

Electricity Prices Prediction

Problem Definition:

- The problem at hand is to predict electricity prices accurately.
- Accurate electricity price predictions are crucial for various stakeholders, including consumers, energy providers, and policymakers.
- Accurate predictions can help consumers make informed decisions about their energy usage, energy providers optimize their operations, and policymakers implement effective energy policies.
- The challenge lies in developing a reliable model that can forecast electricity prices with a high degree of accuracy.

Design Thinking:

1. Data Source:

- Identify reliable sources for historical electricity price data. These sources might include government agencies, energy market websites, or utility companies.
- Consider collecting additional relevant data, such as weather information, demand patterns, supply data, and regulatory changes, as these factors can influence electricity prices.

2. Data Preprocessing:

- Understand the quality and availability of historical electricity price data. Identify missing values, outliers, and data anomalies.
- Convert timestamps into a usable format for time-series analysis.
- Normalize or scale the data if necessary to ensure all features are on the same scale.
- Split the data into training, validation, and test sets to evaluate your model properly.

3. Feature Engineering:

- Explore creative ways to engineer features, such as creating lag features for past prices or using domain knowledge to construct relevant variables.
- Use domain knowledge to engineer features that could be relevant, such as holidays, energy market events, or economic indicators.

4. Model Selection:

- Experiment with various machine learning algorithms suitable for time-series forecasting, such as:
- ARIMA (AutoRegressive Integrated Moving Average)
- LSTM (Long Short-Term Memory) networks
- Gradient Boosting algorithms (e.g., XGBoost, LightGBM)
- Explore various model architectures and hyperparameters to see which ones align best with your problem.

5. Model Training:

- Create a preliminary version of your model and train them on a portion of your dataset.
- Implement techniques like cross-validation to fine-tune model parameters and prevent overfitting.
- Explore rolling-window approaches for time-series data to simulate real-time forecasting.

6. Evaluation:

- Assess the model's performance using appropriate evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).
- Consider using quantile regression to predict price percentiles, which can be crucial for risk management in energy trading.
- Establish monitoring systems to track the model's performance over time, and set up alerts for significant deviations or degradation in accuracy.

7.

Innovation for Electricity price prediction

Introduction Of Electricity price prediction:

Electricity price prediction is the process of using various techniques, such as statistical analysis, machine learning, and data modeling, to forecast the future cost of electricity. This prediction is vital for consumers, businesses, and energy companies to make informed decisions regarding energy consumption, cost management, and resource allocation. It considers factors like historical data, supply and demand dynamics, weather conditions, renewable energy integration, and consumer behavior to provide accurate estimates of future electricity prices. Accurate price predictions support cost savings, efficient energy management, and the integration of renewable energy sources into the grid.

About Dataset:

Dataset link:<https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction>

This is the link about the Electricity price prediction ,this link is got form www.kaggle.com/data.

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.

This dataset contains 38015 rows and 18 columns. By which It is useful for electricity price prediction.

Details about columns:

- **Date & Time:** Date and time of the record

- Holiday: contains the name of the holiday if the day is a national holiday ■ Holiday Flag: contains 1 if it's a bank holiday otherwise 0
- Day Of Week: contains values between 0-6 where 0 is Monday ■ Week Of Year: week of the year
- Day: Day of the date
- Month: Month of the date ■ Year: Year of the date
- Period Of Day: half-hour period of the day
- For cast Wind Production: forecasted wind production ■ System Load EA: forecasted national load
- SMPEA: forecasted price
- ORK Temperature: actual temperature measured ■ ORK Windspeed: actual windspeed measured
- CO2Intensity: actual CO2 intensity for the electricity produced

- Actual Wind Production: actual wind energy production ■ SystemLoadEP2: actual national system load
- SMPEP2: the actual price of the electricity consumed (labels or values to be predicted)

Details of libraries:

Reducing electricity costs typically involves optimizing energy consumption, improving energy efficiency, and sometimes integrating renewable energy sources. Python offers a wide range of libraries and tools that can be used in various aspects of electricity price reduction. Here are some key libraries and tools:

Pandas: Pandas is essential for data manipulation and analysis. You can use it to preprocess and analyze electricity consumption data, perform data-driven decisions, and forecast energy usage.

How to install:

Step 1: Open your command prompt

or terminal. Step 2: Check if pip is

installed:

Step 3: pip install pandas

Step 4: Wait for the installation to finish.

Step 5: Verify the installation by running a Python script

that includes: `import pandas as pd`

`print(pd.version)`

If you see the version number, pandas is successfully installed.

Num Py : NumPy is crucial for numerical computing. It can be used for mathematical calculations related to electricity usage and cost reduction strategies.

How to install:

Step 1: Open your command prompt

or terminal. Step 2: Check if pip is

installed:

Step 3: pip install NumPy

Step 4: Wait for the installation to finish.

Step 5: Verify the installation by running a Python script that includes:

`import numpy as np`

`print(np.__version__)`

If you see the version number, pandas is successfully installed.

Matplotlib and Seaborn: These libraries are excellent for data visualization. They can help you present insights and trends in a clear and understandable way.

How to install:

Step 1: Open your command prompt

or terminal. Step 2: Check if pip is

installed:

Step 3: pip install matplotlib

Step 4: Wait for the installation to finish.

Step 5: Verify the installation by running a Python script that includes:

```
import matplotlib
```

```
print(matplotlib.__version__)
```

If you see the version number, pandas is successfully installed.

is successfully installed. (If you see an error, it means that the installation failed. Try again.)

How to install:

Step 1: Open your command prompt

or terminal. Step 2: Check if pip is

installed:

Step 3: pip install scikit-learn

Step 4: Wait for the installation to finish.

Step 5: Verify the installation by running a Python script that includes:

```
import sklearn
```

```
print(sklearn.__version__)
```

If you see the version number, pandas is successfully installed.

How to train and test dataset:

The train-test split is used to estimate the performance of machine learning algorithms that are

applicable for prediction-based Algorithms/Applications. This method is a fast and easy procedure to perform such that we can compare our own machine learning model results to machine results.

By default, the Test set is split into 30 % of actual data and the training set is split into 70% of the actual data.

We need to split a dataset into train and test sets to evaluate how well our machine learning model performs. The train set is used to fit the model, and the statistics of the train set are known. The second set is called the test data set, this set is solely used for predictions.

Syntax:

```
train_test_split(*arrays, test_size=None,  
train_size=None, random_state=None, shuffle=True,  
stratify=None)
```

code:

```
# import  
modules  
import  
pandas  
as pd  
from sklearn.linear_model import  
LinearRegression from  
sklearn.model_selection import  
train_test_split  
  
# read the dataset  
df = pd.read_csv("Electricity price.csv")  
  
# get the  
location  
s X =  
df.iloc[:,  
:-1]  
y = df.iloc[:, -1]  
  
# split the dataset  
X_train, X_test, y_train, y_test =  
train_test_split( X, y,  
test_size=0.05, random_state=0)
```

Metrics used for the accuracy check for electricity price prediction:

Evaluation metrics are tied to machine learning tasks, There are different metrics for the tasks classification and regression. Some metrics, like precision-recall, are useful for multiple tasks. Classification and regression are examples of supervised learning, which constitutes a machine learning applications. Using different metrics for performance evaluation, we should to improve our model's overall predictive power before we roll it out for production on unseen Without doing a proper evaluation of the Machine Learning model by using different metrics, and only depending on accuracy, can lead to a problem when the respective model is deployed on unseen data and may end in poor predictions.

Classification Metrics in Machine Learning

Classification is about predicting the class labels given input data. In binary classification, there are only two possible output classes(i.e., Dichotomy). In

multiclass classification, more than two possible classes can be present. I'll focus only on binary classification

Accuracy

Accuracy simply measures how often the classifier correctly predicts. We

can define accuracy as the ratio of the number of correct predictions and the total number of predictions.

The four commonly used metrics for evaluating classifier performance are:

1. **Accuracy:** The proportion of correct predictions out of the total predictions.
2. **Precision:** The proportion of true positive predictions out of the total positive predictions (precision = true positives / (true positives + false positives)).
3. **Recall (Sensitivity or True Positive Rate):** The proportion of true positive predictions out of the total actual positive instances (recall = true positives / (true positives + false negatives)).
4. **F1 Score:** The harmonic mean of precision and recall, providing a balance between the two metrics (F1 score = $2 * ((\text{precision} * \text{recall}) / (\text{precision} + \text{recall}))$).

1.

Libraries for Electricity price prediction

```
import csv
```

```
from sklearn.model_selection import
```

```
train_test_split
```

```
Numpy
```

```
Pandas
```

```
from sklearn.preprocessing import
```

```
onehotEncoder
```

```
from sklearn.preprocessing import
```

```
LabelEncoder
```

```
import matplotlib.pyplot
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.linear_model import
```

```
linearRegression
```

```
import standard scaler from sklearn.ensemble
```

```
import randomforestclassifier
```

Importing the Data Set:

```
import csv
```

```
file = open('data.csv',
```

```
'rU') reader =
```

```
csv.reader(file)
```

```
matrix=[]
```

```
for row in reader:
```

```
matrix.append(row)
```

```
print(matrix)
```

Handling the Missing Data:

In this we use the sklearn.preprocessing import imputer it is used to fill the missing data.

The imputer is an estimator used to fill the missing values in datasets. For numerical values, it uses mean, median, and constant. For categorical values, it uses the most frequently used and constant value.

Encode categorical Data:

For Encoding the categorical data using the **one-hot encoding** and it is used to convert a categorical variables into a binary matrix.

This is commonly used in machine learning algorithms that cannot work with categorical data directly.

We can use libraries like scikit-learn or keras to perform one-hot encoding easily.

Splitting the Data Set:

In this we are using the **from sklearn.model_selection import train_test_split**

library for splitting the data set as the test data and the trained data from the data set.

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

The dataframe gets divided into X_train, X_test, y_train and y_test. **X_train and y_train** sets are used for training and fitting the model. The **X_test and y_test** sets are used for testing the model if it's predicting the right outputs/labels.

Feature Scaling:

Sklearn library offers us with **StandardScaler()** function to standardize the data values into a standard format.

The library is **from sklearn.preprocessing import standard scaler**.

Syntax:

```
object = StandardScaler()
```

```
object.fit_transform(data)
```

Csv File:



data.csv

Feature engineering

- Explore creative ways to engineer features, such as creating lag features for past prices or using domain knowledge to construct relevant variables.

- Use domain knowledge to engineer features that could be relevant, such as holidays, energy market events, or economic indicators.

Model Training:

- Create a preliminary version of your model and train them on a portion of your dataset.
- Implement techniques like cross-validation to fine-tune model parameters and prevent overfitting.
- Explore rolling-window approaches for time-series data to simulate real-time forecasting.

code:

```
import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

from sklearn.model_selection import
train_test_split from sklearn.metrics import
mean_squared_error

from sklearn.ensemble import RandomForestRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.linear_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor

df=pd.read_csv("/content/data.csv",
low_memory=False) df.head()
```

Output:

1	44	1	11	2011	1	321.80	3198.66	49.26	6	11.1	605.42
1	44	1	11	2011	2	328.57	3080.71	49.10	5	11.1	589.97
1	44	1	11	2011	3	335.60	2945.56	48.04	6	9.3	585.94
1	44	1	11	2011	4	342.90	2849.34	33.75	6	11.1	571.52

All model trining:

Code:

```
x_train, x_test, y_train, y_test=train_test_split(X,y, test_size=2, random_state=42)
```

```
#LinearRegression
```

```
linear_model=LinearRe
```

```
gression()
```

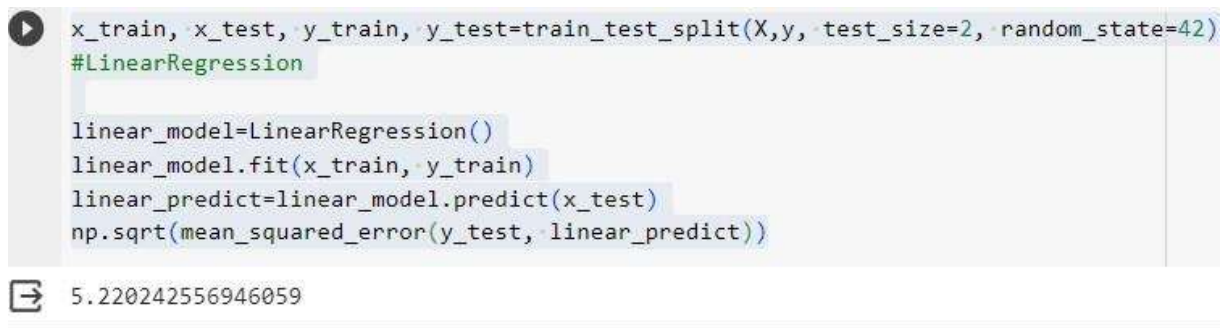
```
linear_model.fit(x_train,
```

```
y_train)
```

```
linear_predict=linear_model.predict(x_test)
```

```
np.sqrt(mean_squared_error(y_test, linear_predict))
```

Output:



A screenshot of a Jupyter Notebook cell. The code is as follows:

```
x_train, x_test, y_train, y_test=train_test_split(X,y, test_size=2, random_state=42)
#LinearRegression

linear_model=LinearRegression()
linear_model.fit(x_train, y_train)
linear_predict=linear_model.predict(x_test)
np.sqrt(mean_squared_error(y_test, linear_predict))
```

The output of the cell is displayed below the code:

```
5.220242556946059
```

KNeighborsRegressor:

Code:

```
#KNeighborsRegressor
```


```
knn_model=KNeighborsRegressor
```


```
() knn_model.fit(x_train, y_train)
```

```
knn_predict=knn_model.predict(x_test)
```

```
print(np.sqrt(mean_squared_error(y_test, knn_predict)))
```

Output:

```
 #KNeighborsRegressor
|
knn_model=KNeighborsRegressor()
knn_model.fit(x_train, y_train)
knn_predict=knn_model.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, knn_predict)))
```

 13.74732321581187

Evaluation:

- Assess the model's performance using appropriate evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).
- Consider using quantile regression to predict price percentiles, which can be crucial for risk management in energy trading.
- Establish monitoring systems to track the model's performance over time, and set up alerts for significant deviations or degradation in accuracy.

Code:

```
some_data=x_test.iloc[50:60]
some_data_label=y_test.iloc[50:60]
]
some_predict=forest_model.predict(some_data)

pd.DataFrame({'Predict':some_predict,'Label':some_data_label})
```

Output:

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

	Predict	Label
4093	149.6479	188.32
22310	36.0076	33.46
8034	59.2229	62.01
35027	75.8247	49.69
23685	73.0210	69.25
268	57.1129	56.21
35261	46.4761	46.64
11905	71.9873	78.52
30903	75.5883	82.36
608	110.6629	415.99