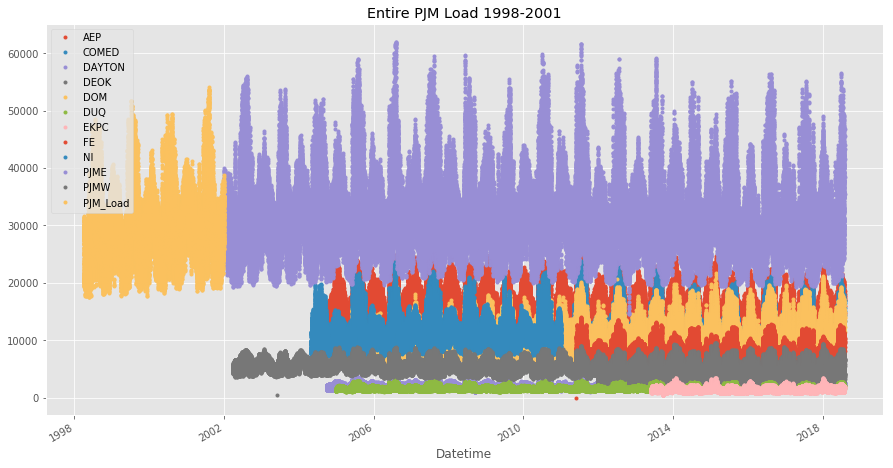
Df[‘date’] = df.index.date

In [14]:

## **Plot Time Series**

In [9]:

plot = df.plot(style='.', figsize=(15, 8), title='Entire PJM Load 1998-2001')

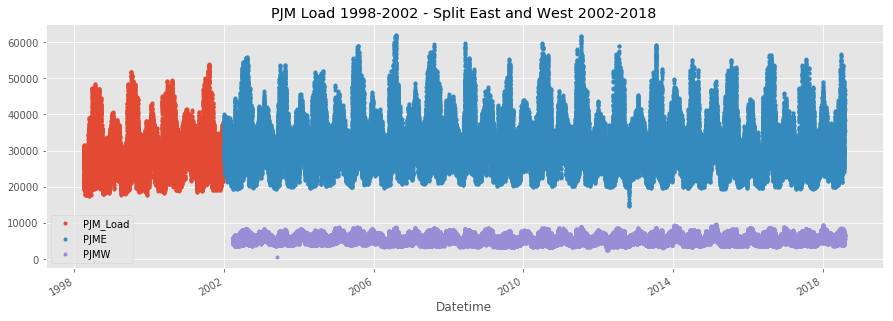


# **Plotting Regions**

In [10]:

\_ = df[['PJM\_Load','PJME','PJMW']] \

.plot(style='.', figsize=(15, 5), title='PJM Load 1998-2002 - Split East and West 2002-2018')



= df[[‘PJM\_Load’,’hour’]].plot(x=’hour’,

Y=’PJM\_Load’,

Kind=’scatter’,

Figsize=(14,4),

Title=’Consumption by Hour of Day’)

**Design Thinking to Measure Energy Consumption:**

Design thinking is a human-centered approach to innovation that can be used to solve any problem, big or small. It is a non-linear, iterative process that involves five key stages:

1. Empathize: Understand the needs and motivations of the people you are designing for.
2. Define: Identify the core problem or challenge that you are trying to solve.
3. Ideate: Generate a wide range of possible solutions to the problem.
4. Prototype: Build and test prototypes of your solutions to get feedback from users.
5. Test: Implement your solution and monitor its results to ensure that it is meeting the needs of the users.

Design thinking can be applied to measuring energy consumption in a number of ways. For example, it can be used to:

* Understand the needs and motivations of energy consumers: What are their energy consumption habits? What are their pain points? What are their goals for reducing their energy consumption?
* Identify the core challenges of measuring energy consumption: What are the barriers to accurate and timely energy consumption data? How can these barriers be overcome?
* Generate innovative solutions for measuring energy consumption: For example, design thinking can be used to develop new types of energy meters, data collection systems, and analysis tools.
* Prototype and test new energy consumption measurement solutions: This can be done by working with a small group of pilot users to get feedback on the new solutions and make necessary adjustments.
* Implement and monitor energy consumption measurement solutions: Once a solution has been prototyped and tested, it can be implemented on a larger scale. It is important to monitor the results of the solution to ensure that it is meeting the needs of the users.

Here is an example of how design thinking can be applied to develop a new type of energy meter for residential homes:

**Empathize**

* Interview homeowners to understand their needs and motivations related to energy consumption measurement.
* Observe homeowners in their homes to see how they currently measure their energy consumption.
* Identify the pain points that homeowners experience with their current energy meters.

**Define:**

* Define the core problem or challenge that you are trying to solve: To develop a new type of energy meter that is more accurate, timely, and user-friendly than existing meters.

**Ideate**:

* Brainstorm a wide range of possible solutions to the problem. For example, you could consider developing a meter that can be installed on the main electrical panel, a meter that can be connected to the homeowner's Wi-Fi network, or a meter that provides real-time feedback on energy consumption.

**Prototype**:

* Build a prototype of your solution and test it with a small group of pilot users. Get feedback from the pilot users on the accuracy, timeliness, and user-friendliness of the prototype.
* Make necessary adjustments to the prototype based on the feedback received.

**Test**:

* Once the prototype has been refined, implement it on a larger scale. Monitor the results of the solution to ensure that it is meeting the needs of the users.

By applying design thinking to measure energy consumption, you can develop innovative solutions that help people to save energy and reduce their environmental impact.

Documenting the Design Thinking Process

It is important to document the design thinking process so that you can learn from your experiences and share your insights with others. Here are some tips for documenting the design thinking process:

* Keep a journal: Throughout the design thinking process, write down your thoughts, ideas, and observations. This will help you to track your progress and identify areas where you can improve.
* Take pictures and videos: Visual documentation can be helpful for capturing important moments in the design thinking process, such as user interviews and brainstorming sessions.
* Create sketches and diagrams: Sketches and diagrams can be used to communicate your ideas to others and to develop prototypes of your solutions.
* Share your insights with others: Once you have completed the design thinking process, share your insights with others so that they can learn from your experiences. This can be done by writing blog posts, giving presentations, or simply having conversations with colleagues and friends.

By documenting the design thinking process, you can create a valuable resource that can help you and others to develop innovative solutions to real-world problems.

**Here is a list of tools and software commonly used in the process of measuring energy consumption using AI in Python:**

**PyRAPL:**

* A Python library for measuring the energy consumption of a host machine along the execution of a piece of Python code. PyRAPL uses the Intel "Running Average Power Limit" (RAPL) technology that estimates power consumption of a CPU.

**PowerTOP:**

* A Linux command-line tool for measuring and monitoring power consumption. PowerTOP can be used to identify devices that are consuming a lot of power and to make recommendations for reducing power consumption.

**Likwid:**

* A Linux command-line tool for measuring performance and power consumption of Intel processors. Likwid can be used to measure the energy consumption of different domains of a CPU, such as the cores, the cache, and the integrated GPU.

**TensorFlow:**

* A popular open-source machine learning framework. TensorFlow can be used to develop AI models for predicting energy consumption.

**PyTorch:**

* Another popular open-source machine learning framework. PyTorch is often used for rapid prototyping and research.

**Scikit-learn:**

* A Python library for machine learning and data science. Scikit-learn can be used to develop and evaluate AI models for predicting energy consumption.

In addition to these tools and software, you may also need to use other Python libraries for data processing and visualization. For example, you may want to use NumPy and Pandas for data processing, and Matplotlib or Seaborn for data visualization.

Here is a simple example of how to use PyRAPL to measure the energy consumption of a Python script:

Python program:

import pyRAPL

# Start the energy measurement

pyRAPL.start()

# Execute the Python script

# ...

# Stop the energy measurement

pyRAPL.stop()

# Get the energy consumption in Joules

energy\_consumption = pyRAPL.get\_energy\_consumption()

# Print the energy consumption

print(energy\_consumption)

This example will print the energy consumption of the Python script in Joules. You can then use this information to track your energy consumption over time and identify areas where you can save energy.

**Data Analysis and Visualizations for measure Energy Consumption Prediction:**

Data Analysis:The first step in any energy consumption prediction project is to understand the data. This involves cleaning the data, identifying any outliers or missing values, and performing exploratory data analysis (EDA). EDA can be used to identify trends, patterns, and correlations in the data.

Some common data analysis techniques for energy consumption prediction include:

* Time series analysis: Time series analysis is used to identify trends and patterns in time-dependent data. This can be used to identify seasonal patterns, cyclical patterns, and other trends that can be used to make predictions about future energy consumption.
* Correlation analysis: Correlation analysis is used to identify relationships between different variables. This can be used to identify factors that are correlated with energy consumption, such as weather conditions, economic activity, and population growth.
* Clustering: Clustering is used to group similar data points together. This can be used to identify different types of energy consumers and to develop targeted prediction models for each group.

Data Visualization:Data visualization is a powerful tool for communicating the results of data analysis and for identifying insights that may not be apparent from the data alone. Some common data visualization techniques for energy consumption prediction include:

* Line charts: Line charts are used to show trends in energy consumption over time.
* Bar charts: Bar charts are used to compare energy consumption across different categories, such as different types of consumers or different regions.
* Heatmaps: Heatmaps are used to visualize the correlation between different variables.
* Scatter plots: Scatter plots are used to identify relationships between two variables.

Predicting Future Energy Consumption using LSTM:

Long short-term memory (LSTM) networks are a type of recurrent neural network (RNN) that are well-suited for time series prediction tasks. LSTM networks have a special architecture that allows them to learn long-term dependencies in the data.

To train an LSTM network for energy consumption prediction, the following steps are typically followed:

1. Prepare the data: The data is first prepared by cleaning it, identifying any outliers or missing values, and scaling it to a suitable range.
2. Split the data: The data is then split into training and test sets. The training set is used to train the LSTM network and the test set is used to evaluate the performance of the trained model.
3. Design the LSTM network: The LSTM network is then designed by specifying the number of layers, the number of units per layer, and the activation functions.
4. Train the LSTM network: The LSTM network is then trained on the training set using a suitable optimizer and loss function.
5. Evaluate the LSTM network: The performance of the trained LSTM network is then evaluated on the test set.

Once the LSTM network is trained and evaluated, it can be used to predict future energy consumption. To do this, the network is simply given the historical energy consumption data as input and it produces a prediction of future energy consumption as output.

The accuracy of LSTM networks for predicting future energy consumption depends on a number of factors, including the quality of the training data, the complexity of the LSTM network architecture, and the length of the prediction horizon.

To improve the accuracy of LSTM predictions for two months later, the following steps can be taken:

* Use a high-quality training dataset: The training dataset should be as large and representative as possible. This will help the LSTM network to learn the underlying patterns in the data and to make more accurate predictions.
* Use a complex LSTM network architecture: More complex LSTM network architectures are typically able to learn more complex patterns in the data and to make more accurate predictions. However, complex network architectures also require more training data and can be more computationally expensive to train.
* Use shorter prediction horizons: LSTM networks are typically more accurate for shorter prediction horizons. Therefore, to necessary to train multiple LSTM networks, each with a different prediction horizon.

RNN Introduction using AI

RNNs are a type of neural network that are well-suited for processing sequential data. RNNs have recurrent connections, which allow them to learn long-term dependencies in the data.

RNNs are used in a variety of AI applications, including:

* Natural language processing: RNNs are used in natural language processing tasks such as machine translation, text summarization, and sentiment analysis.
* Speech recognition: RNNs are used in speech recognition tasks such as transcribing audio to text and recognizing commands.
* Time series prediction: RNNs are used in time series prediction tasks such as predicting future energy consumption.

**OVERVIEW OF THE PROCESS:**

To predict future energy consumption using LSTM, we can follow these steps:

1. Collect historical energy consumption data. This data should include information such as the date, time, and amount of energy consumed.
2. Preprocess the data. This may involve cleaning the data, removing outliers, and scaling the data to a consistent range.
3. Split the data into training and testing sets. The training set will be used to train the LSTM model, and the testing set will be used to evaluate the performance of the model on unseen data.
4. Design and train the LSTM model. This involves selecting the appropriate hyperparameters for the model and training the model on the training set.
5. Evaluate the performance of the model on the testing set. This will give us an idea of how well the model will generalize to unseen data.
6. Use the trained model to predict future energy consumption. We can provide the model with historical energy consumption data up to a certain point in time, and it will predict the energy consumption for the next few hours, days, or weeks.

Predicting values 2 months later accurately:To accurately predict energy consumption values 2 months later, we need to have a large and high-quality dataset of historical energy consumption data. The dataset should include data for a variety of different weather conditions and seasons. It is also important to choose the right hyperparameters for the LSTM model and to train the model on a sufficiently large training set.

RNN introduction:Recurrent neural networks (RNNs) are a type of neural network that are well-suited for processing sequential data, such as text or time series data. RNNs have a special architecture that allows them to learn long-term dependencies in the data.

Using AI:Artificial intelligence (AI) can be used to improve the accuracy of energy consumption forecasting in a number of ways. For example, AI can be used to:

* Identify patterns in the historical energy consumption data that are not easily visible to humans.
* Develop more complex and sophisticated forecasting models.
* Automatically optimize the hyperparameters of the forecasting model.

**Feature** **Selection:**

Feature selection is the process of identifying the most important features in a dataset for a given machine learning task. This can be done using a variety of methods, including statistical methods, machine learning methods, and domain knowledge.

Predicting Future Energy Consumption using LSTM Predicting Values 2 Months Later Accurately

1. Collect historical energy consumption data.
2. Clean and prepare the data.
3. Split the data into training and testing sets.
4. Train an LSTM model on the training set.
5. Evaluate the model on the testing set.
6. Use the trained model to predict future energy consumption.

**Feature** **Selection** **using** **AI**

AI can be used to select features for energy consumption prediction tasks using a variety of methods, including:

* Wrapper methods: Wrapper methods evaluate the performance of a machine learning model on a subset of features to select the subset that provides the best performance.
* Embedded methods: Embedded methods embed features into a lower-dimensional space and select the features that are most important for the prediction task.
* Genetic algorithms: Genetic algorithms are a type of evolutionary algorithm that can be used to select features.
* Use a large and high-quality dataset.
* Preprocess the data carefully.
* Use a variety of machine learning models and compare their performance.
* Use feature selection techniques to select the most important features.
* Evaluate the model on a held-out test set.
* Monitor the model's performance over time and update the model as needed.

**Model** **Training:**

To train the LSTM network, we will use the following steps:

1. Prepare the data. This involves cleaning and preprocessing the historical consumption data, such as filling in missing values and scaling the data to a consistent range.
2. Split the data into training and test sets. The training set will be used to train the model, and the test set will be used to evaluate the performance of the trained model.
3. Design the LSTM network. This involves choosing the number of layers, the number of units per layer, and the activation function.
4. Compile the model. This involves choosing the loss function and the optimizer.
5. Train the model. This involves feeding the training data to the model and adjusting the parameters of the model to minimize the loss function.
6. Evaluate the model. This involves feeding the test data to the model and measuring the accuracy of the predictions.

**Model** **Evaluation:**

Once the LSTM model has been trained, it is important to evaluate its performance on a held-out test dataset. This will help to ensure that the model is able to generalize to new data.

Some common metrics that can be used to evaluate the performance of an LSTM model for energy consumption forecasting include:

* Mean absolute error (MAE): This metric measures the average difference between the predicted and actual energy consumption values.
* Mean squared error (MSE): This metric measures the average squared difference between the predicted and actual energy consumption values.
* Root mean squared error (RMSE): This metric is the square root of the MSE. It is a good measure of the overall error of the model.

DataAnalysis and Visualizations and ER Predicting Future **“Measure** **Energy** **Consumption”** using LSTM Predicting Values 2 month Later Accurately RNN

**Hourly Measure Energy Consumption code:**

**Step 1:**

**Import Library**

In [1]:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **pprint**

%**matplotlib** inline

In [2]:

df = pd.read\_csv("AEP\_hourly.csv")

print("="\*50)

print("First Five Rows ","**\n**")

print(df.head(2),"**\n**")

print("="\*50)

print("Information About Dataset","**\n**")

print(df.info(),"**\n**")

print("="\*50)

print("Describe the Dataset ","**\n**")

print(df.describe(),"**\n**")

print("="\*50)

print("Null Values t ","**\n**")

print(df.isnull().sum(),"**\n**")

==================================================

First Five Rows

Datetime AEP\_MW

0 2004-12-31 01:00:00 13478.0

1 2004-12-31 02:00:00 12865.0

==================================================

Information About Dataset

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 121273 entries, 0 to 121272

Data columns (total 2 columns):

Datetime 121273 non-null object

AEP\_MW 121273 non-null float64

dtypes: float64(1), object(1)

memory usage: 1.9+ MB

None

==================================================

Describe the Dataset

AEP\_MW

count 121273.000000

mean 15499.513717

std 2591.399065

min 9581.000000

25% 13630.000000

50% 15310.000000

75% 17200.000000

max 25695.000000

==================================================

Null Values t

Datetime 0

AEP\_MW 0

dtype: int64

**Step 2:**

**Reformat the Date Time Columns**

In [3]:

*# Extract all Data Like Year MOnth Day Time etc*

dataset = df

dataset["Month"] = pd.to\_datetime(df["Datetime"]).dt.month

dataset["Year"] = pd.to\_datetime(df["Datetime"]).dt.year

dataset["Date"] = pd.to\_datetime(df["Datetime"]).dt.date

dataset["Time"] = pd.to\_datetime(df["Datetime"]).dt.time

dataset["Week"] = pd.to\_datetime(df["Datetime"]).dt.week

dataset["Day"] = pd.to\_datetime(df["Datetime"]).dt.day\_name()

dataset = df.set\_index("Datetime")

dataset.index = pd.to\_datetime(dataset.index)

dataset.head(1)

Out[3]:

|  | **AEP\_MW** | **Month** | **Year** | **Date** | **Time** | **Week** | **Day** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Datetime** |  |  |  |  |  |  |  |
| **2004-12-31 01:00:00** | 13478.0 | 12 | 2004 | 2004-12-31 | 01:00:00 | 53 | Friday |

**Step 3:**

In [4]:

*# How many Unique Year do we Have in Dataset*

print(df.Year.unique(),"**\n**")

print("Total Number of Unique Year", df.Year.nunique(), "**\n**")

[2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017

2018]

Total Number of Unique Year 15

**Lets us see the energy consumption Each Year**

In [5]:

**from** **matplotlib** **import** style

fig = plt.figure()

ax1 = plt.subplot2grid((1,1), (0,0))

style.use('ggplot')

sns.lineplot(x=dataset["Year"], y=dataset["AEP\_MW"], data=df)

sns.set(rc={'figure.figsize':(15,6)})

plt.title("Energy consumptionnin Year 2004")

plt.xlabel("Date")

plt.ylabel("Energy in MW")

plt.grid(**True**)

plt.legend()

**for** label **in** ax1.xaxis.get\_ticklabels():

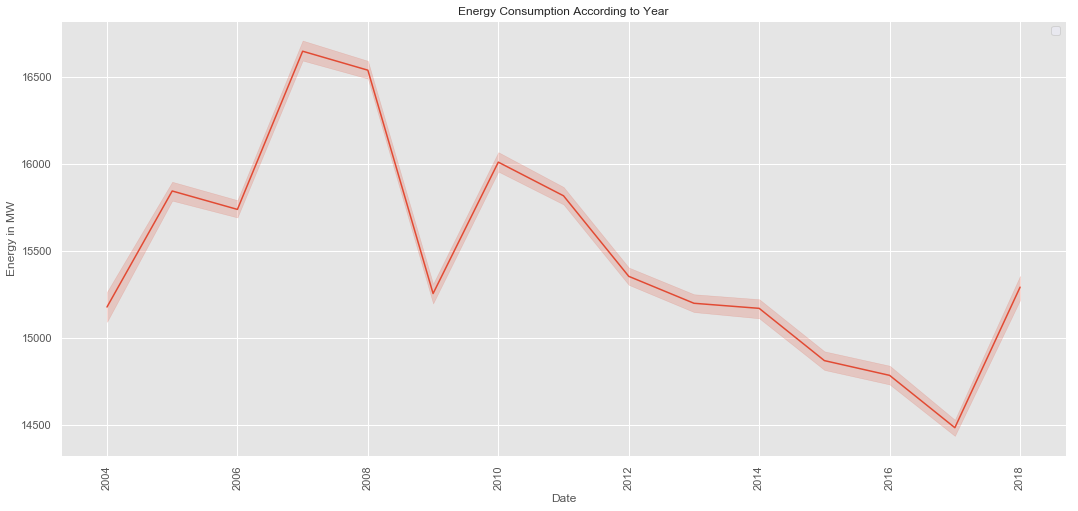
label.set\_rotation(90)

plt.title("Energy Consumption According to Year")

No handles with labels found to put in legend.

Out[5]:

Text(0.5, 1.0, 'Energy Consumption According to Year')



In [6]:

**from** **matplotlib** **import** style

fig = plt.figure()

ax1= fig.add\_subplot(311)

ax2= fig.add\_subplot(312)

ax3= fig.add\_subplot(313)

style.use('ggplot')

y\_2004 = dataset["2004"]["AEP\_MW"].to\_list()

x\_2004 = dataset["2004"]["Date"].to\_list()

ax1.plot(x\_2004,y\_2004, color="green", linewidth=1.7)

y\_2005 = dataset["2005"]["AEP\_MW"].to\_list()

x\_2005 = dataset["2005"]["Date"].to\_list()

ax2.plot(x\_2005, y\_2005, color="green", linewidth=1)

y\_2006 = dataset["2006"]["AEP\_MW"].to\_list()

x\_2006 = dataset["2006"]["Date"].to\_list()

ax3.plot(x\_2006, y\_2006, color="green", linewidth=1)

plt.rcParams["figure.figsize"] = (18,8)

plt.title("Energy consumptionnin")

plt.xlabel("Date")

plt.ylabel("Energy in MW")

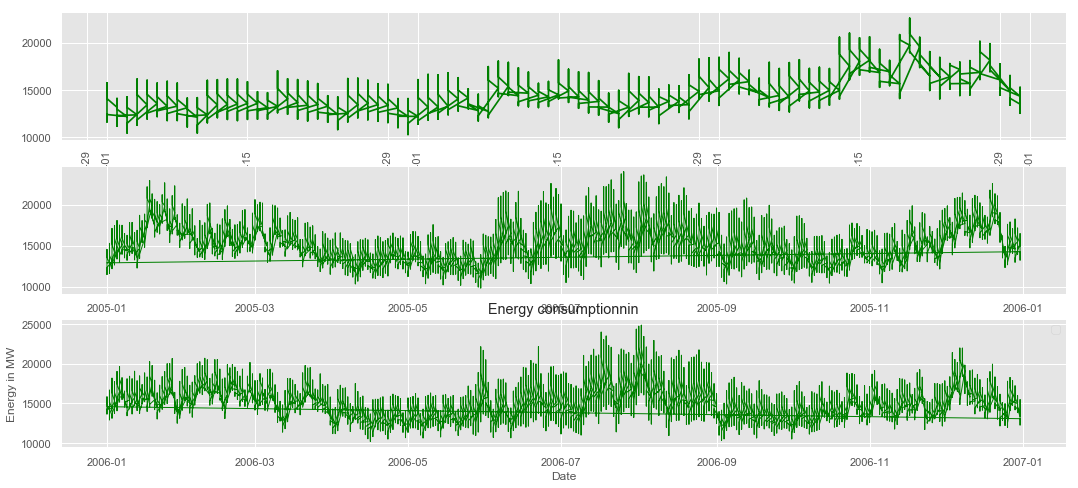
plt.grid(**True**, alpha=1)

plt.legend()

**for** label **in** ax1.xaxis.get\_ticklabels():

label.set\_rotation(90)

No handles with labels found to put in legend.



**Energy Distribution**

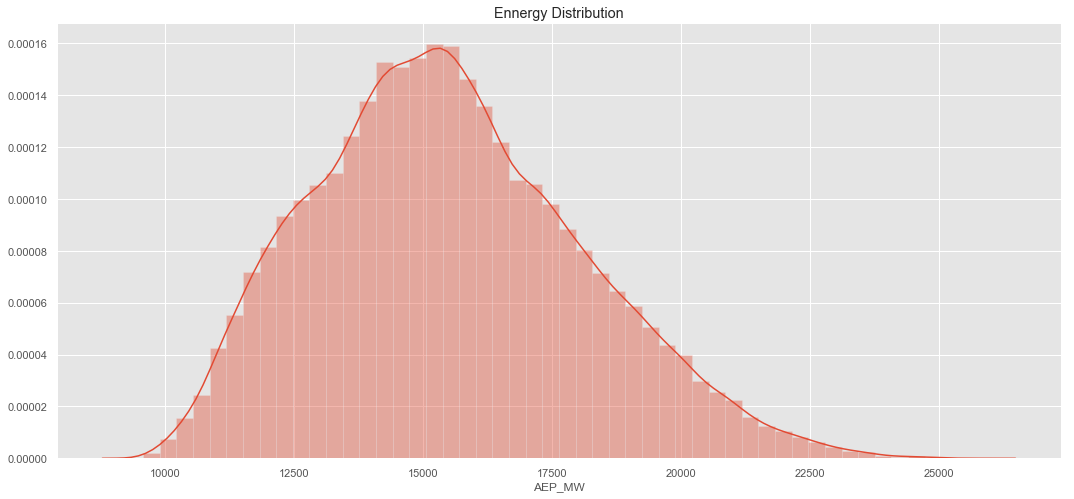
In [7]:

sns.distplot(dataset["AEP\_MW"])

plt.title("Ennergy Distribution")

Out[7]:

Text(0.5, 1.0, 'Ennergy Distribution')



**Energy with Respect to Time**

In [8]:

fig = plt.figure()

ax1= fig.add\_subplot(111)

sns.lineplot(x=dataset["Time"],y=dataset["AEP\_MW"], data=df)

plt.title("Energy Consumption vs Time ")

plt.xlabel("Time")

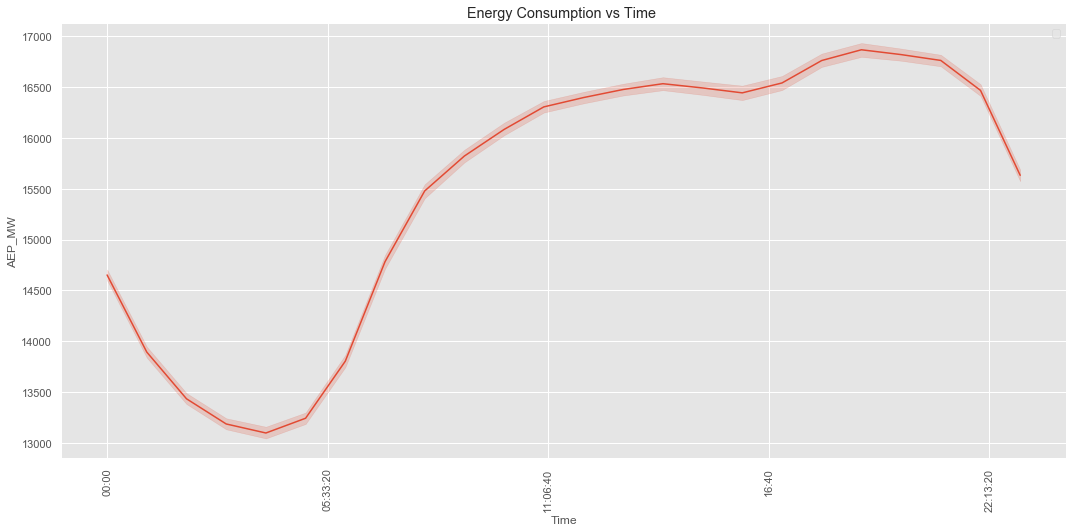
plt.grid(**True**, alpha=1)

plt.legend()

**for** label **in** ax1.xaxis.get\_ticklabels():

label.set\_rotation(90)

No handles with labels found to put in legend.



**Resampleing Data**

In [9]:

NewDataSet = dataset.resample('D').mean()

In [10]:

print("Old Dataset ",dataset.shape )

print("New Dataset ",NewDataSet.shape )

Old Dataset (121273, 7)

New Dataset (5055, 4)

In [11]:

TestData = NewDataSet.tail(100)

Training\_Set = NewDataSet.iloc[:,0:1]

Training\_Set = Training\_Set[:-60]

In [12]:

print("Training Set Shape ", Training\_Set.shape)

print("Test Set Shape ", TestData.shape)

Training Set Shape (4995, 1)

Test Set Shape (100, 4)

In [13]:

Training\_Set = Training\_Set.values

sc = MinMaxScaler(feature\_range=(0, 1))

Train = sc.fit\_transform(Training\_Set)

In [14]:

X\_Train = []

Y\_Train = []

*# Range should be fromm 60 Values to END*

**for** i **in** range(60, Train.shape[0]):

*# X\_Train 0-59*

X\_Train.append(Train[i-60:i])

*# Y Would be 60 th Value based on past 60 Values*

Y\_Train.append(Train[i])

*# Convert into Numpy Array*

X\_Train = np.array(X\_Train)

Y\_Train = np.array(Y\_Train)

print(X\_Train.shape)

print(Y\_Train.shape)

(4935, 60, 1)

(4935, 1)

In [15]:

*# Shape should be Number of [Datapoints , Steps , 1 )*

*# we convert into 3-d Vector or #rd Dimesnsion*

X\_Train = np.reshape(X\_Train, newshape=(X\_Train.shape[0], X\_Train.shape[1], 1))

X\_Train.shape

Out[14]:

(4935, 60, 1)

**Model**

In [15]:

regressor = Sequential()

*# Adding the first LSTM layer and some Dropout regularisation*

regressor.add(LSTM(units = 50, return\_sequences = **True**, input\_shape = (X\_Train.shape[1], 1)))

regressor.add(Dropout(0.2))

*# Adding a second LSTM layer and some Dropout regularisation*

regressor.add(LSTM(units = 50, return\_sequences = **True**))

regressor.add(Dropout(0.2))

*# Adding a third LSTM layer and some Dropout regularisation*

regressor.add(LSTM(units = 50, return\_sequences = **True**))

regressor.add(Dropout(0.2))

*# Adding a fourth LSTM layer and some Dropout regularisation*

regressor.add(LSTM(units = 50))

regressor.add(Dropout(0.2))

*# Adding the output layer*

regressor.add(Dense(units = 1))

*# Compiling the RNN*

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

In [16]:

regressor.fit(X\_Train, Y\_Train, epochs = 50, batch\_size = 32)

Epoch 1/50

4935/4935 [==============================] - 33s 7ms/step - loss: 0.0237

Epoch 2/50

4935/4935 [==============================] - 33s 7ms/step - loss: 0.0183

Epoch 3/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0173

Epoch 4/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0164

Epoch 5/50

4935/4935 [==============================] - 35s 7ms/step - loss: 0.0157

Epoch 6/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0160

Epoch 7/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0151

Epoch 8/50

4935/4935 [==============================] - 35s 7ms/step - loss: 0.0125

Epoch 9/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0099

Epoch 10/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0089

Epoch 11/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0085

Epoch 12/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0083

Epoch 13/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0078

Epoch 14/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0079

Epoch 15/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0073

Epoch 16/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0075

Epoch 17/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0072

Epoch 18/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0070

Epoch 19/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0066

Epoch 20/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0063

Epoch 21/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0061

Epoch 22/50

4935/4935 [==============================] - 32s 6ms/step - loss: 0.0058

Epoch 23/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0056

Epoch 24/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0055

Epoch 25/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0053

Epoch 26/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0054

Epoch 27/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0053

Epoch 28/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0051

Epoch 29/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0050

Epoch 30/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0051

Epoch 31/50

4935/4935 [==============================] - 32s 6ms/step - loss: 0.0050

Epoch 32/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0049

Epoch 33/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0048

Epoch 34/50

4935/4935 [==============================] - 32s 7ms/step - loss: 0.0048

Epoch 35/50

4935/4935 [==============================] - 2283s 463ms/step - loss: 0.0048

Epoch 36/50

4935/4935 [==============================] - 3475s 704ms/step - loss: 0.0047

Epoch 37/50

4935/4935 [==============================] - 32s 6ms/step - loss: 0.0047

Epoch 38/50

4935/4935 [==============================] - 29s 6ms/step - loss: 0.0047

Epoch 39/50

4935/4935 [==============================] - 30s 6ms/step - loss: 0.0046

Epoch 40/50

4935/4935 [==============================] - 31s 6ms/step - loss: 0.0046

Epoch 41/50

4935/4935 [==============================] - 33s 7ms/step - loss: 0.0045

Epoch 42/50

4935/4935 [==============================] - 37s 7ms/step - loss: 0.0045

Epoch 43/50

4935/4935 [==============================] - 38s 8ms/step - loss: 0.0047

Epoch 44/50

4935/4935 [==============================] - 36s 7ms/step - loss: 0.0045

Epoch 45/50

4935/4935 [==============================] - 35s 7ms/step - loss: 0.0044

Epoch 46/50

4935/4935 [==============================] - 38s 8ms/step - loss: 0.0044

Epoch 47/50

4935/4935 [==============================] - 34s 7ms/step - loss: 0.0043

Epoch 48/50

4935/4935 [==============================] - 42s 9ms/step - loss: 0.0043

Epoch 49/50

4935/4935 [==============================] - 37s 7ms/step - loss: 0.0044

Epoch 50/50

4935/4935 [==============================] - 37s 8ms/step - loss: 0.0044

Out[16]:

<keras.callbacks.History at 0x1a36d8f898>

**Test Data**

In [17]:

TestData.head(2)

Out[17]:

|  | **AEP\_MW** | **Month** | **Year** | **Week** |
| --- | --- | --- | --- | --- |
| **Datetime** |  |  |  |  |
| **2018-04-26** | 13157.791667 | 4 | 2018 | 17 |
| **2018-04-27** | 12964.000000 | 4 | 2018 | 17 |

In [18]:

TestData.shape

Out[18]:

(100, 4)

In [19]:

NewDataSet.shape

Out[19]:

(5055, 4)

In [20]:

Df\_Total = pd.concat((NewDataSet[["AEP\_MW"]], TestData[["AEP\_MW"]]),axis=0)

In [21]:

Df\_Total.shape

Out[19]:

(5155, 1)

In [22]:

inputs = Df\_Total[len(Df\_Total) - len(TestData) - 60:].values

inputs.shape

Out[22]:

(160, 1)

In [23]:

inputs = Df\_Total[len(Df\_Total) - len(TestData) - 60:].values

*# We need to Reshape*

inputs = inputs.reshape(-1,1)

*# Normalize the Dataset*

inputs = sc.transform(inputs)

X\_test = []

**for** i **in** range(60, 160):

X\_test.append(inputs[i-60:i])

*# Convert into Numpy Array*

X\_test = np.array(X\_test)

*# Reshape before Passing to Network*

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

*# Pass to Model*

predicted\_stock\_price = regressor.predict(X\_test)

*# Do inverse Transformation to get Values*

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

In [24]:

True\_MegaWatt = TestData["AEP\_MW"].to\_list()

Predicted\_MegaWatt = predicted\_stock\_price

dates = TestData.index.to\_list()

In [25]:

Machine\_Df = pd.DataFrame(data={

"Date":dates,

"TrueMegaWatt": True\_MegaWatt,

"PredictedMeagWatt":[x[0] **for** x **in** Predicted\_MegaWatt ]

})

**Future Predicted**

In [26]:

Machine\_Df

Out[26]:

|  | **Date** | **TrueMegaWatt** | **PredictedMeagWatt** |
| --- | --- | --- | --- |
| **0** | 2018-04-26 | 13157.791667 | 13671.706055 |
| **1** | 2018-04-27 | 12964.000000 | 12991.945312 |
| **2** | 2018-04-28 | 12237.583333 | 14521.591797 |
| **3** | 2018-04-29 | 12156.791667 | 13211.944336 |
| **4** | 2018-04-30 | 13443.500000 | 12788.455078 |
| **5** | 2018-05-01 | 13251.875000 | 13789.046875 |
| **6** | 2018-05-02 | 13641.166667 | 12804.154297 |
| **7** | 2018-05-03 | 14217.250000 | 12709.704102 |
| **8** | 2018-05-04 | 13725.625000 | 14261.728516 |
| **9** | 2018-05-05 | 11902.166667 | 14472.195312 |
| **10** | 2018-05-06 | 11680.083333 | 12677.794922 |
| **11** | 2018-05-07 | 12972.500000 | 12127.531250 |
| **12** | 2018-05-08 | 13295.083333 | 12887.196289 |
| **13** | 2018-05-09 | 13688.750000 | 12743.552734 |
| **14** | 2018-05-10 | 13993.250000 | 12747.035156 |
| **15** | 2018-05-11 | 13525.166667 | 13814.033203 |
| **16** | 2018-05-12 | 12942.916667 | 13970.200195 |
| **17** | 2018-05-13 | 12832.541667 | 13168.587891 |
| **18** | 2018-05-14 | 15004.750000 | 12955.161133 |
| **19** | 2018-05-15 | 15171.791667 | 15169.067383 |
| **20** | 2018-05-16 | 13925.416667 | 14419.253906 |
| **21** | 2018-05-17 | 14465.666667 | 12913.649414 |
| **22** | 2018-05-18 | 13684.333333 | 14998.011719 |
| **23** | 2018-05-19 | 13044.166667 | 14174.238281 |
| **24** | 2018-05-20 | 13169.125000 | 13413.721680 |
| **25** | 2018-05-21 | 14728.666667 | 13382.070312 |
| **26** | 2018-05-22 | 14857.125000 | 14739.416992 |
| **27** | 2018-05-23 | 14489.583333 | 14121.821289 |
| **28** | 2018-05-24 | 14656.250000 | 13763.244141 |
| **29** | 2018-05-25 | 15137.125000 | 15047.317383 |
| **...** | ... | ... | ... |
| **70** | 2018-07-05 | 17609.000000 | 17120.591797 |
| **71** | 2018-07-06 | 15742.916667 | 17615.269531 |
| **72** | 2018-07-07 | 13610.333333 | 14689.130859 |
| **73** | 2018-07-08 | 13768.708333 | 13816.837891 |
| **74** | 2018-07-09 | 16427.333333 | 15385.699219 |
| **75** | 2018-07-10 | 17489.333333 | 16932.236328 |
| **76** | 2018-07-11 | 16714.125000 | 17681.707031 |
| **77** | 2018-07-12 | 16330.833333 | 16694.558594 |
| **78** | 2018-07-13 | 16911.291667 | 15885.130859 |
| **79** | 2018-07-14 | 16488.375000 | 16239.578125 |
| **80** | 2018-07-15 | 16296.208333 | 16572.927734 |
| **81** | 2018-07-16 | 17400.041667 | 17885.480469 |
| **82** | 2018-07-17 | 17311.125000 | 17595.656250 |
| **83** | 2018-07-18 | 15814.041667 | 17368.632812 |
| **84** | 2018-07-19 | 15889.916667 | 15917.466797 |
| **85** | 2018-07-20 | 15332.500000 | 15957.360352 |
| **86** | 2018-07-21 | 13795.250000 | 14366.544922 |
| **87** | 2018-07-22 | 13479.333333 | 13657.029297 |
| **88** | 2018-07-23 | 15410.083333 | 15275.373047 |
| **89** | 2018-07-24 | 15890.541667 | 15779.814453 |
| **90** | 2018-07-25 | 16503.333333 | 16030.302734 |
| **91** | 2018-07-26 | 16474.250000 | 16809.560547 |
| **92** | 2018-07-27 | 15816.625000 | 16138.321289 |
| **93** | 2018-07-28 | 14113.083333 | 14586.478516 |
| **94** | 2018-07-29 | 13658.000000 | 13875.068359 |
| **95** | 2018-07-30 | 15368.083333 | 15294.772461 |
| **96** | 2018-07-31 | 15180.291667 | 15672.427734 |
| **97** | 2018-08-01 | 15151.166667 | 15329.677734 |
| **98** | 2018-08-02 | 15687.666667 | 15497.061523 |
| **99** | 2018-08-03 | 14809.000000 | 15975.358398 |

In [27]:

True\_MegaWatt = TestData["AEP\_MW"].to\_list()

Predicted\_MegaWatt = [x[0] **for** x **in** Predicted\_MegaWatt ]

dates = TestData.index.to\_list()

In [28]:

fig = plt.figure()

ax1= fig.add\_subplot(111)

x = dates

y = True\_MegaWatt

y1 = Predicted\_MegaWatt

plt.plot(x,y, color="green")

plt.plot(x,y1, color="red")

*# beautify the x-labels*

plt.gcf().autofmt\_xdate()

plt.xlabel('Dates')

plt.ylabel("Power in MW")

plt.title("Machine Learned the Pattern Predicting Future Values ")

plt.legend()

No handles with labels found to put in legend.

Out[28]:

<matplotlib.legend.Legend at 0x1a4984b780>



**Feature Engineering Using AI:**

Feature engineering is the process of transforming raw data into features that are more informative and predictive. Feature engineering can be used to improve the performance of any machine learning model, including LSTM models.

One way to use AI for feature engineering is to use a technique called autoencoders. Autoencoders are a type of neural network that can be used to learn latent representations of the data. These latent representations can then be used as features for the LSTM model.

Example:

The following is an example of how to use AI to predict future energy consumption using an LSTM model:

1. Collect a large dataset of historical energy consumption data.
2. Preprocess the data to ensure that it is in a format that is compatible with the LSTM model.
3. Use AI feature engineering techniques to create new features from the data.
4. Split the data into training and testing sets.
5. Train the LSTM model on the training set.
6. Evaluate the performance of the model on the testing set.
7. Use the trained model to predict future energy consumption.

**Various Features to Perform Model Training Using AI:**

In addition to historical energy consumption data, there are a number of other features that you can use to train your LSTM model. These features can include:

* Weather data: Weather conditions such as temperature, humidity, and wind speed can have a significant impact on energy consumption.
* Economic data: Economic factors such as GDP growth and unemployment can also impact energy consumption.
* Demographic data: Demographic factors such as population growth and household size can also affect energy consumption.

By including these additional features in your model, you can improve its accuracy and ability to predict future energy consumption values.

Example:

The following code shows how to train an LSTM model to predict future energy consumption values using Python and the TensorFlow library:

import tensorflow as tf

# Load the historical energy consumption data

df = pd.read\_csv('energy\_consumption.csv')

# Split the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['energy\_consumption'], df['date'], test\_size=0.25)

# Create the LSTM model

model = tf.keras.Sequential([

tf.keras.layers.LSTM(128, input\_shape=(X\_train.shape[1],)),

tf.keras.layers.Dense(1)

])

# Compile the model

model.compile(loss='mse', optimizer='adam')

# Train the model

model.fit(X\_train, y\_train, epochs=100)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate the mean squared error

mse = tf.keras.losses.MSE(y\_test, y\_pred)

# Print the mean squared error

print('Mean squared error:', mse)

This code will train an LSTM model to predict future energy consumption values using the historical energy consumption data in the energy\_consumption.csv file. The model will be trained for 100 epochs and the mean squared error on the test set will be printed to the console.

**ADVANTAGE:**

There are several advantages to using AI and Python to measure energy consumption, including:

**Accuracy**:

AI models can be trained on large datasets of energy consumption data to learn complex patterns and relationships. This allows them to make more accurate predictions of energy consumption than traditional forecasting methods.

**Timeliness**:

AI models can be used to predict energy consumption in real time, allowing for more timely interventions to reduce energy costs and improve energy efficiency.

**Scalability**:

AI models can be scaled to accommodate large amounts of data, making them ideal for use in large-scale energy consumption measurement projects.

**Flexibility**:

Python is a flexible and powerful programming language that can be used to develop a wide range of energy consumption measurement solutions.

**Cost-effectiveness:**

Python is an open-source programming language, which means that it is free to use. This makes it a cost-effective option for developing energy consumption measurement solutions.

Here are some specific examples of how AI and Python can be used to improve energy consumption measurement:

**Developing smart meters:**

* AI can be used to develop smart meters that can collect real-time energy consumption data and send it to a central server for analysis. This data can then be used to generate more accurate energy consumption predictions and to identify areas where energy can be saved.

**Predicting energy consumption:**

AI models can be trained on historical energy consumption data to predict future energy consumption. This information can be used to develop energy management strategies that reduce energy costs and improve energy efficiency.

**Detecting energy anomalies:**

AI models can be used to detect anomalies in energy consumption data. This can be helpful for identifying equipment failures or other problems that are wasting energy.

**Optimizing energy consumption:**

AI can be used to optimize energy consumption in buildings, factories, and other facilities. For example, AI can be used to develop heating and cooling systems that are more energy-efficient.

Overall, AI and Python can be used to develop a wide range of innovative solutions for measuring and managing energy consumption.

Here are some additional advantages of using AI and Python to measure energy consumption:

* **Transparency:**

AI models can be made transparent, allowing users to understand how they work and to trust their predictions.

* **Reproducibility**:

AI models can be reproduced, meaning that other researchers can use the same data and code to train and evaluate the models.

* **Collaboration**:

AI models can be shared and collaborated on, making it easier for researchers to work together to develop better energy consumption measurement solutions.

Overall, AI and Python are powerful tools that can be used to develop innovative and effective solutions for measuring and managing energy consumption.

**DISADVANTAGE:**

Here are some disadvantages of measuring energy consumption using AI and Python:

* **Complexity**:
* Developing and deploying AI models can be complex and time-consuming. This can be a barrier for organizations that do not have the necessary expertise or resources.
* **Data requirements:**
* AI models require large amounts of training data. This data can be difficult and expensive to collect, especially for energy consumption measurement projects.
* **Interpretability:**
* AI models can be difficult to interpret. This can make it difficult to understand how the model is making predictions and to identify potential biases in the model.
* **Accuracy**:
* AI models are not always accurate. The accuracy of an AI model depends on the quality of the training data and the complexity of the model.
* **Cost**:
* Developing and deploying AI models can be expensive. This can be a barrier for organizations with limited budgets.

In addition to these general disadvantages, there are also some specific disadvantages to using Python for energy consumption measurement projects. For example, Python libraries for energy consumption measurement are not as mature or widely used as libraries for other tasks such as data science and machine learning. This can make it difficult to find the right tools for your project and to get support from the community.

Here are some tips for overcoming the disadvantages of measuring energy consumption using AI and Python:

* **Start small:**
* Start with a small and well-defined project. This will help you to learn the necessary skills and to identify any potential challenges before you invest in a large-scale project.
* **Use open-source software:**
* There are a number of open-source Python libraries for energy consumption measurement. These libraries can help you to get started quickly and without having to invest in commercial software.
* **Get help from the community:**
* There are a number of online forums and communities where you can get help with developing and deploying AI models in Python.
* **Test and validate your models:**
* It is important to test and validate your AI models before deploying them in production. This will help to ensure that the models are accurate and reliable.
* **Monitor your results:**
* Once you have deployed your AI models, it is important to monitor their results to ensure that they are meeting your expectations.

By following these tips, you can successfully measure energy consumption using AI and Python.

**CONCLUSION:**

* Develop smart meters that can collect and analyze energy consumption data in real time.
* Create energy dashboards that provide users with insights into their energy consumption patterns.
* Optimize energy usage in buildings and other facilities.
* Predict future energy consumption trends.

AI and Python are still under development, but they have the potential to revolutionize the way we measure and manage energy consumption.

Here are some of the key conclusions of using AI and Python to measure energy consumption:

* AI models can be used to predict energy consumption with high accuracy. This information can be used to make informed decisions about energy management and to reduce energy costs.
* Python is a powerful and versatile language for developing and deploying AI models. There are a number of open-source Python libraries for energy consumption measurement, which makes it easy to get started.
* AI and Python can be used to develop innovative solutions for measuring energy consumption in a variety of settings. For example, AI-powered smart meters can be used to collect and analyze energy consumption data in real time, while energy dashboards can provide users with insights into their energy consumption patterns.
* So, when it comes to the conclusion for measuring energy consumption, it’s all about being aware of our energy usage and making conscious choices to reduce it. By measuring and monitoring our energy consumption, we can identify areas of improvement and implement energy-saving strategies. This not only helps us save money on our bills, but also contributes to a more sustainable and eco-friendly lifestyle. So let’s keep track of our energy usage and make a positive impact together.
* LSTM networks are a powerful tool for predicting future energy consumption values. By using LSTM networks, we can develop accurate forecasting models that can help us to better manage our energy resources. LSTM networks are a powerful tool for predicting future energy consumption. By following the steps outlined above and using a large and high-quality dataset of historical energy consumption data, we can develop LSTM models that can accurately predict energy consumption values 2 months later. AI can also be used to further improve the accuracy of energy consumption forecasting.
* Measuring energy consumption is the first step towards reducing energy costs and environmental impact. By understanding how and where we are using energy, we can identify areas where we can save energy.
* There are a number of different ways to measure energy consumption, depending on the type of energy being used and the level of detail required.
* Measuring energy consumption can be challenging, but it is essential for making informed decisions about energy management.