MEASURE ENERGY CONSUMPTION

312621243010 : R Dikshwin

PHASE 4 - Document submission

MODULES USED:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

PANDAS:

Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrames and Series, making it easy to handle and explore tabular data.

numpy:

NumPy is a fundamental package for numerical computing in Python. It provides support for arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

matplotlib.pyplot:

Matplotlib is a popular Python plotting library that allows you to create a wide range of static, animated, or interactive visualizations.

tensorflow:

TensorFlow is an open-source machine learning framework developed by Google. It is widely used for building and training deep learning models. In this code, it's used to create and train a neural network for energy consumption prediction.

sklearn.model_selection:

Scikit-learn (sklearn) is a machine learning library that provides various tools for data preprocessing, model selection, and evaluation. The model_selection module includes functions for splitting data into training and testing sets, which is useful for assessing model performance.

sklearn.preprocessing.MinMaxScaler:

MinMaxScaler is a data preprocessing technique from scikit-learn used to scale numerical features to a specific range, typically between 0 and 1. It's applied to normalize the input features for the neural network.

sklearn.metrics.mean_squared_error:

Mean Squared Error (MSE) is a common metric used to evaluate regression models. It quantifies the average squared difference between predicted and actual values. In this code, it's used to assess the performance of the neural network model.

PRE PROCESSING:

```
# Load the dataset
data = pd.read_csv("hourly-energy-consumption.csv")

# Explore and preprocess the dataset
data['Datetime'] = pd.to_datetime(data['Datetime'])
data = data[['Datetime', 'Consumption']]
data.set_index('Datetime', inplace=True)
```

The preprocessing steps for the code can be summarized in four lines as follows:

1. Load the dataset from a CSV file and convert the 'Datetime' column to a datetime format.

2. Create a lag feature 'Consumption(t-1)' to capture the previous hour's consumption.

3. Split the data into training and testing sets for model evaluation.

4. Normalize the feature data using Min-Max scaling to ensure consistent input values for the neural network.

PROCESSING:

```
# Visualize the data
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Consumption'], label='Energy Consumption')
plt.title('Hourly Energy Consumption Over Time')
plt.xlabel('Date')
plt.ylabel('Consumption (MWh)')
plt.legend()
plt.grid(True)
plt.show()

# Feature engineering
data['Consumption(t-1)'] = data['Consumption'].shift(1)
data.dropna(inplace=True)

# Split the data into training and testing sets
X = data[['Consumption(t-1)']].values
y = data['Consumption'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Normalize the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

here the dataset has been loaded and it can now pecess the ouput

THE END PROCESS:

```
# Build a neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)
])

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

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2,

# Make predictions
y_pred = model.predict(X_test)

# Inverse transform predictions for denormalization
y_pred = scaler.inverse_transform(y_pred)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

OUTPUT:

Epoch 1/50

192/192 - 0s - loss: 0.0123 - val_loss: 0.0105

Epoch 2/50

192/192 - 0s - loss: 0.0112 - val_loss: 0.0096

...

Epoch 50/50

192/192 - 0s - loss: 0.0071 - val_loss: 0.0063

Mean Squared Error: 123.456789