Phase 3: Report submission

electricity price prediction:

introduction:

The efficient and accurate prediction of electricity prices is of paramount importance for various stakeholders in the energy sector, including power utilities, consumers, and energy traders. The ability to anticipate future electricity prices enables better decision-making, resource allocation, and cost management. In this project, we embark on the journey of building a robust electricity prices prediction model, a vital tool for optimizing energy-related strategies.

Our endeavor begins with the critical first step of loading and preprocessing the historical electricity price dataset. This dataset encapsulates a wealth of information, containing key attributes such as date, hour, temperature, electricity demand, and electricity price. Through meticulous data preparation, we aim to lay the foundation for a powerful predictive model that can unlock insights into the dynamic electricity market.

Dataset:

```
Date,Hour,Temperature,Demand,Price
2023-10-01,1,65,500,0.12
2023-10-01,2,64,520,0.13
2023-10-01,3,63,510,0.15
2023-10-01,4,62,505,0.14
2023-10-01,5,61,490,0.16
2023-10-01,6,60,480,0.18
2023-10-01,7,62,490,0.19
2023-10-01,8,65,530,0.21
2023-10-01,9,70,580,0.23
2023-10-01,10,75,600,0.25
```

Dataset explanation:

- **Date**: Date of the recorded data.
- **Hour**: Hour of the day.
- **Temperature**: Temperature at the given hour.
- **Demand**: Electricity demand at the given hour.
- **Price**: Electricity price at the given hour.

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

# Step 1: Load the dataset

data = pd.read_csv('electricity_prices.csv')

# Step 2: Data Exploration and Understanding

summary_stats = data.describe()

correlations = data.corr()

# Step 3: Data Cleaning

missing_values = data.isnull().sum()

# Handle outliers (if necessary)

# Example: Remove rows with electricity price outliers

data = data[data['Price'] < 1000]

# Step 4: Feature Selection/Engineering

selected_features = data[['Temperature', 'Demand']]

# Create new features (if needed)

# Example: Extract day of the week from the date

data['Day_of_Week'] = pd.to_datetime(data['Date']).dt.dayofweek

# Step 5: Splitting the Data

X = selected_features

y = data['Price']

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

 # Step 6: Data Scaling/Normalization

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

Code explnation:

 # Step 1: Load the dataset

import pandas as pd

data = pd.read_csv('electricity_prices.csv')

- **Explanation (Step 1):** In this step, we import the pandas library and use it to read the dataset from a CSV file named "electricity_prices.csv." This loads the data into a DataFrame, which is a versatile data structure for handling structured data.

 # Step 2: Data Exploration and Understanding

summary_stats = data.describe()

correlations = data.corr()

Explanation (Step 2): Here, we perform data exploration. The describe() method generates summary statistics (e.g., mean, standard deviation) for each numerical column in the dataset. The corr() method computes the correlation matrix, providing insights into how different columns are related.**Explanation (Step 2):** Here, we perform data exploration. The describe() method generates summary statistics (e.g., mean, standard deviation) for each numerical column in the dataset. The corr() method computes the correlation matrix, providing insights into how different columns are related.

 # Step 3: Data Cleaning

missing_values = data.isnull().sum()

Handle outliers (if necessary)

Example: Remove rows with electricity price outliers

```
data = data[data['Price'] < 1000]
```

Explanation (Step 3): In this step, we check for missing values using the `isnull().sum()` function, which counts the number of missing values in each column. We also demonstrate how to handle outliers by filtering the dataset. In this example, rows with electricity prices exceeding 1000 have been removed.

```
# Step 4: Feature Selection/Engineering

selected_features = data[['Temperature', 'Demand']]

# Create new features (if needed)

# Example: Extract day of the week from the date

data['Day_of_Week'] = pd.to_datetime(data['Date']).dt.dayofweek
```

Explanation (Step 4): We perform feature selection by creating a subset of the dataset with only the relevant features, in this case, "Temperature" and "Demand." Additionally, we demonstrate feature engineering by adding a new feature, "Day_of_Week," by extracting the day of the week from the "Date" column.

```
# Step 5: Splitting the Data

from sklearn.model_selection import train_test_split

X = selected_features

y = data['Price']

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

Explanation (Step 5): Here, we split the dataset into a training set (`X_train`, `y_train`) and a testing set (`X_test`, `y_test`). We use the `train_test_split` function from scikit-learn to do this. The `test_size` parameter determines the proportion of data used for testing (20% in this example), and `random_state` ensures reproducibility.

```
# Step 6: Data Scaling/Normalization

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

Explanation (Step 6): In this step, we perform data scaling using the StandardScaler from scikit-learn. Scaling is crucial for many machine learning algorithms. It ensures that all features have a similar scale, preventing some features from dominating others during modeling. fit_transform scales the training data, and transform applies the same scaling to the testing data.

This code prepares your dataset for machine learning, covering essential steps in data preprocessing, such as cleaning, feature selection/engineering, and scaling. It is ready for use in training predictive models to forecast electricity prices.

Conclusion:

In this project, we embarked on the critical initial phase of building an electricity prices prediction model by loading and preprocessing a historical dataset. We meticulously prepared the data to enable subsequent machine learning modeling. The process involved data loading, exploration, and cleaning, as well as feature selection and engineering. We further split the data for training and testing, and applied scaling for uniformity. This rigorous data preparation lays the groundwork for the development of a robust predictive model, vital for informed decision-making in the dynamic energy sector.