Phase 3 : DEVELOPMENT PART1

Name : Dilli babu D

Register Number : 312621243011

College Name : Thangavelu engineering college

Project 4: Electricity Prices Prediction

Objective :

▯ The electricity price prediction task is based on a case study where you need to predict daily price of electricity based on the daily consumption of heavy machinery used by businessess.

Problem Statement :

▯ Create a predictive model that utilizes historical electricity prices and relevant factors to forecast future electricity prices, assisting energy providers and consumers in making informed decisions regarding consumption and investment.

Problem Definition :

▯ The problem is to develop a predictive model that uses historical electricity prices and relevant factors to forecast future electricity prices. The objective is to create a tool that assists both energy providers and consumers in making informed decisions regarding consumption and investment by predicting future electricity prices. This project involves data preprocessing, feature engineering, model selection, training, and evaluation.

CODE EXPLANATION :

Load your electricity price dataset

```
import pandas as pd
data = pd.read_csv('Electricity.csv')
```

- .Imported the pandas library into our environment
- .Passed the filepath to read_csv to read the data into memory as a pandas dataframe

```
# Handle missing values (if any)
data.fill(data.mean(), inplace=True)
# Remove duplicate values (if any)
data = data.drop_duplicates()
```

- . If ‘first’,it consider first value as unique & rest of the same values as duplicate
- . If ‘last’,it consider last value as unique & rest of the same values as duplicate
- . If false,it consider all the same values as duplicate

- . Inplace : Boolean values,removes rows with duplicates if true
- . Return type: Dataframe with removed dulpiacte rows depending on arguments passed

```
# Time-based features
data['Date'] = pd.to_datetime(data['Date'])
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['DayOfWeek'] = data['Date'].dt.dayofweek
# Lagged variables
data['ElectricityPrice_Lag1'] = data['ElectricityPrice'].shift(1)
data['ElectricityPrice_Lag7'] = data['ElectricityPrice'].shift(7)
```

- . The object to convert to a datetime. If a **DataFrame** is provided, the method expects minimally the following columns: **"year"**, **"month"**, **"day"**. The column "year" must be specified in 4-digit format.

```
from statsmodels.tsa.arima_model import ARIMA
# Define the ARIMA order (p, d, q)
p = 1  # Example value
d = 1  # Example value
q = 1  # Example value
# Create the ARIMA model
```

```
model = ARIMA(data['ElectricityPrice'], order=(p, d, q))
# Fit the model to the data
model_fit = model.fit()
# Print the summary of the model
print(model_fit.summary())
```

- . A nonseasonal ARIMA model is classified as an "ARIMA(p,d,q)" model, where:
  - **p** is the number of autoregressive terms,
  - **d** is the number of nonseasonal differences needed for stationarity, and
  - **q** is the number of lagged forecast errors in the prediction equation.

- .The forecasting equation is constructed as follows. First, let y denote the dth difference of Y, which means:

If d=0: $y_t = Y_t$

If d=1: $y_t = Y_t - Y_{t-1}$

If d=2: $y_t = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}) = Y_t - 2Y_{t-1} + Y_{t-2}$

```
# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train, test = data['ElectricityPrice'][:train_size],
data['ElectricityPrice'][train_size:]
# Initialize and fit the ARIMA model on the training data
model = ARIMA(train, order=(p, d, q))
```

```
model_fit = model.fit()
# Print the summary of the model
print(model_fit.summary())
```

- . It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced.

- .There is no optimal split percentage.
- .You must choose a split percentage that meets your project's objectives with considerations that include:

  Computational cost in training the model.
  Computational cost in evaluating the model.
  Training set representativeness.
  Test set representativeness.

Nevertheless, common split percentages include:

  Train: 80%, Test: 20%
  Train: 67%, Test: 33%
  Train: 50%, Test: 50%

```
# Make predictions on the test set
predictions = model_fit.forecast(steps=len(test))
```

```
# Calculate MAE, MSE, RMSE (import necessary libraries)
from sklearn.metrics import mean_absolute_error,
mean_squared_error
import math
mae = mean_absolute_error(test, predictions)
mse = mean_squared_error(test, predictions)
rmse = math.sqrt(mse)
# Print the evaluation results
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
```

- **. raw_values:**

  Returns a full set of errors in case of multioutput input.

- **.uniform_average:**
  Errors of all outputs are averaged with uniform weight.

- **.squared*bool, default=True***
  If True returns MSE value, if False returns RMSE value.