ELECTRICITY PRICES PREDICTION:

project tiles : pre predicted for electricity prices prediction

Phase 4: Development part 2:

FEATURE ENGINEERING:

1.Temporal Features:

Time of Day: Electricity prices often follow a daily pattern. Create features indicating the time of day, such as morning, afternoon, evening, and night.

Day of Week: Prices might vary based on the day of the week. Create binary features for weekdays and weekends.

Month/Season: Some regions have seasonal price variations. Create features for different months or seasons.

- *Public Holidays: Prices might be different on public holidays. Include binary features indicating whether a day is a public holiday or not.

2. Historical Price Data:

Lagged Prices: Use past electricity prices as features. Lagged features (prices from previous hours or days) can capture trends and patterns.

- *Price Change: Calculate the difference between the current price and the price at the same time on the previous day.

3Weather-Related Features:

Temperature: High or low temperatures can affect electricity demand. Include current temperature or temperature forecasts.

Precipitation: Rainy or snowy weather might affect electricity generation and consumption.

Wind Speed: Relevant for areas where wind power is a significant energy source.

4. Economic Indicators:

GDP Data: Economic indicators might influence electricity demand.

Unemployment Rate: High unemployment might correlate with lower industrial activity and thus lower electricity demand.

5. Regulatory Factors:

Policy Changes: Changes in energy policies or regulations can affect prices. Include binary features for major policy changes.

6. Energy Production and Consumption:

Renewable Energy Production: If applicable, include data on renewable energy production (solar, wind) as these sources can influence overall supply.

Electricity Consumption: Historical data on electricity consumption can be a useful predictor.

7. Market Data:

Stock Market Indices: Economic health might be correlated with electricity demand.

Crude Oil Prices: For areas where electricity generation relies on oil, oil prices can be a relevant feature.

8. Technical Indicators:

Moving Averages: Calculate moving averages of past prices to capture trends.

Volatility Measures: Standard deviation of past prices might indicate market volatility.

9. Social Media and News Data:

Sentiment Analysis: Analyse social media or news sentiment regarding energy-related topics. Negative sentiments might indicate potential price fluctuations.

10. Interaction Features:

Combine Features: Create interaction features between related variables. For example, the interaction between high temperature and high humidity might affect electricity demand differently than each factor individually.

11. Cyclical Patterns:

Fourier Transforms: Use Fourier transforms to capture cyclical patterns in the data, especially for daily and yearly seasonality.

12. Geographical Features (If Applicable):

Location: If your data covers multiple regions, include geographical information as features. Different regions might have different price patterns.

13. Machine Learning-Generated Features:

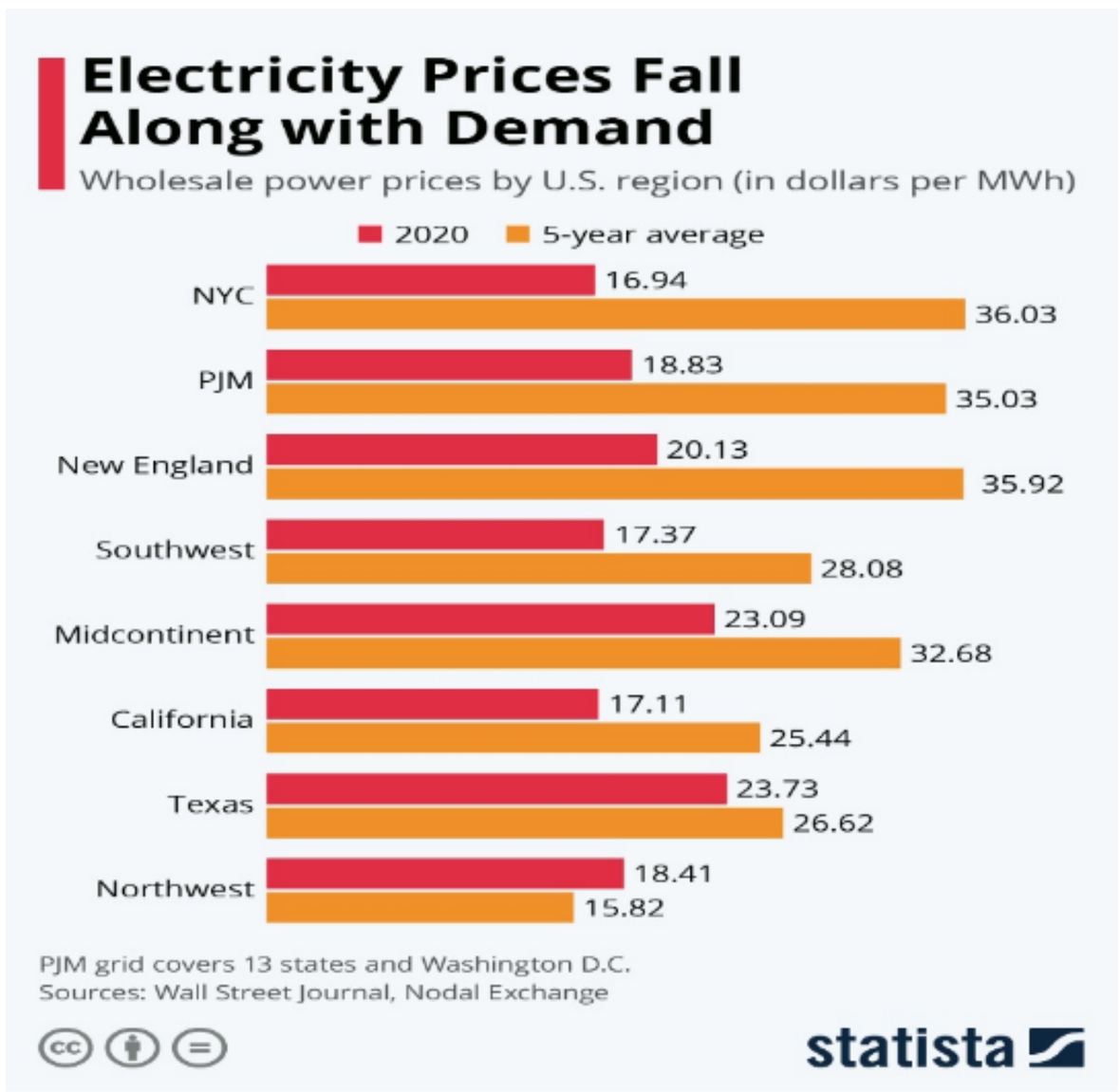
Clustering: Use clustering algorithms to group similar time periods together and include cluster labels as features.

```
import pandas as pd
```

```
import numpy as np
import datetime


# Load your electricity price data into a Pandas DataFrame
# Replace 'your_data.csv' with the path to your dataset.
data = pd.read_csv('your_data.csv')

# Convert the timestamp column to a datetime format
data['timestamp'] = pd.to_datetime(data['timestamp'])


# Feature 1: Date-related features
data['year'] = data['timestamp'].dt.year
data['month'] = data['timestamp'].dt.month
data['day'] = data['timestamp'].dt.day
data['hour'] = data['timestamp'].dt.hour
data['weekday'] = data['timestamp'].dt.weekday  # 0 for Monday, 6 for Sunday


# Feature 2: Lagged electricity prices (past prices)
for lag in range(1, 25):  # Consider the past 24 hours
    data[f'lag_{lag}'] = data['electricity_price'].shift(lag)


# Feature 3: Rolling statistics (e.g., 6-hour rolling mean)
data['rolling_mean'] = data['electricity_price'].rolling(window=6).mean()


# Feature 4: Weather data (if available)
# Load and preprocess weather data, merge it with the electricity price data
# Make sure the weather data has timestamps or date-related information.


# Feature 5: Holidays (if applicable)
```

```
# Create a binary feature indicating whether a given date is a holiday.


# Feature 6: Special events (if applicable)

# Create binary features indicating whether a significant event occurred on a given date.


# Feature 7: Time since the last major event (if applicable)
# Calculate the time elapsed since the last significant event.


# Feature 8: Price patterns (e.g., time of day, day of the week, seasonality)


# Feature 9: Market-related data (e.g., supply and demand information, market indices)


# Feature 10: Historical price trends (e.g., long-term price trend)


# You can continue to engineer more features depending on your dataset and domain
knowledge.


# Drop rows with missing values in the new features
data = data.dropna()


# Split the data into features (X) and the target variable (y)
X = data.drop(['timestamp', 'electricity_price'], axis=1)
y = data['electricity_price']


# Now, you can use X and y to train a machine learning model for electricity price prediction.


# Example model training
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

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

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions with the trained model
predictions = model.predict(X_test)
```



MODEL TRAINING:

1. *Data Collection*:

- Gather historical electricity price data. This data may include information about electricity demand, supply, weather conditions, and other relevant features. You can often obtain this data from government agencies, energy market operators, or research organizations.

2. *Data Preprocessing*:

- Data Cleaning: Remove any missing or erroneous data points.

- Feature Engineering: Create additional features that may be relevant to the prediction task, such as time of day, day of the week, holidays, and weather conditions.

- Normalization/Scaling: Scale the data to ensure that all features have the same magnitude, which can help the model converge faster.

3. *Data Splitting*:

- Split your dataset into training, validation, and test sets. The training set is used to train the model, the validation set is used for hyperparameter tuning, and the test set is used to evaluate the model's performance.

4. *Model Selection*:

- Choose an appropriate machine learning or statistical model for your prediction task. Common choices include time series models, regression models, neural networks, and ensemble methods.

5. *Model Training*:

- Train the selected model using the training data. This involves adjusting model parameters to minimize a chosen loss function, such as mean squared error (MSE) for regression problems.

6. *Hyperparameter Tuning*:

- Optimize the model's hyperparameters using the validation set. This may involve adjusting learning rates, batch sizes, the number of hidden layers in a neural network, or other model-specific parameters.

7. *Model Evaluation*:

- Evaluate the model's performance on the test set using appropriate evaluation metrics. For electricity price prediction, metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are commonly used.

8. *Model Deployment*:

- Once you are satisfied with the model's performance, deploy it to make real-time predictions. This may involve setting up a web service or integrating it into an existing system.

9. *Monitoring and Maintenance*:

- Continuously monitor the model's performance in the real world. Electricity price patterns can change over time due to market dynamics, regulatory changes, or seasonal variations. You may need to retrain the model periodically to keep it accurate.

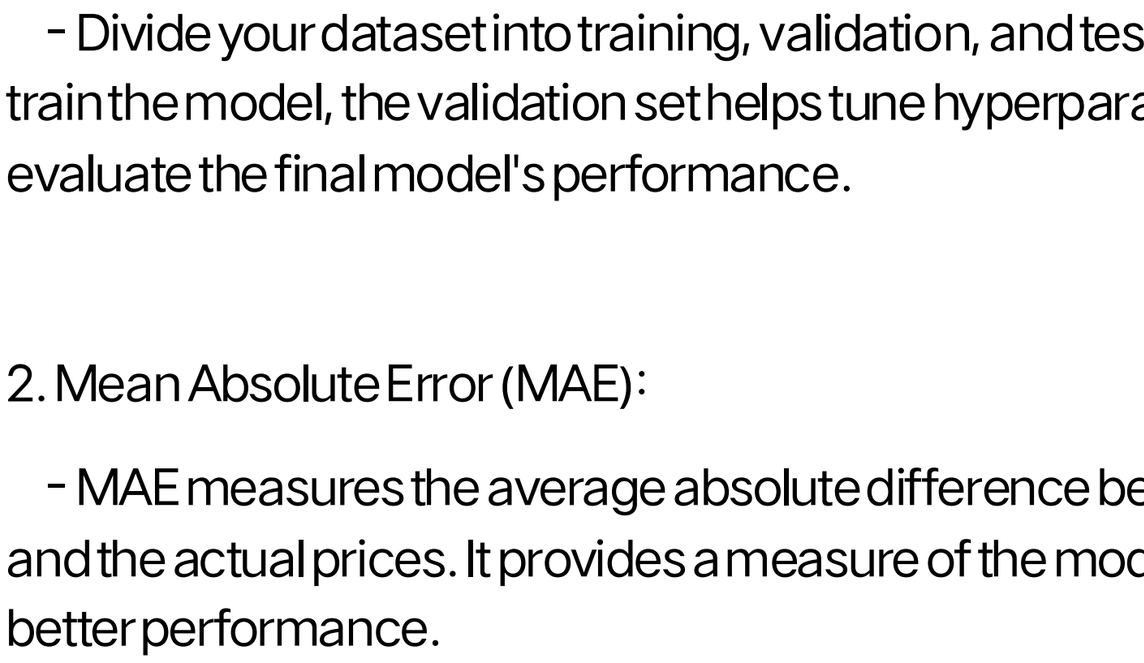
10. *Consider External Factors*:

- Take into account external factors that can influence electricity prices, such as government policies, fuel prices, and energy source availability. Incorporating such information into your model can improve its accuracy.

Code:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
# Assuming 'X' contains your features and 'y' contains the target (electricity prices)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

from sklearn.metrics import mean_squared_error, r2_score
# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2) Score: {r2}")
```



EVALUATION:

1. Data Splitting:

- Divide your dataset into training, validation, and testing sets. The training set is used to train the model, the validation set helps tune hyperparameters, and the testing set is used to evaluate the final model's performance.

2. Mean Absolute Error (MAE):

- MAE measures the average absolute difference between the predicted electricity prices and the actual prices. It provides a measure of the model's accuracy. A lower MAE indicates better performance.

3. Mean Squared Error (MSE):

- MSE measures the average squared difference between the predicted and actual prices. It is sensitive to outliers and gives more weight to larger errors. Smaller MSE values are desirable.

4. Root Mean Squared Error (RMSE):

- RMSE is the square root of the MSE and is a more interpretable metric because it is in the same units as the target variable. Like MSE, lower RMSE values indicate better performance.

5. Mean Absolute Percentage Error (MAPE):

- MAPE calculates the percentage difference between predicted and actual prices. It provides a measure of relative error. It is important when you want to understand the percentage accuracy of your predictions.

6. R-squared (R^2) or Coefficient of Determination:

- R-squared measures the proportion of the variance in the target variable that is explained by the model. A higher R^2 indicates a better fit. However, it can be misleading if not used in conjunction with other metrics.

7. Time-Series Specific Metrics:

- Depending on the characteristics of your data, you may need to consider time-series specific metrics such as Autocorrelation, Seasonal Decomposition, and ACF/PACF plots to assess the model's ability to capture time-dependent patterns.

8. Visualizations:

- Plotting actual vs. predicted prices over time can provide a visual assessment of how well the model performs, especially in capturing trends, seasonality, and anomalies in the data.

9. Cross-Validation:

- Utilize cross-validation techniques, such as k-fold cross-validation, to assess the model's stability and generalization performance on different subsets of the data.

10. Domain Expertise:

- Always consult with domain experts to evaluate the practical significance of the model's performance. They can provide insights into the implications of model predictions on electricity pricing. # Import necessary libraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

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

```
from sklearn.linear_model import LinearRegression  # Replace with your chosen model


# Load your historical electricity price data into a pandas DataFrame
# Ensure your data includes features (independent variables) and the target variable (electricity prices)
# Example: data = pd.read_csv("electricity_prices.csv")

# Split the data into features (X) and the target variable (y)
X = data.drop("Price", axis=1)
y = data["Price"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Replace the following code with your actual model training code
# Example: model = LinearRegression()
# model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model's performance
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Display the evaluation metrics
print(f"Mean Absolute Error: {mae}")
```

```
print(f"Mean Squared Error: {mse}")

print(f"Root Mean Squared Error: {rmse}")

print(f"R-squared (R2) Score: {r2}")
```

DONE BY:

AMBATI CHARAN KUMAR REDDY

AU720921244005

JCT COLLEGE OF ENGINEERING AND TECNOLOGY