ELECTRICITY PRICE PREDICTION

PHASE 5 : Document

TEAM MEMBERS:

S. NISHA (612721104064)

S. NIVETHA (612721104067)

P. PAVITHRA (612721104069)

S. PARAMESHWARI (61272

PROBLEM STATEMENT :

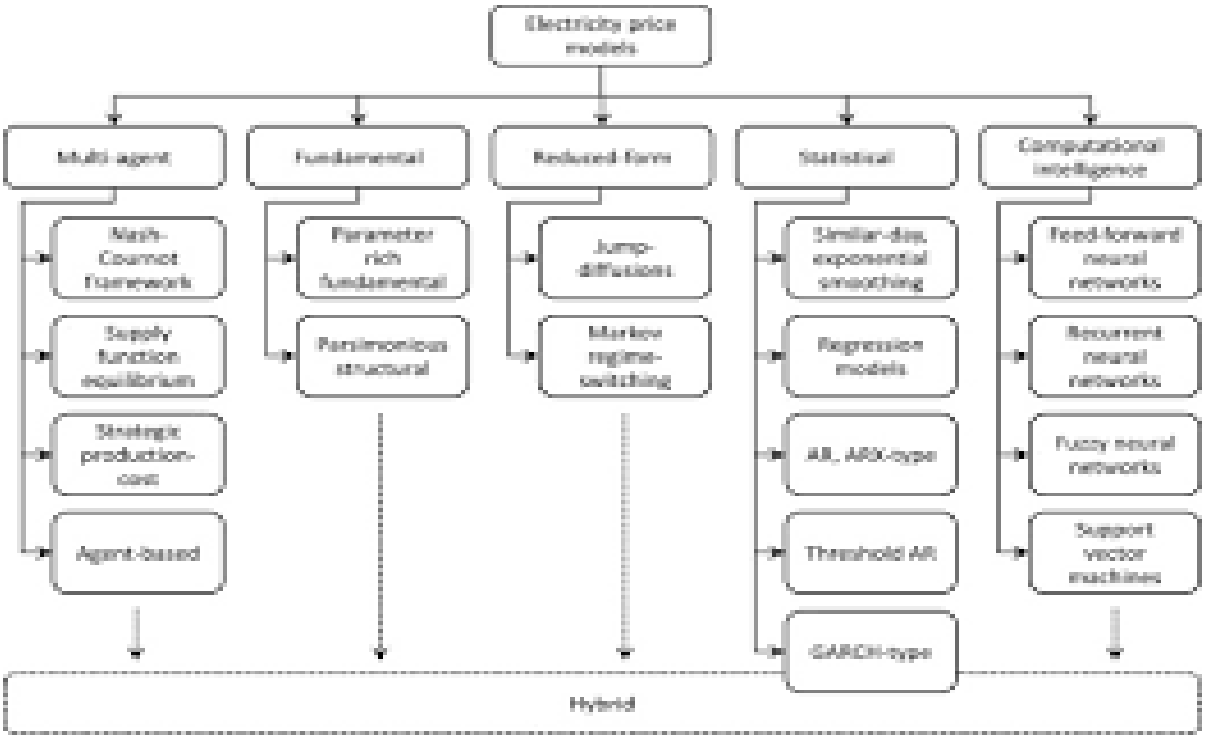
Electricity has attained a very important place in every household on this earth. The demand for electricity in household are derived from the electricity consuming kind of determinants used by the household. Against this backdrop, the present paper makes an analysis about the pattern of electricity consumption in the household sector of Malang city. The aim of this research are to prove the relationship electricity demand to variables which are included in this research, then to determine the level of utility that households get in electrical energy consumption.

Based on a demographic variables into the estimation, it can support the idea of Anderson (1973) and Matsukawa (2000) on Nababan's research (2015) which states the demand of energy by a household not only reflect the revenue and cost (price), but also reflect the demographic characteristics and social characteristics, such in where the household is located, as this may affect the function of household utilities.

This times at Malang City, that affects variables of demand for household electricity consumption can be developed more widely and variated. These variables may develop because of change changes in the neighborhood, the house building forms, and mainly because of changes in technology, etc.

The model equation of demand in this papers can used in analysis because, the changing of variables can explaning the changing of utility in electricity usage, then comparing with a household function. Including to non-business household or business household in Malang City.

DESIGH THINKING PROCESS:



Electricity consumption is one among widely studied section of computer architecture for more than decades. Electricity Adoption is one of the parameter in Machine Learning. It is one of the emerging field in the research. It keeps eye an eye on high accuracy without any kind of computation constraint.

Predicting electricity prices in the future involves a combination of engineering, modeling, and evaluation techniques. The electricity market is influenced by a multitude of factors, including supply and demand, weather conditions, fuel prices, government policies, and more. Here are some ideas for predicting electricity price.

FUTURE ENGINEERING:

Predicting future electricity prices is a complex task that often involves various factors such a supply and demand dynamics, weather conditions, energy policies and more. Engineers and data scientists use various techniques , includes time serious analysis , machine learning , and statistical modeling , to make these predictions . These models analyze historical price data , market conditions and other relevant variables to forecast future prices. Keep in mind that accurate predictions can be challenging due to the volatility of energy and unforeseen events.

PREDICTING FUTURE ELECTRICITY PRICES THROUGH MODELLING :

1. DATA COLLECTION:

   Gather historical data on electricity prices, which should include relevant factors like demand , generation mix , weather patterns , and market trends.

2. DATA PREPROCESSING :

   Clean and preprocess the data to handle missing values , outliers and format it for modelling.

3. FEATURE ENGINEERING :

Create or select relevant features that may impact electricity prices , such as time of day . weather conditions , holidays , or economic indicators.

4. MODEL SELECTION:

Choose an appropriate modelling technique , which could include time series analysis (e.g., ARIMA or GARCH models) or machine learning algorithms (e.g., regression, neural networks or decision trees).

5. MODEL TRAINING :

Use historical data to train the chosen model. This involves learning the relationships between the features and electricity prices.

6. MODEL EVALUATION :

Assess the models' performance using metrices like Mean ABSOLUTE ERRORE (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE) through cross- validation or holdout validation datasets.

7. HYPERPARAMETER TUNING:

Optimize the models' hyperparameter to improve its predictive accuracy.

8. FUTURE PRICE PREDICTION :

Apply the trained model to make predictions for future electricity precise based on new data

9. MONITORING AND UPDATES:

Continuously monitor the models performance and update it as new data becomes available or market conditions change .

Price forecasting is predicting a commodity/ product /service price by evaluating various factors like its characteristics, demand, seasonal ,trends ,other commodities prices etc.,

Analyzing Electricity Price Time Series Data using Python: Time Series Decomposition and Price Forecasting using a Vector Autoregression (VAR) Model

```
def retrieve _time _series(api, series _ID):
 """
Return the time series data frame, based on API and unique Series ID
api: API that we're connected to series _ID:
string. Name of the series that we want to pull from the EIA API
"""
 #Retrieve Data By Series ID
Series _search = api .data _by _series(series=series _ID)
 ##Create a pandas
data frame from the retrieved time series df = pd .Data Frame(series _search)
return df
 ###Execute in the main block
 #Create EIA API using your specific
  API key aPi _key = "YOR API KEY HERE"
 api = eia .API(api _key)
```

```
 #Pull the electricity price data
 series _ID='ELEC.PRICE.TX-ALL.M'
electricity _df= retrieve _time  _series(api, series _ID)
electricity _df .reset _index(level=0, in place=True)
#Rename the columns for easer analysis
electricity _df .rename(columns={'index' :'Date',
 electricity _df .columns[1]:'Electricity _Price'}, in  place=True)
```

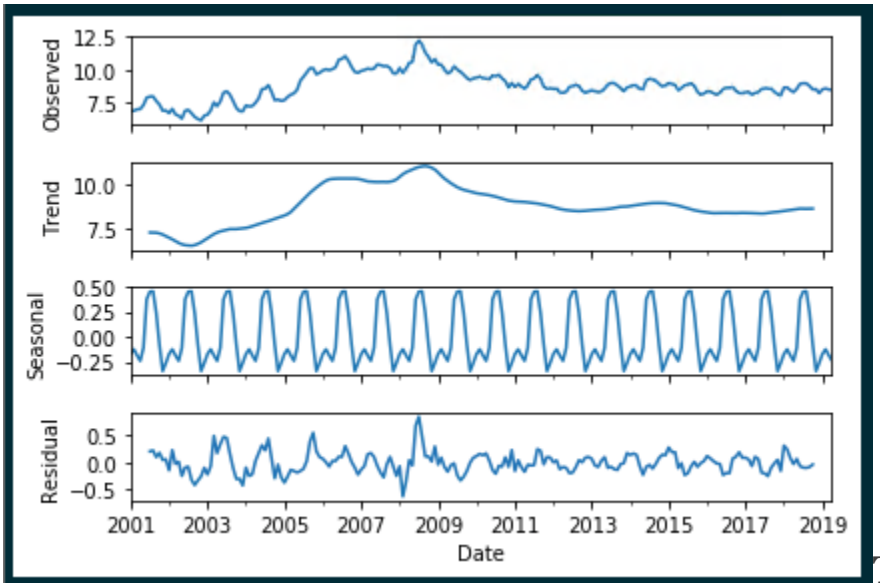
	Date	Electricity_Price
Date		
2001-01-01	2001-01-01	6.90
2001-02-01	2001-02-01	6.91
2001-03-01	2001-03-01	7.02
2001-04-01	2001-04-01	7.04
2001-05-01	2001-05-01	7.34
2001-06-01	2001-06-01	7.90
2001-07-01	2001-07-01	7.98
2001-08-01	2001-08-01	8.00
2001-09-01	2001-09-01	7.65
2001-10-01	2001-10-01	7.38
2001-11-01	2001-11-01	6.92
2001-12-01	2001-12-01	6.93
2002-01-01	2002-01-01	6.73
2002-02-01	2002-02-01	7.06
2002-03-01	2002-03-01	6.64
2002-04-01	2002-04-01	6.53
2002-05-01	2002-05-01	6.32
2002-06-01	2002-06-01	6.91

Snapshot of the time series data for electricity prices, pulled via the EIA API

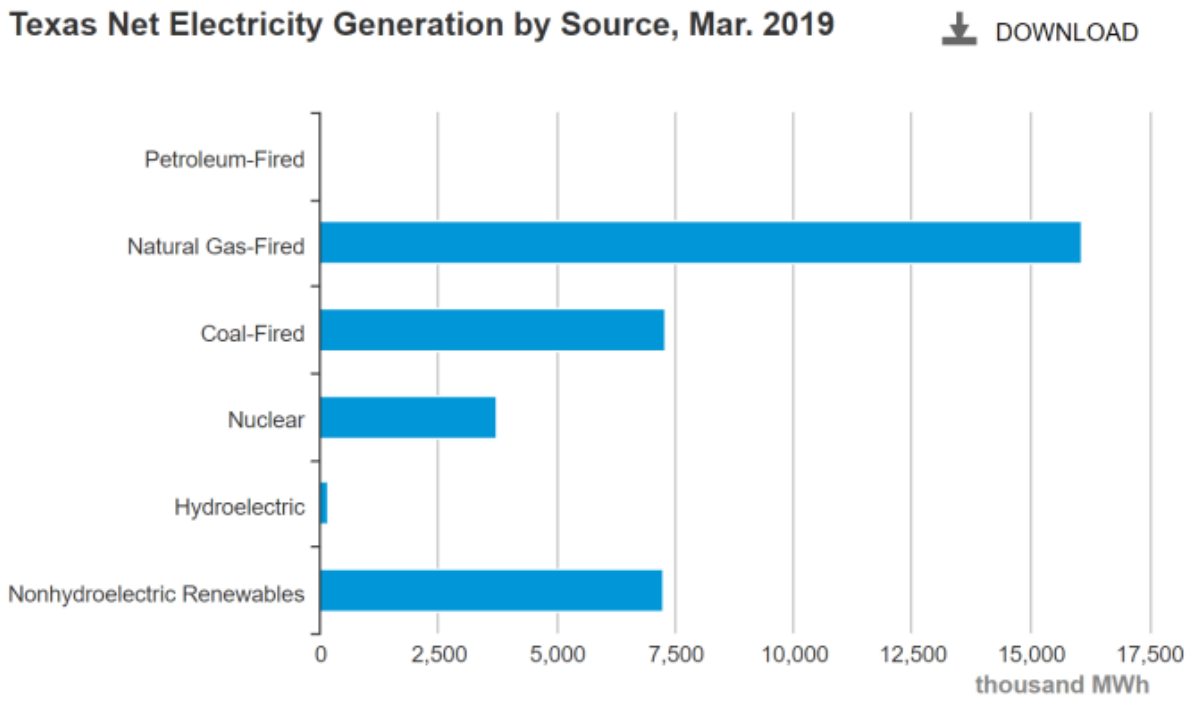
First, let's look at whether or not the monthly electricity data displays seasonality and a trend. To do this, we use the seasonal_decompose() function in the statsmodels.tsa.seasonal package. This function breaks down a time series into its core components: trend, seasonality, and random noise. The code and its outputs are displayed below:

```
def decompose_time_series(series):
  """
  Decompose a time series and plot it in the console
```

```
    Arguments:
        series: series. Time series that we want to decompose
    Outputs:
        Decomposition plot in the console
    """
    result = seasonal_decompose(series, model='additive')
    result.plot()
    pyplot.show()
#Execute in the main block
#Convert the Date column into a date object
electricity_df['Date']=pd.to_datetime(electricity_df['Date'])
#Set Date as a Pandas DatetimeIndex
electricity_df.index=pd.DatetimeIndex(electricity_df['Date'])
#Decompose the time series into parts
decompose_time_series(electricity_df['Electricity_Price'])
```



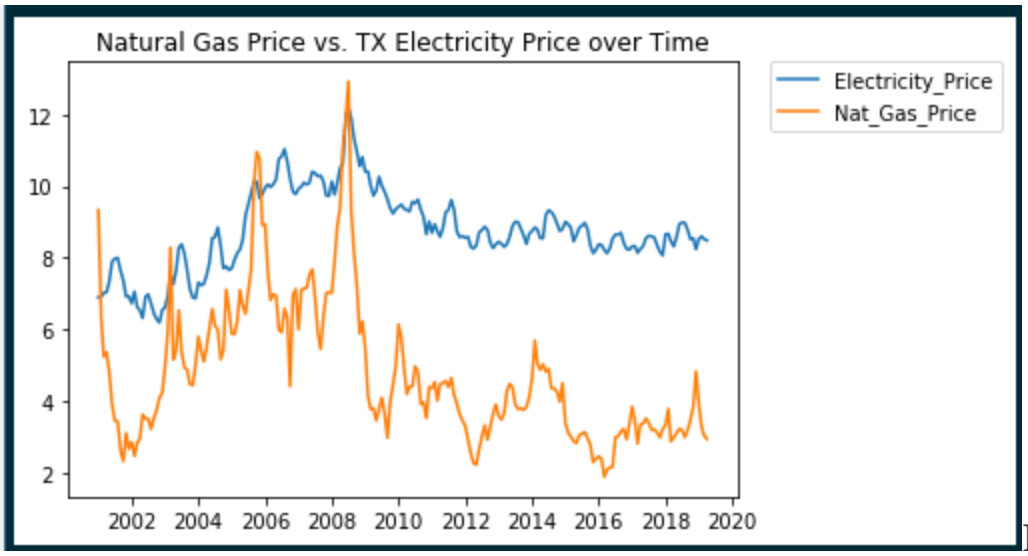
Time Series Decomposition: Monthly Electricity Prices in TX



Although we're analyzing electricity data going back to 2001 and the breakdown above is from March 2019, for the sake of simplicity we'll assume that natural gas has been one of the main sources of electricity generation in Texas for the past 15-20 years. Consequently, we're going to pull the time series of natural gas prices via the EIA API and compare it side-by-side to the TX electricity price time series:

```
#Pull in natural gas time series data
series_ID='NG.N3035TX3.M'
nat_gas_df=retrieve_time_series(api, series_ID)
nat_gas_df.reset_index(level=0, inplace=True)
#Rename the columns
nat_gas_df.rename(columns={'index':'Date',
        nat_gas_df.columns[1]:'Nat_Gas_Price_MCF'},
        inplace=True)
#Convert the Date column into a date object
nat_gas_df['Date']=pd.to_datetime(nat_gas_df['Date'])
#Set Date as a Pandas DatetimeIndex
nat_gas_df.index=pd.DatetimeIndex(nat_gas_df['Date'])
#Decompose the time series into parts
decompose_time_series(nat_gas_df['Nat_Gas_Price_MCF'])


#Merge the two time series together based on Date Index
master_df=pd.merge(electricity_df['Electricity_Price'], nat_gas_df['Nat_Gas_Price_MCF'],
                left_index=True, right_index=True)
master_df.reset_index(level=0, inplace=True)


#Plot the two variables in the same plot
```

```
plt.plot(master_df['Date'],
         master_df['Electricity_Price'], label="Electricity_Price")
plt.plot(master_df['Date'],
         master_df['Nat_Gas_Price_MCF'], label="Nat_Gas_Price")
# Place a legend to the right of this smaller subplot.
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Natural Gas Price vs. TX Electricity Price over Time')
plt.show()
```



Plotted Natural Gas Prices and Electricity Prices over Time

There appears to be more variation in the natural gas price time series (the highs are especially high and the lows are especially low), but for the most part, the trends in both time series appear very similar. In fact, it's highly likely we could estimate TX electricity price using natural gas price as a proxy.

There are a couple approaches that we can take to make a model stationary. They are as follows:

1. Difference the time series. Data differencing is, more or less, calculating the instantaneous velocity at each data point, or the amount the time series changes from one value to the next.
2. Transform the time series. This can be done by applying a log or power transformation to the data.

   Next, in order for a VAR model to work, the sampling frequency (i.e. daily, monthly, yearly data) needs to be the same. If it isn't, the data needs to be converted to the same frequency using imputation/linear interpolation, etc.

   In the code below, each time series is transformed using the numpy natural log function, and then differenced by one interval:

```
#Transform the columns using natural log

master_df['Electricity_Price_Transformed']=np.log(master_df['Electricity_Price'])

master_df['Nat_Gas_Price_MCF_Transformed']=np.log(master_df['Nat_Gas_Price_MCF'])



#Difference the data by 1 month

n=1

master_df['Electricity_Price_Transformed_Differenced'] = master_df['Electricity_Price_Transformed'] -
master_df['Electricity_Price_Transformed'].shift(n)

master_df['Nat_Gas_Price_MCF_Transformed_Differenced'] =
master_df['Nat_Gas_Price_MCF_Transformed'] - master_df['Nat_Gas_Price_MCF_Transformed'].shift(n)
```

   determine that a time series is stationary, the test must return a p-value of less than .05. In the Python code below, we run the Augmented Dickey-Fuller test on the transformed, differenced time series, determining that they are both stationary (the p-values returned are .000299 and 0 for the electricity and natural gas time series, respectively):

```
def augmented_dickey_fuller_statistics(time_series):
    """
    Run the augmented Dickey-Fuller test on a time series
    to determine if it's stationary.
    Arguments:
        time_series: series. Time series that we want to test
    Outputs:
        Test statistics for the Augmented Dickey Fuller test in
        the console
    """
    result = adfuller(time_series.values)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t%s: %.3f' % (key, value))
#Execute in the main block
#Run each transformed, differenced time series thru the Augmented Dickey Fuller test
print('Augmented Dickey-Fuller Test: Electricity Price Time Series')
 augmented_dickey_fuller_statistics(master_df['Electricity_Price_Transformed_Differenced'].dropna())
print('Augmented Dickey-Fuller Test: Natural Gas Price Time Series')
```

```
augmented_dickey_fuller_statistics(master_df['Nat_Gas_Price_MCF_Transformed_Differenced'].dropna()
)
```

```
Augmented Dickey-Fuller Test: Electricity Price Time Series
ADF Statistic: -4.397904
p-value: 0.000299
Critical Values:
        1%: -3.463
        5%: -2.876
        10%: -2.574
Augmented Dickey-Fuller Test: Natural Gas Price Time Series
ADF Statistic: -11.519473
p-value: 0.000000
Critical Values:
        1%: -3.461
        5%: -2.875
        10%: -2.574
```

Outputs for the Augmented Dickey-Fuller Test for the electricity price time series and the natural gas price time series, respectively

Now that we've made our two time series stationary, it's time to fit the data to a VAR model. The Python code below successfully builds the model and returns a summary of the results, where we use a 95/5 percent split for the training/validation sets:

```
#Conver the dataframe to a numpy array

master_array=np.array(master_df[['Electricity_Price_Transformed_Differenced',

                        'Nat_Gas_Price_MCF_Transformed_Differenced']].dropna())



#Generate a training and test set for building the model: 95/5 split
```

```
training_set = master_array[:int(0.95*(len(master_array)))]

test_set = master_array[int(0.95*(len(master_array))):]



#Fit to a VAR model

model = VAR(endog=training_set)

model_fit = model.fit()

#Print a summary of the model results

model_fit.summary()
```

```
Out[52]:
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Fri, 19, Jul, 2019
Time:                     18:31:26
----------------------------------------------------------------------
No. of Equations:         2.00000    BIC:                    -10.8008
Nobs:                     174.000    HQIC:                   -10.8655
Log likelihood:           461.353    FPE:                 1.82802e-05
AIC:                     -10.9097    Det(Omega_mle):      1.76658e-05
----------------------------------------------------------------------

Results for equation y1
==============================================================================
            coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const         0.001441         0.002454            0.587           0.557
L1.y1         0.223214         0.074353            3.002           0.003
L1.y2         0.041002         0.018430            2.225           0.026
==============================================================================

Results for equation y2
==============================================================================
            coefficient       std. error           t-stat            prob
------------------------------------------------------------------------------
const        -0.003560         0.010134           -0.351           0.725
L1.y1        -0.177961         0.307066           -0.580           0.562
L1.y2         0.038476         0.076113            0.506           0.613
==============================================================================

Correlation matrix of residuals
            y1        y2
y1    1.000000  0.215946
y2    0.215946  1.000000
```

VAR Model Summary, with y1=Electricity Price Time Series, and y2=Natural Gas Price Time Series

What we really want to focus on in the model summary above is the equation for y1, where y1 estimates the electricity prices in the state of Texas based on lagged values of itself and lagged natural gas prices.

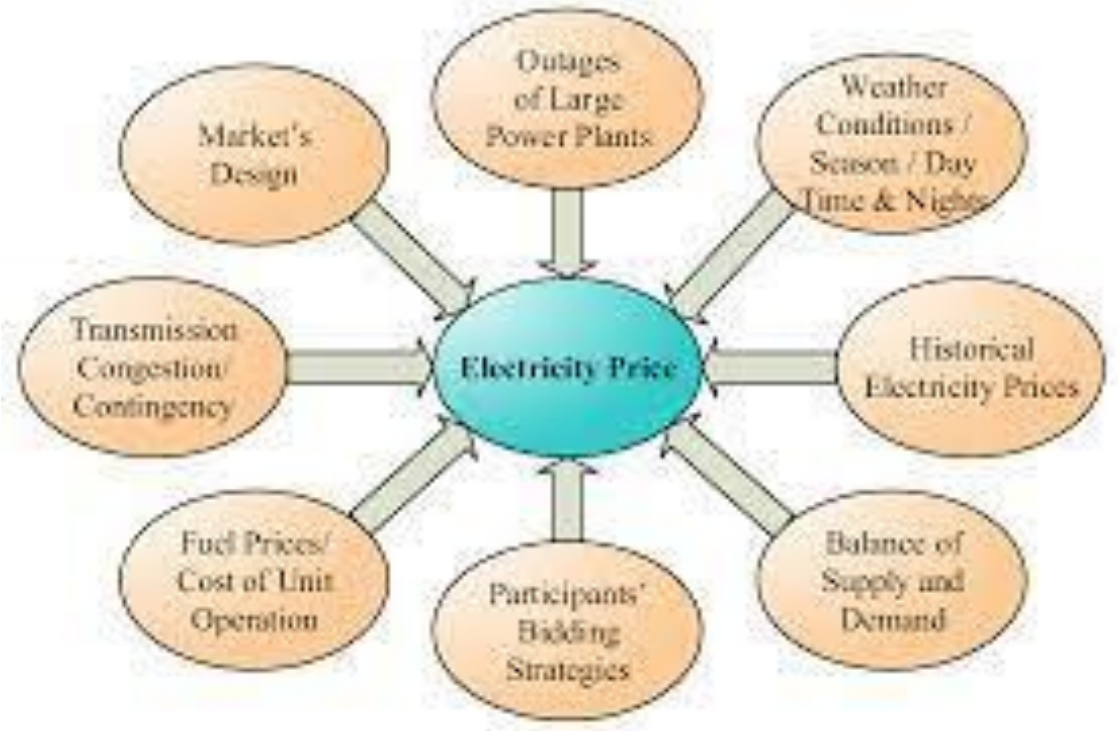
```
def calculate_model_accuracy_metrics(actual, predicted):

  """
```

```
Output model accuracy metrics, comparing predicted values

to actual values.

Arguments:

   actual: list. Time series of actual values.

   predicted: list. Time series of predicted values

Outputs:

   Forecast bias metrics, mean absolute error, mean squared error,

   and root mean squared error in the console

"""

#Calculate forecast bias

forecast_errors = [actual[i]-predicted[i] for i in range(len(actual))]

bias = sum(forecast_errors) * 1.0/len(actual)

print('Bias: %f' % bias)

#Calculate mean absolute error

mae = mean_absolute_error(actual, predicted)

print('MAE: %f' % mae)

#Calculate mean squared error and root mean squared error

mse = mean_squared_error(actual, predicted)

print('MSE: %f' % mse)

rmse = sqrt(mse)

print('RMSE: %f' % rmse)
```

```
#Execute in the main block

#Un-difference the data

for i in range(1,len(master_df.index)-1):

   master_df.at[i,'Electricity_Price_Transformed']= master_df.at[i-
1,'Electricity_Price_Transformed']+master_df.at[i,'Electricity_Price_Transformed_Differenced_PostProce
ss']



#Back-transform the data

master_df.loc[:,'Predicted_Electricity_Price']=np.exp(master_df['Electricity_Price_Transformed'])



#Compare the forecasted data to the real data

print(master_df[master_df['Predicted']==1][['Date','Electricity_Price', 'Predicted_Electricity_Price']])

#Evaluate the accuracy of the results

calculate_model_accuracy_metrics(list(master_df[master_df['Predicted']==1]['Electricity_Price']),

                    list(master_df[master_df['Predicted']==1 ['Predicted_Electricity_Price']))
```

The accuracy of forecasting short- or medium-term electricity loads and prices is critical in the energy market. The amplitude and duration of abnormally high prices and load spikes can be detrimental to retailers and production systems. Therefore, predicting these spikes to effectively manage risk is critical

.



An electricity price multi-bi-forecasting method is proposed using a new MIMO scheme that exports both the PF and IF results. A multivariable forecasting scheme is adopted using electricity price data combined with electric load data.

Load-forecasting methods can be classified into two broad categories: parametric methods and artificial intelligence–based methods. The artificial intelligence methods are further classified into neural network–based methods and fuzzy logic–based methods.

EVALUTION OF ELECTRICITY PRICE PREDICTION:

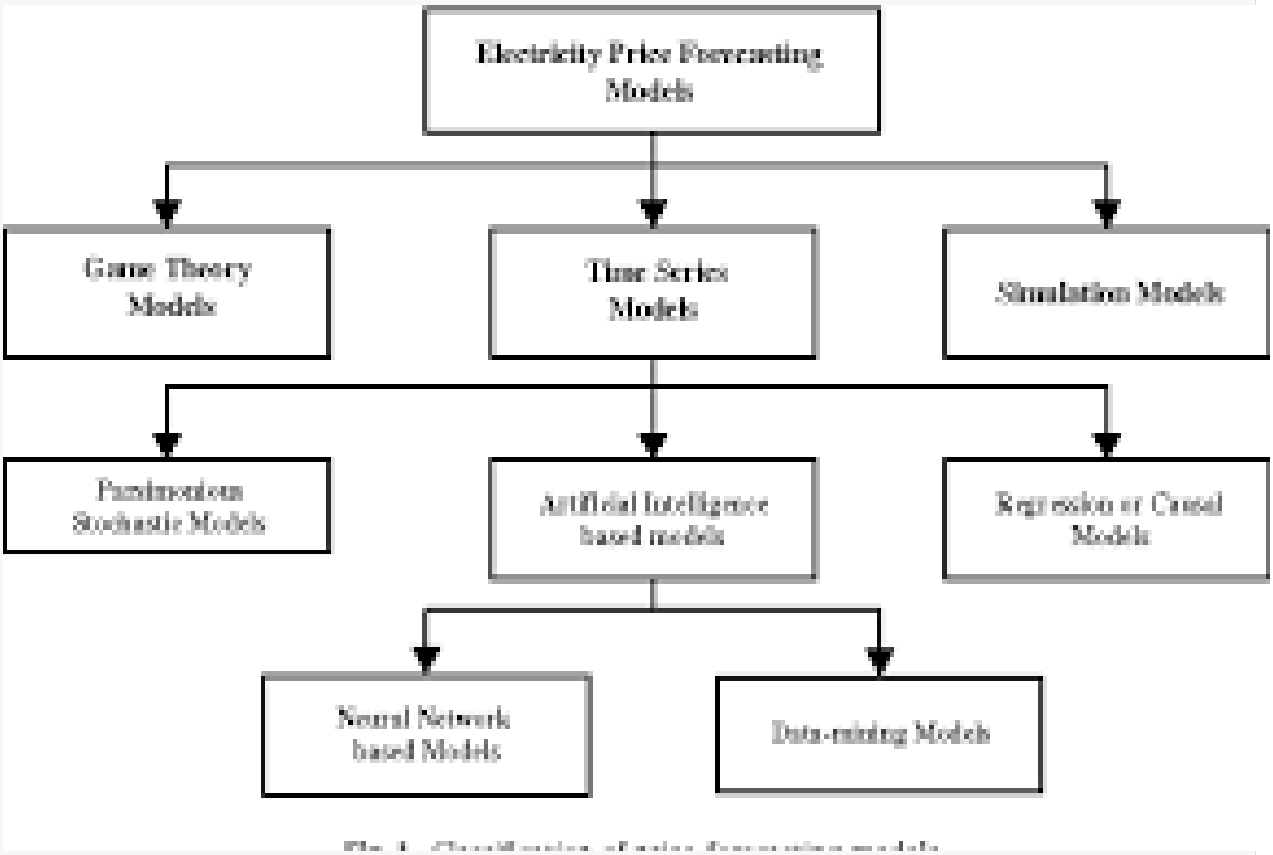
The evaluation of electricity price prediction models is crucial to assess their accuracy and usefulness. There are several commonly used metrics and methods to evaluate the performance of these models:

1. **Mean Absolute Error (MAE):** MAE measures the average absolute difference between predicted and actual prices. It gives an idea of the magnitude of errors.
2. **Mean Squared Error (MSE):** MSE measures the average of the squared differences between predicted and actual prices. It amplifies the impact of larger errors.
3. **Root Mean Squared Error (RMSE):** RMSE is the square root of MSE. It provides a measure of the average error in the same unit as the target variable (electricity price).
4. **Mean Absolute Percentage Error (MAPE):** MAPE calculates the average percentage difference between predicted and actual prices. It's useful for understanding the relative error.
5. **R-squared (R^2):** R-squared measures the proportion of the variance in the dependent variable (electricity price) that is predictable from the independent variables (features) in the model. A higher R-squared indicates a better fit. **Adjusted R-squared:** Adjusted R-squared is a modified version of R-squared that accounts for the number of predictors in the model. It helps prevent overfitting by penalizing the inclusion of unnecessary variables.
6. **Residual Analysis:** Examining the distribution of residuals (the differences between predicted and actual values) can reveal patterns or biases in the model's predictions. A good model should have residuals that are approximately normally distributed, centered around zero, and exhibit homoscedasticity (constant variance).
7. **Time Series Analysis:** For electricity price prediction, you may need to use time series-specific metrics, such as AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion), which help evaluate the quality of time series models.

8. **Cross-Validation:** Use techniques like k-fold cross-validation to assess how well the model generalizes to new data. This helps prevent overfitting and provides a more realistic evaluation of the model's performance.
9. **Out-of-Sample Testing:** Evaluate the model's performance on a separate test dataset that it has not seen during training. This simulates how the model will perform in real-world scenarios.

10. **Domain Expert Evaluation:** Consult domain experts who have knowledge of the electricity market to get qualitative feedback on the model's performance and its practical utility.
11. **Benchmarking:** Compare the model's performance with existing industry benchmarks or with simpler forecasting methods to gauge its improvement.
12. **Economic Impact Assessment:** In some cases, it might be useful to assess the economic impact of using the model for electricity price prediction, such as potential cost savings or revenue gains.

Keep in mind that the choice of evaluation metrics and methods can vary depending on the specific characteristics of your dataset, the type of model you're using (e.g., regression, time series, machine learning), and the goals of your electricity price prediction task. It's essential to consider the context and requirements of your application when selecting the mostappropriate evaluation approach.



Typical data sources for electricity price predictions include a combination of historical energy market data, environmental data, and other relevant information. Here are some common data sources:

1. **Historical Electricity Price Data:**
    - Hourly, daily, or sub-hourly electricity price data from energy markets, which is often publicly available. This data is

essential for training predictive models and understanding price trends.

2. **Load Data:**
   - Historical electricity load data, which represents the amount of electricity consumed over time. Load data is critical for understanding demand patterns and their impact on prices.
3. **Generation Data:**
   - Information on power generation sources, including data on the availability and output of various energy sources such as coal, natural gas, nuclear, renewables (e.g., wind and
4. **Weather Data:**
   - Meteorological data, including temperature, humidity, wind speed, and solar irradiance. Weather conditions can significantly affect electricity supply and demand.
5. **Market Data:**
   - Data related to market fundamentals, such as supply and demand forecasts, infrastructure changes, grid conditions, regulatory changes, and market rules.
6. **Fuel Price Data:**
   - Data on the prices of fuels used in electricity generation, such as coal, natural gas, and oil. These prices can impact the cost of generation and, subsequently, electricity prices.
7. **Transmission and Grid Data:**
   - Information about the state of the electricity grid, transmission constraints, and grid stability. This data can influence electricity prices due to transmission losses and congestion.
8. **Economic Indicators:**

- Economic data, such as GDP growth, employment rates, and industrial activity, which can provide insights into overall energy consumption and its relationship to economic conditions.

9. **Publicly Available Reports:**
   - Reports and publications from energy market operators, regulatory bodies, and industry associations can provide valuable insights into market conditions and forecasts.
10. **Energy Exchange Data:**
    - Data from energy exchanges and trading platforms that provide information on real-time or future energy prices, trading volumes, and market liquidity.
11. **Financial Data:**
    - Data related to financial markets, including futures and options prices, can provide additional insights into market expectations.

12. **Machine Learning Features:**
    - Engineered features specific to machine learning models, such as lagged variables, moving averages, and technical indicators, to capture patterns in the data.
    - Predictions and data related to renewable energy availability, especially for wind and solar generation. These forecasts are crucial for understanding the variable nature of renewables.
13. **Regulatory and Policy Information:**

- Information on government policies, subsidies, tax incentives, and regulations affecting the energy sector. Changes in policy can have a substantial impact on electricity prices.

14. **Market Sentiment and News Data:**

Social media sentiment analysis, news articles, and public sentiment data can help capture public perception, market sentiment, and speculative factors that influence energy prices.

Data sources may vary based on the specific goals of your electricity price prediction project, the region and market you are focused on, and the availability of data. Data from multiple sources is often combined and preprocessed to build predictive models

Program to calculate electricity bill

```c
/**
 * C program to calculate total electricity bill
 */

#include <stdio.h>

int main()
{
    int unit;
    float amt, total_amt, sur_charge;

    /* Input unit consumed from user */
    printf("Enter total units consumed: ");
    scanf("%d", &unit);


    /* Calculate electricity bill according to given conditions */
    if(unit <= 50)
    {
        amt = unit * 0.50;
    }
    else if(unit <= 150)
```

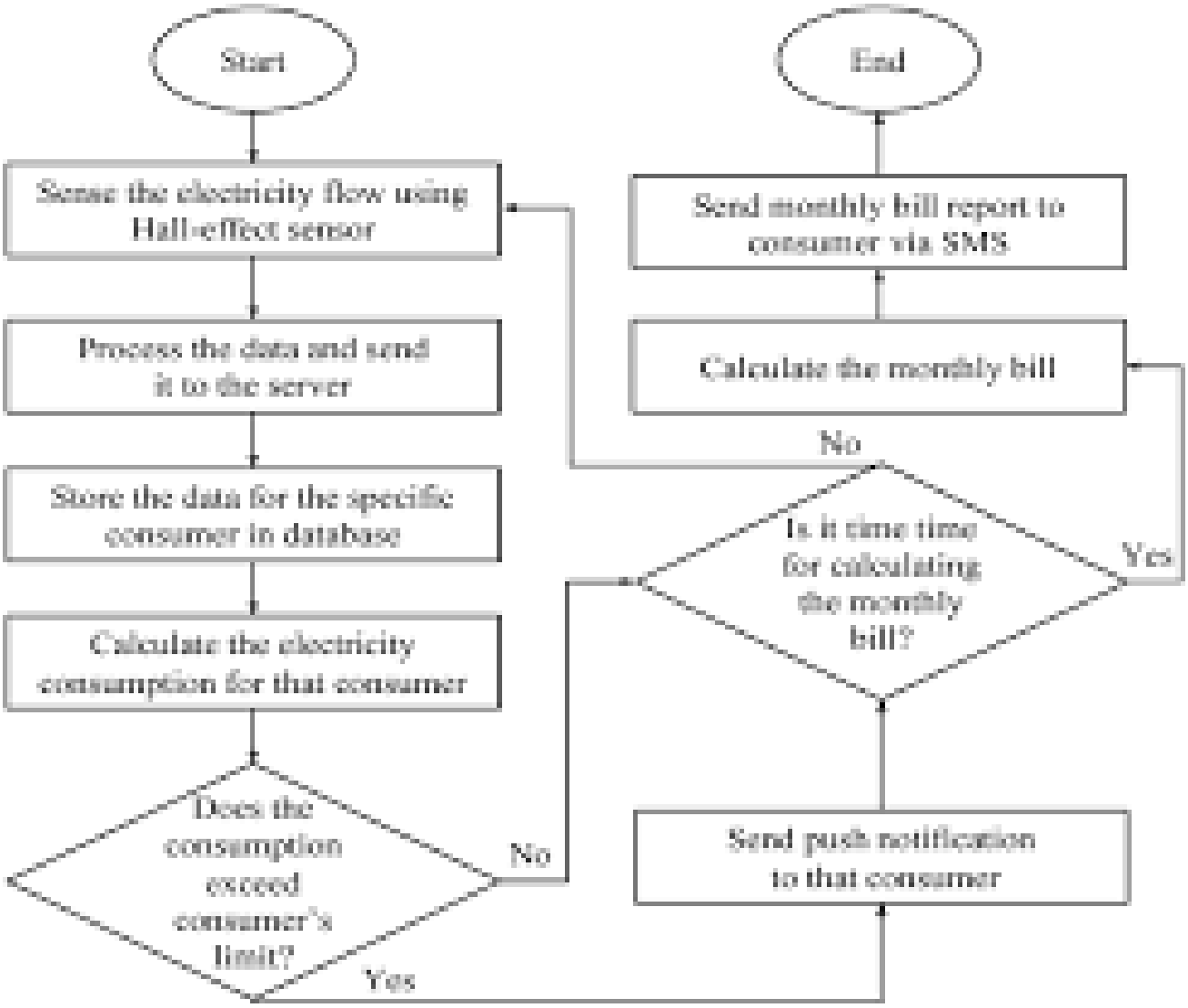
```
    {
        amt = 25 + ((unit-50) * 0.75);
    }
    else if(unit <= 250)
    {
        amt = 100 + ((unit-150) * 1.20);
    }
    else
    {
        amt = 220 + ((unit-250) * 1.50);
    }

    /*
     * Calculate total electricity bill
     * after adding surcharge
*/
    sur_charge = amt * 0.20;
    total_amt  = amt + sur_charge;

    printf("Electricity Bill = Rs. %.2f", total_amt);

    return 0;
}
```

Copy

Note: `%.2f` is used to print fractional values only up to 2 decimal places. You can also use `%f` to print fractional values normally up to six decimal

OUTPUT:

```
ter total units consumed: 150



Electricity Bill = Rs. 120.00
```



The electricity bill is calculated by multiplying the unit generated by the cost per unit. ie, if the unit generated is 190(say) and the cost per unit is 10Rs so the total electricity bill is 190×10 = 1900Rs

Program to calculate Electricity Bill

- Given an integer **U** denoting the amount of KWh units of electricity consumed, the task is to calculate the electricity bill with the help of the below charges:

  - *1 to 100 units –*
  - *100 to 200 units –*
  - *200 to 300 units –*
  - *above 300 units –*

  Examples:

  Input: *U = 250*
  Output: *3500*
  Explanation:
  *Charge for the first 100 units – 10*100 = 1000*
  *Charge for the 100 to 200 units – 15*100 = 1500*
  *Charge for the 200 to 250 units – 20*50 = 1000*
  Total Electricity Bill = 1000 + 1500 + 1000 = 3500
  Input: *U = 95*
  Output: *950*
  Explanation:
  *Charge for the first 100 units – 10*95 = 950*
  Total Electricity Bill = 950

  Approach 1: The idea is to identify the charge bar in which it falls and then calculate the bill according to the charges mentioned above. Below is the illustration of the steps:

- Check units consumed is less than equal to the 100, If yes then the total electricity bill will be:

-
- Else if, check that units consumed is less than equal to the 200, if yes then total electricity bill will be:

-
- Else if, check that units consumed is less than equal to the 300, if yes then total electricity bill will be:

-
- Else if, check that units consumed greater than 300, if yes then total electricity bill will be:

-

Below is the implementation of the above approach:

- C++
- Java
- Python3
- C#
- Javascript

```
// C++ implementation to calculate the
```

```
// electricity bill
#include<bits/stdc++.h>
using namespace std;


// Function to calculate the
// electricity bill
int calculateBill(int units)
{

    // Condition to find the charges
    // bar in which the units consumed
    // is fall
    if (units <= 100)
    {
        return units * 10;
    }
```

```
    else if (units <= 200)
    {
        return (100 * 10) +
               (units - 100) * 15;
    }
    else if (units <= 300)
    {
        return (100 * 10) +
               (100 * 15) +
               (units - 200) * 20;
    }
    else if (units > 300)
    {
        return (100 * 10) +
               (100 * 15) +
               (100 * 20) +
```

```
                (units - 300) * 25;
    }
    return 0;
}


// Driver Code
int main()
{
    int units = 250;
    cout << calculateBill(units);
}


// This code is contributed by spp____
```

Output

```
3500
```

Time Complexity: O(1)

Auxiliary Space: O(1)

Approach 2 : In this approach, we can use an array to store the different rate of charges and their respective range of units. This

approach can make the code more readable and easier to maintain. Here's how the code would look like:

- C++
- Java
- Python3
- C#
- Javascript

```
#include <bits/stdc++.h>

using namespace std;



const int n = 4;



// Function to calculate the electricity bill

int calculateBill(int units)

{

    int charges[n] = { 10, 15, 20, 25 };

    int range[n] = { 100, 100, 100, INT_MAX };

    int bill = 0;
```

```
    for (int i = 0; i < n; i++) {
        if (units <= range[i]) {
            bill += charges[i] * units;
            break;
        }
        else {
            bill += charges[i] * range[i];
            units -= range[i];
        }
    }
    return bill;
}


// Driver code
int main()
{
```

```
    int units = 250;

    cout << calculateBill(units);

    return 0;

}
```

Output
3500

TIME COMPLEXITY :

The time complexity of the **calculateBill** function is **O(n)**, where **n** is the number of ranges of units and their respective charges. This is because the function uses a **for** loop to iterate through the **range** and **charges** arrays, and for each iteration, it performs a constant amount of work (calculating the bill based on the units consumed).
Since n is a constant value, the time complexity can be considered as **O(1)** in the best-case scenario. The function takes a constant amount of time to run, regardless of the number of units consumed.
Auxiliary Space : The space complexity of this code is **O(n)**, where **n** is the number of rate of charges. This is because the program uses two arrays, **charges** and **range**, both of which have a size of n elements. The arrays take up **2 * n * sizeof(int) bytes** of memory. In this case, **n = 4**, so the total memory occupied by the arrays is **2 * 4 * sizeof(int)**.

The price of electricity depends on many factors. Predicting the price of electricity helps many businesses understand how much electricity they have to pay each year. The Electricity Price Prediction task is based on a case study where you need to predict the daily price of electricity based on the daily consumption of heavy machinery used by businesses. So if you want to learn how to predict the price of electricity, then this article is for you. In this article, I will walk you through the task of electricity price prediction with machine learning using **Python**.

Electricity Price Prediction (Case Study)

Suppose that your business relies on computing services where the power consumed by your machines varies throughout the day. You do not know the actual cost of the electricity consumed by the machines throughout the day, but the organization has provided you with historical data of the price of the electricity consumed by the machines. Below is the information of the **data** we have for the task of forecasting electricity prices:

1. Date Time: Date and time of the record
2. Holiday: contains the name of the holiday if the day is a national holiday
3. Holiday Flag: contains 1 if it's a bank holiday otherwise 0
4. Day Of Week: contains values between 0-6 where 0 is Monday
5. Week Of Year: week of the year
6. Day: Day of the date
7. Month: Month of the date
8. Year: Year of the date

9. Period Of Day: half-hour period of the day
10. Forcast Wind Production: forecasted wind production
11. System Load EA forecasted national load
12. SMPEA: forecasted price
13. ORK Temperature: actual temperature measured
14. ORK Windspeed: actual windspeed measured
15. CO2Intensity: actual C02 intensity for the electricity produced
16. Actual Wind Production: actual wind energy production
17. SystemLoadEP2: actual national system load
18. SMPEP2: the actual price of the electricity consumed (labels or values to be predicted)

So your task here is to use this data to train a machine learning model to predict the price of electricity consumed by the machines. In the section below, I will take you through the task of electricity price prediction with machine learning using Python.

Electricity Price Prediction using Python

I will start the task of electricity price prediction by importing the necessary Python libraries and the dataset that we need for this task:

```
1
import pandas as pd
2
import num py as np
3
data = pd.read_csv("https://raw.githubusercontent.com/amankharwal/Website-data/master/electricity.csv")
4
print(data .head())
```

```
      Date Time 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]
```

Data .inform()

```
<class 'pandas .core. frame .Data Frame'>
Range Index: 38014 entries, 0 to 38013
Data columns (total 18 columns):
 #   Column              Non-Null Count  D type
---  ------              --------------  -----
 0   Date Time           38014 non-null  object
 1   Holiday             38014 non-null  object
 2   Holiday Flag        38014 non-null  int64
 3   Day Of Week         38014 non-null  int64
 4   Week Of Year        38014 non-null  int64
 5   Day                 38014 non-null  int64
 6   Month               38014 non-null  int64
 7   Year                38014 non-null  int64
 8   Period Of Day       38014 non-null  int64
 9   Forecast Wind Production  38014 non-null  object
 10  System Load EA      38014 non-null  object
 11  SMPEA               38014 non-null  object
 12  ORK Temperature     38014 non-null  object
 13  ORK Windspeed       38014 non-null  object
 14  CO2Intensity        38014 non-null  object
 15  Actual Wind Production  38014 non-null  object
 16  SystemLoadEP2       38014 non-null  object
 17  SMPEP2              38014 non-null  object
D types: int64(7), object(11)
memory usage: 5.2+ MB
```

I can see that so many features with numerical values are string values in the dataset and not integers or float values. So before moving further, we have to convert these string values to float values:

```
data["Forecast Wind Production"] = pd.  to _numeric
(data["Forecast Wind Production"], errors= 'coerce')
data["System  load EA"] = pd. to_ numeric(data["System Load EA"],

errors= 'coerce')
data["SMPEA"] = pd.to_ numeric(data["SMPEA"], errors= 'coerce')
```

```
data["ORK Temperature"] = pd. to_ numeric(data["ORK Temperature"],
 errors= 'coerce')
data["ORK Windspeed"] = pd.to_ numeric(data["ORK Wind speed"],
errors= 'coerce')
data["CO2Intensity"] = pd.to_  numeric(data["CO2Intensity"],
errors= 'coerce')
data["Actual Wind  Production"] = pd. to _numeric
(data["Actual Wind Production"],
 errors= 'coerce')
data["SystemLoadEP2"] = pd .to _numeric(data["SystemLoadEP2"],
errors= 'coerce')
data["SMPEP2"] = pd.to _numeric(data["SMPEP2"], errors= 'coerce')
```

view rawelectricity1.py hosted with ♡ by GitHub

Now let's have a look at whether this dataset contains any null values or not:

```
data. Is  null().sum()
```

Date Time 0
Holiday 0
Holiday Flag 0
Day Of Week 0
Week Of Year 0
Day 0
Month 0
Year 0
Period Of Day 0
Forecast Wind Production 5
System Load EA 2
SMPEA 2
ORK Temperature 295
ORK Windspeed 299
CO2Intensity 7
Actual Wind Production 5
SystemLoadEP2 2
SMPEP2 2
D type: int64

So there are some columns with null values, I will drop all these rows containing null values from the dataset:

```
data = data. Dropna  ()
```

Now let's have a look at the correlation between all the columns in the dataset:

```
import seaborn as sns
import matplotlib .py plot as plt
correlations = data. corr(method='person')
Plt .figure(fig  size=(16, 12))
Sns  .heat map(correlations, c map="cool warm", cannot=True)
Plt .show()
```

Electricity Price Prediction Model

Now let's move to the task of training an electricity price prediction model. Here I will first add all the important features to x and the target column to y, and then I will split the data into training and test sets:

```
x = data[["Day", "Month", "Forecast Wind Production", "System Load EA",
          "SMPEA", "ORK Temperature", "ORK Windspeed", "CO2Intensity",
          "Actual Wind Production", "SystemLoadEP2"]]
y = data["SMPEP2"]
from sk learn  .model _selection import train _test _split
X train, x test, y train, y test = train _test _split(x, y,
                                                      Test _size=0.2,
                                                      Random _state=42)
```

As this is the problem of regression, so here I will choose the Random Forest regression algorithm to train the electricity price prediction model:

```
1
from sk  learn. ensemble import Random Forest Regressor
2
model = Random Forest  Regressor()
3
model. fit(x train, y train)
```

Random Forest Regressor(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)

Now let's input all the values of the necessary features that we used to train the model and have a look at the price of the electricity predicted by the model:

```
#features = [["Day", "Month", "Forecast Wind Production", "System Load EA", "SMPEA",
"ORK Temperature", "ORK Wind speed", "CO2Intensity", "Actual Wind Production",
"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)
```

array([65.1696])

So this is how you can train a machine learning model to predict the prices of electricity.

CONCLUSION:

Predicting the price of electricity helps a lot of companies to understand how much electricity expenses they have to pay every year. I hope you liked this article on the task of electricity price prediction with machine learning using Python. Feel free to ask your valuable questions in the comments section below.