Technical Report – PowerAI Project

Team Members:

- **Amira Boudaoud** — *Référente de l'équipe | Data and AI*
- **Ghofrane Ben Rhaiem** — *Data and AI*
- **Antonin Timbert** — *Data Engineering*
- **Mohamed Talhi** — *Business Intelligence and Analysis*
- **Hind Karim El Alaoui** — *Business Intelligence and Analysis*

Technical Supervisors (Experts):

- **Leonardo DE AQUINO MARQUES**
- **Dario VIEIRA CONCEICAO**

Academic Year: 2024/2025

Table of Contents

Abstract :

The technical report for PowerAI Project presents a comprehensive analysis and development of an unsupervised anomaly detection system for monitoring electrical equipment in a data center environment. The project, conducted by a multidisciplinary team during the 2024/2025 academic year, aimed to address the challenge of identifying unexpected failures or anomalies in critical systems such as Uninterruptible Power Supplies (UPS), energy meters, and Power Distribution Units (PDUs). The team employed a robust technological stack, including Python, scikit-learn, TensorFlow/Keras, and visualization tools like matplotlib and seaborn, to preprocess, analyze, and model the data. The workflow involved data acquisition, exploratory data analysis (EDA), dataset partitioning into functional subsets, and the implementation of advanced machine learning techniques like Isolation Forest, LSTM Autoencoders, and Dense Autoencoders. The project also included the development of a web application using Streamlit and Flask for real-time anomaly visualization and reporting. Key challenges such as handling large datasets, lack of labeled anomalies, and model optimization were addressed through iterative refinement and expert feedback. The report highlights the team's collaborative efforts, individual contributions, and the broader impact of the solution for data center operations, emphasizing its potential to reduce downtime and improve efficiency.

1. General Project Analysis

This section provides a comprehensive overview of the methodology, technologies, and steps taken throughout the PowerAI project.

1.1 Problem Definition

Modern electrical installations rely on critical equipment such as UPS systems, energy meters (met_1, met_2) and PDUs (Power Distribution Units) to ensure safe and continuous power supply.

Any unexpected failure or anomaly in these systems can lead to operational downtime, equipment damage, or safety risks. However, manually monitoring dozens of parameters in real time is not practical.

The main goal of this project is to develop and test unsupervised anomaly detection methods that can automatically detect unusual behavior in the equipment data, helping maintenance teams identify risks in advance.

The problem is divided into subsets (UPS, meters, PDUs) to make the analysis more interpretable and precise. Each subset uses a tailored set of variables relevant to its function (currents, voltages, power, battery metrics, etc.).

1.2 Technological Stack

The technical implementation of this project is based on open-source tools and widely used data science libraries:

- Python (main programming language)
- Data manipulation: pandas, numpy
- Machine Learning: scikit-learn (Isolation Forest, PCA)
- Deep Learning: tensorflow / keras (LSTM Autoencoder)
- Visualization: matplotlib, seaborn
- Jupyter Notebook (for interactive analysis)
- CSV files (to store and exchange prepared datasets)
- Version control: Git
- Collaboration tools: Teams (sharing documents and logs)

This stack allows fast prototyping, reproducibility, and flexibility for testing multiple anomaly detection approaches.

1.3 Dataset Description

The dataset used in this project, named **merged_dataset.csv**, contains detailed electrical measurements collected from an industrial site covering the full electrical path from the external power grid to final equipment consumption.

The data is organized into four main blocks:

1. External Power Supply:

This section of the dataset tracks the incoming power from the external grid, with columns such as:

- met_va_1, met_vb_1, met_vc_1 – Phase voltages (V)
- met_ia_1, met_ib_1, met_ic_1 – Phase currents (A)
- met_p_1 – Active power (kW)
- met_fp_1 – Power factor (unitless, 0 to 1)

These columns help monitor supply quality and efficiency.

2. UPS (Uninterruptible Power Supply)

The UPS block ensures stable power delivery and backup during grid disturbances. The dataset includes:

- Input voltages: ups_va_in, ups_vb_in, ups_vc_in
- Input frequency: ups_hz_in
- Output voltages: ups_va_out, ups_vb_out, ups_vc_out
- Output currents: ups_ia_out, ups_ib_out, ups_ic_out
- Output frequency: ups_hz_out
- Battery operations: ups_bat_i_charge (charging current), ups_bat_i_discharge (discharge current), ups_p_cell_v and ups_n_cell_v (battery cell voltages)

3. PDUs (Power Distribution Units)

The PDUs redistribute the UPS output to local equipment.

The dataset tracks:

- pdu1_kwh to pdu8_kwh – Energy delivered per unit
- pdu1_fp to pdu8_fp – Power factor per PDU
- pdu1_i to pdu8_i – Currents per PDU circuit

4. Energy Consumption Metrics

The dataset includes multiple energy metering columns to measure and track total and seasonal usage:

- met_kwh_1, met_kwh_2 – Total site energy in kWh (site 1 & site 2)
- met_kwh_month_1, met_kwh_month_2 – This month's total
- met_kwh_jan_1 to met_kwh_dec_1 – Monthly breakdown for the year
- met_kwh_y_1, met_kwh_y_2 – Total annual energy for the current year
- met_kwh_1y_1, met_kwh_1y_2 – Previous year's total

These allow seasonal comparisons and help detect peaks or improvements.

Time Period

The dataset spans a continuous monitoring period, sampled at regular intervals.

The full period covers several months, providing enough data to analyze:

- Daily and weekly usage patterns
- Seasonal peaks
- Long-term trends in power quality and consumption

The **data_hora** column stores precise timestamps for each measurement, making it possible to apply time series models and windowing.

1.3 Workflow and Methodology

1.3.1. Project Overview

This project addresses the challenge of **anomaly detection in electricity consumption and flow in a data center**, with a focus on:

- Understanding the electrical infrastructure
- Splitting and analyzing the data into functional subsets
- Applying and comparing AI-based anomaly detection models
- Deploying the models through an intuitive web application

1.3.2 Data Acquisition and Preprocessing

Our project began with two proprietary database files (**.db format**) containing power monitoring data from a large-scale datacenter. To facilitate analysis, we developed a Python conversion script that:

- Extracted all tables from the SQLite databases
- Performed schema analysis to understand relationships
- Merged relevant tables into a unified **CSV** format
- Ensured timestamp synchronization across all data sources

The resulting merged dataset contained 704,248 records with 165 parameters spanning multiple power system components. Initial temporal analysis revealed the data covered the period from (2024-10-31 20:31:19) to (2025-02-21 14:16:54), though we identified several gaps in the time series that required handling in later sections.

1.3.3. Dataset Exploration and Splitting

3.1 Exploratory Data Analysis (EDA)

Performed on the raw dataset, Checked for:

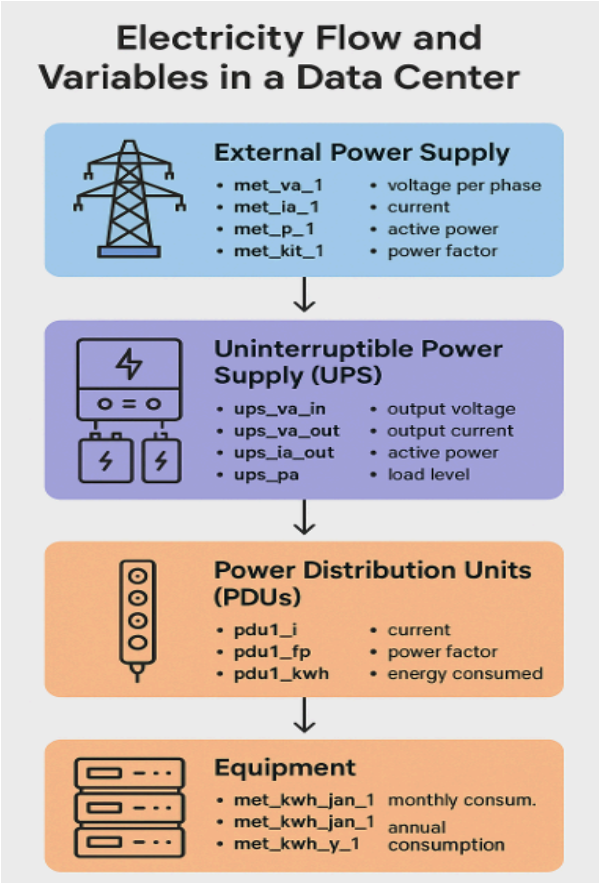
- Null values
- Duplicates
- Unique values
- Statistical summaries

Visualization:

- Time series plots
- Distributions per phase (Plotted general distributions of voltage, current, and power factor.)
- Initial anomaly patterns

Based on both domain knowledge and after many discussions between the team members and structural data characteristics, we split the dataset into **four functional subsets**, inspired by the actual power flow in a data center (appendix 1: [POWER_AI_Topic.pdf]).

3.2 Dataset Partitioning



- **External Power Supply (Meter 1 & Meter 2)**
  - Voltage (met_va_1, met_vb_1, met_vc_1, met_v_1)
  - Current (met_ia_1, met_ib_1, met_ic_1, met_i_1, met_in_1)
  - Power factor (met_fp_1, met_fp_2)
- **UPS (Uninterruptible Power Supply)**
  - Input/output voltages & currents
  - Battery charge/discharge levels
  - Mode of operation
- **PDUs (Power Distribution Units)**

    - Output currents: pduX_i
    - Power factors: pduX_fp
- **Equipment Consumption**

Resulted in 4 dataFrames as follow :

Stage	Subset Name	Shape (rows and columns)	Description
1	External Power Supply	(704248, 65)	Voltage, current, power factor from the grid
2	UPS	(704248, 40)	Battery, internal conversions, output voltages
3	PDU	(704248, 25)	Final power distribution lines
4	Equipment Consumption	(704248, 37)	Final measured loads

We validated this split using a Python script to ensure **non-overlapping columns** and correct feature assignments.

Each member then took responsibility for a specific subset, thus detailed explanation of each subset analysis would be provided in the second part of the report (individual contribution)

- Amira: Meter 1 + Meter 2 (external power supply)
- Antonin: PDU + part of Meter 2
- Mohammed: UPS
- Ghofrane: Equipment consumption

1.3.4. Subset-Specific Analysis Approach

1.3.4.1 Methodology

For each subsystem, we implemented a tiered analytical approach:

- **Stage 1:** Basic Statistical Profiling
    - Descriptive statistics (mean, variance, skewness)
    - Temporal distribution analysis

    - Phase-to-phase correlation studies

- **Stage 2:** Visualization Suite
    - Time-series plots of key parameters (Figure 2)
    - Histograms with kernel density estimation
    - Rolling window statistics (2/4/6 minute intervals)

- **Stage 3:** Anomaly Detection Framework

We employed three complementary techniques:

    - Contextual Anomaly Detection: considers multiple time-adjacent values instead of isolated ones.
    - Analyzes measurements relative to operational context (e.g., higher current tolerance during backup mode)
    - Uses sliding window comparison (typically 15-30 minute windows)

- **Stage 4:** Domain-Based Sanity Checks (hard thresholds and rules based on electrical standards (e.g., acceptable current limits) :
    - Electrical rule validation.
    - Threshold comparisons against:
    - Manufacturer specifications
    - Regulatory requirements (IEC 62040)
    - Historical percentiles (99.7% typically)

- **Stage 5:** Statistical Methods
    - Modified Z-score (robust to outliers)
    - Interquartile Range (IQR) with 1.5x multiplier
    - Dynamic percentile thresholds (adaptive to load changes)

1.3.4.2 Advanced Algorithm Implementation

After establishing baseline performance, we deployed three machine learning approaches:

- Isolation Forest (Unsupervised, tree-based)
    - Effective for point anomalies
    - Trained on 5-month "normal operation" data

    - Contamination parameter tuned via grid search

- LSTM Autoencoders(Captures temporal dependencies for time-series)
    - Captures temporal dependencies
    - Architecture: 64-32-16-32-64 units
    - Trained on 5-month "normal operation" data

- Dense Autoencoders (Neural network that learns data reconstruction)
    - For multivariate pattern detection
    - Bottleneck layer = 1/4 input dimension
    - Regularized with L2 (λ=0.001)

1.3.5. Expert-Guided Refinement & Project Restructuring

Based on expert feedback, we restructured our approach:
Instead of operating on broader subsets, we narrowed the focus to clearly defined groups, each representing a functional subcomponent.

The expert proposed the following groups:

- **Meter 1 and Meter 2**

met_va_1, met_vb_1, met_vc_1, met_v_1 → **Voltages**
met_ia_1, met_ib_1, met_ic_1, met_i_1, met_in_1 → **Currents**
met_fp_1, met_ia_seq_*_1, met_i_imb_1, met_i_leak_1 → Other key metrics **(power factor and sequence)**

- **UPS**

Inputs: ups_va_in, ups_vb_in, ups_vc_in, ups_hz_in
Outputs: ups_va_out, ups_vb_out, ups_vc_out, ups_ia_out, ups_ib_out, ups_ic_out
Battery: ups_bat_i_charge, ups_bat_i_discharge, ups_p_cell_v, ups_n_cell_v

- **PDU**

pdu1_i to pdu8_fp

1.3.6. Final Anomaly Detection Workflow

For each group (e.g., Meter 1 Voltage):

- **Windowed transformation (e.g., 12 samples = 2 min window)**
  Introduced configurable time windows:
    - Base window: 12 samples (2 minutes)
    - Alternative windows: 16/25/30 samples (3/5/6 minutes)
- **Team Work Allocation**

Finalized responsibilities as:

- Amira: Meter 1 + Meter 2 voltage/current
- Antonin: PDU systems + Meter 2 power quality and sequence data
- Ghofrane: UPS systems

- **Model Development Process**
    - **Iterative Training Approach**

For each subsystem, we executed:

- **Data Preparation:**
    - Min-max normalization (LSTM)
    - Standard scaling (autoencoder)
    - Sequential windowing with timestamp preservation (Train models for each window size : **2-3-5 and 6 minutes window**)

- **Architecture Search and analysis :**
    - Tested 5 LSTM variants (32-128 units)
    - Evaluated 3 autoencoder depths
    - Optimized dropout rates (0.1-0.3)
    - Tried different isolation forest algorithms which lead to not satisfying results.

- **Threshold Optimization:**
    - Percentile-based (97-99.9%)
    - F1-maximizing dynamic thresholds
    - Ensemble voting across methods

- **Validation Framework**

Used expert-provided anomaly logs containing:

- validated events (timestamps + duration) - ground truth logs
- Time series plots of the different subsets between 29/12 and 23/02

- *Metrics Calculated for each results :*
    - Total anomalies detected
    - True Positives, False Positives, False Negatives
    - Precision, Recall, and F1-Score

The **best model** and window size were selected based on these metrics.

Remark : the different results for different window size and models would be provided in each subpart in the second part of the report (individual contributions)

1.3.7.Saving and Loading Models (Deployment Prep)

To preserve results and prepare for deployment:

- Trained models were saved as .keras files
- Scalers were saved using joblib.

Folder structure:

```
anomaly_detection_project/
├── meter1_voltage/
│   ├── model.keras
│   ├── scaler.pkl
├── meter1_current/
├── ups/
├── pdu/
...
```

1.3.8. Web Application Design

1.3.8.1 Frontend: Streamlit

Users will:

- Choose dataset type (e.g., Meter 1, UPS)
- Choose signal type (e.g., Voltage, Current)
- Upload CSV file
- View plotted results + anomaly points

- View table summary of detected anomalies

1.3.8.1 Backend: Streamlit

Handles:

- Preprocessing
- Model and scaler loading (based on file/selection)
- Anomaly detection logic
- Response with results to frontend

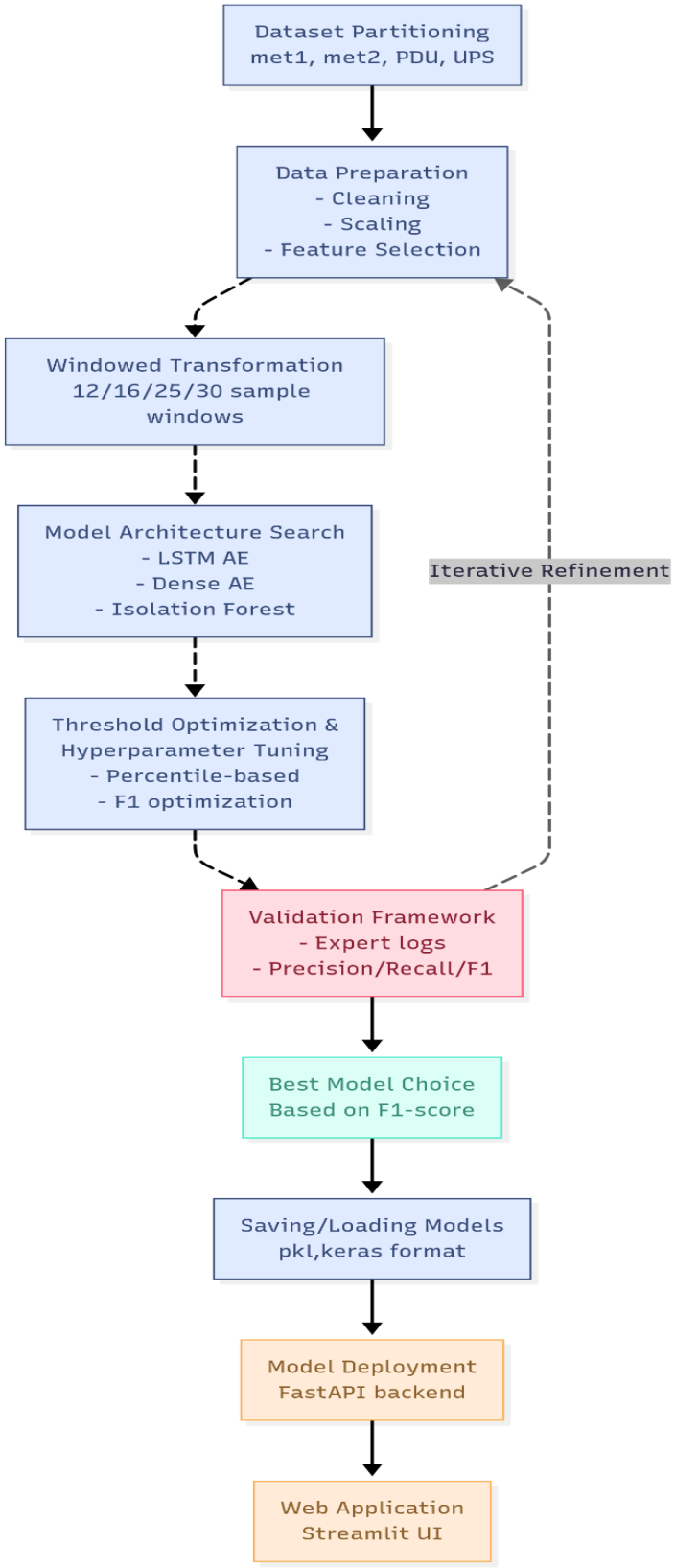
1.3.9. Conclusion

This project followed a rigorous flow:

- Data cleaning, merging, and intelligent splitting
- In-depth EDA and domain exploration
- Iterative anomaly detection experiments across window sizes and models
- Final model selection using expert logs and precision-recall metrics
- Deployment planning with saved models and frontend/backend design
- Complete web interface integration
- Test multiple edge cases and real-world CSV uploads

This pipeline provides a solid foundation for anomaly detection in data center environments, with a focus on realistic, operational deployment and model evaluation grounded in expert feedback.

conceptual flowchart of the anomaly detection pipeline



1.4 Challenges and Solutions

- Challenge 1: Large Dataset Size (165 parameters × 700K+ rows)
    - **Issue**: Training deep models (LSTM, autoencoders) required high memory and long runtime.
    - **Solution**:
        - Used data subset partitioning (Meter1, UPS, etc.).
        - Applied downsampling and windowing.
        - Trained models with early stopping and batch processing.
- Challenge 2: Lack of Ground Truth Labels
    - **Issue**: No labeled anomalies in original dataset.
    - **Solution**:
        - Used expert-provided event logs.
        - Matched model output timestamps with validated intervals.
- Challenge 3: High False Positives in Simple Methods
    - **Issue**: Basic Z-score and thresholding flagged many false positives.
    - **Solution**:
        - Used multivariate models (autoencoder, LSTM).
        - Introduced contextual windowing to reduce noise.
- Challenge 4: Model Selection Across Time Windows
    - **Issue**: Different window sizes impacted performance.
    - **Solution**:
        - Tested across 12, 16, 25, and 30-sample windows.
        - Selected best configuration based on F1-score.
- Challenge 5: Model Preservation in Google Colab
    - **Issue**: Colab runtime resets caused loss of trained models.
    - **Solution**:
        - Saved models and scalers to Google Drive using **.keras** and **.pkl** formats.
- Challenge 6: Backend Integration
    - **Issue**: Integrating Streamlit frontend with Flask backend for inference.
    - **Solution**:
        - Standardized model input/output shapes.
        - Created modular Flask endpoints for each subset.

This structured and modular approach allowed us to overcome technical constraints while delivering a scalable and extensible anomaly detection system.

1.5 Results and Impact

Project Outcomes

The project successfully delivered a working end-to-end anomaly detection pipeline tailored for electricity data in a data center environment. Major accomplishments included:

- **Functional Pipeline**:
  - Complete flow from raw .db to deployed model.
  - Standardized feature engineering and preprocessing.
- **Scalable Model Development**:
  - LSTM and Autoencoder models tuned across various subsets.
  - Window-based temporal aggregation improved anomaly accuracy.
- **Model Evaluation**:
  - Used expert-validated logs to assess model outputs.
  - Achieved high F1-scores for multiple configurations.
- **Operational Deployment Readiness**:
  - Saved models ready for inference (.keras + .pkl).
  - Web interface supports CSV uploads and live anomaly plotting.
- **User Experience Focus**:
  - Frontend enables non-technical users to visualize anomalies.

Broader Impact

- **For Data Centers**:
  - Real-time fault detection.
  - Reduced downtime and improved efficiency.
- **For Future Work**:
  - Pipeline is modular and expandable.
  - Can be retrained with newer data.
  - Framework allows testing new anomaly detection algorithms.

- **For Educational Value**:
    - Demonstrates a complete AI engineering pipeline.
    - Showcases effective teamwork, domain integration, and deployment practices.

This solution not only solves a critical operational problem but also lays the foundation for intelligent monitoring systems in complex infrastructures.

2. Individual Contributions

2.1 Amira Boudaoud

As part of the team project, I was initially responsible for the global preprocessing and exploratory data analysis (EDA) of the full dataset. This phase involved identifying the data structure, understanding column distributions, handling missing values, and verifying data consistency. This foundational analysis was crucial in preparing the data for more focused processing in the following stages.

Subsequently, I focused on the **external power supply datasets,** which corresponded to Meter 1 and Meter 2 readings. These subsets are central to understanding the incoming energy from the grid and include critical parameters such as voltage, current, power factor, frequency, and sequence components.

Stage 1: Initial Dataset Split – External Power Supply

The external power supply data was derived from the original dataset, which contained numerous metering and sensor data points. The goal here was to isolate measurements from Meter 1 and Meter 2 and divide them into manageable subgroups based on electrical parameters.

```
(704248, 65)
```

```
df_external.columns
```

```
Index(['met_va_1', 'met_vb_1', 'met_vc_1', 'met_v_1', 'met_hz_1', 'met_ia_1',
       'met_ib_1', 'met_ic_1', 'met_i_1', 'met_in_1', 'met_pa_1', 'met_pb_1',
       'met_pc_1', 'met_p_1', 'met_sa_1', 'met_sb_1', 'met_sc_1', 'met_s_1',
       'met_fpa_1', 'met_fpb_1', 'met_fpc_1', 'met_fp_1', 'met_va_seq_p_1',
       'met_va_seq_n_1', 'met_va_zero_n_1', 'met_va_groud_1', 'met_v_imb_1',
       'met_ia_seq_p_1', 'met_ia_seq_n_1', 'met_ia_zero_n_1', 'met_i_imb_1',
       'met_i_leak_1', 'met_va_2', 'met_vb_2', 'met_vc_2', 'met_v_2',
       'met_hz_2', 'met_ia_2', 'met_ib_2', 'met_ic_2', 'met_i_2', 'met_in_2',
       'met_pa_2', 'met_pb_2', 'met_pc_2', 'met_p_2', 'met_sa_2', 'met_sb_2',
       'met_sc_2', 'met_s_2', 'met_fpa_2', 'met_fpb_2', 'met_fpc_2',
       'met_fp_2', 'met_va_seq_p_2', 'met_va_seq_n_2', 'met_va_zero_n_2',
       'met_va_groud_2', 'met_v_imb_2', 'met_ia_seq_p_2', 'met_ia_seq_n_2',
       'met_ia_zero_n_2', 'met_i_imb_2', 'met_i_leak_2', 'data_hora'],
      dtype='object')
```

Figure 1 : column names of external energy supply dataframe

This subset was divided into the following categories:

1. Voltage Measurements

- met_va_1, met_vb_1, met_vc_1 (Meter 1 phases)
- met_va_2, met_vb_2, met_vc_2 (Meter 2 phases)
- met_v_1, met_v_2 (average voltages)
- met_v_imb_1, met_v_imb_2 (imbalance indicators)

2. Current Measurements

- met_ia_1, met_ib_1, met_ic_1 and met_ia_2, met_ib_2, met_ic_2
- met_i_1, met_i_2, met_in_1, met_in_2
- met_i_imb_1, met_i_imb_2, met_i_leak_1, met_i_leak_2

3. Power and Frequency

- Active Power: met_pa_X, met_pb_X, met_pc_X, met_p_X
- Apparent Power: met_sa_X, met_sb_X, met_sc_X
- Power Factor: met_fpa_X, met_fpb_X, met_fpc_X
- Frequency: met_hz_1, met_hz_2

4. Sequence Components

- Voltage Sequences: met_va_seq_p_X, met_va_seq_n_X, met_va_zero_n_X
- Current Sequences: met_ia_seq_p_X, met_ia_seq_n_X, met_ia_zero_n_X

Purpose: These partitions allowed us to treat voltage, current, and power factor data separately for targeted anomaly detection.

```
PARAMETER_GROUPS = {
    'voltage': {
        'phase': ['met_va_1', 'met_vb_1', 'met_vc_1', 'met_va_2', 'met_vb_2', 'met_vc_2'],
        'average': ['met_v_1', 'met_v_2'],
        'imbalance': ['met_v_imb_1', 'met_v_imb_2']
    },
    'current': {
        'phase': ['met_ia_1', 'met_ib_1', 'met_ic_1', 'met_ia_2', 'met_ib_2', 'met_ic_2'],
        'average': ['met_i_1', 'met_i_2'],
        'neutral': ['met_in_1', 'met_in_2'],
        'imbalance': ['met_i_imb_1', 'met_i_imb_2'],
        'leakage': ['met_i_leak_1', 'met_i_leak_2']
    },
    'power': {
        'active': ['met_pa_1', 'met_pb_1', 'met_pc_1', 'met_pa_2', 'met_pb_2', 'met_pc_2'],
        'total': ['met_p_1', 'met_p_2'],
        'apparent': ['met_sa_1', 'met_sb_1', 'met_sc_1', 'met_sa_2', 'met_sb_2', 'met_sc_2'],
        'factor': ['met_fpa_1', 'met_fpb_1', 'met_fpc_1', 'met_fpa_2', 'met_fpb_2', 'met_fpc_2']
    },
    'frequency': ['met_hz_1', 'met_hz_2'],
    'sequence_components': {
        'voltage': ['met_va_seq_p_1', 'met_va_seq_n_1', 'met_va_zero_n_1',
                    'met_va_seq_p_2', 'met_va_seq_n_2', 'met_va_zero_n_2'],
        'current': ['met_ia_seq_p_1', 'met_ia_seq_n_1', 'met_ia_zero_n_1',
                    'met_ia_seq_p_2', 'met_ia_seq_n_2', 'met_ia_zero_n_2']
    }
}
```

```
df_met_1 = df_external[[col for col in df_external.columns if '_1' in col] + ['data_hora']].copy()
df_met_2 = df_external[[col for col in df_external.columns if '_2' in col] + ['data_hora']].copy()
```

Figure 2: column partition for both meter 1 and meter 2 data

Daily and Hourly Time Series Analysis

To explore behavioral patterns, I performed time-based aggregations and plotted:

- **Daily Trends**: Highlighted weekday vs weekend changes, long-term shifts.
- **Hourly Trends**: Showed intra-day operational cycles, power demand peaks, or periodic anomalies.

Sample Plots:

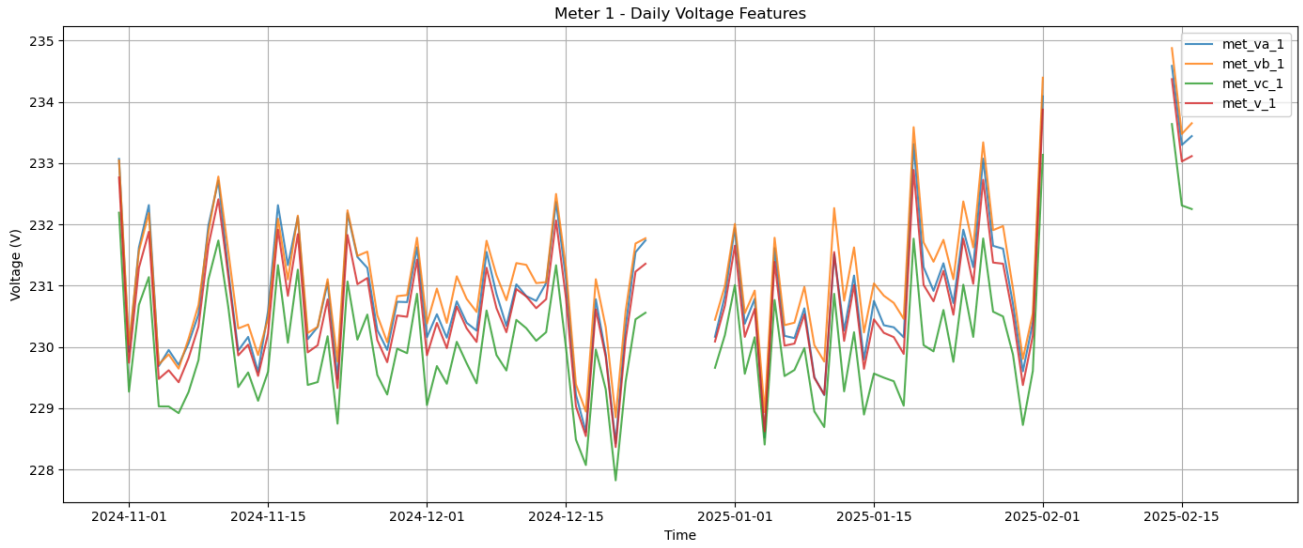


Figure 3 : all voltage features for Meter 1, daily average

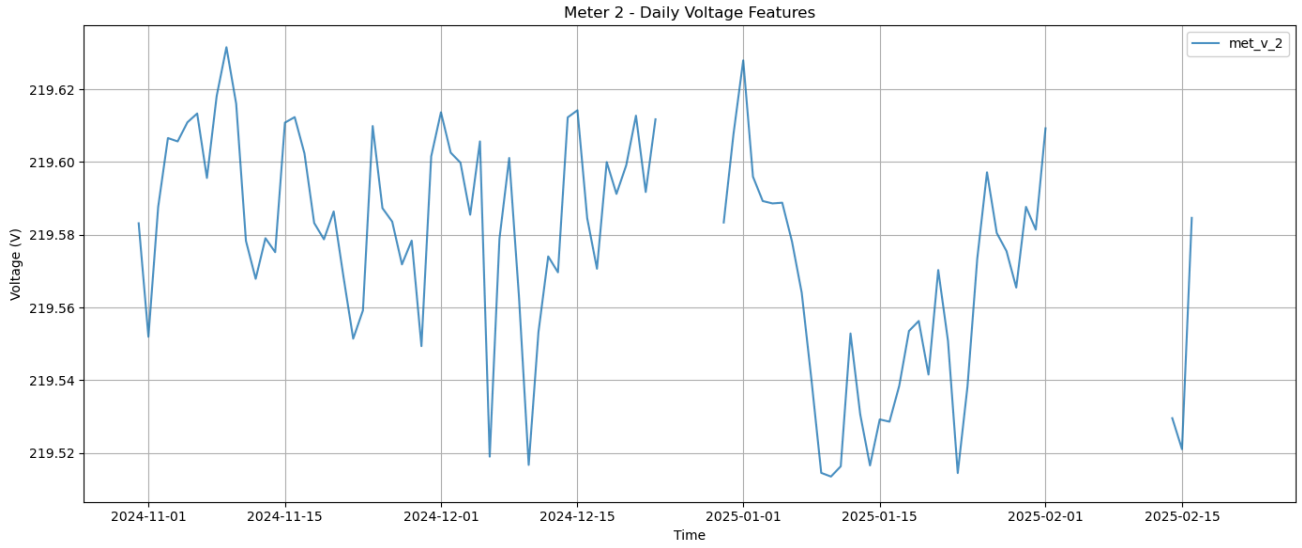


Figure 4 : average voltage feature for Meter 2, daily

Data Decomposition: Train/Test Time Windows

Based on visual and domain-specific analysis, I defined operational windows:

- **Normal periods** (for training models)
- **Unknown/anomalous periods** (for testing)

This decomposition was critical to ensure that models trained only on clean patterns.

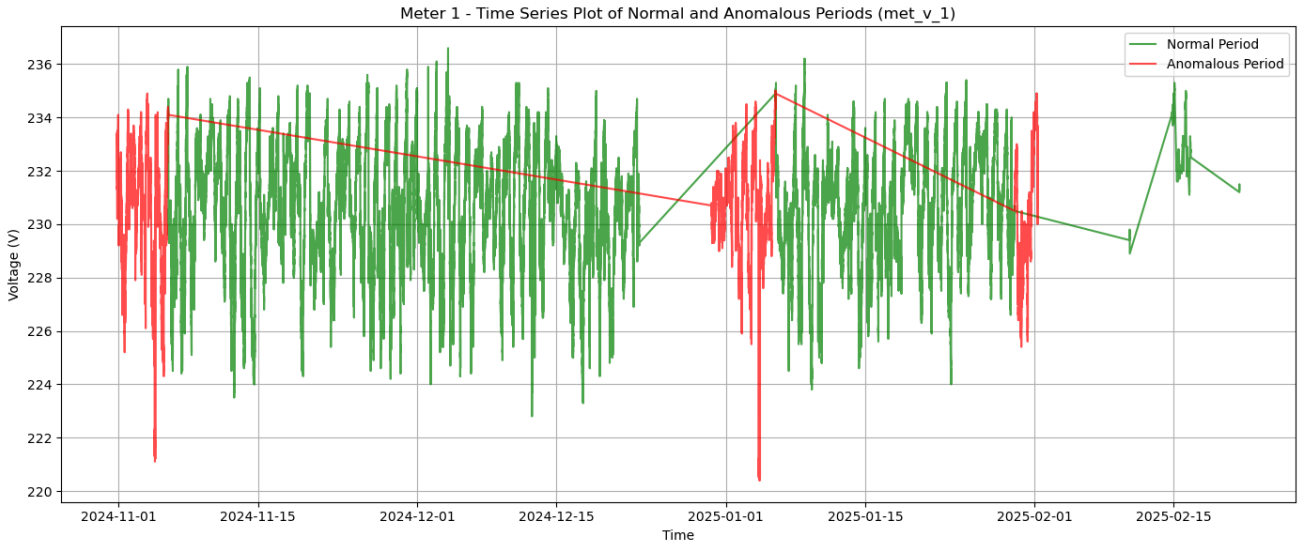


Figure 5 : Annotated Timeline Plot with Green/Red Zones for Train/Test Split

Machine Learning Algorithms for Anomaly Detection

To detect anomalies in voltage and current measurements for Meter 1 and Meter 2, I employed three complementary approaches:

1. Isolation Forest

- Used for fast, unsupervised anomaly detection.
- Trained using scikit-learn's IsolationForest with tuned contamination levels:
    - Meter 1: contamination=0.003
    - Meter 2: contamination=0.04
- Detected anomalies were plotted in red over time series data.

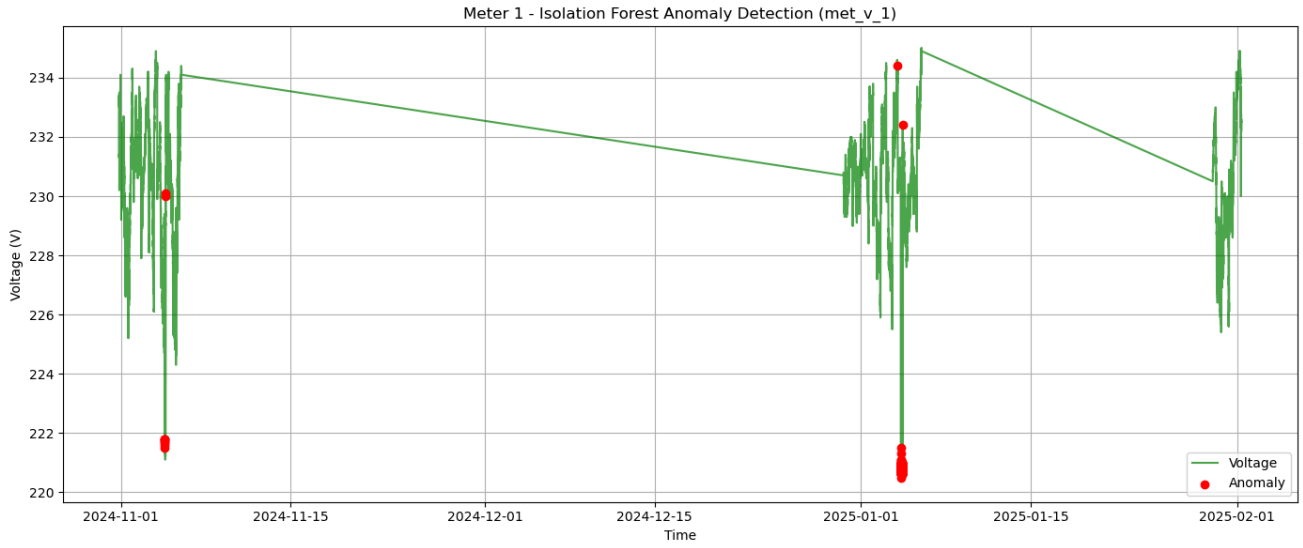


Figure 6 : example model output_Meter 1 - Isolation Forest Anomaly Detection (met_v_1)

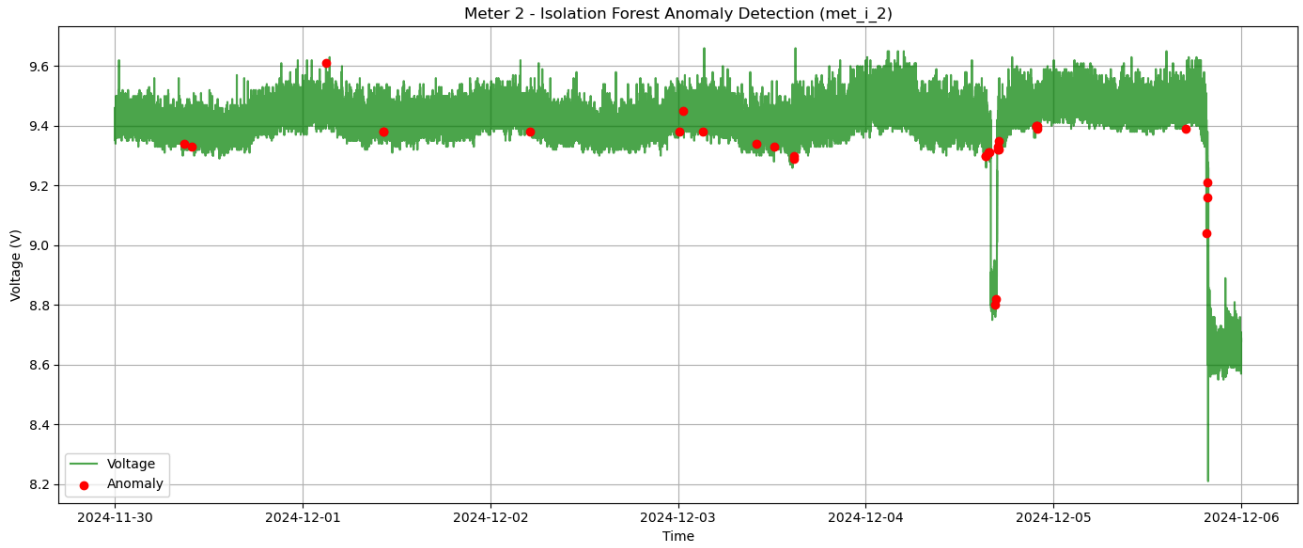


Figure 7: example model output_Meter 2 - Isolation Forest Anomaly Detection (met_i_2)

2. Dense Autoencoder

- Feedforward neural network trained to reconstruct normal sequences.
- Architecture:
  - Input layer
  - Encoder: Dense(32) → Dense(16) → Dense(8)

    - Decoder: Dense(16) → Dense(32) → Output
- Trained on normal data; reconstruction error used as anomaly score.
- Threshold set using 98th percentile of reconstruction errors.

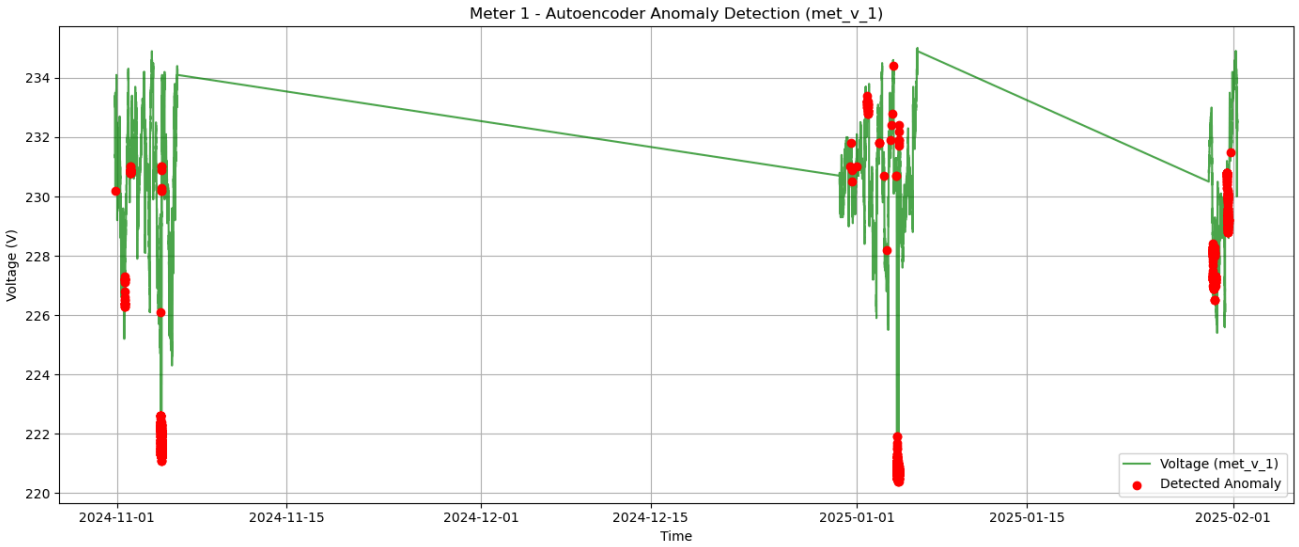


Figure 8 : Meter 1 - Autoencoder Anomaly Detection (met_v_1)

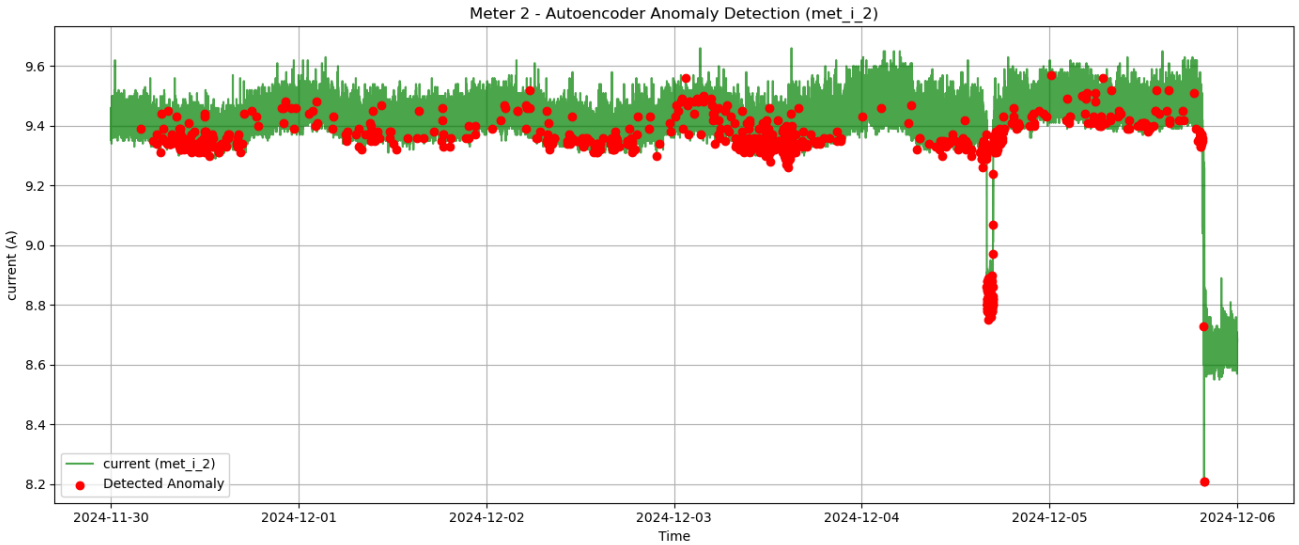


Figure 9: Meter 2 - Autoencoder Anomaly Detection (met_i_2)

3. LSTM Autoencoder

- Sequential model capturing temporal dependencies.
- Architecture:

    - LSTM(64) → RepeatVector → LSTM(64) → TimeDistributed(Dense)
- Input reshaped into 3D sequences (windowed format)
- Used MinMaxScaler and validation split for training.
- Anomalies detected via reconstruction error percentile thresholding.

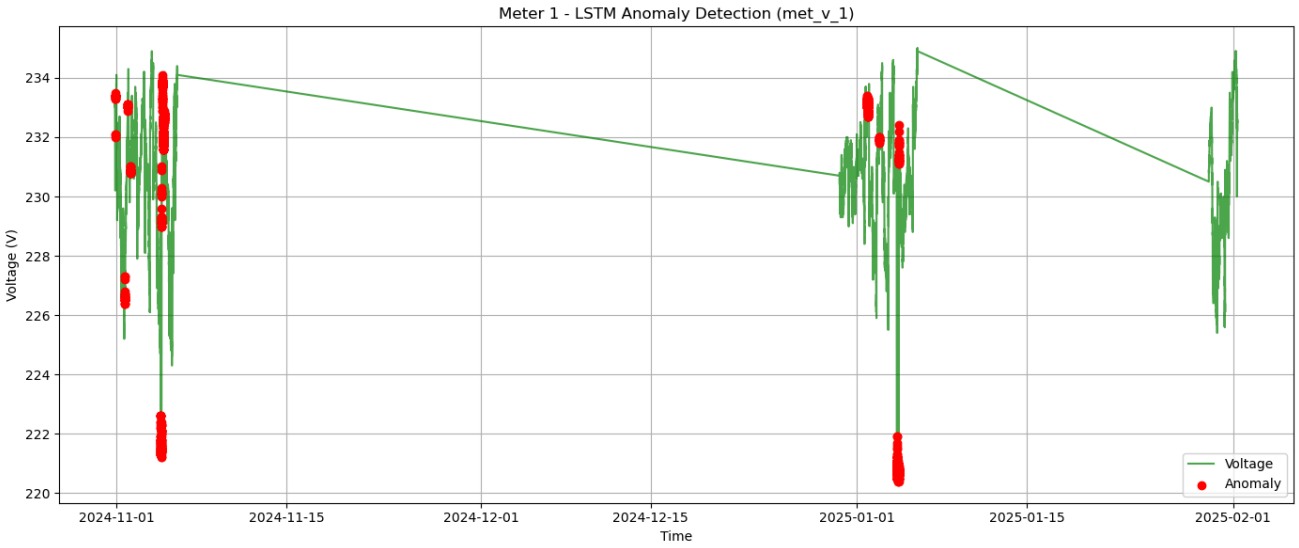


Figure 10 : Meter 1 - LSTM Anomaly Detection (met_v_1)

Transition to Stage 2: Expert Feedback & Redefinition

After initial model development, we faced challenges in validating anomalies. Although models flagged anomalies, we lacked labeled ground truth to assess their accuracy. This emphasized the importance of incorporating the anomaly logs provided by the technical expert. These logs helped validate predictions and assess model precision.

However, even with logs, the number of false positives remained high. This led to the restructuring of our dataset based on expert guidance. The expert advised segmenting the data into clearer subsets: Meter 1, Meter 2, UPS, and PDU, along with specific groups of parameters (e.g., voltage, current, power factor…). This allowed for more focused, interpretable analysis and model training.

Stage 2 :Data Refinement and Meter-Based Anomaly Modeling

In this stage, I focused on:

- **Meter 1**: Voltage, current, power factor, and sequence components
- **Meter 2**: Voltage and current only

```
# Split general dataset into Meter 1 full sets
df_met1 = df[[col for col in df.columns if '_1' in col] + ['data_hora']].copy()
```

```
# Split general dataset into  Meter 2 full sets
df_met2 = df[[col for col in df.columns if '_2' in col] + ['data_hora']].copy()
```

```
# Split met_1 data
df_met1_v = df_met1[['met_va_1', 'met_vb_1', 'met_vc_1', 'met_v_1']].copy()
df_met1_i = df_met1[['met_ia_1', 'met_ib_1', 'met_ic_1', 'met_i_1', 'met_in_1']].copy()
df_met1_fp = df_met1[['met_fpa_1', 'met_fpb_1', 'met_fpc_1', 'met_fp_1']].copy()
df_met1_seq = df_met1[['met_ia_seq_p_1', 'met_ia_seq_n_1', 'met_ia_zero_n_1', 'met_i_imb_1', 'met_i_leak_1']].copy()
```

```
# Split met_2 data
df_met2_v = df_met2[['met_va_2', 'met_vb_2', 'met_vc_2', 'met_v_2']].copy()
df_met2_i = df_met2[['met_ia_2', 'met_ib_2', 'met_ic_2', 'met_i_2', 'met_in_2']].copy()
df_met2_fp = df_met2[['met_fpa_2', 'met_fpb_2', 'met_fpc_2', 'met_fp_2']].copy()
df_met2_seq = df_met2[['met_ia_seq_p_2', 'met_ia_seq_n_2', 'met_ia_zero_n_2', 'met_i_imb_2', 'met_i_leak_2']].copy()
```

Figure 11: code snippets showing how Meter 1 and Meter 2 data were split

Windowing Function and Purpose

```
def window_dataframe(df, window_size=12):
    """
    Returns a DataFrame with columns grouped by variable, then window index.
    E.g., va-1-1, va-1-2, ..., vb-1-1, vb-1-2, ...
    """
```

This function transforms a time series into overlapping fixed-size windows. Each row becomes a window, capturing temporal behavior. It is particularly important for feeding data into LSTM models.

We applied this windowing to all Meter 1 and 2 parameter subsets. The following pipeline was used:

Phase 1: Model Selection and Evaluation

For each parameter group (voltage, current, etc.), we tested Autoencoder and LSTM models across multiple window sizes (12, 16, 25). The goal was to identify the combination with the **best precision and recall** using the **expert-defined anomaly intervals.**

```
# Show summary
summary_voltage_1 = pd.DataFrame(results)
summary_voltage_1
```

	Model	Window Size	Total Detected	True Positives	False Positives	False Negatives
0	Autoencoder	12	45	3	42	2
1	LSTM	12	45	2	43	3
2	Autoencoder	16	34	3	31	2
3	LSTM	16	34	2	32	3
4	Autoencoder	25	22	0	22	5
5	LSTM	25	22	0	22	5

Figure 12: voltage meter 1 example : summary table showing performance by model + window size

This phase helped us determine the optimal configuration (e.g., Autoencoder + 16 window) for each parameter group.

Phase 2: Final Model Training and Saving (Voltage Example)

Once the best model and window were selected, we retrained the model using the full training set and saved it for deployment.

- **Best window**: 16 (3 min)
- **Model**: Autoencoder
- **Preprocessing**: StandardScaler
- **Model Architecture**: Dense (64 → 32 → 16 → 32 → 64)

Post training:

- Computed reconstruction errors
- Defined threshold (99.8th percentile)
- Flagged anomalies
- Plotted detected points on voltage time series

Models and scalers were saved in a structured directory:

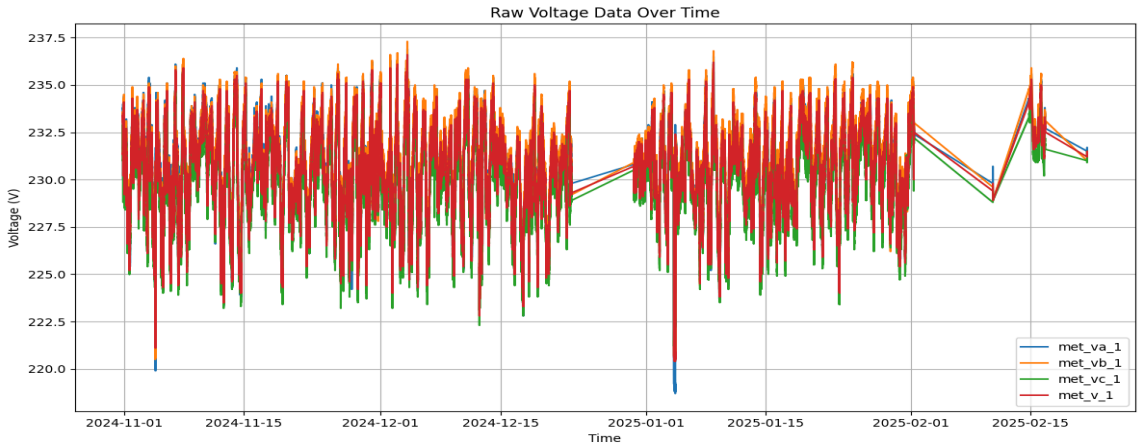
```
anomaly_detection_project/
├── meter1_voltage/
│   ├── autoencoder_model.keras
│   ├── scaler.pkl
│   └── summary.csv
...
```

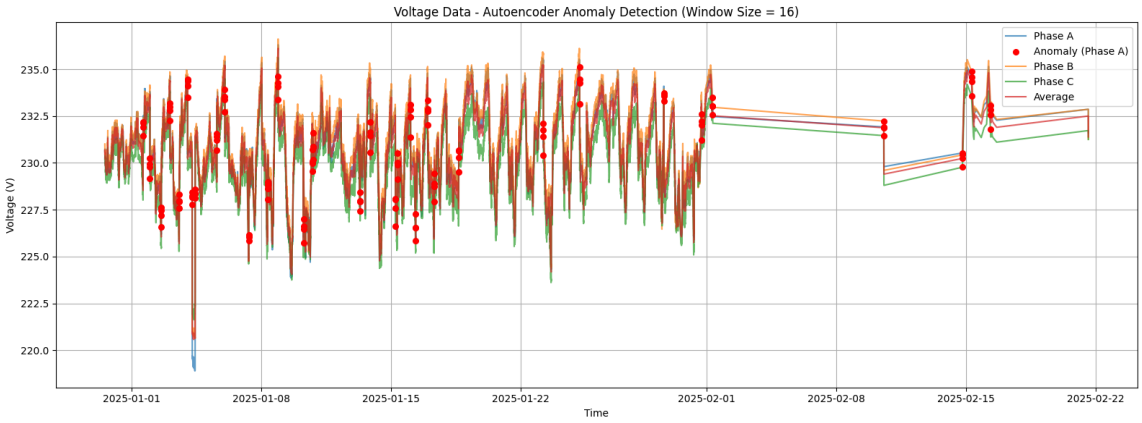
Code was prepared to reload models for future inference.

Results for all Subsets (Meter 1 & 2)

Following the same approach described above, we trained models for the following:

- **Meter 1**
  - **Voltage**: Autoencoder (16 window)





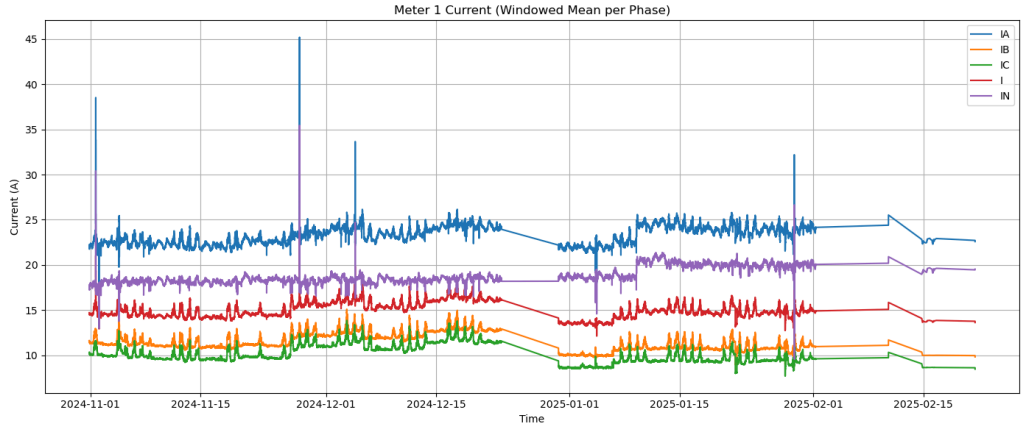
Voltage Anomaly Detection Performance Summary:

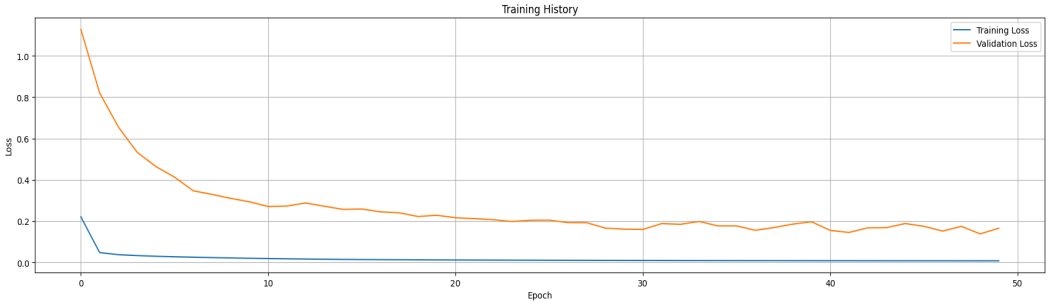
	Model	Window Size	Total Detected	True Positives	False Positives	False Negatives
0	Autoencoder	16	34	3	31	2

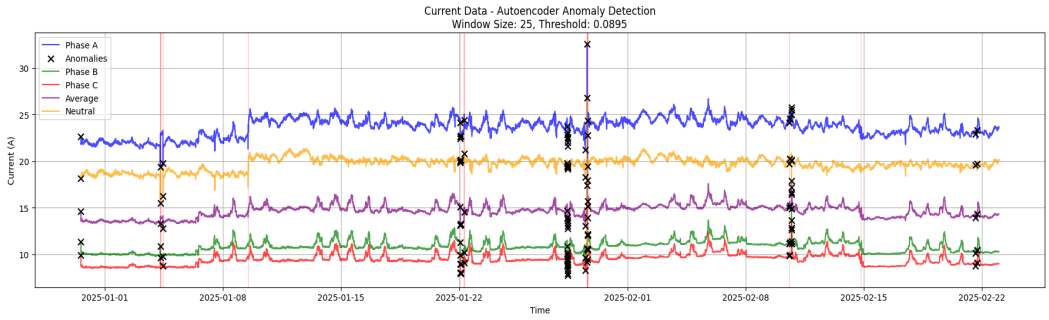
```
Detailed Results:
Total anomalies detected: 34
True positives: 3
False positives: 31
False negatives: 2

Detected Voltage Anomaly Timestamps:
DatetimeIndex(['2025-01-01 15:12:12', '2025-01-02 00:03:39',
               '2025-01-02 14:46:58', '2025-01-03 02:04:00',
               '2025-01-03 13:57:19', '2025-01-04 00:46:33',
               '2025-01-04 06:41:13', '2025-01-04 10:35:13',
               '2025-01-05 14:42:05', '2025-01-06 01:02:40',
               '2025-01-07 07:57:29', '2025-01-08 08:06:52',
               '2025-01-08 21:58:15', '2025-01-10 06:43:06',
               '2025-01-10 18:18:02', '2025-01-10 19:10:34',
               '2025-01-13 08:05:51', '2025-01-13 21:19:51',
               '2025-01-15 05:44:32', '2025-01-15 07:59:33',
               '2025-01-16 01:15:26', '2025-01-16 07:57:28',
               '2025-01-16 23:26:39', '2025-01-17 07:58:40',
               '2025-01-18 15:37:05', '2025-01-23 05:06:24',
               '2025-01-25 03:57:46', '2025-01-29 17:05:57',
               '2025-01-31 17:23:54', '2025-02-01 07:50:49',
               '2025-02-10 13:39:30', '2025-02-14 19:21:37',
               '2025-02-15 07:06:15', '2025-02-16 07:37:39'],
              dtype='datetime64[ns]', name='window_time', freq=None)

Missed Voltage Anomaly Intervals (False Negatives):
From 2025-01-04 10:33:00 to 2025-01-04 10:35:00
From 2025-02-14 21:30:00 to 2025-02-14 21:33:00
```

Figure 13 : 1. raw voltage data met1 over time/ 2.current signal with anomaly points overlayed / summary result for voltage data

- **Current**: Autoencoder (25 window)





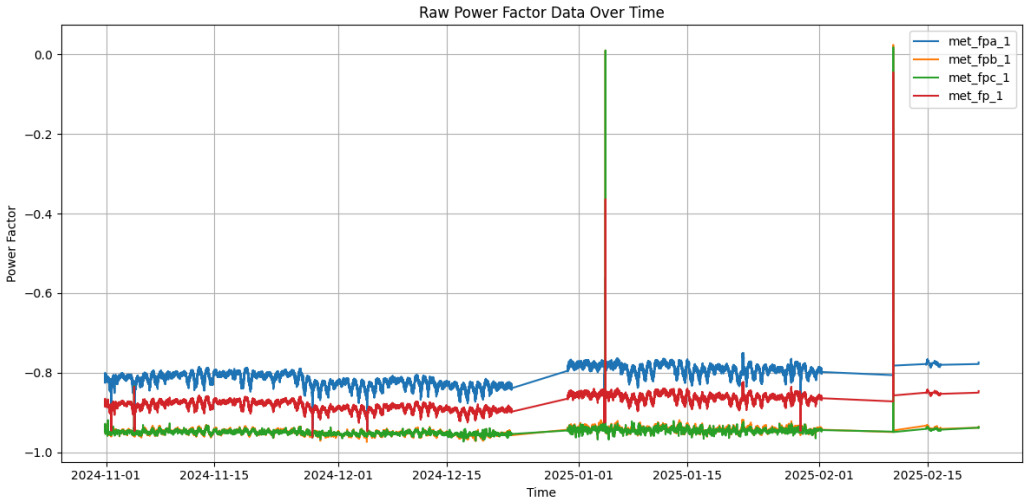


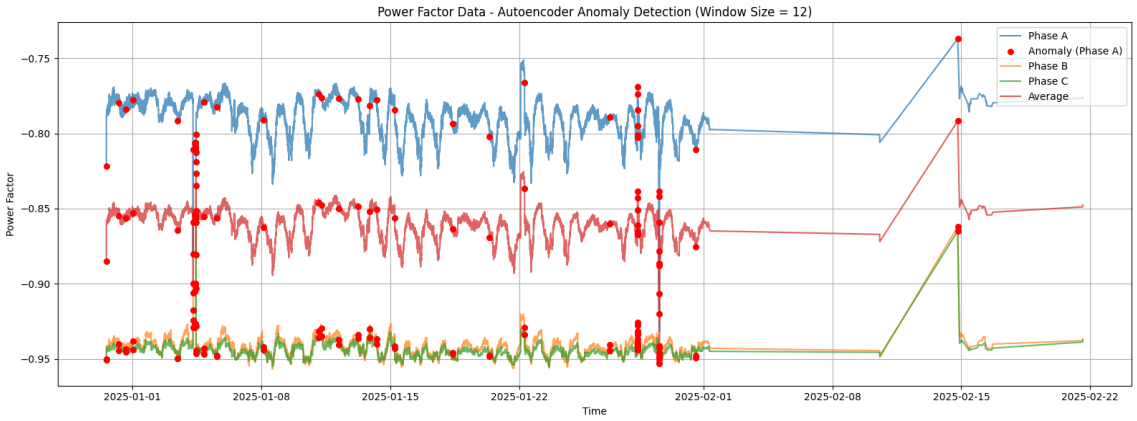

```
Best Performing Model:
Model              Autoencoder
Window Size                 25
Threshold             0.100842
Total Detected              29
Precision             0.241379
Recall                0.583333
F1-score              0.341463
Name: 4, dtype: object

Complete Results:
```

	Model	Window Size	Threshold	Total Detected	Precision	Recall	F1-score
0	Autoencoder	12	0.090693	61	0.131148	0.400000	0.197531
2	Autoencoder	16	0.083368	46	0.173913	0.444444	0.250000
4	Autoencoder	25	0.100842	29	0.241379	0.583333	0.341463
1	LSTM	12	0.015720	61	0.081967	0.250000	0.123457
3	LSTM	16	0.015135	46	0.130435	0.333333	0.187500
5	LSTM	25	0.015310	29	0.137931	0.333333	0.195122

Figure 14 : 1. raw Current data met1 over time/ 2.voltage signal with anomaly points overlayed / summary result for current data

- **Power Factor**: Autoencoder (12 window)





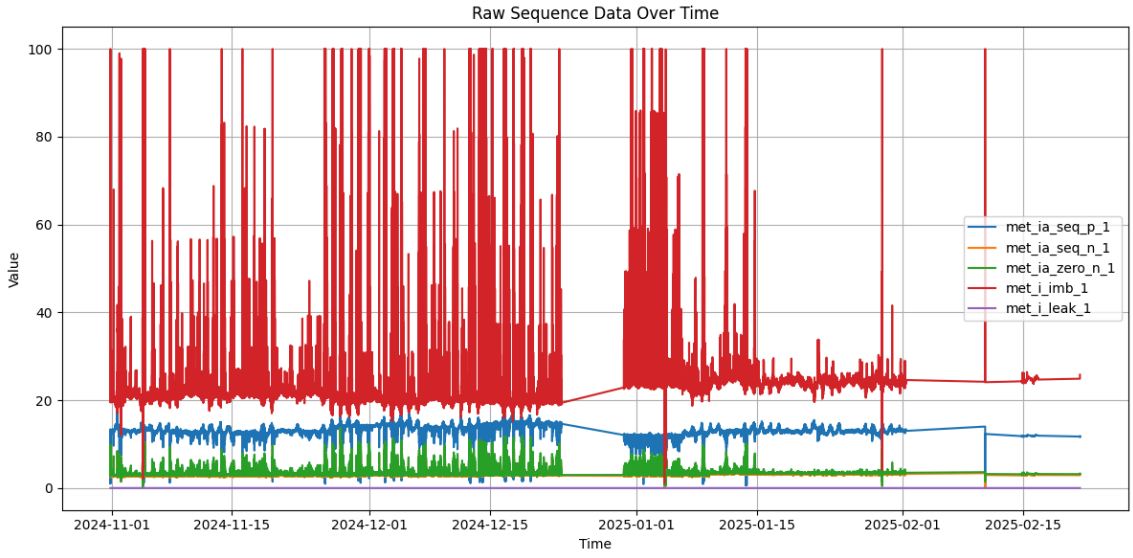
Anomaly Detection Performance Summary (Power Factor Data):

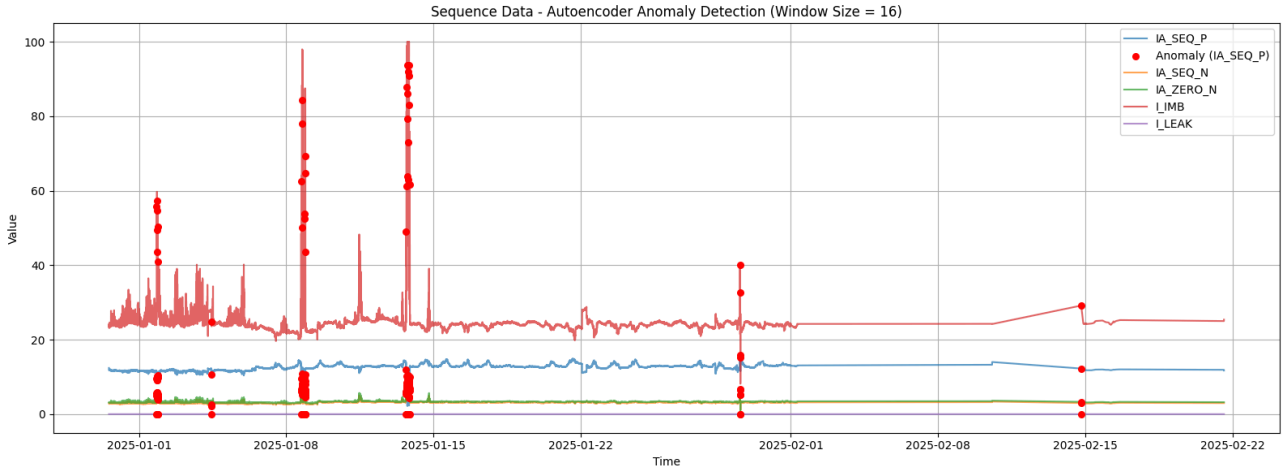
	Model	Window Size	Total Detected	True Positives	False Positives	False Negatives
0	Autoencoder	12	45	10	35	3

```
Detected Anomaly Timestamps (Power Factor Data):
DatetimeIndex(['2024-12-30 13:43:43', '2024-12-31 05:31:13',
               '2024-12-31 14:50:51', '2025-01-01 00:33:06',
               '2025-01-03 10:29:16', '2025-01-04 06:42:36',
               '2025-01-04 06:50:49', '2025-01-04 08:32:03',
               '2025-01-04 09:02:56', '2025-01-04 09:44:26',
               '2025-01-04 09:46:30', '2025-01-04 09:50:37',
               '2025-01-04 09:52:40', '2025-01-04 09:54:44',
               '2025-01-04 10:09:09', '2025-01-04 10:17:23',
               '2025-01-04 10:33:50', '2025-01-04 10:35:54',
               '2025-01-04 20:44:46', '2025-01-05 14:17:24',
               '2025-01-08 02:55:02', '2025-01-11 02:39:59',
               '2025-01-11 05:49:53', '2025-01-12 04:30:31',
               '2025-01-13 05:39:54', '2025-01-13 21:07:26',
               '2025-01-14 05:53:59', '2025-01-15 05:42:29',
               '2025-01-18 09:02:49', '2025-01-20 09:15:44',
               '2025-01-22 06:50:08', '2025-01-26 21:25:32',
               '2025-01-28 09:53:22', '2025-01-28 09:55:46',
               '2025-01-28 09:57:49', '2025-01-28 10:04:10',
               '2025-01-28 10:06:13', '2025-01-28 10:16:31',
               '2025-01-28 10:24:45', '2025-01-29 13:52:32',
               '2025-01-29 13:56:39', '2025-01-29 14:26:42',
               '2025-01-29 14:30:50', '2025-01-31 13:29:54',
               '2025-02-14 19:21:37'],
              dtype='datetime64[ns]', name='window_time', freq=None)

Missed Anomaly Intervals (False Negatives - Power Factor Data):
From 2025-01-04 06:40:00 to 2025-01-04 06:42:00
From 2025-01-22 01:42:00 to 2025-01-22 01:44:00
From 2025-02-10 13:47:00 to 2025-02-10 13:48:00
```

Figure 15 : 1. raw PF data met1 over time/ 2.pf signal with anomaly points overlayed / summary result for pf data

- **Sequence**: Autoencoder (16 window)





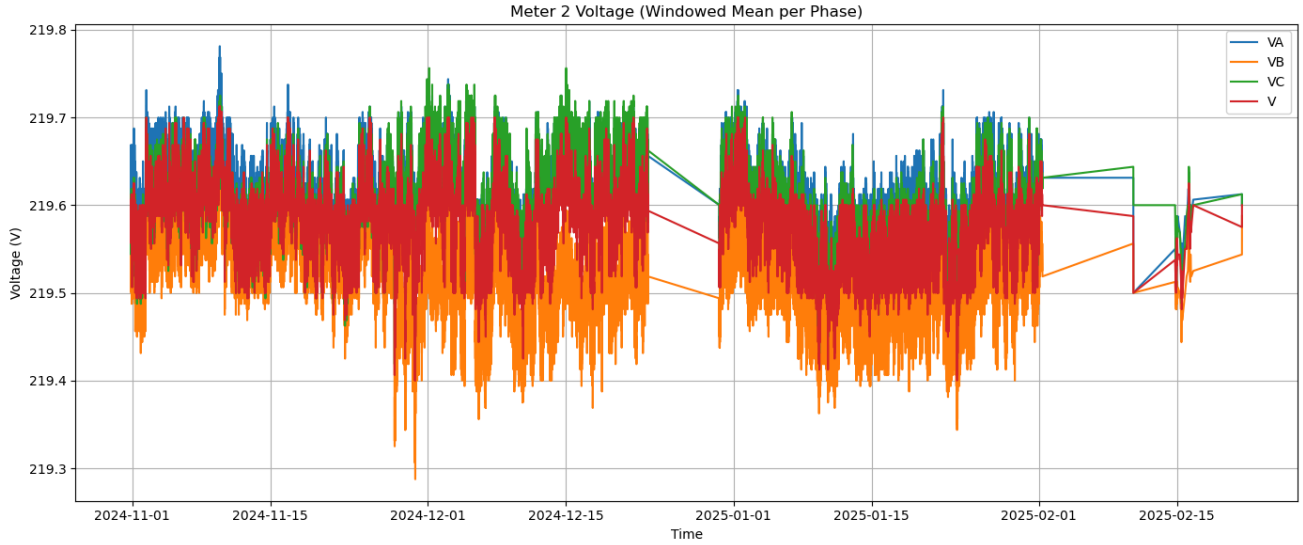
Anomaly Detection Performance Summary (Sequence Data):

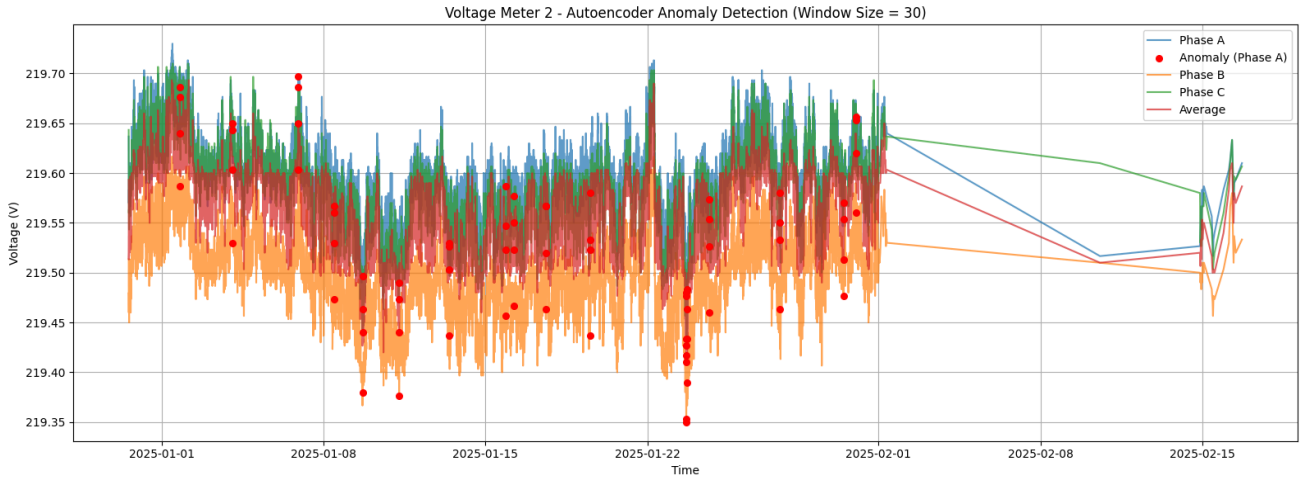
	Model	Window Size	Total Detected	True Positives	False Positives	False Negatives
0	Autoencoder	16	34	25	9	9

```
Detected Anomaly Timestamps (Sequence Data):
DatetimeIndex(['2025-01-01 20:15:03', '2025-01-01 20:20:33',
               '2025-01-01 20:23:17', '2025-01-01 20:37:00',
               '2025-01-01 20:47:59', '2025-01-01 21:18:11',
               '2025-01-01 22:02:27', '2025-01-04 10:35:13',
               '2025-01-08 17:42:12', '2025-01-08 18:09:41',
               '2025-01-08 18:15:10', '2025-01-08 18:23:25',
               '2025-01-08 21:03:10', '2025-01-08 21:22:24',
               '2025-01-08 21:33:23', '2025-01-08 21:47:07',
               '2025-01-08 21:49:54', '2025-01-13 16:54:46',
               '2025-01-13 17:26:31', '2025-01-13 17:50:50',
               '2025-01-13 18:06:16', '2025-01-13 18:19:17',
               '2025-01-13 18:30:47', '2025-01-13 18:33:42',
               '2025-01-13 19:17:11', '2025-01-13 19:22:58',
               '2025-01-13 19:34:48', '2025-01-13 19:47:18',
               '2025-01-13 19:54:49', '2025-01-13 20:18:40',
               '2025-01-13 20:44:04', '2025-01-29 13:53:54',
               '2025-01-29 13:56:39', '2025-02-14 19:21:37'],
              dtype='datetime64[ns]', name='window_time', freq=None)

Missed Anomaly Intervals (False Negatives - Sequence Data):
From 2025-01-01 20:12:00 to 2025-01-01 20:13:00
From 2025-01-02 19:20:00 to 2025-01-02 19:21:00
From 2025-01-02 19:22:00 to 2025-01-02 19:23:00
From 2025-01-03 18:01:00 to 2025-01-03 18:02:00
From 2025-01-03 18:03:00 to 2025-01-03 18:04:00
From 2025-01-11 10:25:00 to 2025-01-11 13:20:00
From 2025-01-14 17:58:00 to 2025-01-14 19:55:00
From 2025-01-29 14:26:00 to 2025-01-29 14:28:00
From 2025-01-29 14:28:00 to 2025-01-29 14:34:00
```

Figure 16 : 1. raw sequence data over time/ 2.sequence signal with anomaly points overlayed / summary result for sequence data

- **Meter 2**
  - **Voltage**: Autoencoder (30 window)





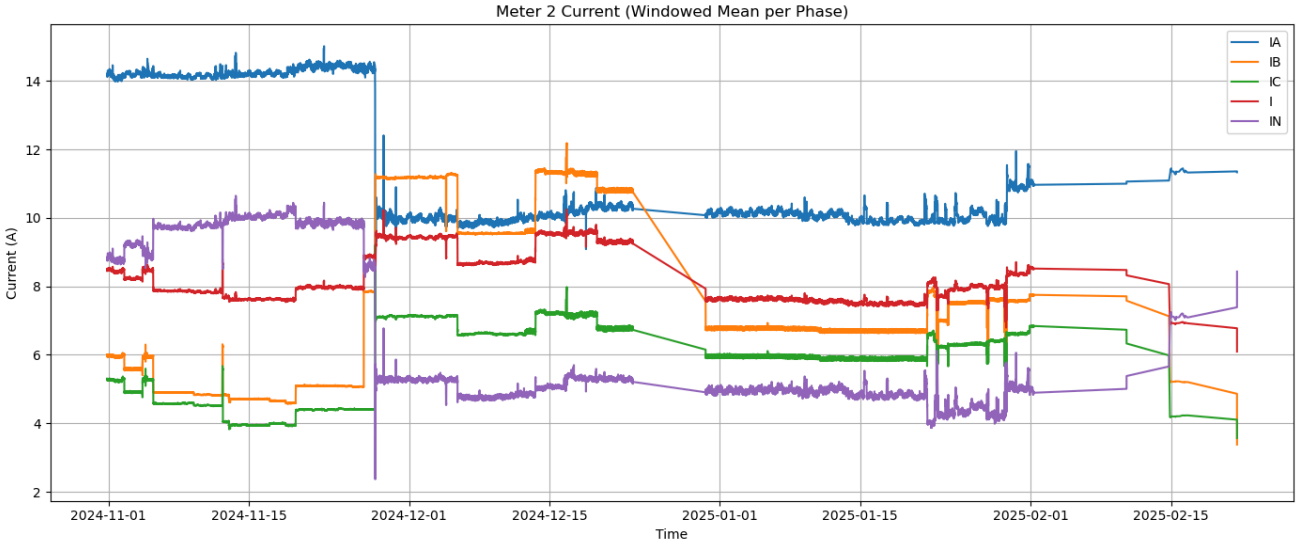
Anomaly Detection Performance Summary (Voltage Meter 2):

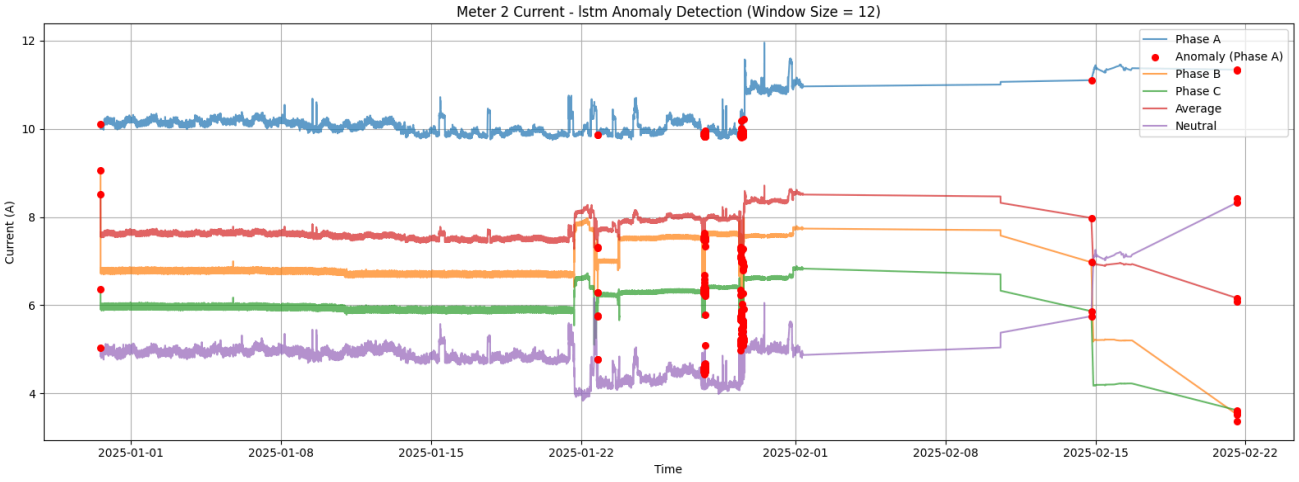
	Model	Window Size	Total Detected	True Positives	False Positives	False Negatives
0	Autoencoder	30	18	0	18	0

```
Detailed Results:
Total anomalies detected: 18
True positives: 0
False positives: 18
False negatives: 0

No missed anomalies (all supervisor intervals detected)
```

Figure 16 : 1. raw Voltage meter 2 data over time/ 2.Voltage signal with anomaly points overlayed / summary result for Voltagedata

- **Current**: LSTM-autoencoder (12 window)





Anomaly Detection Performance Summary (Meter 2 Current):

	Model	Window Size	Total Detected	True Positives	False Positives	False Negatives
0	lstm	12	45	5	40	57

Figure 16 : 1. raw current meter 2 data over time/ 2.current signal with anomaly points overlaid / summary result for current data

2.2 Ghofrane Ben Rhaiem

Part 1 : Analysis of Equipment Energy Consumption

1.1 Data Extraction

To begin, I worked with the raw dataset merged_dataset.csv, which contains over 165 parameters from multiple equipment types.
My first step was to identify and extract only the columns related to overall energy consumption, specifically:

- met_kwh_1, met_kvah_1 (meter 1)
- met_kwh_2, met_kvah_2 (meter 2)
- pdu1_kwh to pdu8_kwh (PDUs)

This allowed me to reduce the dataset to focus on total energy use without being distracted by unrelated electrical signals.

1.2 Data Cleaning

I then cleaned the energy subset by:

Converting the timestamp column data_hora into proper datetime format, handling invalid entries by coercing to NaT.
Sorting the data chronologically to prepare it for time series operations.

Verifying there were no missing values in the core kWh or kVAh columns. If gaps existed, I tested forward-fill methods to maintain consistency.

1.3 Subset Export

After cleaning, I saved the energy-only dataset as df_energy.csv.
This file is reusable by the team for:

- Simple EDA,
- Benchmarking,
- Comparing with anomaly detection results later.

1.4 Time Series Analysis

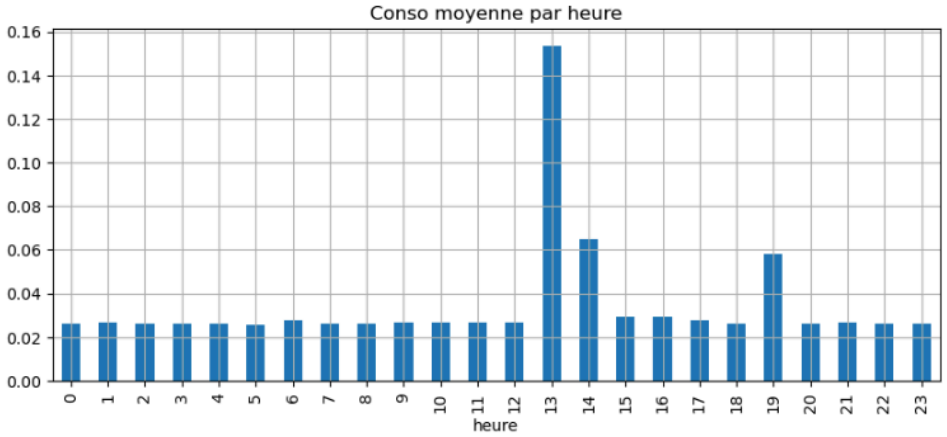
To understand the behavior of energy consumption over time, I performed several time series visualizations:

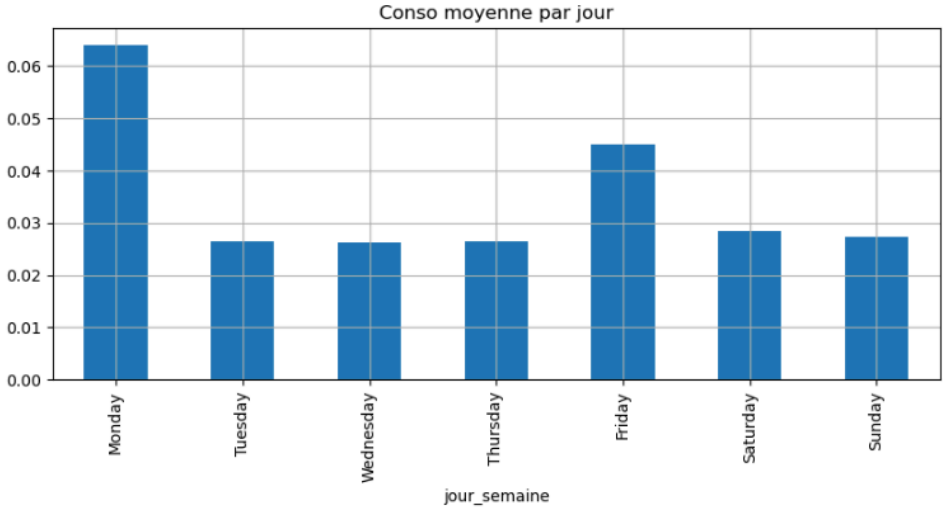
- Plotted hourly energy trends for met_kwh_1 and met_kwh_2 to observe daily usage patterns.
- Created daily total plots to spot long-term trends or seasonality.
- Plotted PDU kWh values individually to detect irregular spikes in specific units.

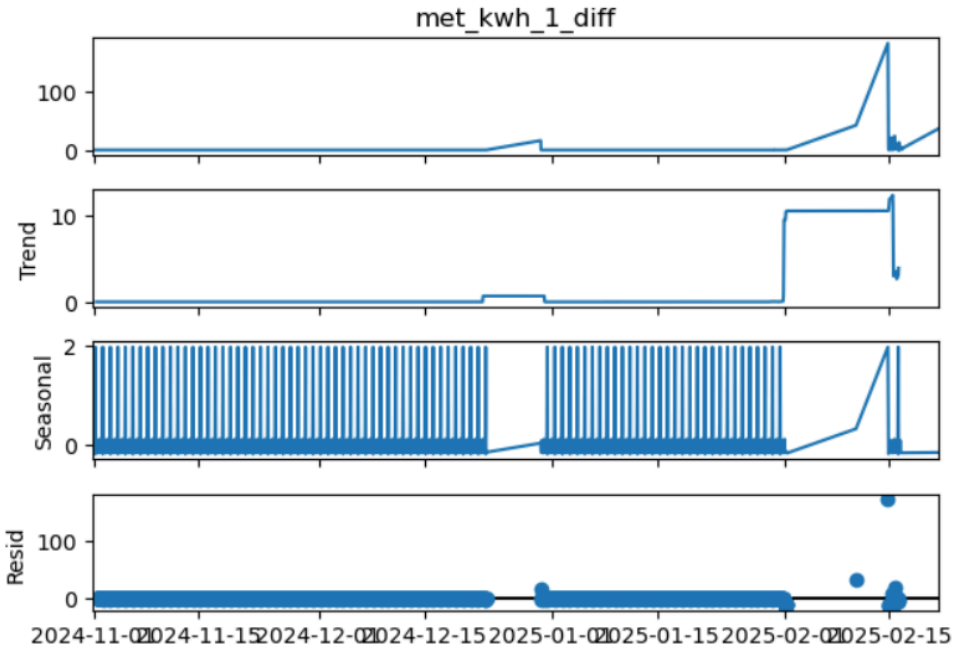
Example: I used matplotlib to overlay multiple meters on the same chart to compare their shapes and highlight potential outliers.

These plots helped reveal:

Expected usage peaks (e.g., working hours vs nights).
Unexpected spikes that could be suspicious or need further investigation.


Conso moyenne par heure
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
heure


Conso moyenne par jour
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
jour_semaine


met_kwh_1_diff
100
0
Trend
10
0
Seasonal
2
0
Resid
100
0
2024-11-01
2024-11-15
2024-12-01
2024-12-15
2025-01-01
2025-01-15
2025-02-01
2025-02-15

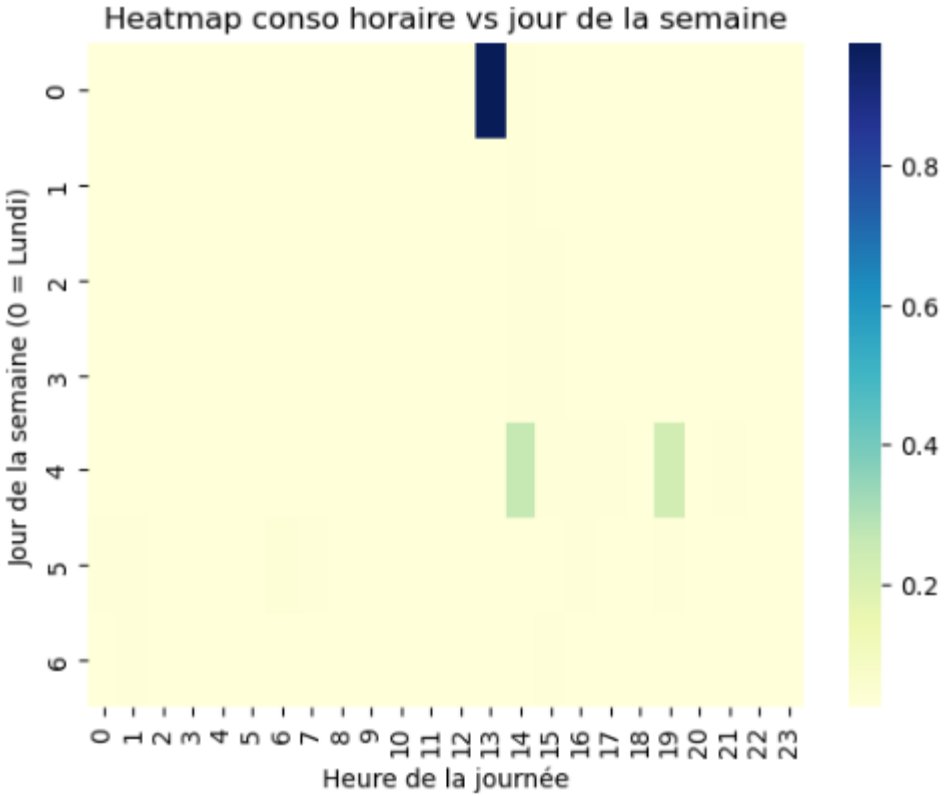
1.5 Correlation Analysis

To check how different meters and PDUs relate, I calculated pairwise correlations:

- Used df_energy.corr() to generate a correlation matrix.
- Visualized the matrix with a seaborn heatmap.

This analysis helped answer:

- Do met_kwh_1 and met_kwh_2 follow the same pattern?
- Are certain PDUs strongly correlated, indicating they are part of the same zone or process?
- Are there outliers that do not correlate at all?



These insights help check for redundancy and detect unexpected behavior when anomalies appear.

1.6 Basic Descriptive Statistics

For each energy column, I computed:

– Minimum and maximum values to check realistic bounds.

– Mean, median, and standard deviation.
– Total energy consumed over the period covered by the dataset.
I summarized these statistics in a small table that can be added in the appendix of the report.

1.7 Insights Shared with the Team

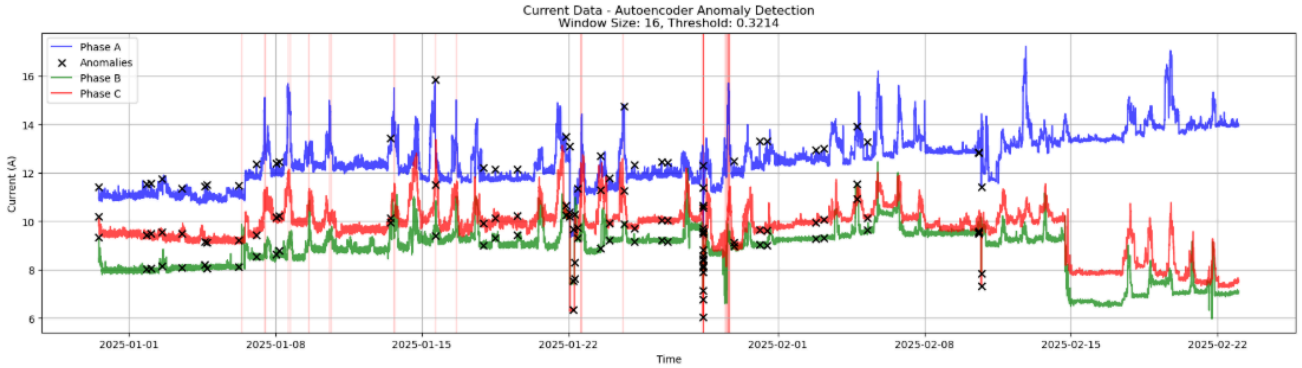
Finally, I documented and shared:

- The periods with suspicious spikes or drops.
- Recommendations for selecting good time windows for anomaly detection tests.
- Confirmation that the cleaned dataset was ready to be used by other models (e.g., UPS or PDU anomaly pipelines).

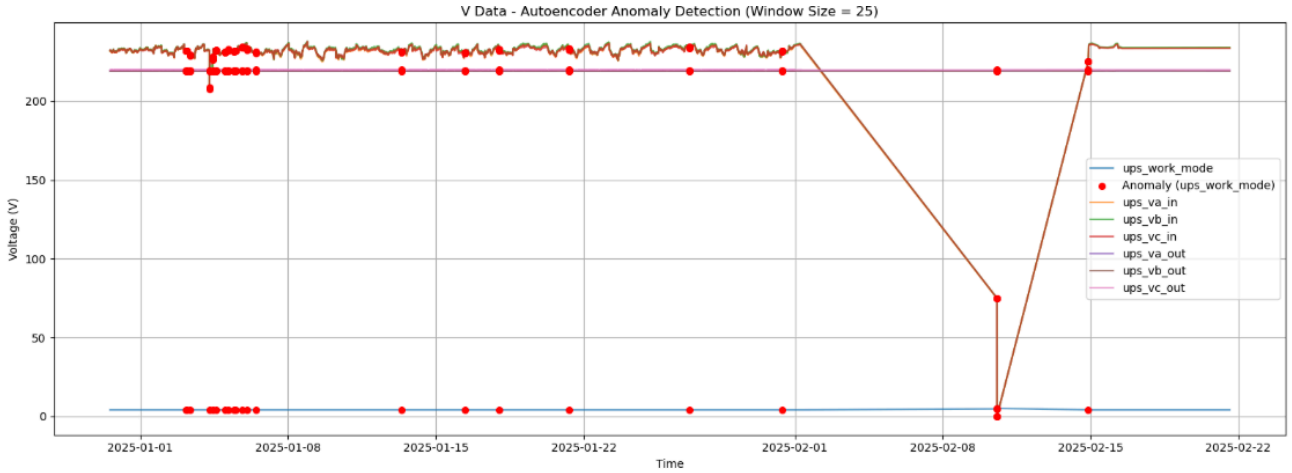
Part 2 – UPS Subsystem Anomaly Detection

2.1 Data Subsetting

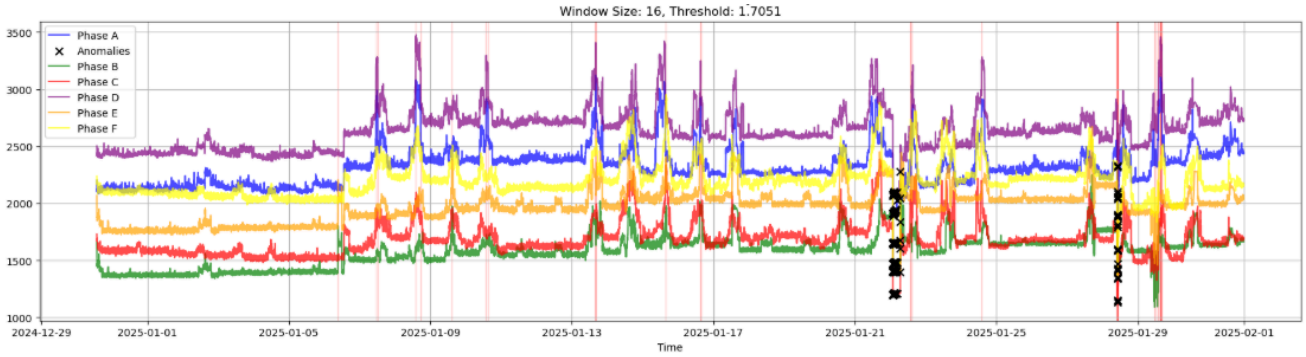
From the full dataset, I selected only UPS parameters.
After feedback from the professor,
I organized them into logical groups:
Currents: ups_ia_out, ups_ib_out, ups_ic_out



Voltages: ups_va_in, ups_vb_in, ups_vc_in, ups_va_out, ups_vb_out, ups_vc_ou

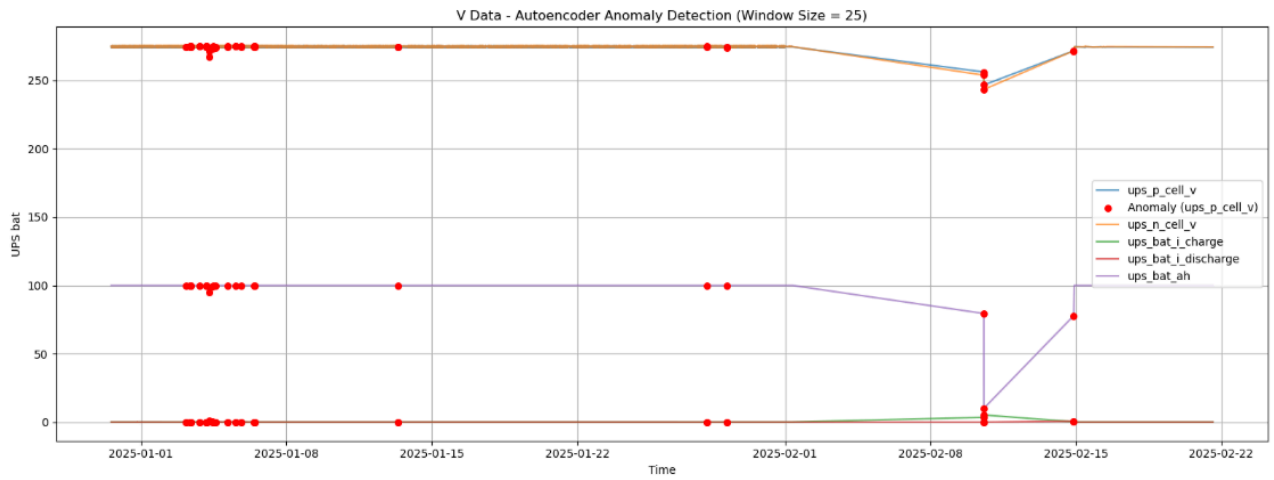


Power: ups_pa, ups_pb, ups_pc, ups_sa, ups_sb, ups_sc



Battery & Cells: ups_p_cell_v, ups_n_cell_v, ups_bat_i_charge, ups_bat_i_discharge, ups_bat_ah

+ ups_work_mode to track operation status.



This ensured that each ML model uses a meaningful subset, not an isolated signal.

2.2 **Rolling Window Feature Engineering**

I applied a rolling window to each subset:

- 12 samples ≈ 2 minutes.
- Following Leonardo's advice, I planned to also test 24–36 samples (4–6 min) for comparison.

For each window:

- Calculated rolling mean to capture the average state.
- Calculated rolling standard deviation to detect fluctuation/volatility.

Combined these into a new feature set per subset.

2.3 Isolation Forest

For each subset:

- Standardized the features using StandardScaler.
- Trained an Isolation Forest:
  contamination=0.01 to assume ~1% of points are outliers.
  Set a fixed random_state for reproducibility.
- Labeled anomalies: points classified as outliers.

Visualized:

- Plots showing e.g., ups_pa or ups_ia_out with red dots marking Isolation Forest anomalies.

2.4 PCA Autoencoder

I then used a PCA-based Autoencoder:

- Reduced the data to principal components covering 95% of variance.
- Reconstructed the inputs from these components.
- Calculated the reconstruction error.
- Set a threshold (99.5th percentile) — points with high errors were flagged as anomalies.

2.5 LSTM Autoencoder

After the unsupervised models, I built and trained an LSTM Autoencoder to capture time sequence patterns:

- Reshaped the rolling data into (samples, time steps, features) windows.
- Defined the LSTM encoder–decoder structure to learn temporal dependencies.
- Trained the model to reconstruct normal windows.
- Detected anomalies where the reconstruction error was above a set limit.

The LSTM Autoencoder added the advantage of capturing temporal context that simple models cannot.

2.6 Model Evaluation: Precision, Recall, and F1-Score

To compare the three models (Isolation Forest, PCA Autoencoder, LSTM Autoencoder) in practice:
I used the manual anomaly logs provided by the professor as the ground truth.
For each model's predictions, I calculated:

- Precision: How many detected anomalies were actually correct.
- Recall: How many real anomalies the model was able to find.
- F1-Score: The harmonic mean of precision and recall, a balanced measure of overall detection performance.

This gave us a clear way to check which model performed best in terms of finding real UPS anomalies with minimal false positives.

2.7 Refinement & Next Steps

After these comparisons, I planned adjustments:

- Tuning window sizes.
- Trying different thresholds.
- Combining model results to reduce noise.

This step-by-step approach helps ensure that the final anomaly detection pipeline for the UPS is robust, interpretable, and aligned with the manual anomaly references.

2.3 Antonin Timbert

1. **Data Acquisition & Understanding**

POWER AI WISE – Framing the Topic, was written to provide a shared foundational understanding of the electrical infrastructure monitored by the POWER AI project. It introduces the key concepts of alternating current (AC) and three-phase systems, which are essential to understanding how energy is transmitted and distributed within a data center.

The document outlines the full electrical path, from the external power grid down to the final consumption by servers and equipment. This path is broken down into four main stages:

1. External Power Supply
2. UPS (Uninterruptible Power Supply)
3. PDUs (Power Distribution Units)
4. Energy Consumption

For each of these stages, the document explains the role played in ensuring stable and efficient energy delivery. It also identifies the key variables measured at each level — including voltage, current, power factor, frequency, and energy consumption — which are critical for monitoring system performance, detecting anomalies, and optimizing energy usage.In particular:
- The power factor is highlighted as a key indicator of efficiency.
- The UPS section details the internal AC/DC/AC conversion process and its role in voltage regulation and backup.
- The PDUs are described as local distributors of clean power, while also offering real-time measurements.
- The final section on energy consumption explains how monthly and yearly metrics can be used to analyze seasonal trends, evaluate improvements, and support cost estimation.

2. Exploratory Data Analysis (EDA)

A comprehensive EDA phase was conducted to better understand the behavior, relationships, and temporal dynamics of the monitored electrical signals. The dataset, which originally featured high-frequency measurements (one value every 10 seconds), was preprocessed to ensure it was suitable for both human analysis and machine learning modeling.

Subset Structuring by Physical Layers

The original dataset was logically divided into four sub-datasets, each reflecting a specific physical layer of the energy delivery chain:

1. PDU Current (*pdu1_i* to *pdu8_i*): captures the electrical load per distribution channel.

2. PDU Power Factor (*pdu1_fp* to *pdu8_fp*): indicates the quality of energy usage per line.
3. MET Power Factor (*met_fpa_2*, *met_fpb_2*, *met_fpc_2*, *met_fp_2*): measures energy efficiency at the global supply level.
4. MET Current & Sequence Data (*met_ia_seq_p_2*, *met_ia_seq_n_2*, *met_ia_zero_n_2*, *met_i_imb_2*, *met_i_leak_2*): tracks imbalance and potential faults.

This separation enables targeted analyses that align with the physical reality of the data center infrastructure.

Temporal Resolution Adjustment

Due to the density of the raw data (1 point every 10 seconds), a downsampling function was introduced to reduce the data to 1 value every 2 minutes, improving both computational efficiency and visual clarity. This was done via a *downsample_df*() utility function, which was consistently applied to all sub-datasets. Additionally, timestamp cleaning and alignment ensured the signals across variables were properly synchronized, especially important for time-based comparisons and modeling.

Correlation Analysis

A correlation matrix was computed for all PDU current signals. This analysis revealed several strong pairwise correlations, indicating that certain lines behave similarly and may be powered by common or synchronized systems. These relationships were visualized using a network graph, where highly correlated PDU pairs were linked — a technique that offers a clear visual representation of structural dependencies between power lines.

Comparative Yearly Trends

To explore long-term patterns, the average values and peak intensities were compared between 2024 and 2025. This comparison revealed:
- Consistent usage on some PDU lines
- Emerging deviations or increased activity on others

Such differences can indicate changes in operational loads, equipment deployments, or evolving inefficiencies.

Time Series Visualization

A dedicated function *plot_weekly_2min_sampling*() was developed to generate weekly plots for each sub-dataset. These visualizations, built from the downsampled data, made it easier to detect:
- Repeating patterns (e.g., daily/weekly cycles)
- Sudden disruptions or outliers
- Underused or inactive lines

These weekly time series plots proved particularly helpful in spotting non-periodic spikes, chronic inefficiencies, or short-term anomalies, and served as a visual complement to the ML-based anomaly detection methods.

3. Anomaly Detection (Unsupervised)

The project employs a multi-model approach to detect anomalies in the electrical data using unsupervised learning algorithms. These models are applied independently to each sub-dataset (e.g., PDU currents, PDU power factor, MET power factor, MET imbalance/leakage), allowing a tailored analysis for each signal group. The goal is to identify unusual patterns or behaviors without relying on labeled data, which is typically unavailable in this type of system monitoring context.

Models Used

Isolation Forest:

This ensemble-based algorithm identifies anomalies by isolating observations in feature space. The fewer splits needed to isolate a point, the more likely it is to be anomalous. It is particularly well-suited for high-dimensional and sparse anomalies. The model outputs a binary *anomaly* label and a continuous *anomaly_score* for each time point.

Autoencoder:

A deep neural network is trained to reconstruct its input. Points with high reconstruction error are flagged as anomalies. This model is effective at capturing

non-linear correlations between variables but does not model time dependencies. Outputs include a binary anomaly flag and a *reconstruction_error* score. Thresholds for anomaly detection can be defined via statistical strategies (e.g., mean + 3×std) or percentiles.

LSTM Predictive Model:

A recurrent neural network (LSTM) is trained to predict the next time step based on a sequence of past values. High prediction errors indicate temporal anomalies — such as unexpected shifts or drifts. This model captures temporal dependencies and is particularly useful for detecting evolving anomalies over time. The output includes a *prediction_error* per sample and a corresponding *anomaly* flag based on adaptive thresholds.

Evaluation Strategy

To compare and interpret the outputs from different models, a central evaluation function — *evaluate_anomaly_models()* — was implemented. This tool summarizes the number of anomalies detected by each method, and optionally visualizes the flagged points using time series plots.

The evaluation is not only quantitative (number of anomalies) but also **qualitative**, helping to ensure that:
- Models don't under-detect (too strict)
- Models don't over-detect (too noisy)
- The anomalies are temporally aligned with significant changes in the signal

This comparative approach is essential to select the most reliable models for each sub-dataset, and to build trust in the outputs by validating consistency across models.

Observations from Application

Isolation Forest often detects **localized, sharp anomalies** based on geometric separation.
The Autoencoder identifies **non-linear outliers** in correlated feature space, especially when reconstruction is hard.

The LSTM model is more sensitive to **pattern disruptions or drifts** over time, making it suitable for detecting **behavioral changes** in the signal.
Each model brings complementary insights, and together they form a **robust ensemble** for anomaly detection in a critical energy infrastructure.

2.4 Hind Karim El Alaoui

As part of the Power AI Wise project, my main contribution was to thoroughly understand and interpret each column in the dataset. I documented the meaning and role of each variable in a Jupyter Notebook, which helped clarify the structure and logic of the data. This understanding allowed us to confidently split the dataset into three main blocks, rather than four. In fact, the met1 and met2 columns represent the same measurement units recorded from two different meters, so they were grouped accordingly. I also ensured the inclusion of the date_time column in each subset to enable time-based anomaly detection. The data used was already pre-cleaned, based on the version cleaned by Amira.

After splitting the dataset, I focused on analyzing the met2 block. Initially, I applied the Isolation Forest algorithm to detect anomalies, but it proved ineffective, as it returned an unrealistically high number of anomalies. After discussing the issue with Amira, we decided to apply the same algorithms that had successfully worked for the met1 block: **Autoencoder** and **LSTM**. These deep learning-based methods allowed us to detect anomalies more accurately, using both the values and temporal patterns in the data.

This approach yielded reliable and meaningful results, showing strong consistency with the expected anomalies provided in the project and leading to improved overall performance.

2.5 Mohamed Talhi:

My contribution focused on the design and implementation of the complete Streamlit anomaly detection application, including the integration of all subsystems (UPS, PDU, MET 1, MET 2). I developed a user-friendly interface capable of processing uploaded CSV files, extracting relevant features, applying preprocessing, and visualizing both raw and anomaly-related outputs.

Part 1 – Full Streamlit App Architecture

I built a modular application in Streamlit structured around a selector interface to choose the system family and subtype, a dynamic loader to import the appropriate models and scalers (.keras or .pkl) depending on the subtype, and a preprocessing pipeline that includes column selection (with alias fallback logic), time window transformation for LSTM inputs, and feature standardization via StandardScaler. I integrated both classical and deep learning models including Isolation Forest, Autoencoders, and LSTM Autoencoders. All logic is automatically adjusted based on the subtype selected by the user, ensuring flexibility and reducing manual errors.

Part 2 – Preprocessing Pipeline and Feature Engineering

I wrote robust functions to transform the raw time series into structured ML-ready data. The extract_X() function selects the required columns per model subtype, handles aliases like met_ to m1_, and validates feature presence. The window_dataframe() function reshapes data into non-overlapping windows for sequence-based models like LSTM. I also implemented rolling window aggregation and compact averaging (for example, every 300 lines) to reduce visualization lag and support long CSV files efficiently. This pipeline ensures that each ML model receives exactly the features and shape it expects, avoiding common dimension mismatch errors.

Part 3 – Model Integration and Error Calculation

For each subtype, the app loads the model (keras or pkl), applies the correct scaling if required, and computes either the reconstruction error for autoencoder and LSTM models or prediction residuals for pipeline-based models. Anomalies are flagged accordingly based on these outputs.

Part 4 – Data Visualization and Anomaly Plotting

I developed custom visualizations to display the raw signal time series before any preprocessing, the compacted time series after aggregation, and the anomaly reconstruction errors with optional thresholds. These visualizations include time series charts of the original uploaded CSV, charts of compacted and processed data, and plots of reconstruction error over time. To address performance issues on large files, I introduced pre-plot downsampling to ensure fast rendering in the Streamlit dashboard.

Part 5 – Error Handling and Interface Robustness

I ensured the app displays clear error messages for missing columns, invalid formats, and model mismatches such as dimension errors. I used try/except blocks to isolate failure points and prevent the app from crashing. I also used the st.cache_resource decorator to optimize model loading time and avoid unnecessary recomputation.

Part 6 – GitHub Deployment and Documentation

I prepared the application for GitHub by writing a requirements.txt file for easy dependency installation, cleaning and organizing the code structure with a dedicated artifacts folder and a modular anomaly_app.py file, and documenting clear instructions to run the application using the command streamlit run anomaly_app.py.

This work ensures that the team can reliably detect and visualize anomalies across all monitored electrical systems, while enabling future improvements and scalability. My contribution provided the operational backbone of the app used to evaluate all models and datasets throughout the project.

Conclusion :

In conclusion, the PowerAI Project successfully delivered an end-to-end anomaly detection pipeline tailored for electrical systems in data centers. The project began with meticulous data preprocessing and partitioning into functional subsets, followed by the application of unsupervised learning models to detect anomalies in voltage, current, power factor, and other critical parameters. Techniques such as Isolation Forest, Autoencoders, and LSTM models were rigorously tested and optimized, with performance evaluated using expert-validated logs. The team overcame significant challenges, including high false positives and computational constraints, by adopting contextual windowing, multivariate models, and modular deployment strategies. The final solution included a user-friendly web interface for real-time monitoring, ensuring practicality for non-technical users. The project's outcomes demonstrate the effectiveness of AI-driven approaches in enhancing operational reliability and predictive maintenance for critical infrastructure. Beyond its immediate applications, the work lays a foundation for future advancements in intelligent monitoring systems, showcasing the value of interdisciplinary collaboration and iterative model refinement in solving complex real-world problems.