SYSTEM FEATURES

CORRELATION ANALYSIS INSIGHTS

Description:

Provides exploratory data analysis tools for understanding relationships between multiple time-series sensor readings using visual and statistical methods.

User Story:

As a user, I want to visualize and calculate correlation between time-series signals to identify synchronous patterns and temporal lags.

Acceptance Criteria:

- Plots time-series signals from multiple streams.
- Generates pairplots to assess linear and non-linear relationships.
- Computes correlation matrices with visual heatmaps.
- Applies cross-correlation to determine time lags between signals.
- Provides shifted visual plots for time-aligned signal comparison.

Preconditions:

- At least **two** time-series sensor streams have been uploaded.
- All selected streams share a **common timestamp format** and are time-aligned.
- Missing values have been handled (imputation or removal).
- User has **read** access to the underlying data source.

Postconditions:

- Correlation matrix and lag analysis report persisted (e.g. JSON/CSV in reports storage).
- Visualization artifacts (heatmap, shifted plots) are cached for the user session.
- Audit log entry created recording parameters and execution time.

Dependencies:

- Python libs: pandas, numpy, scipy, seaborn or equivalent plotting library.
- Time-series ingestion service (TBD): must expose aligned streams.
- Authentication/Authorization service for data access.

Z-Score Anomaly Detection

Description:

Applies Z-score-based statistical analysis to detect anomalies by identifying data points that lie several standard deviations away from the mean.

User Story:

As a user, I want to identify extreme spikes or drops in the sensor data using a statistical thresholding technique.

Acceptance Criteria:

- Calculates rolling mean and standard deviation for each signal.
- Flags values exceeding the Z-score threshold.
- Displays anomalies on the signal plot with visual markers.
- Allows adjustable threshold settings.
- Outputs detected anomalies in CSV/JSON format.

Preconditions:

- Target sensor's time-series is **clean**—no nulls or non-numeric entries.
- User has input a **rolling window size** and **Z-threshold**.
- Historical data of at least **[TBD]** points exists for stable statistics.

Postconditions:

- Anomaly flags written to anomaly store and made available via API.
- Plots annotated with anomaly markers are saved/exported.
- User settings (window size, threshold) persisted for future runs.

Dependencies:

- Python libs: pandas, scipy.stats, plotting tool (e.g., matplotlib or plotly).
- Configuration service for storing user thresholds.
- Data quality service to validate input before processing.

Developed By: Alireza Montazeri

LSTM-Based Anomaly Detection

Description:

Uses an LSTM (Long Short-Term Memory) neural network to model time-series data, identify patterns, and detect deviations based on prediction errors.

User Story:

As a user, I want to use deep learning to detect time-dependent anomalies in signals with complex temporal patterns.

Acceptance Criteria:

- Constructs sequences using sliding window technique.
- Trains LSTM model on historical data.
- Predicts next values and calculates deviation.
- Flags data points where prediction error exceeds threshold.
- Visualizes both original and predicted signals with anomaly highlights.

Preconditions:

- A historical dataset labeled “normal” (or sufficiently large unlabeled) is available.
- Data preprocessed: **scaled/normalized**, converted into sliding-window sequences.
- Model hyperparameters configured (window size, layers, epochs).
- Compute resources (GPU/CPU) allocated for training/inference.

Postconditions:

- Trained LSTM model artifact stored in model registry.
- Prediction errors logged and anomaly flags output to results store.
- Dashboard updated with original vs. predicted plot overlays.

Dependencies:

- Deep-learning framework (e.g., TensorFlow or PyTorch).
- Feature store for retrieving pre-processed sequences.
- Model registry service for versioning and deployment.

Developed By: Georgia Quach, Hai Nam Le

Isolation Forest Anomaly Detection

Description:

Implements the Isolation Forest algorithm to detect anomalies by randomly partitioning the dataset and isolating outliers.

User Story:

As a user, I want to use a tree-based model to detect anomalous patterns in high-dimensional time-series data.

Acceptance Criteria:

- Generates temporal features (rolling mean, std dev, lag).
- Fits Isolation Forest model on derived features.
- Computes anomaly scores and classifies data points.
- Displays anomaly results overlaid on signal plots.
- Provides downloadable anomaly report.

Preconditions:

- Temporal features generated (rolling mean, std dev, lag features) and stored.
- Feature set **scaled** (e.g. via Standard or MinMax scaler).
- User has defined contamination parameter (expected outlier ratio).

Postconditions:

- Isolation Forest model artifact and feature scaler stored.
- Anomaly scores and labels exported to anomalies database.
- Overlaid signal plots generated and saved for user review.

Dependencies:

- scikit-learn IsolationForest implementation.
- Feature engineering pipeline service.
- Reporting/storage service for anomaly results.

Developed By: Li Wan, Arnav Ahuja

One-Class SVM Anomaly Detection

Description:

Utilizes a One-Class SVM algorithm to detect deviations from learned "normal" behavior in time-series sensor data.

User Story:

As a user, I want to model normal patterns in my sensor data and detect when new readings deviate from expected behavior.

Acceptance Criteria:

- Extracts statistical features from each sensor stream.
- Trains One-Class SVM on normal data segments.
- Predicts anomalies as outliers outside the learned boundary.
- Highlights anomalous segments visually.
- Outputs result set with timestamps and anomaly indicators.

Preconditions:

- Dataset of **normal-only** behavior available for fitting.
- Features extracted and **scaled** consistently with training.
- User has specified SVM kernel type and nu (outlier proportion).

Postconditions:

- One-Class SVM model and scaler artifact stored in registry.
- New data classified; outlier flags written to results store.
- Visual report highlighting detected outliers generated.

Dependencies:

- scikit-learn OneClassSVM implementation.
- Preprocessing pipeline for feature extraction and scaling.
- Model management service for deployment and version control.

Developed By: Joy Jayesh Patel, Mariam Suleman Banda