PHASE 4

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	3196.66	49.26	6	11.1	605.42
1	44	1	11	2011	2	328.57	3060.71	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=LinearRegression()

linear_model.fit(x_train, y_train)

linear_predict=linear_model.predict(x_test)

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

Output:

```
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))
```

```
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