ELECTRICITY PRICES PREDICTION

(GROUP 2-PHASE 4)

DEVELOPMENT PART 1

TEAM MEMBERS:

1. PRAVEENA.E

2. PAVITHRA.S

3. NISHA.J

Building an electricity price prediction model involves several steps, including data preprocessing, feature engineering, model training, and evaluation. Below, I'll outline each of these steps in more detail:

1. Data Preprocessing:

- Load the dataset: You can use the Pandas library to load the dataset from the provided link.

```python
import pandas as pd
data = pd.read_csv(&quot;your_dataset_path.csv&quot;)
```

- Explore the data: Get an understanding of the data by examining its structure, checking for missing values, and performing basic statistics and data visualization.

```python
data.info()
data.describe()
data.head()
```

- Handle missing values: Depending on the dataset, you may need to deal with missing values. You can choose to impute missing data or drop rows/columns with too many missing values.

- Convert date and time columns: If your dataset contains date and time information, consider converting them into a datetime format. This allows you to extract features like day of the week, month, hour, etc., which can be useful for modeling.

2. Feature Engineering:

- Create new features: Based on domain knowledge, create new features that could potentially be predictive of electricity prices. For example, you might want to create lag features, rolling statistics, or one-hot encode categorical variables.

- Feature selection: Not all features are equally relevant. Use techniques like correlation analysis or feature importance from machine learning models to select the most informative features.

3. Model Training:

- Split the data: Split your dataset into training and testing sets. You can use the `train_test_split` function from Scikit-learn.

```python
from sklearn.model_selection import train_test_split

X = data.drop(&#39;target_column_name&#39;, axis=1) # Features
y = data[&#39;target_column_name&#39;] # Target variable
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Choose a model: Select a machine learning model suitable for regression tasks. Common choices include Linear Regression, Decision Trees, Random Forests, Gradient Boosting, or Neural Networks.

- Train the model: Fit the selected model to the training data.

```python
from sklearn.linear_model import LinearRegression # Replace with your chosen model
model = LinearRegression() # Replace with your chosen model
model.fit(X_train, y_train)
```

4. Evaluation:

- Predict electricity prices: Use the trained model to make predictions on the test dataset.

```python
y_pred = model.predict(X_test)
```

- Evaluate the model: Use appropriate regression metrics to assess the model's performance. Common metrics include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2) score.

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

- Visualize results: Consider visualizing the actual vs. predicted values to get a better understanding of the model's performance.

5. Fine-Tuning and Deployment (Optional):

- Depending on the results, you may want to fine-tune hyperparameters or try different models to improve performance.

- If the model performs well, you can deploy it for real-world predictions.

Remember that building an effective prediction model often involves iterative steps, and you may need to try different models and feature engineering techniques to achieve the best results. Additionally, you can use libraries like Scikit-learn and TensorFlow/Keras (for deep learning) to streamline the modeling process.

PROGRAM:

```
In [1]: import pandas as pd
        data = pd.read_csv("C:\\Users\\lenovo\\Downloads\\Electricity\\New folder\\Electricity.csv", low_memory=False)
        data.head()
```

Out[1]:

	DateTime	Holiday	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay	ForecastWindProduction	SystemLoadEA	SMPEA	ORKTemperature	ORKWindspeed	CO2Int
0	01/11/2011 00:00	NaN	0	1	44	1	11	2011	0	315.31	3388.77	49.26	6.00	9.30	
1	01/11/2011 00:30	NaN	0	1	44	1	11	2011	1	321.80	3196.66	49.26	6.00	11.10	
2	01/11/2011 01:00	NaN	0	1	44	1	11	2011	2	328.57	3060.71	49.10	5.00	11.10	
3	01/11/2011 01:30	NaN	0	1	44	1	11	2011	3	335.60	2945.56	48.04	6.00	9.30	
4	01/11/2011 02:00	NaN	0	1	44	1	11	2011	4	342.90	2849.34	33.75	6.00	11.10	

```
In [2]: data.info()
```

```
<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                 1536 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.describe()
```

Out[3]:

	HolidayFlag	DayOfWeek	WeekOfYear	Day	Month	Year	PeriodOfDay
count	38014.000000	38014.000000	38014.000000	38014.000000	38014.000000	38014.000000	38014.000000
mean	0.040406	2.997317	28.124586	15.739412	6.904246	2012.383859	23.501105
std	0.196912	1.999959	15.587575	8.804247	3.573696	0.624956	13.853108
min	0.000000	0.000000	1.000000	1.000000	1.000000	2011.000000	0.000000
25%	0.000000	1.000000	15.000000	8.000000	4.000000	2012.000000	12.000000
50%	0.000000	3.000000	29.000000	16.000000	7.000000	2012.000000	24.000000
75%	0.000000	5.000000	43.000000	23.000000	10.000000	2013.000000	35.750000
max	1.000000	6.000000	52.000000	31.000000	12.000000	2013.000000	47.000000

In [21]:
```
data=data[['ForecastWindProduction',
       'SystemLoadEA', 'SMPEA', 'ORKTemperature', 'ORKWindspeed',
       'CO2Intensity', 'ActualWindProduction', 'SystemLoadEP2', 'SMPEP2']]
```

In [22]:
```
data.isin(['?']).any()
```

Out[22]:
```
ForecastWindProduction    True
SystemLoadEA              True
SMPEA                     True
ORKTemperature            True
ORKWindspeed              True
CO2Intensity              True
ActualWindProduction      True
SystemLoadEP2             True
SMPEP2                    True
dtype: bool
```

dtype: bool

In [23]:
```
for col in data.columns:
    data.drop(data.index[data[col] == '?'], inplace=True)
```

In [24]:
```
data=data.apply(pd.to_numeric)
data=data.reset_index()
data.drop('index', axis=1, inplace=True)
```

In [25]:
```
data.corrwith(data['SMPEA']).abs().sort_values(ascending=False)
```

Out[25]:
```
SMPEP2                    1.000000
SMPEA                     0.618158
SystemLoadEP2             0.517081
SystemLoadEA              0.491096
ActualWindProduction      0.083434
ForecastWindProduction    0.079639
ORKWindspeed              0.035436
CO2Intensity              0.035055
ORKTemperature            0.009087
dtype: float64
```

In [30]:
```
X=data.drop('SMPEA', axis=1)
y=data['SMPEA']
```

In [31]:
```
x_train, x_test, y_train, y_test=train_test_split(X,y, test_size=0.2, random_state=42)
```

In [32]:
```
from sklearn.metrics import mean_squared_error
linear_model=LinearRegression()
linear_model.fit(x_train, y_train)
linear_predict=linear_model.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, linear_predict)))
```

```
23.286827374230228
```

```
In [34]: some_data=x_test.iloc[50:60]
         some_data_label=y_test.iloc[50:60]
         some_predict=linear_model.predict(some_data)
         pd.DataFrame({'Predict':some_predict,'Label':some_data_label})
```

Out[34]:

	Predict	Label
4093	126.718172	122.47
22310	40.939104	33.78
8034	45.506013	60.91
35027	57.615353	61.36
23685	67.683748	62.09
268	66.142103	49.76
35261	56.396384	30.93
11905	63.495308	61.50
30903	71.314507	82.26
608	227.258421	119.70

```
In [37]: from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

         mae = mean_absolute_error(some_data_label, some_predict)

         print(mae)

19.84973669824718

In [38]: mse = mean_squared_error(some_data_label, some_predict)
         print(mse)

1296.140388436656

In [39]: rmse = mean_squared_error(some_data_label, some_predict, squared=False)

36.00194978659706

In [40]: r2 = r2_score(some_data_label, some_predict)

In [41]: print(r2)

-0.45791346385622456

In [ ]:
```

THANK YOU

SUBMITTED BY-

PRAVEENA.E