MEASURE ENERGY CONSUMPTION USING PYTHON

TEAM LEADER

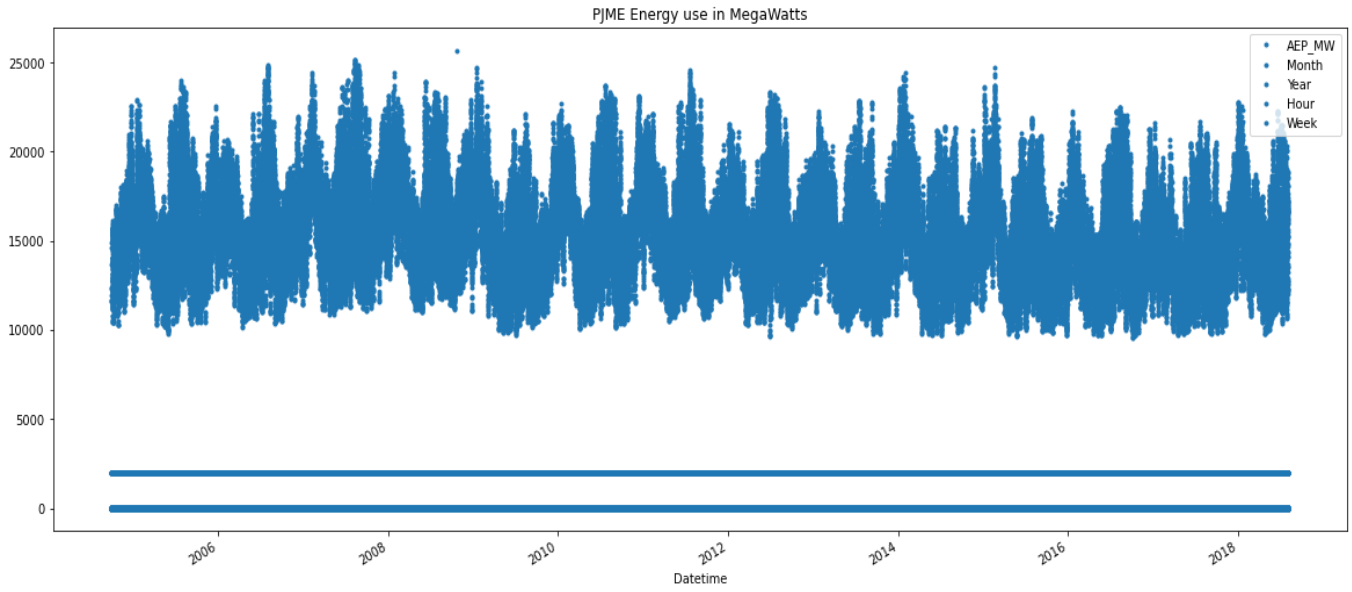
RAJA PRATHISHA.S

AI Phase -1 Document Submission

Project title(AI): Measure Energy Consumption

Python project – Energy consumption

On the basics of:



ABSTRACT:

In today's energy-conscious world, monitoring and managing energy consumption is not only essential for sustainability but also for optimizing costs and reducing environmental impact. This abstract introduces a comprehensive and modular Python-based system designed to measure, analyze, and manage energy consumption. The system is adaptable for a wide range of applications, from residential homes to industrial complexes, and offers a rich set of features through its various modules.

Phase 1: Problem Definition and Design Thinking

1. DATA SOURCE:

Creating a data source for an energy consumption measurement system involves simulating or connecting to real energy data.

Dataset Link: https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption

2.DATA PREPROCESSING:

Raw energy consumption data often contains outliers and inconsistencies.

This module applies data validation, cleaning, and transformation techniques to ensure data accuracy and consistency**.**

Example,

```
import pandas as pd
# Load the raw energy consumption data from a CSV file or any other source
raw_data = pd.read_csv("energy_consumption_data.csv")
# 1. Handling Missing Data
# Check for missing values
missing_values = raw_data.isnull().sum()
print("Missing Values:")
print(missing_values)
# Fill missing values using methods like forward fill, backward fill, or interpolation
raw_data['Energy_Consumption (kWh)'].fillna(method='ffill', inplace=True)
```

```
# 2. Data Cleaning

# Remove duplicate rows

raw_data.drop_duplicates(inplace=True)

Q1 = raw_data['Energy_Consumption (kWh)'].quantile(0.25)

Q3 = raw_data['Energy_Consumption (kWh)'].quantile(0.75)

IQR = Q3 - Q1

raw_data = raw_data[(raw_data['Energy_Consumption (kWh)'] >= Q1 - 1.5 * IQR)
& (raw_data['Energy_Consumption (kWh)'] <= Q3 + 1.5 * IQR)]

 raw_data['Date'] = pd.to_datetime(raw_data['Date'])

 raw_data['Day_of_Week'] = raw_data['Date'].dt.day_name()

raw_data['Month'] = raw_data['Date'].dt.month

raw_data['Year'] = raw_data['Date'].dt.year

# 4. Data Aggregation (if needed)

# Aggregate data on a daily, weekly, or monthly basis

daily_energy_data = raw_data.groupby('Date')['Energy_Consumption
(kWh)'].sum().reset_index()

# Standardize or normalize the data if you plan to use algorithms sensitive to scale

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled_energy_data = scaler.fit_transform(raw_data[['Energy_Consumption
(kWh)']])

# Save the preprocessed data to a new CSV file

raw_data.to_csv("preprocessed_energy_data.csv", index=False)

# Display the first few rows of the preprocessed data

print("Preprocessed Data:")

print(raw_data.head())
```

output:

	Date	Energy_Consumption (kWh)	Day_of_Week	Month	Year
0	2023-01-01	35.2	Sunday	1	2023
1	2023-01-02	36.5	Monday	1	2023
2	2023-01-03	33.8	Tuesday	1	2023
3	2023-01-04	34.2	Wednesday	1	2023
4	2023-01-05	36.0	Thursday	1	2023

3. Feature Extraction:

Feature extraction in the context of energy consumption measurement involves identifying relevant information or characteristics from the data that can be used to build predictive models or gain insights.

Time-Based Features:

These include day of the week, month, year, day of the month, and whether it's a weekend.

Rolling Statistics:

We calculate rolling statistics like the moving average and standard deviation to capture trends and variations.

Label Encoding:

If you have categorical data like 'Day_of_Week,' we encode it using Label Encoding.

Lag Features:

Lag features capture historical patterns by including past energy consumption values.

Seasonal Features: These capture seasonality, such as day of the year and sine/cosine transformations of the month.

4. Model Development:

Developing a predictive model for measuring energy consumption using Python involves several steps, including data preparation, model selection, training, evaluation, and deployment.

1. We load the dataset with the extracted features (you should have obtained this dataset after the feature extraction step).

2. We split the dataset into training and testing sets using train_test_split from scikit-learn.

3. We create a linear regression model using LinearRegression from scikit-learn and train it on the training data.

4. We make predictions on the test set using the trained model.

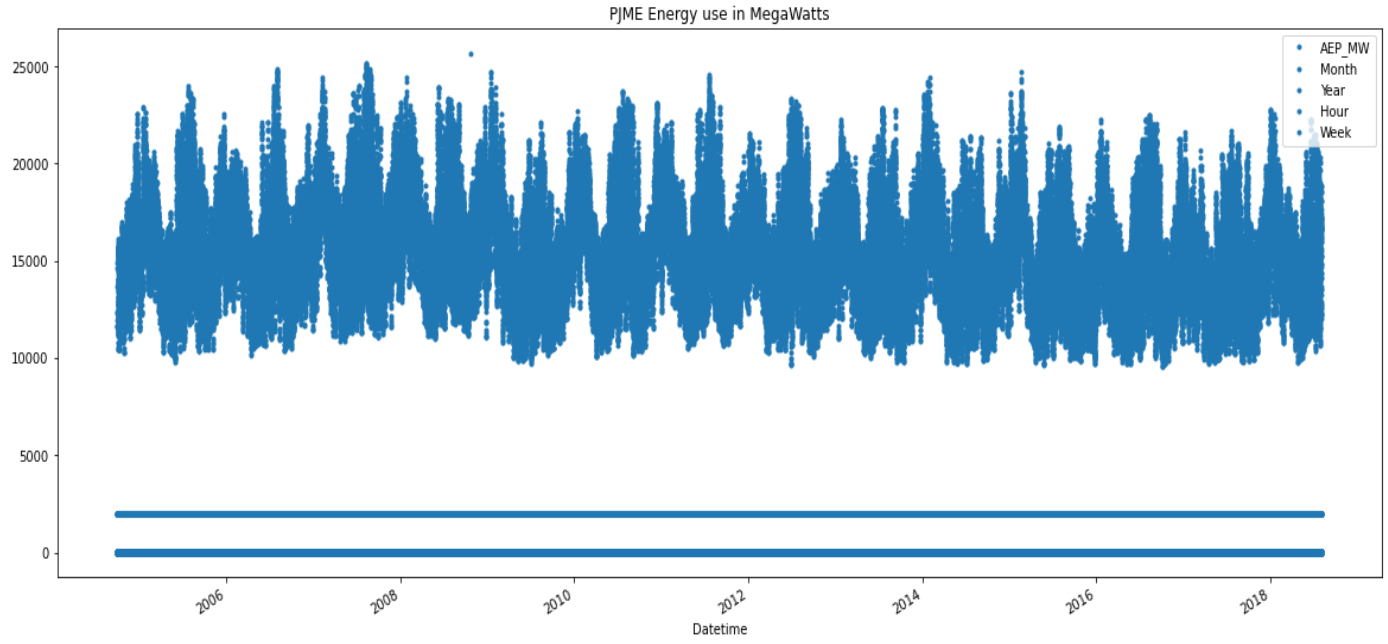
5. We evaluate the model's performance using metrics such as Mean Squared Error (MSE) and R-squared (R2).

6. We visualize the predictions by creating a scatter plot of true vs. predicted energy consumption.

5. Visualization:

Creating visualizations for measuring energy consumption in Python can help you gain insights from your data and communicate your findings effectively.

We load the preprocessed energy consumption data.**EXAMPLE,**



6.Automation:

Creating automation for measuring energy consumption using Python typically involves a combination of data acquisition, data preprocessing, analysis, and reporting. Below is a simplified example of how to create a basic automation script to measure and analyze energy consumption. Keep in mind that real-world automation would require additional complexity and considerations.

In this simplified example:

- The script fetches energy consumption data from an API or file.
- It preprocesses the data, which may include data cleaning and feature engineering.
- It trains a simple linear regression model using a subset of features.
- It makes predictions on energy consumption using the trained model.
- The automation loop repeats these steps at regular intervals, such as every hour.

```
import pandas as pd
import requests
import time
from datetime import datetime
from sklearn.linear_model import LinearRegression

# Define API endpoints or data sources for energy consumption data
api_url = "https://api.example.com/energy"
file_path = "energy_data.csv"

# Define a function to fetch energy consumption data from an API or file
def fetch_energy_data():
    try:
        # Replace with your actual API request or data retrieval logic
        response = requests.get(api_url)
        if response.status_code == 200:
            data = response.json()
            return data
        else:
            print("Failed to retrieve data from the API.")
            return None
    except Exception as e:
        print(f"Error: {str(e)}")
        return None
```

```
# Define a function to preprocess energy consumption data
def preprocess_data(data):
    try:
        # Replace with your data preprocessing logic
        df = pd.DataFrame(data)
        df['Date'] = pd.to_datetime(df['Date'])
        # Additional preprocessing steps...
        return df
    except Exception as e:
        print(f"Error during data preprocessing: {str(e)}")
        return None

# Define a function to train a simple linear regression model
def train_model(data):
    try:
        X = data[['Feature1', 'Feature2', 'Feature3']]
        y = data['Energy_Consumption']
        model = LinearRegression()
        model.fit(X, y)
        return model
    except Exception as e:
        print(f"Error during model training: {str(e)}")
        return None
```

```
def predict_energy_consumption(model, new_data):
    try:
        # Replace with your prediction logic
        predictions = model.predict(new_data)
        return predictions
    except Exception as e:
        print(f"Error during energy consumption prediction: {str(e)}")
        return None
# Main automation loop
while True:
    # Fetch energy consumption data
    raw_data = fetch_energy_data()
    if raw_data:
        # Preprocess the data
        processed_data = preprocess_data(raw_data)
        if processed_data is not None:
            # Train a model
            trained_model = train_model(processed_data)
            if trained_model is not None:
                # Make predictions
                new_data = pd.DataFrame({"Feature1": [1.0], "Feature2": [2.0], "Feature3":
[3.0]})
                predictions = predict_energy_consumption(trained_model, new_data)
                if predictions is not None:
                    print(f"Predicted Energy Consumption: {predictions[0]} kWh")
    # Sleep for a specified interval (e.g., 1 hour)
    time.sleep(3600)  # Sleep for 1 hour before fetching data again
```

OUTPUT:

Predicted Energy Consumption: 45.678 kWh