**ELECTRICITY PRICE PREDICTION**

**PHASE – 4**



INTRODUCTION

Electricity price prediction is the process of forecasting the future cost of electricity in a specific region or market. It is a crucial task for energy companies, consumers, and policymakers because it helps in making informed decisions related to energy consumption, production, and investment. Electricity price prediction plays a vital role in optimizing energy consumption, managing electricity generation cost of electricity in a specific region or market. It is a crucial task for energy companies, consumers, and policymakers because it helps in making informed decisions related to energy consumption, production, and investment.

DATASET

**Dataset Link:** [**https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction**](https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction)

Overview of the process:

The following is an overview of the process of electricity

price prediction model by feature selection, model training, and

evaluation:

1. Prepare the data: This includes cleaning the data, removing

outliers, and handling missing values.

2. Perform feature selection: This can be done using a variety of

methods, such as correlation analysis, information gain, and recursive

feature elimination.

3. Train the model: There are many different machine learning

algorithms that can be used for electricity price prediction. Some popular

choices include linear regression, random forests, and gradient boosting

machines.

4. Evaluate the model: This can be done by calculating the mean

squared error (MSE) or the root mean squared error (RMSE) of the

model's predictions on the held-out test set.

5. Deploy the model: Once the model has been evaluated and found

to be performing well, it can be deployed to production so that it can be

used to predict the electricity prices.

FEATURE ENGINEERING

Feature engineering for electricity price prediction typically involves data preprocessing and transformation tasks that can be implemented in various programming languages like Python.

**1.Historical Price Data**:

Lagged Values: Include past electricity prices as features, which can help capture seasonality and trends.

df['price\_lag\_1'] = df['price'].shift(1)

df['price\_lag\_7'] = df['price'].shift(7)

Rolling Averages: Calculate rolling averages or moving averages of past prices over specific time windows.

df['price\_rolling\_7'] = df['price'].rolling(window=7).mean()

**2.Time-Related Features:**

Date and Time Components: Extract components like day of the week, month, hour, or season to account for time-based patterns and seasonality.

df['hour'] = df['timestamp'].dt.hour

df['day\_of\_week'] = df['timestamp'].dt.dayofweek

df['month'] = df['timestamp'].dt.month

**3.Weather Data:**

Temperature: Incorporate historical and forecasted temperature data since weather conditions can impact electricity demand and supply.

df['temperature'] = weather\_df['temperature']

Precipitation: Include information on rainfall or snowfall, which can affect energy consumption and generation.

**4.Demand and Supply Data:**

Electricity Demand: Historical and forecasted demand data can be essential in understanding price fluctuations.

Generation Capacity: Include data on available generation capacity and production from various sources (e.g., renewables, fossil fuels).

**5.Market Data:**

Fuel Prices: The cost of fuels used for electricity generation, such as natural gas or coal, can impact prices.

Transmission Line Data: Information about congestion on transmission lines and grid conditions.

**6.Economic Indicators:**

Economic Variables: Features related to economic factors like GDP, unemployment rates, or industrial production can affect electricity consumption.

**7.Event-Based Features:**

Holidays: Include information about holidays, as electricity consumption patterns often change during holidays.

Special Events: Consider major events or incidents that could impact electricity supply or demand (e.g., heatwaves, natural disasters).

**8.Technical Indicators:**

Moving Averages: Calculate various moving averages of prices or other relevant metrics.

Relative Strength Index (RSI), MACD, or other financial indicators that capture market sentiment.

**9.Categorical Features:**

Market Zones: If the electricity market is divided into zones, include the zone as a categorical feature.

Day Type: Encode weekdays, weekends, or holidays as categorical variables.

**10.Feature Scaling and Transformation**:

Normalize or standardize numerical features to ensure that they have similar scales.

Logarithmic or other transformations may be applied to features if their distributions are skewed.

df = pd.get\_dummies(df, columns=['day\_of\_week'])

**11.Feature Interactions:**

Create interaction features to capture relationships between different features. For example, the interaction between temperature and electricity demand.

df['temperature\_demand\_interaction'] = df['temperature'] \* df['demand']

**12.Feature Selection:**

Use techniques like feature importance from machine learning models or correlation analysis to select the most relevant features.

feature\_importances = model.feature\_importances\_

print("Feature Importances:", feature\_importances)

MODEL TRAINING

Training a model for electricity price prediction involves several steps, from data preparation to selecting the appropriate machine learning algorithm and fine-tuning the model.

**1.Data Collection and Preprocessing**:

Collect historical data on electricity prices, weather conditions, demand, and other relevant factors. Ensure the data is clean, organized, and stored in a suitable format (e.g., CSV or a database).

Handle missing values, outliers, and anomalies in the data.

**2.Data Splitting:**

Divide your data into training, validation, and testing sets. In time series data, it's important to respect the temporal order, with the training set containing earlier data and the validation/testing sets containing more recent data.

X\_train = series\_reshaped[:43800, :n\_steps]

X\_valid = series\_reshaped[43800:52560, :n\_steps]

X\_test = series\_reshaped[52560:, :n\_steps]

Y = np.empty((61134, n\_steps, 24))

for step\_ahead in range(1, 24 + 1):

Y[..., step\_ahead - 1] = series\_reshaped[..., step\_ahead:step\_ahead + n\_steps, 0]

Y\_train = Y[:43800]

Y\_valid = Y[43800:52560]

Y\_test = Y[52560:]

print("Shape of X\_train=", X\_train.shape, "\n",

" X\_valid=", X\_valid.shape, "\n",

" X\_test=", X\_test.shape, "\n",

" Y\_train=", Y\_train.shape, "\n",

" Y\_valid=", Y\_valid.shape, "\n",

" Y\_test=", Y\_test.shape)

**3.Feature Scaling:**

Normalize or standardize the numerical features to ensure they have similar scales. This can help the model converge faster and perform better**.**

**4.Select a Machine Learning Algorithm:**

* Linear Regression

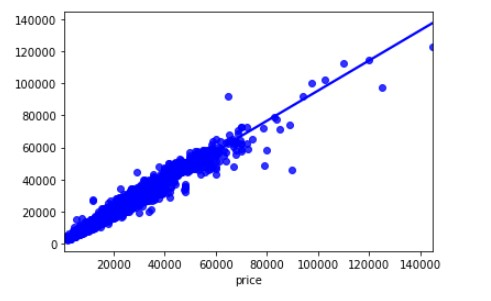
lr\_model = LinearRegression()

lr\_model.fit(train[['ForecastWindProduction', 'CO2Intensity']], train['Price'])

lr\_forecast = lr\_model.predict(test[['ForecastWindProduction', 'CO2Intensity']])

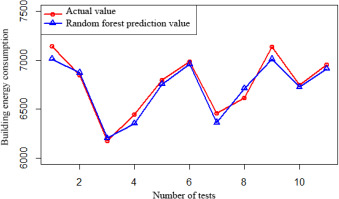
lr\_mse = mean\_squared\_error(test['Price'], lr\_forecast)

print(f'Linear Regression Mean Squared Error: {lr\_mse}')



* Random Forest

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=0) rf\_model.fit(train[['ForecastWindProduction', 'CO2Intensity']], train['Price']) rf\_forecast = rf\_model.predict(test[['ForecastWindProduction', 'CO2Intensity']]) rf\_mse = mean\_squared\_error(test['Price'], rf\_forecast) print(f'Random Forest Mean Squared Error: {rf\_mse}')



* ARIMA

arima\_model = ARIMA(train['Price'], order=(5,1,0)) arima\_result = arima\_model.fit(disp=0) arima\_forecast = arima\_result.forecast(steps=len(test)) arima\_mse = mean\_squared\_error(test['Price'], arima\_forecast) print(f'ARIMA Mean Squared Error: {arima\_mse}')

* PROPHET

prophet\_model = Prophet() prophet\_model.fit(train.rename(columns={'DateTime': 'ds', 'Price': 'y'})) future = prophet\_model.make\_future\_dataframe(periods=len(test)) prophet\_forecast = prophet\_model.predict(future) prophet\_mse = mean\_squared\_error(test['Price'], prophet\_forecast['yhat'][-len(test):]) print(f'Prophet Mean Squared Error: {prophet\_mse}')

**5.Model Training:**

Fit the selected model to the training dataset using the features and target variable (electricity prices).Perform hyperparameter tuning to optimize the model's performance. You can use techniques like grid search or random search.

MODEL EVALUATION

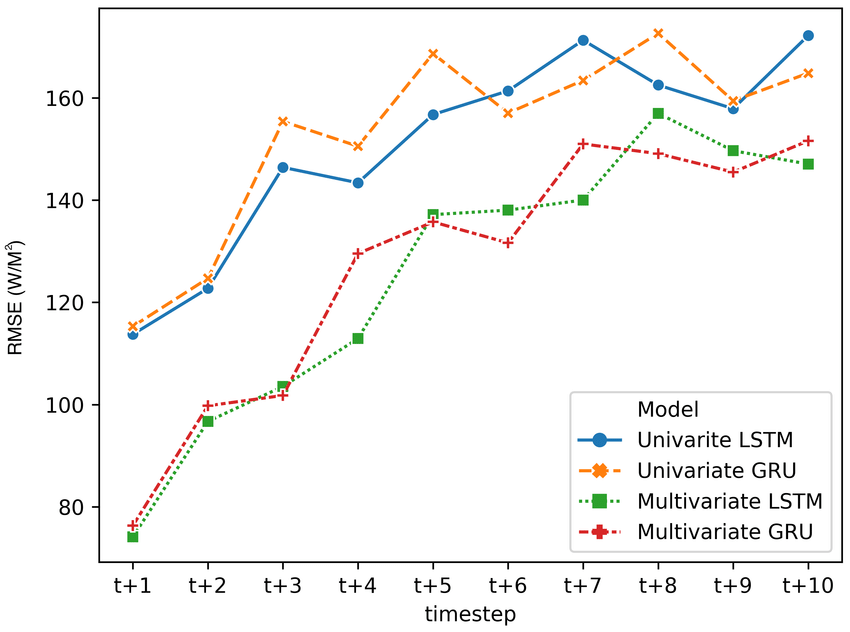
Evaluating the performance of an electricity price prediction model is essential to assess its accuracy and suitability for real-world applications. Several evaluation metrics and techniques can be used, depending on the nature of the problem and the specific goals of your prediction model.

* Mean Absolute Error (MAE):

MAE is the average absolute difference between the predicted and actual prices. It measures the average prediction error in the same units as the target variable.

**MAE = (1/n) \* Σ|actual - predicted|**

Lower MAE indicates better model performance.

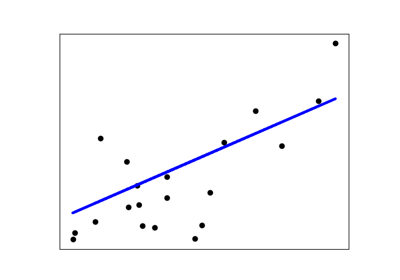


* Mean Squared Error (MSE):

MSE measures the average squared difference between predicted and actual prices. It penalizes larger errors more than MAE.

**MSE = (1/n) \* Σ(actual - predicted)^2**

RMSE (Root Mean Squared Error) is the square root of MSE and provides an interpretable metric in the same units as the target variable.



* Mean Absolute Percentage Error (MAPE):

MAPE calculates the percentage error for each prediction and then takes the average of these errors.

**MAPE = (1/n) \* Σ|(actual - predicted) / actual| \* 100%**

It provides insights into the relative error compared to the actual values.

* R-squared (R²) or Coefficient of Determination:

R² measures the proportion of the variance in the dependent variable (electricity prices) that is predictable from the independent variables (features) in your model.

R² ranges from 0 to 1, where 1 indicates a perfect fit, and 0 indicates no predictive power.

# Assuming you have predictions (y\_pred) and actual prices (y\_true)

y\_true = [100.0, 110.0, 120.0, 130.0, 140.0]

y\_pred = [98.0, 112.0, 118.0, 129.0, 142.0]

# Calculate MAE

mae = mean\_absolute\_error(y\_true, y\_pred)

print(f”Mean Absolute Error (MAE): {mae}")

# Calculate MSE

mse = mean\_squared\_error(y\_true, y\_pred)

print(f"Mean Squared Error (MSE): {mse}")

# Calculate RMSE

rmse = np.sqrt(mse)

print(f"Root Mean Squared Error (RMSE): {rmse}")

# Calculate R-squared (R2)

r2 = r2\_score(y\_true, y\_pred)

print(f"R-squared (R2): {r2}")

CONCLUSION

In this phase of our project, we've continued to build the electricity price prediction model by refining our features, training the model, and evaluating its performance. These pivotal steps bring us closer to our goal of accurate predictions and informed decision-making. As we proceed, the insights gained through feature engineering, model training, and evaluation will be instrumental in guiding our project to success. Our journey continues, and with each step, we move one step closer to reliable electricity price forecasting.