**ELECTRICITY PRICES PREDICTION**

**Python libraries**

**Pandas:**

* pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

**Matplotlib:**

* Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

**Seaborn:**

* Seaborn is another open-source Python library, one that is based on Matplotlib (which focuses on plotting and data visualization) but features Pandas’ data structures. Seaborn is often used in ML projects because it can generate plots of learning data. Of all the Python libraries, it produces the most aesthetically pleasing graphs and plots, making it an effective choice if you’ll also use it for marketing and data analysis.

**SciPy:**

* SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

**PYTORCH:**

* PyTorch is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation umbrella. It is free and open-source software released under the modified BSD license.

**DATASET READINGS AND ACTIVITIES**

**About Dataset:**

* The Human Activity Recognition database was built from the recordings of 30 study participants performing activities of daily living (ADL) while carrying a waist-mounted smartphone with embedded inertial sensors. *The objective is to classify activities into one of the six activities performed*.

**1. Data Collection:**

* Gather historical data on electricity prices. This data can often be obtained from government energy agencies, utility companies, or online platforms that provide datasets.

**2. Data Preprocessing:**

**Clean the data:**

Handle missing values, outliers, and inconsistencies.

**Format the data:**

Ensure the data is in a suitable format for analysis.

**Feature selection:**

Identify relevant features that might influence electricity prices, such as demand, weather conditions, time of day, etc.

**3. Data Exploration:**

Explore the data to understand patterns, correlations, and trends. Visualization tools can be helpful here.

**4. Data Splitting:**

Split the dataset into training and testing sets. The training set is used to train the machine learning model, and the testing set is used to evaluate its performance.

**5. Model Selection:**

Choose an appropriate machine learning algorithm for prediction. Common choices include regression algorithms like Linear Regression, Decision Trees, or more advanced methods like Random Forest or Neural Networks.

**6. Model Training:**

Train the selected model using the training dataset. Adjust hyperparameters for better performance if necessary.

**7. Model Evaluation:**

Evaluate the model's performance using the testing dataset. Common metrics for regression problems include Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).

**8. Prediction and Analysis:**

Use the trained model to make predictions on new data. Analyze the results and assess the accuracy of the predictions.

**9. Iterative Process:**

Depending on the performance of the model, you might need to iterate by adjusting features, trying different algorithms, or fine-tuning parameters to improve prediction accuracy.

**DATASET CLEANING:**

* Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled.

**TRAIN AND TEST :**

**1. Splitting the Dataset:**

Randomly divide your cleaned dataset into two parts: the training set and the testing set. A common split ratio is 80:20 or 70:30, where 80% (or 70%) of the data is used for training the model, and the remaining 20% (or 30%) is used for testing.

**Python Example using scikit-learn:**

from sklearn.model\_selection import train\_test\_split

# Assuming X contains features and y contains target variable (electricity prices)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

In this example, X represents the features, and y represents the target variable (electricity prices). test\_size parameter specifies the proportion of the dataset to include in the test split. random\_state ensures reproducibility of the split.

**2. Training the Model:**

Use the training data (X\_train and y\_train) to train your machine learning model. Choose an appropriate algorithm based on the nature of your problem (regression, in this case) and the characteristics of your data.

**Python Example using Linear Regression:**

from sklearn.linear\_model import LinearRegression

# Create a linear regression model

model = LinearRegression()

# Train the model

model.fit(X\_train, y\_train)

**3. Evaluating the Model:**

After training the model, use the testing data (X\_test) to make predictions. Then, compare these predictions with the actual values (y\_test) to evaluate the model's performance.

**Python Example:**

# Make predictions on the test data

predictions = model.predict(X\_test)

# Evaluate the model using appropriate metrics (e.g., Mean Absolute Error, Mean Squared Error, R-squared)

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

mae = mean\_absolute\_error(y\_test, predictions)

mse = mean\_squared\_error(y\_test, predictions)

r2 = r2\_score(y\_test, predictions)

print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("R-squared:", r2)

These metrics will give you an indication of how well your model performs on unseen data. Lower MAE and MSE values and a higher R-squared value indicate a better-performing model.

**ACCURACY AND OTHER METRIC FOR PREDICTION EVALUATION**:

**Mean Absolute Error (MAE):**

* MAE measures the average absolute errors between predicted and actual values. It gives you an idea of how far off your predictions are on average.

**Mean Squared Error (MSE):**

* MSE measures the average of the squared errors between predicted and actual values. It amplifies larger errors, making it more sensitive to outliers.

**Root Mean Squared Error (RMSE):**

* RMSE is the square root of MSE, which gives you an interpretable measure of the average error in the same unit as the predicted values.

**Mean Absolute Percentage Error (MAPE):**

* MAPE expresses the prediction errors as a percentage of the actual values. It's useful when you want to understand the accuracy of your predictions in relation to the actual values.

**R-squared (R²):**

* R-squared measures the proportion of the variance in the dependent variable (electricity prices in this case) that is predictable from the independent variables (features) in your model. It ranges from 0 to 1, with 1 indicating a perfect fit.

**Root Mean Squared Percentage Error (RMSPE):**

* RMSPE is the square root of the average of the squared percentage errors. It's particularly useful when dealing with percentage data, providing a relative measure of accuracy.

When evaluating your prediction model, it's essential to consider a combination of these metrics to get a comprehensive understanding of its performance. Lower values of MAE, MSE, RMSE, MAPE, and RMSPE indicate better accuracy, while a higher R-squared value indicates a better fit of the model.

**1. Scatter Plots:**

Create scatter plots with actual prices on one axis and predicted prices on the other. Points close to the diagonal line indicate accurate predictions.

**Python Example using Matplotlib:**

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

plt.scatter(y\_test, predictions, color='blue', label='Actual vs. Predicted')

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Electricity Price Predictions')

plt.legend()

plt.show()

**2. Residual Plots:**

Residual plots show the difference between actual and predicted values. A random spread of points around the horizontal line at 0 suggests a good model fit.

**Python Example using Seaborn:**

import seaborn as sns

residuals = y\_test - predictions

plt.figure(figsize=(8, 6))

sns.residplot(predictions, residuals, lowess=True, line\_kws={'color': 'red', 'lw': 1})

plt.xlabel('Predicted Prices')

plt.ylabel('Residuals')

plt.title('Residual Plot')

plt.show()

**3. Line Plot for Time Series Prediction (if applicable):**

If your data involves time series, you can create a line plot showing actual prices over time and overlay it with predicted prices to observe trends.

**Python Example using Matplotlib:**

plt.figure(figsize=(12, 6))

plt.plot(test\_data.index, y\_test, label='Actual Prices', color='blue')

plt.plot(test\_data.index, predictions, label='Predicted Prices', color='red')

plt.xlabel('Time')

plt.ylabel('Prices')

plt.title('Electricity Price Predictions over Time')

plt.legend()

plt.show()

**4. Error Distribution Visualization:**

Plotting the distribution of errors (the differences between actual and predicted prices) can provide insights into the model's performance.

**Python Example using Seaborn:**

plt.figure(figsize=(8, 6))

sns.histplot(residuals, kde=True, color='green')

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.title('Error Distribution')

plt.show()

These visualizations can help you assess the accuracy of your predictions, identify patterns, and understand where the model performs well or struggles. Feel free to adapt these examples based on your specific dataset and visualization preferences.

**VISUALIZATION:**

**1. Scatter Plots:**

Create scatter plots with actual prices on one axis and predicted prices on the other. Points close to the diagonal line indicate accurate predictions.

**Python Example using Matplotlib:**

import matplotlib.pyplot as plt

\plt.figure(figsize=(8, 6))

plt.scatter(y\_test, predictions, color='blue', label='Actual vs. Predicted')

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Electricity Price Predictions')

plt.legend()

plt.show()

**2. Residual Plots:**

Residual plots show the difference between actual and predicted values. A random spread of points around the horizontal line at 0 suggests a good model fit.

**Python Example using Seaborn:**

import seaborn as sns

residuals = y\_test - predictions

plt.figure(figsize=(8, 6))

sns.residplot(predictions, residuals, lowess=True, line\_kws={'color': 'red', 'lw': 1})

plt.xlabel('Predicted Prices')

plt.ylabel('Residuals')

plt.title('Residual Plot')

plt.show()

**3. Line Plot for Time Series Prediction (if applicable):**

If your data involves time series, you can create a line plot showing actual prices over time and overlay it with predicted prices to observe trends.

**Python Example using Matplotlib:**

plt.figure(figsize=(12, 6))

plt.plot(test\_data.index, y\_test, label='Actual Prices', color='blue')

plt.plot(test\_data.index, predictions, label='Predicted Prices', color='red')

plt.xlabel('Time')

plt.ylabel('Prices')

plt.title('Electricity Price Predictions over Time')

plt.legend()

plt.show()

**4. Error Distribution Visualization:**

Plotting the distribution of errors (the differences between actual and predicted prices) can provide insights into the model's performance.

**Python Example using Seaborn:**

plt.figure(figsize=(8, 6))

sns.histplot(residuals, kde=True, color='green')

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.title('Error Distribution')

plt.show()

These visualizations can help you assess the accuracy of your predictions, identify patterns, and understand where the model performs well or struggles. Feel free to adapt these examples based.

on your specific dataset and visualization preferences.

**PHASE 2:INNOVATION**

**EXPLANATION:**

* Electricity price prediction refers to the use of mathematical models and algorithms to forecast future electricity prices based on historical data and various influencing factors. Machine learning techniques, such as regression algorithms, neural networks, and time series analysis, are commonly employed for this purpose.
* The prediction process typically involves collecting historical data on electricity prices and relevant features such as demand, weather conditions, time of day, and economic indicators. This data is then cleaned and preprocessed to handle missing values, outliers, and other inconsistencies.
* Machine learning models are trained using the preprocessed data to learn the patterns and relationships between the input features and electricity prices. These models are then used to make predictions on new, unseen data. The accuracy of the predictions is evaluated using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared, which measure the difference between the predicted values and the actual electricity prices.
* Electricity price prediction has practical applications in various sectors, including energy trading, renewable energy integration, and consumer decision-making. Accurate predictions enable businesses and consumers to plan their energy usage, optimize energy-related expenses, and make informed decisions in the dynamic energy market.

**KAGGLE:**

A subsidiary of Google, it is an online community of data scientists and machine learning engineers. Kaggle allows users to find datasets they want to use in building AI models, publish datasets, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

[**https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction**](https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction)

**REST OF EXPLANATION:**

**Time-Series Considerations:**

Electricity price data often has a time-based component. Time-series models like ARIMA (AutoRegressive Integrated Moving Average) or more advanced ones like SARIMA (Seasonal ARIMA) are designed to handle such temporal patterns effectively.

**Feature Engineering:**

Expert knowledge in the domain can help in creating meaningful features. For instance, creating lag features (past prices) or rolling statistics (average prices over a period) can enhance the model's predictive power.

**Handling Non-Linearity:**

Electricity prices might follow non-linear patterns. In such cases, machine learning algorithms like Decision Trees, Random Forests, or neural networks can capture these complex relationships better than linear models.

**Ensemble Methods:**

Consider using ensemble methods like Random Forests or Gradient Boosting. These techniques combine multiple models to improve accuracy and generalizability.

**Hyperparameter Tuning:**

Fine-tune your model's hyperparameters using techniques like grid search or randomized search. This process helps you find the best combination of hyperparameters for your specific model.

**Regularization and Overfitting:**

Regularization techniques like L1 (Lasso) and L2 (Ridge) regularization can prevent overfitting, especially in complex models like neural networks. Regularization penalizes large coefficients, encouraging simpler and more generalizable models.

**Handling Categorical Data:**

If your dataset includes categorical variables (like types of energy sources), consider techniques like one-hot encoding or feature embedding to convert them into numerical values for the model to understand.

**Anomaly Detection**:

Implement anomaly detection algorithms to identify unusual patterns or outliers in the data. Anomalies can significantly impact predictions, so it's essential to handle them appropriately.

**Model Interpretability**:

For certain applications, understanding the factors contributing to predictions is crucial. Techniques like SHAP (SHapley Additive exPlanations) values or Partial Dependence Plots can help interpret complex model predictions.

**Continuous Monitoring and Updating:**

Markets change, and so should your model. Regularly update the model with fresh data to ensure it adapts to new trends and patterns in the electricity market.

**Ethical and Fairness Considerations:**

Be mindful of biases in the data and the potential impact on predictions. Ethical considerations are crucial, especially if the predictions are used to make important decisions.

Remember, the effectiveness of your prediction model not only depends on the algorithms and techniques but also on a deep understanding of the domain and the data you're working with. Always be critical of your results and iterate on your approach to achieve the best possible predictions.

**METRICS USED FOR THE ACCURACY CHECK:**

**Mean Absolute Error (MAE):**

MAE represents the average of the absolute errors between the predicted and actual values. It gives a linear measure of the prediction accuracy.

**Mean Squared Error (MSE):**

MSE measures the average of the squared errors between predicted and actual values. It amplifies the impact of larger errors compared to MAE.

**Root Mean Squared Error (RMSE):**

RMSE is the square root of MSE and provides an interpretable measure in the same unit as the target variable.

​

**Mean Absolute Percentage Error (MAPE):**

MAPE calculates the average percentage difference between predicted and actual values. It is particularly useful when you want to understand the relative error in percentage terms.

**R-Squared (R²) Score:**

R² measures the proportion of the variance in the dependent variable (electricity prices) that is predictable from the independent variables (features) in the model. A higher R² indicates a better fit of the model to the data.

**Adjusted R-Squared:**

Adjusted R² adjusts the R² score for the number of predictors in the model. It penalizes the addition of unnecessary predictors that do not improve the model significantly.