# SMART METER DIGITAL TWIN

## INTRODUCTION

The Smart Meter Digital Twin is an innovative system that integrates real-time data analytics, predictive modeling, and monitoring functionalities to provide enhanced insights into energy consumption and system health. It mirrors the physical smart meter infrastructure in a virtual environment, enabling advanced energy management and decision-making.

## System Features

**Real-Time Monitoring**

* Displays live data on energy consumption, voltage, current, and power factor.
* Visual dashboards with interactive graphs and KPIs.
* Alerts for abnormal or unexpected readings.

**Energy Prediction**

* Uses LSTM and XGBoost models for forecasting future energy consumption.
* Incorporates historical data, weather, and time-based features.
* Provides short-term and long-term usage trends.

**Energy Billing Prediction**

* Predicts monthly and upcoming bills based on consumption trends.
* Considers slab rates and time-of-use pricing.
* Sends early alerts if predicted bill exceeds user thresholds.

**Fault Detection**

* Detects anomalies such as voltage spikes, current leakage, and missing data.
* Uses classification models (e.g., XGBoost, Isolation Forest) for early fault prediction.
* Maintains a fault history log for diagnostics.

**Anomaly Prediction**

* Identifies outliers in consumption or power quality.
* Generates daily and weekly reports highlighting anomalies.
* Supports alert mechanisms via email/SMS.

## Technologies Used

**Frontend:** Streamlit for UI with dark/light themes.

**Backend:** Python (pandas, NumPy, scikit-learn, TensorFlow, XGBoost).

**Visualization:** Matplotlib, Plotly, Streamlit Charts.

**Storage:** CSV data files / Firebase (optional).

**Notifications:** Email/SMS API integration.

## Code Implementation Overview

**Real-Time Monitoring**

**Purpose**: Simulate and display smart meter readings in real-time.

**Python Class (RealTimeMonitor):**

* Reads historical smart meter data from a CSV.
* Streams data row-by-row like a live feed.
* Supports viewing of the most recent readings and resetting the stream.

**Streamlit Dashboard:**

* Configurable refresh rate and data source selection.
* Displays key metrics: Voltage, Phase Current, Power Factor, Frequency.
* Plots real-time energy consumption using st.line\_chart().
* Shows meter location on a map using pydeck.
* Uses a loop to simulate continuous data updates at user-defined intervals.

**Technologies:**

* pandas for data handling
* Streamlit for interactive UI
* pydeck for geographical visualization
* time.sleep() for real-time simulation delay

**Output**:

A responsive dashboard continuously updating smart meter readings and visualizations.

Interactive experience for users to monitor their energy data in near real-time.

Energy Prediction (Daily & Monthly)

**Purpose:** Predict energy consumption for the next day and the next 30 days using LSTM deep learning models.

**Model Training Process:**

* Data is preprocessed and resampled to daily totals.
* The KWHhh column (energy usage) is scaled using MinMaxScaler.
* **For day prediction:** 7-day sequences are created to predict the next day.
* **For month prediction:** 30-day sequences are created to predict the next 30 days.
* LSTM models are built using keras.Sequential:
* Two-layer LSTM architecture with dropout for regularization.
* For monthly prediction, RepeatVector and TimeDistributed layers enable sequence-to-sequence learning.
* Models are trained using early stopping to avoid overfitting.
* After training, models are evaluated using RMSE and MAE metrics.
* Models and scalers are saved using joblib and model.save().

**Streamlit Interface:**

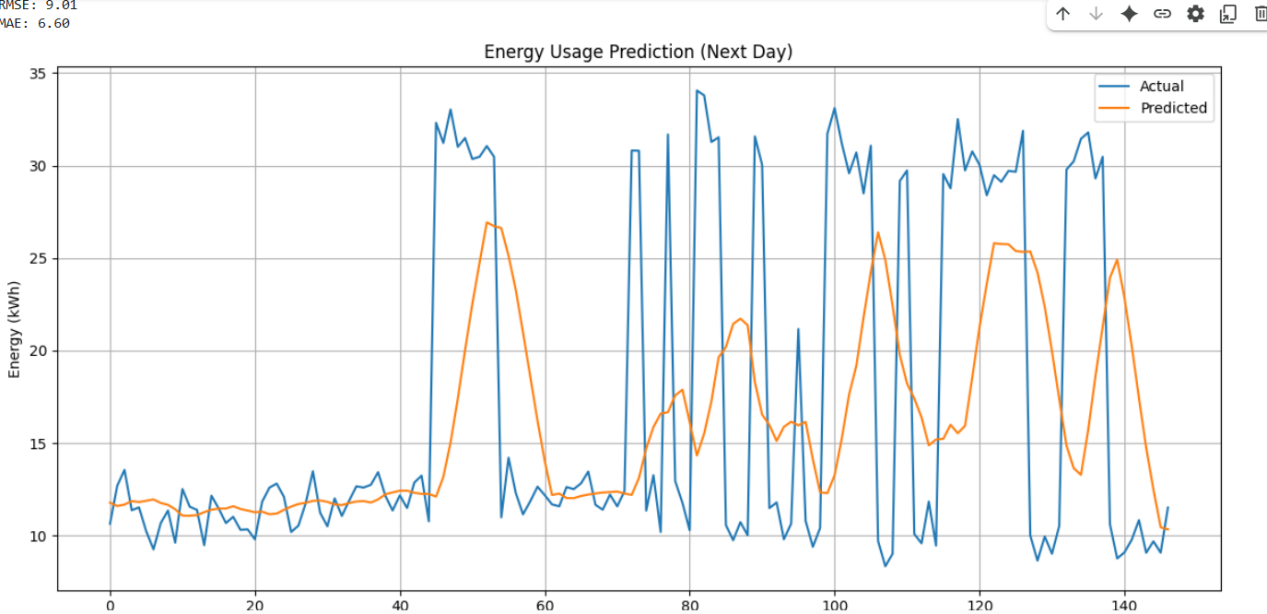
* Lets users upload CSVs with the last 7 or 30 days of data.
* Loads the appropriate model and scaler based on prediction type.
* Displays single value prediction (next day) or a 30-day forecast chart.
* Allows users to download prediction results as CSV.

**Technologies Used:**

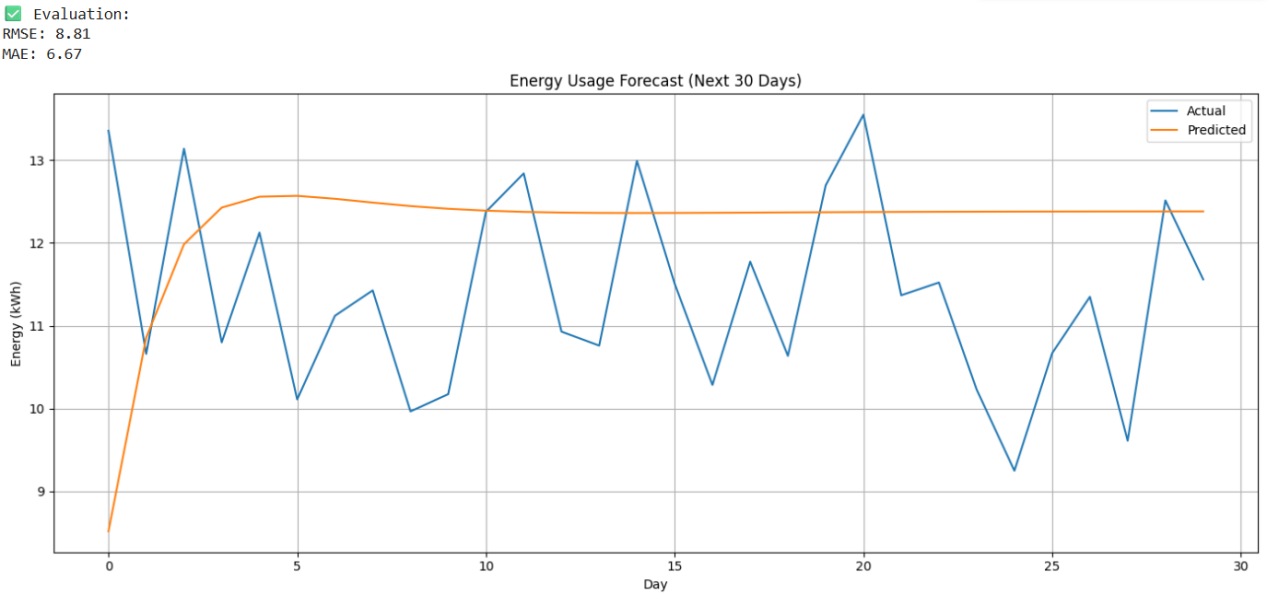
* Keras for model creation and training.
* Joblib for model/scaler serialization.
* Streamlit for user interface and visualization.
* Matplotlib for internal training evaluation plots.

**Output:**

**NEXT DAY OUTPUT**

****

**NEXT MONTH PREDICTION OUTPUT**

****

Energy Billing Prediction

**Purpose:** Estimate future electricity bills based on current consumption patterns and tariff structures.

**Model Training:**

* Uses a RandomForestRegressor model trained on a dataset with past energy usage and billing details.
* Important features include:
* Energy\_Consumption\_KWh
* Units\_Consumed\_KWh
* Tariff\_Per\_KWh
* Average\_Daily\_Consumption\_KWh
* The target is Projected\_Bill (predicted future bill).
* Model is trained on 80% of the data and evaluated on the remaining 20% using RMSE and R² score.
* A scatter plot visually compares actual vs. predicted bills.
* The model is saved as bill\_predictor\_model.pkl for reuse.

**Streamlit Interface:**

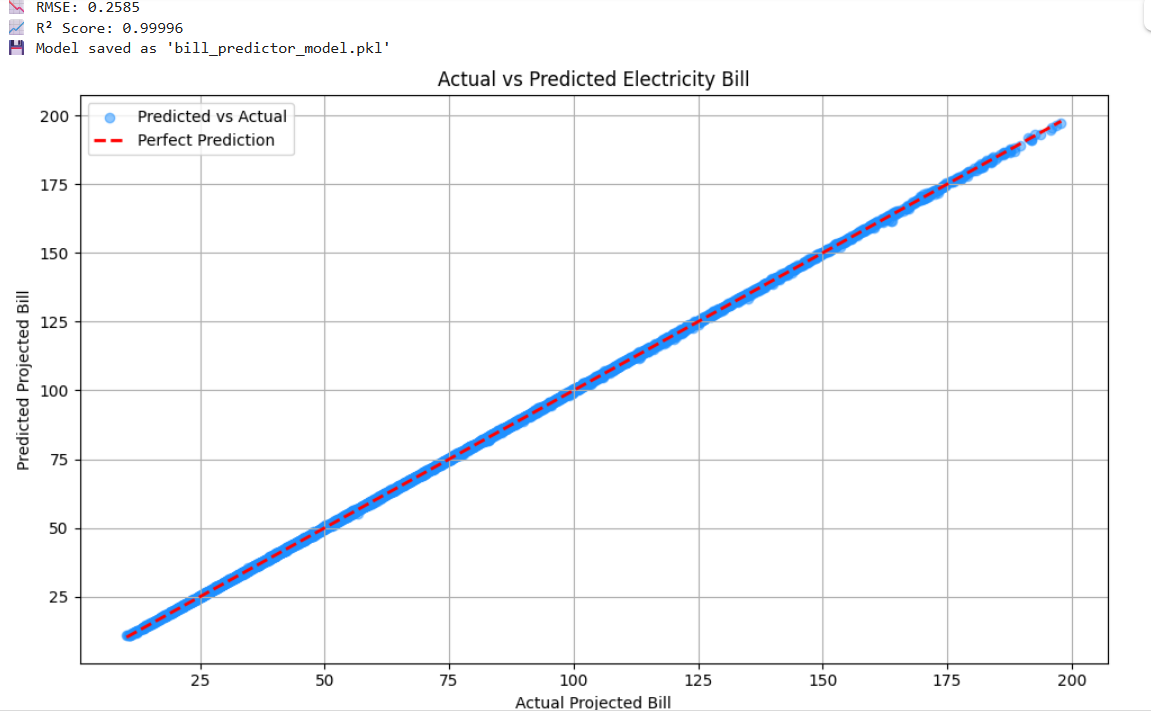
* Users upload their Training\_Data.csv file.
* The app displays predicted bills and flags cases where predicted bills are >20% higher than the previous bill.
* For such anomalies, alert emails are sent using SMTP Gmail integration.
* Alert Mechanism:
* Emails are personalized and highlight the predicted bill vs. previous bill.
* Alerts encourage proactive energy savings by users.

**Technologies Used:**

* scikit-learn for model training
* joblib for saving the model
* smtplib, email.mime for email notifications
* Streamlit for the interactive dashboard

Output:

* Interactive bill prediction dashboard
* Anomaly detection for high-usage customers
* Email alerting system to notify users



Fault Detection & Anomaly Monitoring

Purpose: Continuously scan live smart meter readings for anomalies or faults and notify stakeholders in real-time.

**Streamlit Dashboard Functionality:**

* Displays live updates of smart meter metrics (Voltage, Power, Power Factor, Frequency).
* Sidebar alerts update instantly upon anomaly detection.

**Anomaly Detection Logic:**

* A custom function (detect\_row\_anomalies) checks each row for known fault conditions such as:
* Voltage outside the 180–250 V range
* Power factor below 0.5
* Negative active power (may indicate faulty readings)
* Frequency outside 48.5–51.5 Hz
* Zero energy consumption blocks (possible outages)

**Live Log Creation:**

Logs are created daily and saved in the logs/ directory.

Each detected anomaly is recorded with a timestamp.

**Email Notification System:**

* If an anomaly is found, an alert is sent via email using the configured Gmail app password.
* smtplib and EmailMessage are used to send alerts.
* Emails are only triggered when anomalies are detected.

**Final Summary:**

* At the end of the monitoring loop, a final summary of all anomalies is displayed.
* Historical logs can be browsed from the Streamlit dashboard.

**Technologies Used:**

* Streamlit for real-time UI
* pandas for row-wise data processing
* smtplib and email.message for alert emails
* os and datetime for logging and file management

**Output:**

* Real-time display of incoming smart meter data.
* Alerts generated both visually and via email.
* Anomaly logs maintained for future reference.

## Anomaly Prediction using Machine Learning

**Model Training Code Summary:**

* **Model**: XGBoost Classifier
* **Dataset**: generated\_anomaly\_training\_data.csv
* **Features Used**:
* Voltage
* SystemPowerFactor
* ActivePower\_kW
* Frequency
* BlockEnergykWh
* **Label**: Binary class 'Anomaly' (1 = anomaly, 0 = normal)

**Steps:**

**Load and preprocess data**

* Convert timestamp to datetime
* Select features and label
* Normalize features with MinMaxScaler

**Train-test split**

* Use 80-20 split
* Stratify by label to preserve anomaly ratio

**Model training**

* Use XGBClassifier
* Handle class imbalance using scale\_pos\_weight
* Set use\_label\_encoder=False, eval\_metric='logloss'

**Evaluate**

* Predict labels and probabilities
* Calculate Confusion Matrix, Classification Report
* Report Accuracy, R², and RMSE

**Visualizations**

* Voltage plot with red dots for predicted anomalies
* Actual vs predicted anomalies line graph
* Confusion matrix heatmap using Seaborn

**Save**

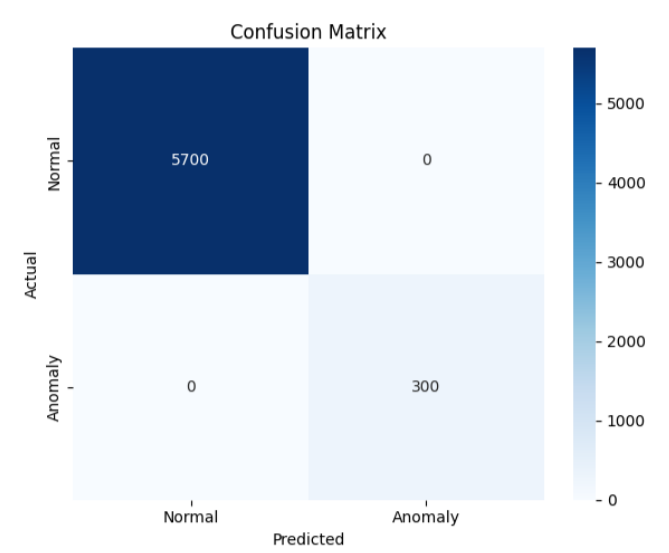
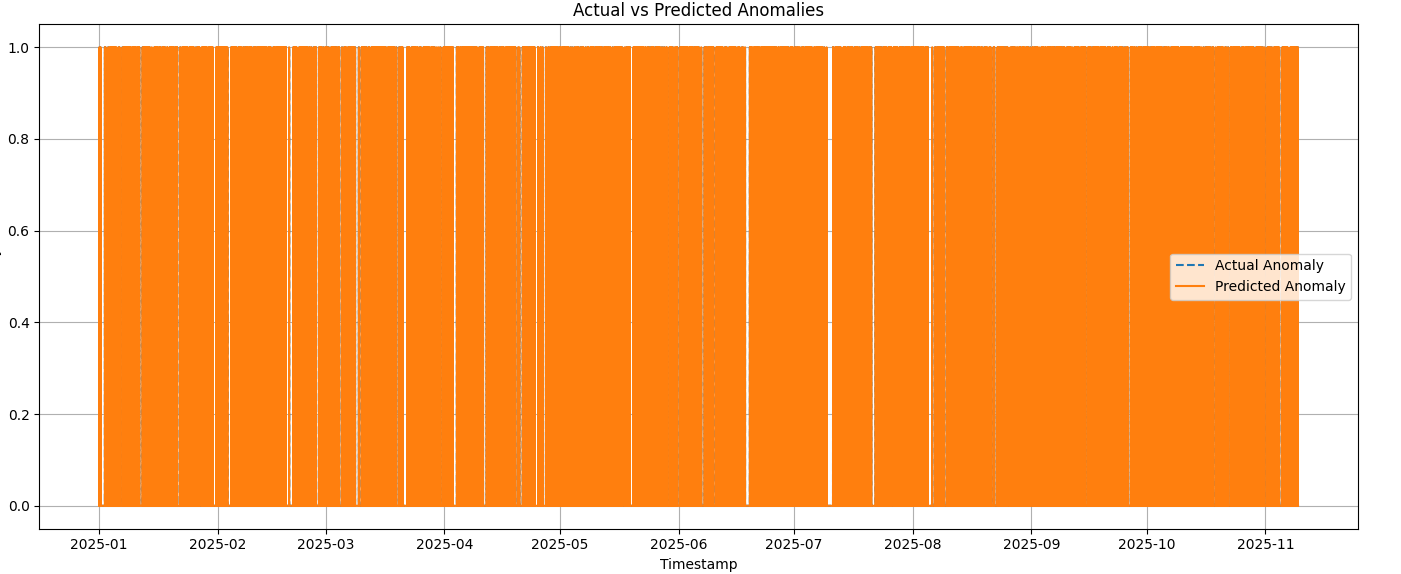
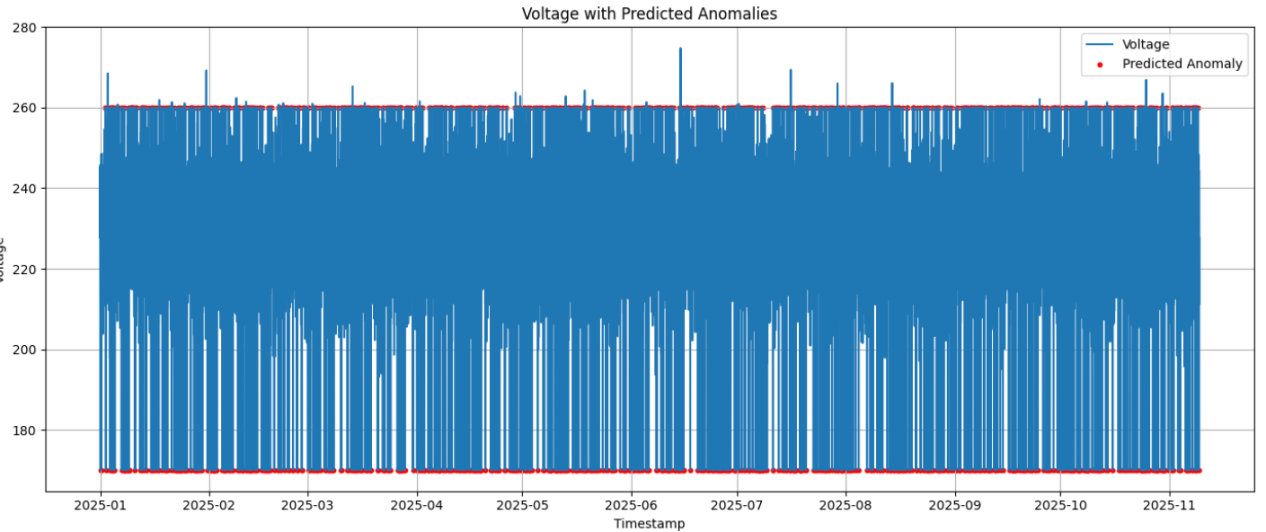
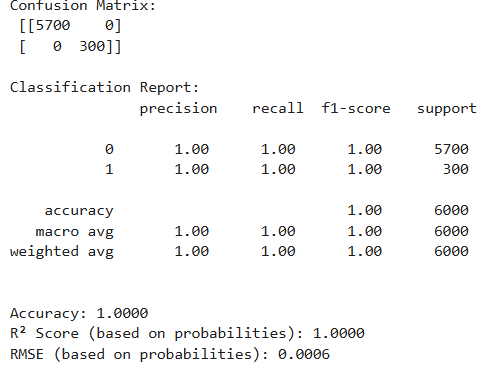
* Save model and scaler using joblib

**Streamlit Interface:**

* Upload smart meter CSV with necessary features
* Model predicts and marks anomalies

**Output include**s:

* Processed dataframe with predictions
* Voltage anomaly timeline chart
* Download button for result CSV



### What the Matrix Shows

|  | **Predicted: Normal** | **Predicted: Anomaly** |
| --- | --- | --- |
| **Actual: Normal** | **5700** | 0 |
| **Actual: Anomaly** | 0 | **300** |

### ✅ Interpretation

**True Positives (TP)** = 300  
➤ These are **actual anomalies** that were **correctly predicted** as anomalies.

**True Negatives (TN)** = 5700  
➤ These are **actual normal** readings correctly predicted as normal.

**False Positives (FP)** = 0  
➤ No normal readings were incorrectly predicted as anomalies. ✔️

**False Negatives (FN)** = 0  
➤ No anomalies were missed — i.e., the model didn’t fail to catch any anomalies. ✔️

### 📈 Performance Metrics (Implied)

Because there are **zero errors**, the model achieved **perfect classification**:

**Accuracy** = (TP + TN) / Total = (5700 + 300) / 6000 = **100%**

**Precision** = TP / (TP + FP) = 300 / (300 + 0) = **100%**

**Recall** = TP / (TP + FN) = 300 / (300 + 0) = **100%**

**F1 Score** = 2 × (Precision × Recall) / (Precision + Recall) = **100%**

**Conclusion**

This Digital Twin system provides an intelligent interface for smart meter management. It combines:

Real-time streaming

* ML-driven energy forecasting
* Billing estimates
* Fault detection
* Predictive anomaly classification
* Alerting via email

Together, these components form a complete digital monitoring and analytics solution for energy infrastructure.

## Real-Time Monitoring

### Code Summary:

The RealTimeMonitor class streams data row-by-row from a CSV, simulating real-time smart meter readings such as voltage, current, power factor, frequency, and location

### Streamlit Dashboard:

* Displays live voltage, current, power factor, and frequency.
* Uses pydeck to visualize meter location.
* Includes a refresh interval and real/simulated data toggle.

## Energy Prediction

### Daily Prediction (Next Day)

* **Model Type**: LSTM
* **Input**: Last 7 days of daily kWh consumption
* **Output**: Next day prediction

#### Steps:

1. Resample to daily frequency
2. Normalize using MinMaxScaler
3. Use sliding window to create sequences (7 days → 1 day)
4. Train an LSTM model with 2 layers and dropout

### Monthly Prediction (Next 30 Days)

* **Model Type**: LSTM with RepeatVector and TimeDistributed
* **Input**: Last 30 days
* **Output**: Next 30-day forecast

#### Steps:

* Same as above but input/output length = 30
* Uses RepeatVector to project input to 30 steps output

### Streamlit Dashboard:

1. Upload CSV of past energy
2. Choose prediction type: day or month
3. Predict and display forecast with line charts
4. Download results

## Energy Billing Prediction

### Model:

**Algorithm**: Random Forest Regressor

### Features Used:

Energy\_Consumption\_KWh

Units\_Consumed\_KWh

Tariff\_Per\_KWh

Average\_Daily\_Consumption\_KWh

### Target:

Projected\_Bill

### Steps:

* Train/test split (80/20)
* Train model and evaluate RMSE, R²
* Save with joblib

### Streamlit Dashboard:

Upload customer training data

Predict next bill using trained model

Compare with previous bill

Detect anomaly if predicted > 1.2 × previous

Email alert to customer using SMTP

## Fault Detection

### Key Features:

Rule-based anomaly detection in real-time

### Criteria:

Voltage < 180V or > 250V

SystemPowerFactor < 0.5

ActivePower\_kW < 0

Frequency < 48.5 or > 51.5 Hz

BlockEnergykWh == 0

### Actions:

* For each fault, it logs the error and displays in sidebar alerts
* Sends real-time email alert using Gmail SMTP if fault is detected

### Dashboard:

* Real-time metrics with anomalies marked
* Generates daily log file in /logs
* Displays previous logs

## 6. Anomaly Prediction using Machine Learning

### Model Training Code Summary:

**Model**: XGBoost Classifier

**Dataset**: generated\_anomaly\_training\_data.csv

### Features Used:

Voltage

SystemPowerFactor

ActivePower\_kW

Frequency

BlockEnergykWh

### Label:

Binary class 'Anomaly' (1 = anomaly, 0 = normal)

### Steps:

Load and preprocess data

Convert timestamp to datetime

Select features and label

Normalize features with MinMaxScaler

Train-test split

Use 80-20 split

Stratify by label to preserve anomaly ratio

### Model Training:

Use XGBClassifier

Handle class imbalance using scale\_pos\_weight

Set use\_label\_encoder=False, eval\_metric='logloss'

### Evaluation:

Predict labels and probabilities

Calculate Confusion Matrix, Classification Report

Report Accuracy, R², and RMSE

### Visualizations:

Voltage plot with red dots for predicted anomalies

Actual vs predicted anomalies line graph

Confusion matrix heatmap using Seaborn

### Save:

Save model and scaler using joblib

### Streamlit Interface:

Upload smart meter CSV with necessary features

Model predicts and marks anomalies

#### Output Includes:

Processed dataframe with predictions

Voltage anomaly timeline chart

Download button for result CSV