Software Requirements Specification (SRS)

IoT Management Data Science

Version: 2.0
Date: May 19, 2025

---

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements for the Data Science module of the Intelligent IoT Management platform. The module allows users to analyze time-series sensor data through timestamp-based and sliding window correlation, stream selection, and anomaly detection. It supports data cleaning, normalization, and interactive visualization of results. Machine learning-based anomaly detection methods have been implemented and evaluated and will be integrated in future versions. This document also details upcoming features which have been completed but are yet to be integrated.

1.2 Scope

This system allows users to:

- Load and validate predefined time-series datasets
- Select time frames for analysis
- Choose window size for rolling correlation
- Choose time stamps for targeted correlation analysis
- Normalize data for consistent comparison
- Compute correlation for the selected time frames
- Compute sliding correlation using window size
- Identify the most correlated stream pairs
- Generate interactive visualizations
- Export correlation results in CSV and JSON formats
- *(Future) Upload and validate custom datasets*
- *(Future) Detect anomalies using user-selected machine learning models*
- *(Future) Visualize ML-based anomaly results within the plotting interface*

1.3 Definitions, Acronyms, and Abbreviations

- **IoT**: Internet of Things
- **CSV**: Comma-Separated Values
- **JSON**: JavaScript Object Notation
- **MAD**: Median Absolute Deviation
- **LSTM**: Long Short-Term Memory neural network
- **SVM**: Support Vector Machine
- **UI**: User Interface
- **EDA**: Exploratory Data Analysis

2. Overall Description

2.1 Product Perspective

The Data Science module is a core analytical component of the larger IoT web platform. It interacts with the backend to load and process predefined sensor datasets and sends results on the front end to create interactive visualizations and exportable correlation outputs. While initial versions use predefined datasets, future enhancements will include user-upload functionality. Machine learning-based anomaly detection has been developed and will be integrated into the interface in future releasees.

2.2 User Needs

- Intuitive UI for selecting time frames, window size, and sensor streams
- Real-time correlation insights over static and sliding windows
- Exportable correlation results in CSV and JSON formats
- Interactive and responsive visualizations
- Access to multiple anomaly detection methods
- Ability to understand and compare stream relationships visually

2.3 Constraints

- Frontend must display results within a few seconds after user interaction
- Currently supported data formats: CSV and JSON (preloaded)
- Data streams must contain consistent and regular timestamps
- At least two sensor streams are required for correlation
- Machine learning outputs are not yet available in the frontend UI

2.4 Assumptions and Dependencies

- The first column in the dataset is a valid timestamp in recognizable format
- The dataset is loaded from the backend (no user-upload currently)
- All required visualization libraries (e.g., Plotly, Chart.js) are available at the front end
- Dataset sizes are assumed to be $\leq$10,000 rows for smooth visualization and processing

3. System Features

3.1 Load Time-Series Data

Description:
Loads a predefined time-series dataset from the backend and validates its structure before further analysis. The system ensures that the data includes a time-tamp column, and it qualifies for correlation processing. *This will be further extended to the "Input Time-Series Data" in future.*

User Story:
As a user, I want the system to validate the pre-loaded dataset so I can be confident it is correctly structured for time-series analysis.

Acceptance Criteria:

- Confirms the presence of a valid timestamp column as the first column
- Validates column headers and stream data types
- Ensures the dataset includes at least two sensor streams in addition to the time column
- Provides internal error messages if formatting issues are detected

Developed By: Mariam Suleman Banda, Mai Teo

3.2 Normalize Data

Description:
Performs data cleaning and time-series validation by ensuring consistent timestamp intervals (e.g., 10-minute gaps) across all streams. Missing values are handled via interpolation, and Min-Max scaling is applied to bring all streams to a comparable numerical range for correlation analysis.

User Story:
As a user, I want my time-series data to be cleaned, interpolated, and normalized so that I can fairly compare sensor readings across different streams over consistent time intervals.

Acceptance Criteria:

- Confirms dataset is structured as time-series (with valid timestamp column)
- Aggregates or interpolates data into consistent 10-minute intervals
- Fills missing values using forward-fill or linear interpolation
- Detects and smooths outliers (if, any)
- Applies Min-Max normalization to each selected stream
- Outputs clean, scaled data ready for correlation or anomaly detection

Developed By: Quang Minh Nhu, Li Wan

3.3 Choose Time Frames

Description:
Allows selection of a time interval through calendar-based timestamp input. This feature enables users to filter the dataset by specifying a start and end time to narrow the scope of analysis.

User Story:
As a user, I want to select a specific time range using timestamps so I can focus my analysis on the most relevant period of the dataset.

Acceptance Criteria:

- User can select start and end timestamps via a calendar input
- Data is filtered to include only rows within the selected time range
- Filtered data is previewed before further analysis
- Correlation and visualization features apply only to this filtered range

Developed By: Keyi Tao

3.4 Choose Window Size

Description:
Allows users to define a window size (number of data points) and apply a sliding window across the dataset. This enables dynamic segmentation of time-series data for rolling correlation analysis, where correlations are calculated for each window as it moves across the timeline.

User Story:
As a user, I want to set a window size and perform rolling correlation, so I can observe how relationships between data streams evolve over time.

Acceptance Criteria:

- User can input or select window sizes
- Data is automatically segmented into overlapping windows based on the selected size
- Correlation is calculated within each window for selected data streams
- Each window's correlation result is visualized through heatmaps or line plots

Developed By: Arnav Ahuja

3.5 Choose Data Streams

Description:
Enables users to select specific data streams (e.g., s1, s2, s3) from the dataset for targeted correlation analysis. This ensures the system focuses only on relevant variables and avoids unnecessary computation.

User Story:
As a user, I want to select specific streams from my dataset so I can focus my analysis on the most relevant variables.

Acceptance Criteria:

- Streams are automatically detected from the dataset header
- Dynamic checkboxes are generated for each stream
- At least two streams must be selected to calculate correlation
- Only selected streams are visualized and processed
- Selected stream names are displayed clearly on plots and outputs

Developed By: Eberenna Ekene Ezenwa, Prethyasha Madhavan

3.6 Get Correlation

Description:
Calculates correlation values across selected data streams within the chosen timestamp range. The results are displayed through visualizations such as heatmaps and line plots to help users understand inter-stream relationships during that specific time window.

User Story:
As a user, I want to analyze correlations between data streams within a specific time range so I can understand how they relate during that period.

Acceptance Criteria:

- Calculates correlation matrix based only on the filtered timestamp range
- Generates a line plot for all pairs for visualization
- Updates automatically when the user modifies the selected streams or time range
- Clearly displays selected stream names and correlation values

Developed By: Li Wan

3.7 Sliding Window Correlation

Description:
Performs rolling correlation analysis across overlapping segments of the dataset, based on the user-defined window size. This feature calculates how correlation between selected data streams evolves over time, with each window generating a separate correlation value.

User Story:
As a user, I want to observe how the correlation between sensor streams changes over time using a sliding window, so I can detect shifting patterns or anomalies.

Acceptance Criteria:

- Applies user-defined window size to segment the dataset into overlapping intervals
- Produces a time series of correlation values for each stream pair
- Results are visualized using line plots of correlation values
- Automatically updates when window size or stream selection changes

Developed By: Arnav Ahuja, Mai Teo

3.8 Identify Most Correlated Streams

Description:
Identifies and displays the pair(s) of sensor streams with the highest correlation value based on the selected timestamp range. This feature helps users quickly focus on the most strongly related signals in the dataset.

User Story:
As a user, I want the system to highlight which sensor streams are most correlated so I can focus on analyzing key relationships

Acceptance Criteria:

- Computes correlation values across all selected stream pairs
- Identifies the stream pair(s) with the highest correlation coefficient
- Displays the most correlated streams visually
- Automatically refreshes results when time range, window size, or stream selection changes

Developed By: Mai Teo

3.9 Get Visualization

Description:
Generates interactive visualizations for the selected streams and time frames, displaying normalized values along with overlaid correlation trends and anomaly points. These plots help users interpret data behavior visually and detect patterns or deviations.

User Story:
As a user, I want to explore visual plots of my selected data streams so I can quickly understand trends, correlations, and anomalies.

Acceptance Criteria:

- Interactive plots support zooming, panning, and stream toggling
- Line plots display stream values across time
- Overlays include correlation trends and anomalies (if any)
- Automatically updates based on user selections (streams, time, or window size)
- Supports clear legends and visual distinction between normal data and anomalies

Developed By: Hai Nam Le, Georgia Quach

3.10 Export as CSV and JSON

Description:
Enables the export of processed correlation results into both CSV and JSON formats. CSV is intended for human-readable summaries and spreadsheets, while JSON is structured for backend integration and web-based data exchange. Both formats include key metadata such as timestamps, stream pairs, and correlation values.

User Story:
As a user or backend developer, I want to export correlation results in CSV and JSON formats so I can reuse them in other tools, systems, or interfaces.

Acceptance Criteria:

- Correlation results include stream names, timestamps, and correlation coefficients
- CSV export provides a flat table format suitable for spreadsheets or reporting
- JSON export uses a nested structure for API-based backend integration
- Supports both timestamp-based and sliding window correlation modes
- Files are auto generated or triggered upon user request or function call
- Output files include headers/keys and are saved with unique, identifiable names

Developed By: Mai Teo, Arnav Ahuja

4. Future Work

4.1 Feature Integrations

These features have been implemented and tested in isolation but are yet to be fully integrated into the frontend/backend workflow. Their inclusion in future sprints will complete the core functionality of the system.

4.1.1 Get Time-Series Data:
Developed by: *Prethyusha Madhavan, Billy Thai*

This feature allows users to upload IoT time-series datasets in CSV or JSON format. It includes backend support for validating file structure, automatically detecting the timestamp column, and providing a preview of the first 10 rows before proceeding. Errors are raised for incorrectly formatted or un-processable files.

The core logic has been implemented and verified, but the feature has not yet been integrated into the frontend interface or fully connected to the correlation and preprocessing pipeline. Integration work is planned for an upcoming trimester.

4.1.2 Input Time-Series Data

Developed by: *Mariam Suleman Banda*

This feature allows users validate the uploaded CSV or JSON format. It includes backend support for validating file structure, automatically detecting the timestamp column, and providing a preview before proceeding. Errors are raised for incorrectly formatted or un-processable files.

The core logic has been implemented and verified on the backend, but the feature has not yet been integrated into the frontend interface. User interaction for upload, preview, and validation feedback remains to be implemented. Full pipeline connectivity to preprocessing, correlation, and visualization components is planned for a future version.

4.1.3 Detect Anomaly

Developed by: *Hai Nam Le, Georgia Quach*

This feature supports anomaly detection using various methods including Z-score, Isolation Forest, and Median Absolute Deviation (MAD). The backend implementation includes logic for detecting and flagging anomalies, generating annotated plots, and producing a structured anomaly report.

The anomaly detection methods have been documented and validated in the team's Machine Learning Report. However, this functionality has not yet been linked to frontend controls (e.g., dropdown method selector) or integrated with the correlation pipeline for real-time analysis. Visualisation and interactivity features will be developed in a future version.

4.2 Machine Learning Model Integration

Initial implementations of four machine learning algorithms for anomaly detection have been completed by the Data Science team. These models were developed and documented in the team's Machine Learning Report. Each model was designed for time-series anomaly detection and tested on the project's datasets. While each method has been implemented and evaluated, none have been integrated into the platform yet. Future work will involve unifying their outputs, enabling user selection from the frontend, and visualizing the results within the system's interface.

4.2.1 Z-Score Method

Developed by: *Alireza Montazeri, Devanshi Tyagi*

The Z-score method provides a statistical baseline for anomaly detection by identifying points that deviate significantly from the mean value. The implementation involves calculating rolling means and standard deviations and marking points where the Z-score exceeds a chosen threshold. This method is simple, explainable, and effective for detecting sudden spikes or drops.

4.2.2 LSTM (Long Short-Term Memory)
Developed by: *Georgia Quach, Hai Nam Le*

The LSTM model is a deep learning-based approach that learns sequential dependencies in time-series data. Georgia implemented this using rolling window sequences to predict future values and flag deviations as anomalies. The implementation includes model architecture, training, loss tracking, and anomaly evaluation.

4.2.3 Isolation Forest
Developed by: *Li Wan, Arnav Ahuja*

The Isolation Forest is an unsupervised algorithm that isolates anomalies using random tree splits. Temporal features such as rolling mean, lag, and standard deviation have been engineered before applying the model to each stream individually. Visualization of flagged anomalies was also included.

4.2.4 One-Class SVM
Developed by: *Joy Jayesh Patel, Mariam Suleman Banda*

The One-Class Support Vector Machine was implemented as an unsupervised method trained on normal data to detect deviations. The model used engineered statistical features per stream and defined an anomaly boundary in high-dimensional space. Hai Nam also provided visual plots comparing normal vs. anomalous segments.

5. External Interface Requirements

5.1 User Interface

- Interactive dashboard for loading predefined datasets and visualising results
- Calendar selectors for choosing time frames
- Input fields and sliders for setting window size
- Checkboxes for selecting data streams
- Visualization area using Plotly for line plots, heatmaps, and anomaly overlays
- Export buttons for downloading results in CSV and JSON formats
- Responsive layout compatible with desktop and tablet screens
- *(Future) Dropdown menu for anomaly detection method*
- *(Future) File upload field for user-submitted datasets*

5.2 Software Interfaces

- Processes predefined data sets stored on the backend (e.g., CSV files)
- Integrates with backend via Python modules and RESTful APIs built with Flask or Django
- Communicates with anomaly detection models and correlation functions via internal API calls
- Output returned as JSON for frontend rendering and optionally stored as CSV for export
- It depends on libraries such as Pandas, NumPy, Scikit-learn, Plotly, and TensorFlow/Keras for analysis and modelling

6. Non-Functional Requirements

6.1 Performance

- Must process and display correlation and visualization results within 3 seconds for datasets with ≤10,000 rows
- Sliding window and anomaly detection operations should be completed within reasonable latency

6.2 Security

- Predefined datasets are stored securely on the backend with restricted access
- Anomaly feedback and model selection input (if enabled) will be stored anonymously and used only for internal evaluation
- *(Future) Uploaded files will be validated, stored temporarily, and deleted after processing to prevent misuse*

6.3 Reliability

- The system must handle missing or corrupted values in datasets gracefully, with fallback interpolation or warnings
- Should provide clear visual and textual feedback if input data is inconsistent, malformed, or fails validation checks
- Backend processes must log and handle runtime errors (e.g., file parsing, JSON encoding) without interrupting the user experience

7. Appendices

7.1 Glossary

- **Time-Series Data**: A sequence of observations recorded at specific time intervals, typically used to monitor changes over time in sensor or system behavior.
- **Normalization**: The process of scaling numerical data—commonly using Min-Max scaling—so values from different streams can be compared on the same scale.
- **Anomaly**: A data point or segment that significantly deviates from the norm or expected trend, identified via statistical or machine learning methods.
- **Heatmap**: A visualization technique where values are represented using a range of colors, often used to display correlation matrices.
- **Correlation**: A statistical measure that describes the strength and direction of a relationship between two data streams.
- **Sliding Window**: A method of segmenting time-series data into overlapping intervals for rolling analysis (e.g., correlation over time).
- **Z-score**: A statistical metric representing how many standard deviations a data point is from the mean.
- **LSTM**: Long Short-Term Memory, a deep learning architecture for learning from sequential data.
- **Isolation Forest (iForest)**: An unsupervised anomaly detection algorithm that isolates outliers through recursive data partitioning.
- **One-Class SVM**: A machine learning algorithm that models normal behavior to mark deviations as anomalies.

8. References

- Python Official Documentation – https://docs.python.org/3/
- Pandas Documentation – https://pandas.pydata.org/
- NumPy Documentation – https://numpy.org/doc/
- Matplotlib Documentation – https://matplotlib.org/stable/
- Plotly Documentation – https://plotly.com/python/
- Scikit-learn Documentation – https://scikit-learn.org/stable/
- TensorFlow (Keras) Documentation – https://www.tensorflow.org/guide/keras
- Flask Documentation – https://flask.palletsprojects.com/
- Django Documentation – https://docs.djangoproject.com/
- Jupyter Notebook – https://jupyter.org/
- LSTM Time-Series Guide – https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
- Isolation Forest (iForest) – https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html
- One-Class SVM – https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html