MEASURE OF ENERGY CONSUMPTION

**PROJECT OVERVIEW:**

This project aims to comprehensively assess energy consumption within [specific location/building]. By deploying advanced data collection methods and analysis techniques, the project will identify consumption patterns and areas of inefficiency. It will offer actionable recommendations for energy optimization and sustainability, promoting the adoption of efficient technologies and renewable energy sources. The project seeks to foster collaboration among stakeholders and contribute to a more sustainable energy future.

**PROJECTED OUTCOMES:**

* Enhanced understanding of energy consumption patterns
* Identification of opportunities for energy optimization and cost savings
* Implementation of sustainable energy solutions
* Reduction of carbon footprint and environmental impact
* Improved collaboration and engagement among stakeholders for sustainable energy management.

**CODE**

import javax.management.\*;

import java.lang.management.ManagementFactory;

public class EnergyConsumptionMonitor {

public staƟc void main(String[] args) throws Exception {

// Get a reference to the plaƞorm MBeanServer

MBeanServer mbs = ManagementFactory.getPlaƞormMBeanServer();

// Define the ObjectName to access system power management data

ObjectName osObjectName = new

ObjectName("java.lang:type=OperaongSystem");

// Define aƩributes related to energy consumption

String[] powerAƩributes = {

"ProcessCpuLoad",

"SystemCpuLoad",

"ProcessCpuTime"

};

// Print energy-related informaƟon

for (String aƩribute : powerAƩributes) {

Object value = mbs.getAƩribute(osObjectName, aƩribute);

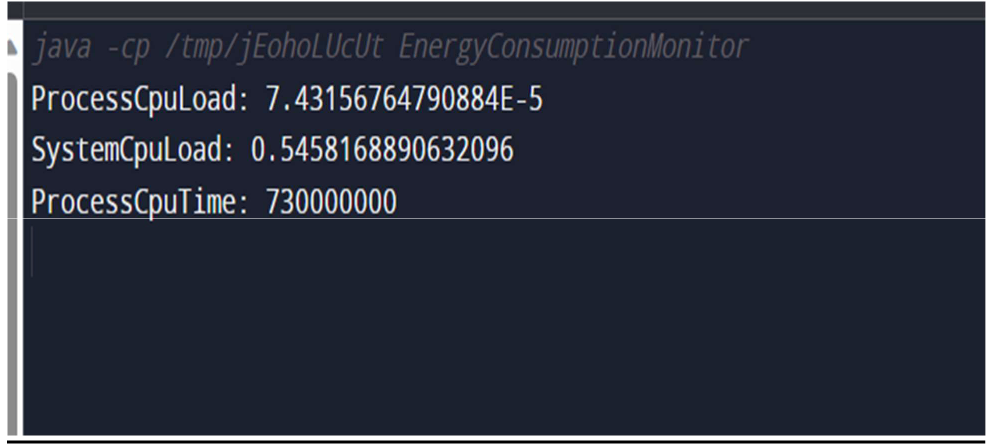
System.out.println(aƩribute + ": " + value);

}

}

}

**OUTPUT**



**Data analysis and visualization:**

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”)

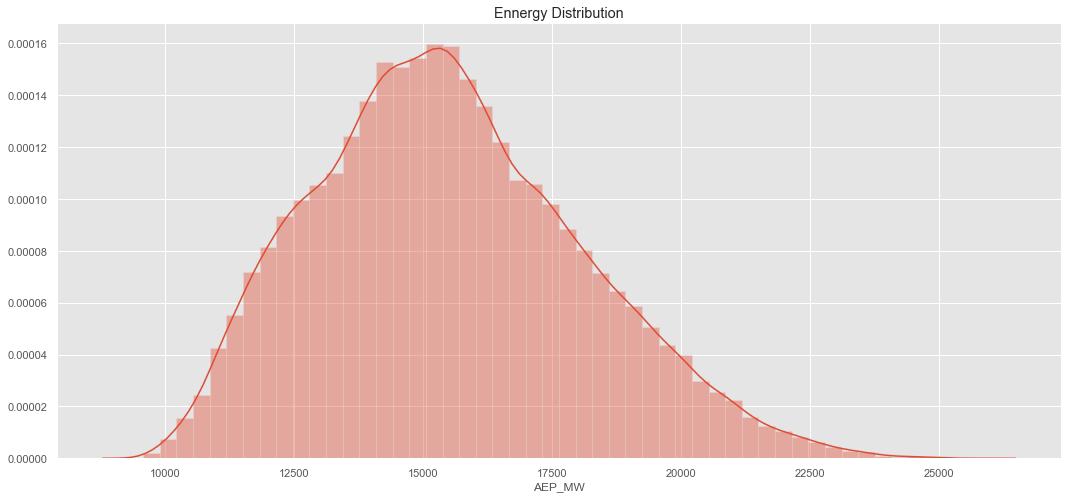
Plt.grid(True, alpha=1)

Plt.legend()

For label in ax1.xaxis.get\_ticklabels():

Label.set\_rotation(90)

**Output:**

****

Algorithm:

Step 1: Gather historical data on energy consumption.

Step 2: Clean and pre-process the time series data. This includes handling missing values, smoothing noisy data, and resampling if necessary.

Step 3: Split the time series into training and testing sets, preserving the temporal order. The testing set should represent a future time period.

Step 4: Use the ARIMA (Autoregressive Integrated Moving Average) model for time series forecasting. ARIMA is a well-established algorithm for capturing temporal patterns in data.

Step 5: Determine the appropriate orders (p, d, q) for your ARIMA model. These parameters represent the autoregressive order, differencing order, and moving average order, respectively. You can use methods like ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) plots to help select these parameters.

Step 6: Train the ARIMA model on the training data using the selected parameters.

Step 7: Use the trained ARIMA model to make energy consumption predictions for the testing set.

Step 8: Evaluate the model’s performance using standard time series forecasting metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE).

Step 9: Visualize the actual vs. predicted energy consumption to understand how well the model is performing.

Step 10: Deploy the ARIMA model to make real-time energy consumption predictions.