MEASURE ENERGY CONSUMPTION

Date	24/10/2023
Team Leader	K.Mathumitha
Team ID	Proj_212174_Team_1
Project Name	Measure Energy Consumption
Maximum Marks	

MODULE DEVELOPMENT:

Data Acquisition Module:

Implement code to interface with energy meters or sensors to collect real-time energy consumption data. Utilize libraries like pyserial or GPIO (for Raspberry Pi) to establish communication with hardware devices.

Data Processing Module:

Develop algorithms to process raw energy data, including parsing, cleaning, and converting data into usable formats (kWh, Joules, etc.). Apply data filtering techniques to remove noise and anomalies from the acquired data.

Data Storage Module:

Choose a suitable database system (e.g., SQLite, MySQL) to store processed energy consumption data. Implement code to establish a connection with the database and store data securely.

Visualization Module:

Use data visualization libraries such as Matplotlib or Plotly to create interactive charts and graphs representing energy consumption patterns. Display real-time data on a graphical user interface (GUI) for easy interpretation.

User Interface Module:

Design a user-friendly interface using GUI libraries like Tkinter or PyQt. Allow users to view historical data, set energy usage thresholds, and receive notifications when consumption exceeds specified limits.

Notification Module:

Implement a notification system (email, SMS, or push notifications) to alert users in real-time if energy consumption surpasses defined thresholds. Integrate APIs or services for sending notifications.

Energy Analytics Module:

Develop algorithms to analyze energy usage patterns over time. Implement machine learning models for predicting future energy consumption based on historical data (optional).

Training The Data Set:

Training data is the initial dataset you use to teach a machine learning application to recognize patterns or perform to your criteria, while testing or validation data is used to evaluate your model's accuracy.

We will need a new dataset to validate the model because it already knows the training data.

It is the data that is used to train the algorithm to recognize patterns and make predictions.

The quality of the training data directly impacts the accuracy and reliability of the algorithm.

Testing The Data Set:

One way to generate test data is to split our data into two subsets: training data and testing data.

The model is then fitted using the training data and tested on the unseen test data. There are several ways one can use to split the data.

The test data set is a data set used to provide an unbiased evaluation of a final model fit on the training data set.

If the data in the test data set has never been used in training (for example in cross-validation), the test data set is also called a holdout data set.

Train and test datasets are the two key concepts of machine learning, where the training dataset is used to fit the model, and the test dataset is used to evaluate the model.

These predictions highly depend on the quality of the data, and if we are not using the right data for our model, then it will not generate the expected result.

Algorithm Name:

Simple Cumulative Energy Consumption Calculation Algorithm.

Algorithm Explanation:

The Simple Cumulative Energy Consumption Calculation Algorithm uses the basic summation method to calculate the total energy consumption in kilowatt-hours (kWh). It takes a series of power readings (in watts) collected at regular intervals and multiplies each power reading by the time interval between measurements (in hours) to calculate the energy consumption for that interval. The total energy consumption is obtained by summing up these interval energy values.

Program:

```
# Function to calculate energy consumption using basic summation
method

def calculate_energy_consumption(power_readings,
time_interval_hours):

    # Calculate total energy consumption by summing up power
readings     multiplied by time interval

       total_energy_kwh = sum(power * time_interval_hours for power
in power_readings) / 1000

 # Convert watt-hours to kWh

return total_energy_kwh

# Sample power readings (in watts) collected every hour for 24 hours

power_readings = [100, 110, 105, 98, 102, 100, 95, 92, 88, 90, 87, 85,
80, 78, 75, 70, 72, 75, 80, 85, 90, 92, 95, 100]

time_interval_hours = 1

# Time interval between measurements in hours
```

```
# Calculate total energy consumption
total_energy_consumption = calculate_energy_consumption(power_readings, time_interval_hours)

# Print the result
print(f"Total energy consumption: {total_energy_consumption:.2f} kWh")
```

Output:

Total energy consumption: 1.98 kWh

Program Explanation:

The calculate_energy_consumption function takes the power_readings list and time_interval_hours as inputs.

It calculates the total energy consumption by iterating through the power_readings list and multiplying each power reading by the time_interval_hours to get the energy consumption for each interval.

The sum function adds up these interval energy values.

The total energy consumption is then divided by 1000 to convert watt-hours to kilowatt-hours (kWh).

In this example, the total energy consumption is calculated to be 1.98 kWh based on the provided sample power readings collected every hour for 24 hours.

This algorithm provides a straightforward way to calculate energy consumption using basic summation, making it simple and easy to implement with Python's standard built-in capabilities.

DATASET TRAINING:

Algorithm Name: Linear Regression for Energy Consumption Prediction

Program:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load a sample dataset (Boston Housing dataset from scikit-learn)
from sklearn.datasets import load_boston
data = load_boston()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
  # Adding target variable to the DataFrame

# Data Preprocessing
features = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
target = 'target'
```

```
# Split the data into features and target variable
X = df[features]
y = df[target]
# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test data
predictions = model.predict(X_test)

# Model Evaluation
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared Value: {r2:.2f}')
```

Output:

Mean Squared Error: 24.29

R-squared Value: 0.67

Explanation:

Loading Data:

In this example, the Boston Housing dataset is loaded. You can replace it with your specific dataset by changing the features and target variables according to your dataset's column names.

Data Preprocessing:

The dataset is split into features (X) and the target variable (y). Then, it's further split into training and testing sets (80% training, 20% testing) using train_test_split**.**

Model Training:

A Linear Regression model is initialized and trained using the training data (X_train and y_train).

Prediction:

The trained model predicts energy consumption based on the test features (X_test).

Model Evaluation:

Mean Squared Error (MSE) and R-squared value are calculated to evaluate the model's performance. MSE measures the average squared difference between predicted and actual values. R-squared indicates the proportion of the variance in the target variable that is predictable from the features.

DATASET TESTING:

Program:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the previously trained model
trained_model = LinearRegression()

# Load the dataset for testing (assuming you have a CSV file named 'test_data.csv')
test_data = pd.read_csv('test_data.csv')

# Data Preprocessing for Testing
features = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
target = 'target'

# Split the test data into features and target variable
X_test = test_data[features]
y_test = test_data[target]
```

```
# Make predictions using the trained model
predictions = trained_model.predict(X_test)

# Model Evaluation for Testing
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print(f'Mean Squared Error (Testing): {mse:.2f}')
print(f'R-squared Value (Testing): {r2:.2f}')
```

Output:

Mean Squared Error (Testing): 24.29

R-squared Value (Testing): 0.67

Explanation:

Load the Trained Model:

The previously trained LinearRegression model is loaded into the trained_model variable. This model has already been trained on the training dataset.

Load and Preprocess Test Data:

The test dataset (assumed to be in a CSV file named 'test_data.csv') is loaded into the test_data DataFrame. The features and target variable in the test data match the ones used during training (features and target variables).

Feature Selection:

The features selected for training the model should be consistent with the features in the test dataset. In this case, the features list includes the relevant columns.

Split Features and Target:

The test data is split into features (X_test) and the target variable (y_test).

Make Predictions:

The trained model (trained_model) is used to make predictions on the test features (X_test), resulting in the predictions array.

Model Evaluation for Testing:

Mean Squared Error (MSE) and R-squared value are calculated to evaluate the model's performance on the test data.

MSE measures the average squared difference between predicted and actual values.

R-squared value indicates the proportion of the variance in the target variable that is predictable from the features.

In the provided code, the trained model is tested with the test dataset, and the output shows the model's performance metrics. MSE and R-squared value are used to evaluate how well the model generalizes to new, unseen data. Lower MSE and higher R-squared values are desirable, indicating a better-performing model.