Phase 4: Development Part 2 - Feature Engineering, Model Training, and Evaluation

In this phase, we will continue to build the electricity price prediction model by focusing on feature engineering, model training, and evaluation. These steps are crucial in developing an accurate predictive model.

1. Feature Engineering

Feature engineering involves creating new features or transforming existing ones to enhance the predictive power of the model. In the context of electricity price prediction, this could include:

Time-Based Features: Extract time-related features such as day of the week, month, and year to capture seasonality and trends.

Lagged Variables: Create lagged variables to account for autocorrelation in time series data. These can be past electricity prices, demand, or supply values.

External Factors: Include external factors like weather conditions and economic indicators that may influence electricity prices.

2. Model Training

After feature engineering, we move on to model training. This step involves selecting an appropriate forecasting model and training it on the preprocessed data. You mentioned considering ARIMA, LSTM, Prophet, and other advanced models. Here's how you can train a model using a popular library like scikit-learn:

3. Evaluation

Once the model is trained, it's crucial to evaluate its performance using appropriate time series forecasting metrics. Some commonly used metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). Here's an example of how to evaluate the model:

By performing these steps, you will enhance your electricity price prediction model's accuracy and have a better understanding of its performance.

Code:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

from math import sqrt

import keras
```

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("Electricity.csv", na_values=['?'])
df.isnull().sum()
df = df.dropna()

#Splitting the independent features and target feature
X = df[['ActualWindProduction', 'SystemLoadEP2', 'SMPEA', 'SystemLoadEA', 'ForecastWindProduction',
    'DayOfWeek', 'Year', 'ORKWindspeed', 'CO2Intensity', 'PeriodOfDay']]
y = df['SMPEP2']
#Train-Validation Split (90% Train set and 10% Validation set)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 42)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
model = keras.Sequential([
      keras.layers.Dense(512, activation="relu", input_shape=[10]),
      keras.layers.Dense(800, activation="relu"),
      keras.layers.Dropout(0.3),
      keras.layers.Dense(1024, activation="relu"),
      keras.layers.Dropout(0.3),
      keras.layers.Dense(1, activation = 'linear'),
      ])
model.summary()
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
#Fitting the model with Early stopping and restoring the best weights
early_stopping = keras.callbacks.EarlyStopping(patience = 10, min_delta = 0.001, restore_best_weights
=True )
```

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    batch_size=50,
    epochs=100,
    callbacks=[early_stopping],
    verbose=1,
)
#Evaluating the model on test set
from sklearn.metrics import mean_absolute_error,r2_score
predictions = model.predict(X_test)
print(f"MAE: {mean_absolute_error(y_test, predictions)}")


print(f"R2_score: {r2_score(y_test, predictions)}")
#XG Boost Regressor Model
from xgboost import XGBRegressor
model2 = XGBRegressor(n_estimators = 8000, max_depth=17, eta=0.1, subsample=0.7,
colsample_bytree=0.8)
model2.fit(X_train, y_train)
pred = model2.predict(X_test)
r2_score(y_test, pred)
mean_absolute_error(y_test, pred)
```

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	5632

dense_1 (Dense)	(None, 800)	410400
dropout (Dropout)	(None, 800)	0
dense_2 (Dense)	(None, 1024)	820224
dropout_1 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 1)	1025

==========

Total params: 1,237,281

Trainable params: 1,237,281

Non-trainable params: 0

MAE: 6.526929660850849

R2_score: 0.8712985185533424