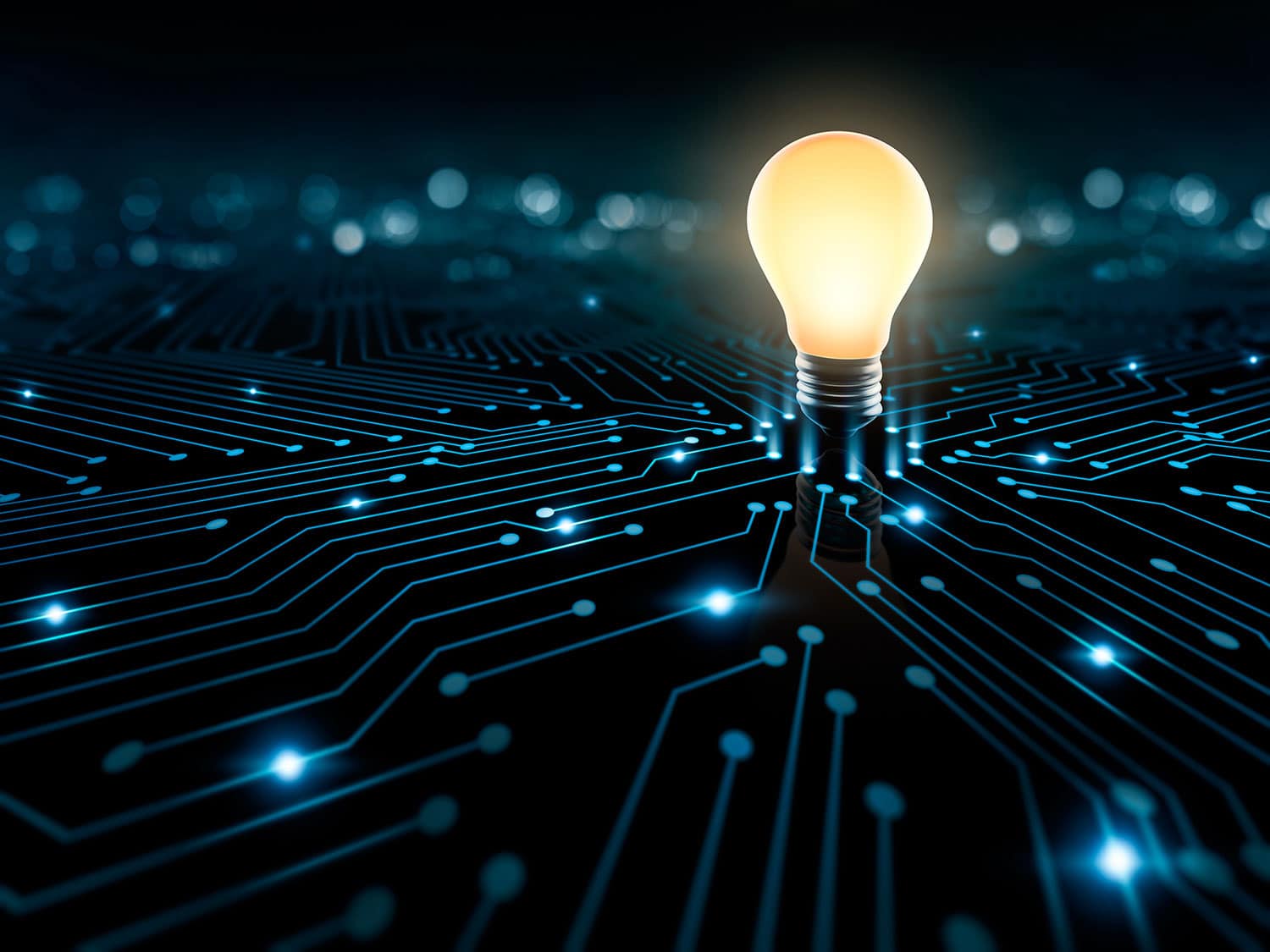
# **ELECTRICITY PRICES PREDICTION**

TEAM MEMBER

DEVIKA NAVAKUMAR

**Phase 2 Submission Document**

**Project** : ELECTRICITY PRICES PREDICTION



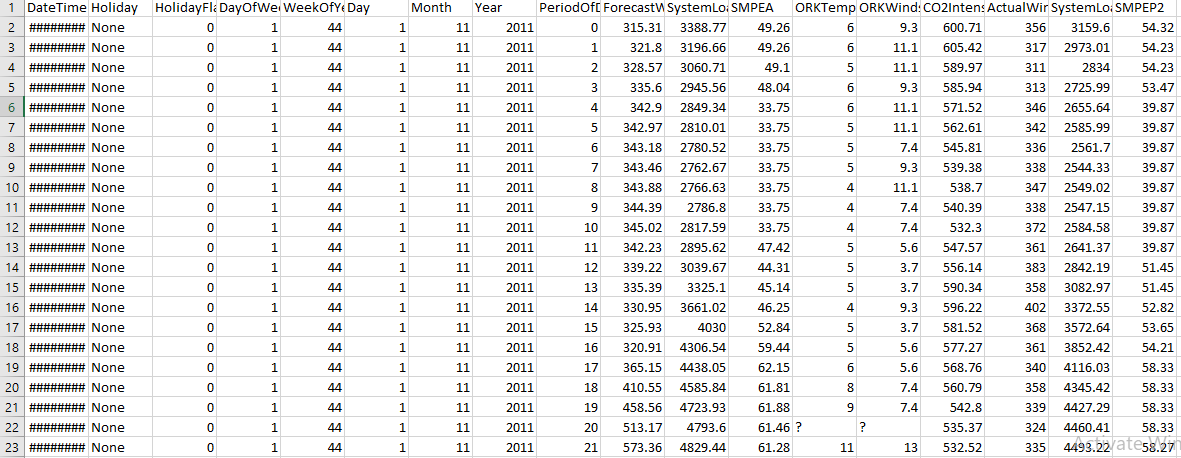
**Introduction:**

* **Electricity price prediction is the process of forecasting the future prices of electricity in a given market or region. This prediction is crucial for various stakeholders in the energy sector, including consumers, producers, and grid operators, as it helps them make informed decisions related to energy consumption, production, and grid management.**
* **This report presents an analysis of electricity price prediction using historical data and machine learning techniques. The goal is to forecast electricity prices accurately to support decision-making in the energy sector. We explore the data, build a predictive model, and evaluate its performance.**

**Data Source**

**A good data source for electricity prices prediction using machine learning should be Accurate, Complete & Accessible.**

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

****

**Data Collection and Preprocessing:**

* **Gather historical data on electricity prices.This data should include timestamp,location and the corresponding electricity prices.**
* **To prepare the data for modeling, we performed the following preprocessing steps;**
* **Handled missing data.**
* **Removed outliers.**
* **Converted date and time into suitable formats.**
* **Normalized prices for consistent scaling.**

**Exploratory Data Analysis (EDA):**

* **Visualize and analyze the dataset to gain insights into the relationships between variables.**
* **Identify correlations and patterns that can inform feature selection and engineering.**
* **Present various data visualizations to gain insights into the dataset.**
* **Explore correlations between features and the target variable (electricity prices).**
* **Discuss any significant findings from the EDA phase that inform feature selection.**

**Feature Engineering:**

* **Create new features or transform existing ones to capture valuable information.**
* **Explain the process of creating new features or transforming existing ones.**
* **Showcase domain-specific feature engineering, such as proximity scores or composite indicators.**
* **Emphasize the impact of engineered features on model performance.**
* **Create relevant features that could impact electricity prices,such as weather data(temperature,humidity),economic indicators,holidays or even events like major sports games.**

**Model Evaluation and Selection:**

* **The model's performance was evaluated using the test dataset. Key metrics included:**
* **Mean Squared Error (MSE)**
* **R-squared (R2)**
* **We opted for a machine learning approach to predict electricity prices. The selected model was known for its ability to handle time series data effectively.**

**Libraries Used:**

* **NumPy : It is a library that is used to work with arrays.**
* **Pandas : It is a python library used for working with data sets.**
* **Matplotlib : It is a python library used for plotting graphs with the help of other libraries like NumPy and Pandas.**

**Program:**

**Electricity Prices Prediction**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

# Load your historical electricity price data (replace 'electricity\_prices.csv' with your dataset)

data = pd.read\_csv('electricity\_prices.csv')

# Assuming your dataset has a 'timestamp' and 'price' column

timestamps = pd.to\_datetime(data['timestamp'])

prices = data['price']

# Visualize the data (optional)

plt.figure(figsize=(12, 6))

plt.plot(timestamps, prices)

plt.title('Electricity Prices Over Time')

plt.xlabel('Timestamp')

plt.ylabel('Price')

plt.show()

# Check for stationarity (using Dickey-Fuller test, or visually)

# If not stationary, perform differencing to make it stationary

d = 1

differenced\_prices = prices.diff(periods=d).dropna()

# Split the data into training and testing sets

train\_size = int(0.8 \* len(differenced\_prices))

train\_data, test\_data = differenced\_prices[:train\_size], differenced\_prices[train\_size:]

# Fit an ARIMA model

p, q = 1, 1 # Example values for the AR and MA orders (you should tune these)

model = ARIMA(train\_data, order=(p, d, q))

model\_fit = model.fit()

# Forecast future prices

forecast\_steps = len(test\_data)

forecast, stderr, conf\_int = model\_fit.forecast(steps=forecast\_steps)

# Invert differencing to get actual price forecasts

forecast = np.cumsum(forecast)

forecast = np.concatenate(([prices.iloc[train\_size - 1]], forecast))

forecast = forecast + prices.iloc[train\_size - 1]

# Visualize the predictions

plt.figure(figsize=(12, 6))

plt.plot(timestamps[train\_size:], test\_data, label='Actual Prices')

plt.plot(timestamps[train\_size:], forecast, label='Predicted Prices', color='red')

plt.title('Electricity Price Prediction with ARIMA')

plt.xlabel('Timestamp')

plt.ylabel('Price')

plt.legend()

plt.show()

Electricity prices prediction using python

**In[1] :**

import pandas as pd

import numpy as np

data = pd.read\_csv("C:\Users\MY PC\Downloads\electricity.csv")

print(data.head())

**Out[1] :**

**DateTime Holiday ... SystemLoadEP2 SMPEP2**

**0 01/11/2011 00:00 None ... 3159.60 54.32**

**1 01/11/2011 00:30 None ... 2973.01 54.23**

**2 01/11/2011 01:00 None ... 2834.00 54.23**

**3 01/11/2011 01:30 None ... 2725.99 53.47**

**4 01/11/2011 02:00 None ... 2655.64 39.87**

**[5 rows x 18 columns]**

**In[2] :**

data.info()

**Out[2] :**

**<class 'pandas.core.frame.DataFrame'>**

**RangeIndex: 38014 entries, 0 to 38013**

**Data columns (total 18 columns):**

**# Column Non-Null Count Dtype**

**--- ------ -------------- -----**

**0 DateTime 38014 non-null object**

**1 Holiday 38014 non-null object**

**2 HolidayFlag 38014 non-null int64**

**3 DayOfWeek 38014 non-null int64**

**4 WeekOfYear 38014 non-null int64**

**5 Day 38014 non-null int64**

**6 Month 38014 non-null int64**

**7 Year 38014 non-null int64**

**8 PeriodOfDay 38014 non-null int64**

**9 ForecastWindProduction 38014 non-null object**

**10 SystemLoadEA 38014 non-null object**

**11 SMPEA 38014 non-null object**

**12 ORKTemperature 38014 non-null object**

**13 ORKWindspeed 38014 non-null object**

**14 CO2Intensity 38014 non-null object**

**15 ActualWindProduction 38014 non-null object**

**16 SystemLoadEP2 38014 non-null object**

**17 SMPEP2 38014 non-null object**

**dtypes: int64(7), object(11)**

**memory usage: 5.2+ MB**

**In[3] :**

data.isnull().sum()

**Out[3] :**

**DateTime 0**

**Holiday 0**

**HolidayFlag 0**

**DayOfWeek 0**

**WeekOfYear 0**

**Day 0**

**Month 0**

**Year 0**

**PeriodOfDay 0**

**ForecastWindProduction 5**

**SystemLoadEA 2**

**SMPEA 2**

**ORKTemperature 295**

**ORKWindspeed 299**

**CO2Intensity 7**

**ActualWindProduction 5**

**SystemLoadEP2 2**

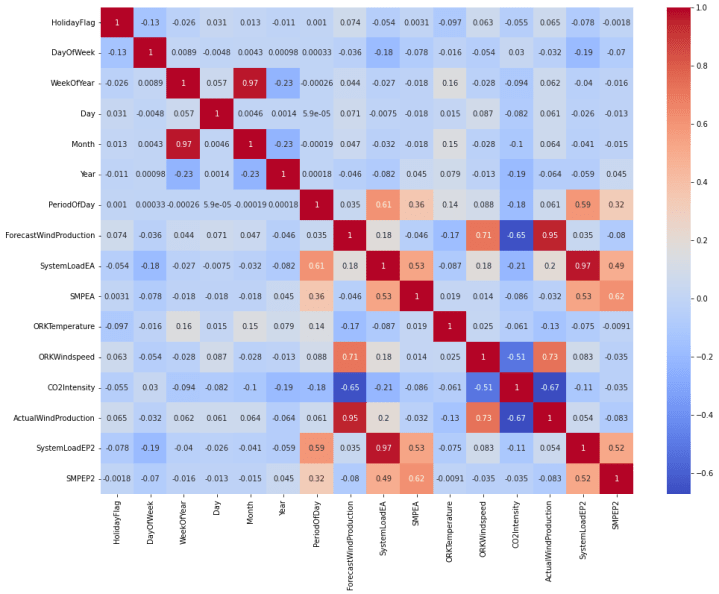
**SMPEP2 2**

**dtype: int64**

**In[4] :**

data = data.dropna()

**Out[4] :**



**In[5] :**

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()

model.fit(xtrain, ytrain)

**Out[5] :**

**RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0, criterion='mse',**

**max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,**

**max\_samples=None, min\_impurity\_decrease=0.0,**

**min\_impurity\_split=None, min\_samples\_leaf=1,**

**min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,**

**n\_estimators=100, n\_jobs=None, oob\_score=False,**

**random\_state=None, verbose=0, warm\_start=False)**

**In[6] :**

#features = [["Day", "Month", "ForecastWindProduction", "SystemLoadEA", "SMPEA", "ORKTemperature", "ORKWindspeed", "CO2Intensity", "ActualWindProduction", "SystemLoadEP2"]]

features = np.array([[10, 12, 54.10, 4241.05, 49.56, 9.0, 14.8, 491.32, 54.0, 4426.84]])

model.predict(features)

**Out[6] :**

**array([65.1696])**