Abstract

"If we could first know where we are and whither we are tending, we could better judge what to do and how to do it". - Abraham Lincoln

Acronyms

DQ	Data Quality
WSN	Wireless Sensor Network
IoT	Internet of Things
FFT	Fast Fourier Transform
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
RFR	Random Forest Regressor
SVM	Support Vector Machine
k-NN	k-Nearest Neighbors
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
OPTICS	Ordering Points To Identify the Clustering Structure
LOF	Local Outlier Factor

1 Methodology

1.1 Project Deliverables

This research project delivers the following outcomes:

- A literature review focussed on data quality monitoring in smart cities.
- A taxonomy of data quality dimensions for object-detecting wireless sensor networks.
- A machine learning pipeline for automated data quality monitoring of pedestrian data streams.
- A pathway to a real-time monitoring system for detecting data quality issues in smart city data streams
- A dashboard for visualising data quality issues.

The key scientific discoveries that this research project seeks to make are:

- Understanding how and why data quality issues occur in smart city data streams.

- Identifying the key data quality dimensions that are important for smart city data streams.
- Finding the limits of sparsity and noise in data used for predictive modelling (what is the minimum quality of data required for accurate predictions?).

1.2 Overview

This methodology section outlines the research design and implementation of the project. The Research Design Section 1.3 focusses on explaining the rationale behind the choices made in the experiment design. The Implementation Section 1.5 gives a brief overview as to how these design decisions have been implemented. A detailed description of the implementation, including a guide explaining how you can run your own experiment using the library, is provided in the package documentation which can be found here.

1.3 Research Design

The research design is structured into the following stages:

1. Data Collection
2. Data Preprocessing
3. Feature Engineering
4. Data Loading
5. Model Training
6. Model Evaluation

1.3.1 Data Collection

The Newcastle Urban Observatory is one of the worlds largest open urban data collection projects. The data collected includes pedestrian counts, air quality, weather, and traffic data. The high velocity data streams are available in near real-time for a wide range of sensor types, including object-detecting wireless sensors - which are the focus of this research project. The depth and breadth of the data collected by the urban observatory make it an ideal case study for investigating for understanding the causes of data quality issues and the effect of issues on predictive modelling. The map in Figure 1 shows the distribution of pedestrian sensors across the city of Newcastle.

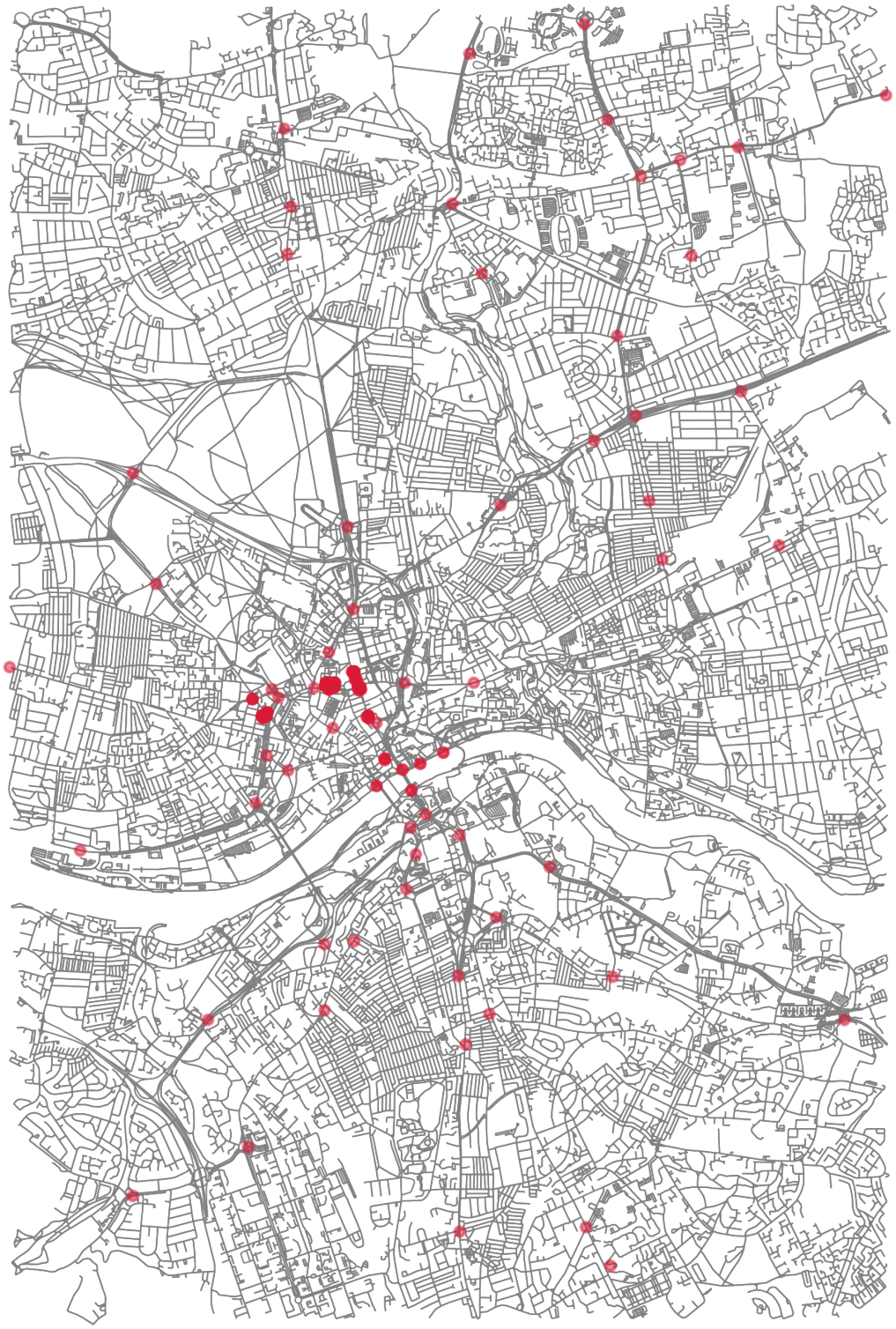


Figure 1: Distribution of sensors in the urban observatory

The data from the pedestrian sensors is collected at 15-minute intervals and each record contains the number of pedestrians detected within a scene over that time period. Figure 2 shows an example of the raw data from a sensor over a period of 18 days in February

2023. The y-axis shows the number of pedestrians recorded by the sensor over a 15-minute period. The x-axis is the timestamp of the recording. The full extent of the data for the same sensor is shown in Figure 3. Both Figure 2 and Figure 3 show that there are data quality issues. Figure 2 shows regular missing data points during the night (when there are no pedestrians) and Figure 3 shows that there are some extended periods of data sparsity.

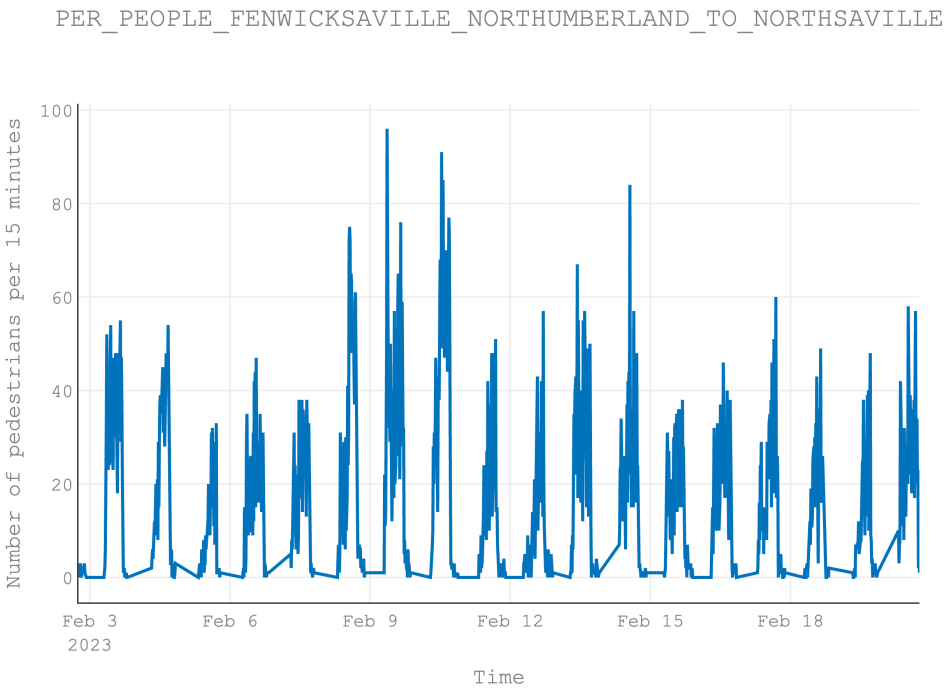


Figure 2: Example of preprocessed data from a sensor

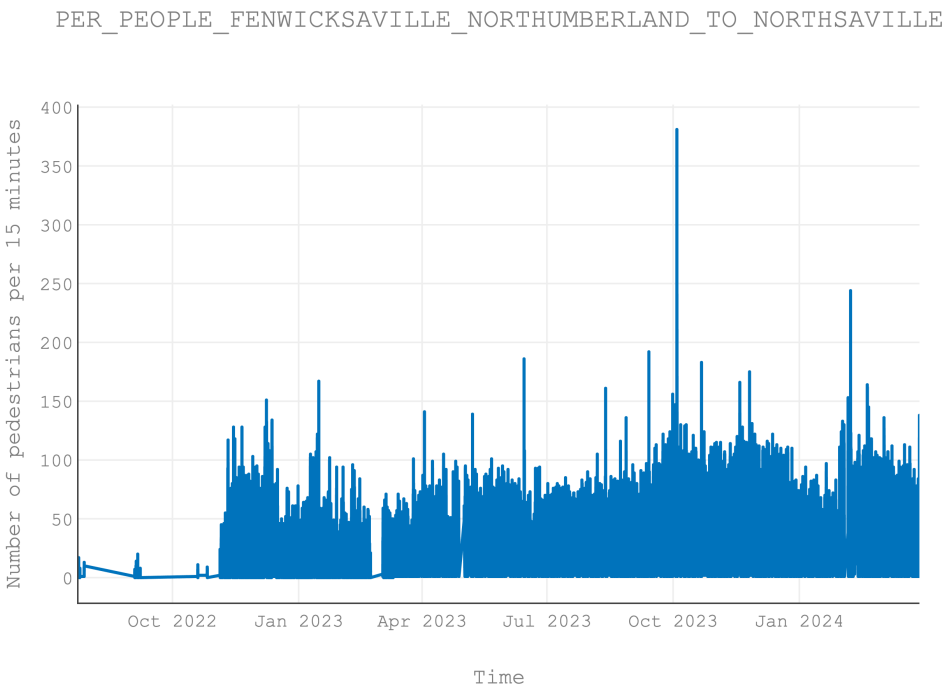


Figure 3: Example of preprocessed data from a sensor - 2-year extent

The data quality issues in the urban observatory data can be summarised as follows:

- Missing data points - these are caused by sensor failures, networks issues, or sensor design issues (some of the sensors do not transmit a reading if there are no pedestrians detected).
- Erroneous data points - these result from a range of issues such as sensor calibration issues, network issues, or environmental factors (e.g. a sensor may be blocked by a parked lorry). Due to the nature of the data (raw sensor footage is unavailable due to data protection guidelines), it is difficult to determine the cause of the erroneous data points, or if they are indeed erroneous. This presents an interesting challenge for the research project.

To more clearly indicate the extent of the missing data points, Figure 4 shows the number of records recorded per day for the same sensor. The plot shows that there are many missing records over the 2-year period. Ideally there should be 96 records per day (4 records per hour for 24 hours). The plot shows that there are approximately zero days where all 96 records are present. It is also evident from Figure 4 that there are a few days where no records are present at all.

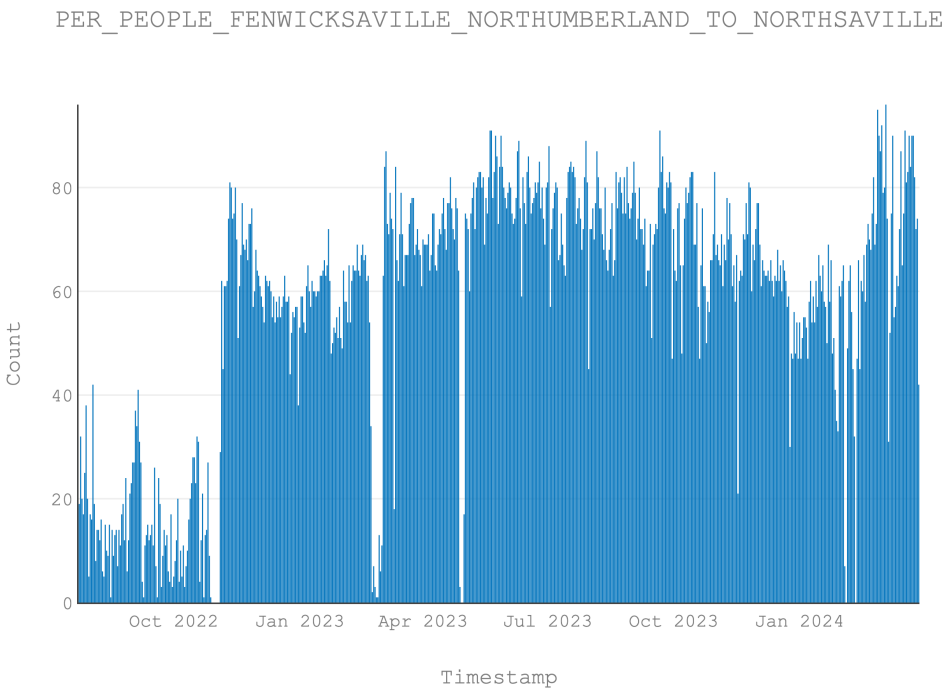


Figure 4: Example of records recorded per day

1.3.2 Data Preprocessing

Data preprocessing is essential to ensure optimal model performance. The neural networks need to be 'aware' of the data gaps to ensure that they are not training on the unrepresentative sequences. ML for time-series models assume that sequential data points are evenly sampled over time, however, imputing the missing data can affect it integrity. For this reason, the data has been left in its raw form, and a consecutive sequence detection and labelling algorithm has been developed (see Section 1.5.4).

Figure 5 shows the outputs of the data preprocessing stage for the same sensor as above. Each new consecutive sequence is shown as a new colour. The sequences in this example a appear to restart with a daily pattern. This means each sequence is less than 96 records long which limits the length of training windows that can be used in the machine learning pipeline (this is explained further in Section 1.4).

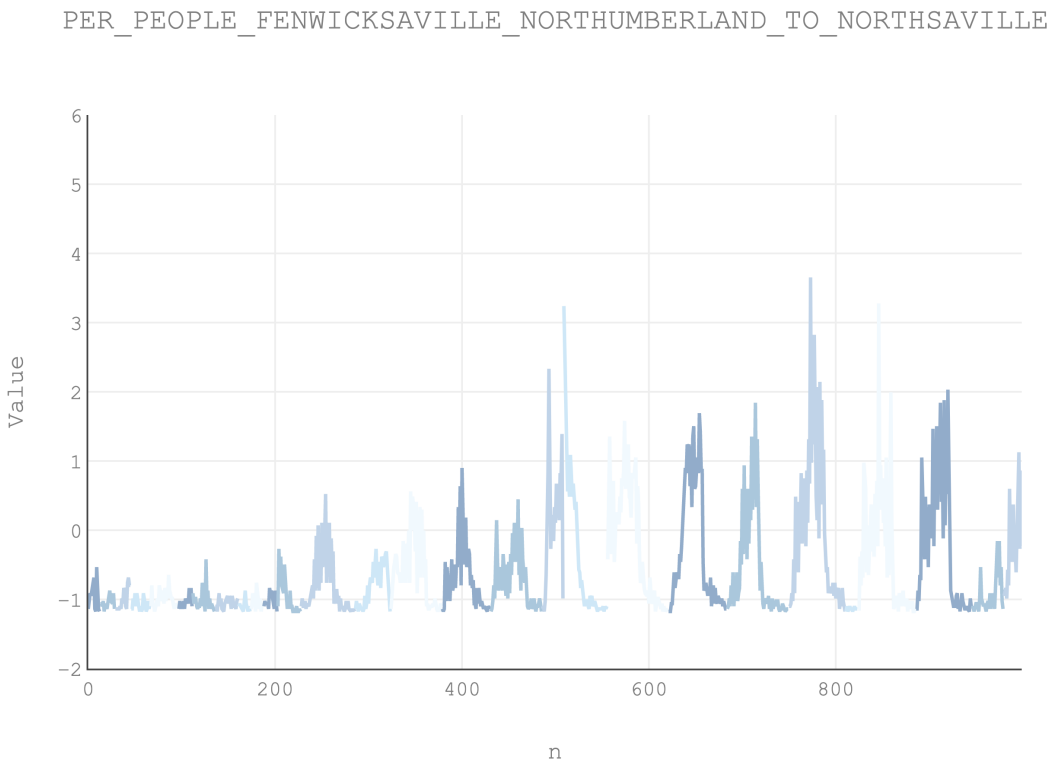


Figure 5: Example of preprocessed data from a sensor

1.3.3 Feature Engineering

Feature engineering is the process of transforming raw data into features that can be used by machine learning algorithms. The features are engineered to capture the underlying patterns in the data. Figure 6 shows an example of engineered frequency features from the same sensor shown previously (frequency features are shown as grey lines and the pedestrian flow data as blue). The uneven looking sinusoids highlight the missing data gaps. The features are calculated using a Lomb-Scargle Periodogram (similar to Fourier Transform but applicable to data with missing records).

The features are then normalised using the Standard Scaler equation shown in Equation 1. The standard scalar normalisation technique is chosen to ensure that any future data points falling outside of the existing range can be scaled appropriately (a min max scaler would not be appropriate in this case). Calculating the mean and standard deviation in the scaling process also serves as a check for future data drift (because movement patterns change over the time the mean is likely to be non-stationary).

Equation 1 Standard Scaler equation

$$X_i' = \frac{X_i - \mu}{\sigma} \tag{1}$$

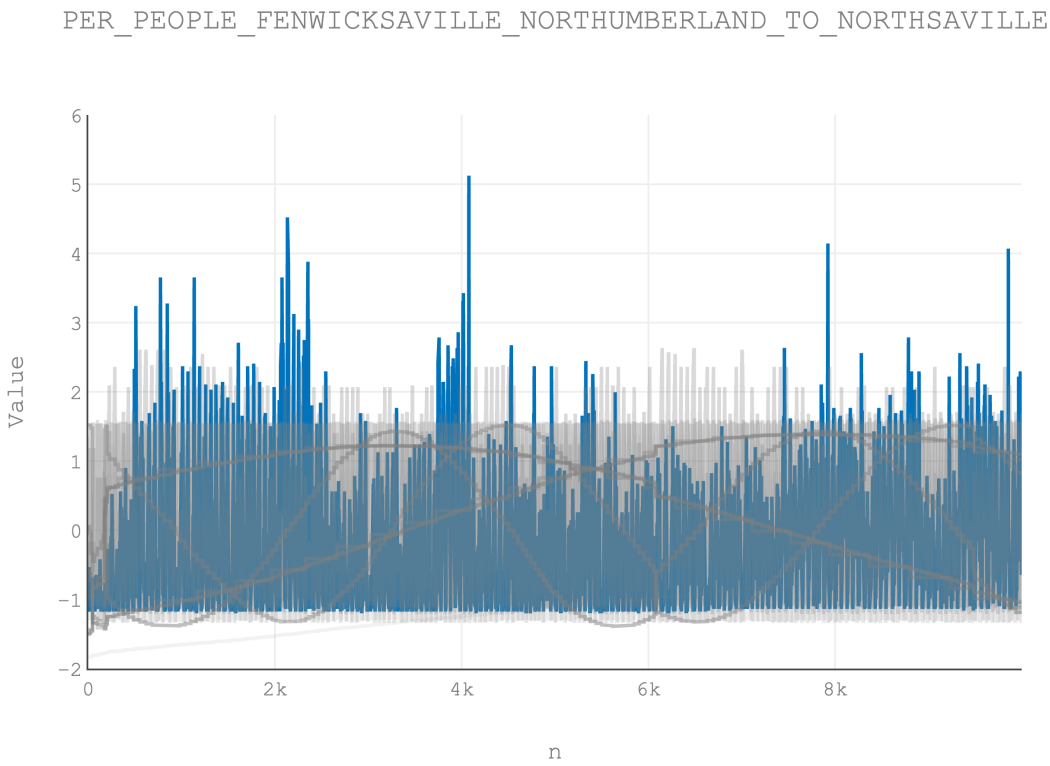


Figure 6: Example of engineered frequency features from a sensor

1.4 Data Loading

For recurrent neural networks (such as LSTMs), the data must be windowed to create standard length sequences of data that can be used for training. An example window is shown in Figure 7. Each window is contains a sequence of data points from a sensor (shown in blue) and a number of additional sequences from engineering features (show in grey). The length of this window is set as a hyper-parameter. The label (shown in red) for each window is the data point a specified number of time steps into the future (the horizon). The window size variable is a hyper-parameter that can be tuned to improve the performance of the model. The horizon hyper-parameter also affects the models performance, but more importantly, the horizon dictates how far into the future the model can predict. In the example plot, the window size is 4 and the horizon is 8. This means that the model is recursively trained to predict the pedestrian flow 8 time steps (2 hours) into the future using the previous 4 time steps (1 hour) of data.

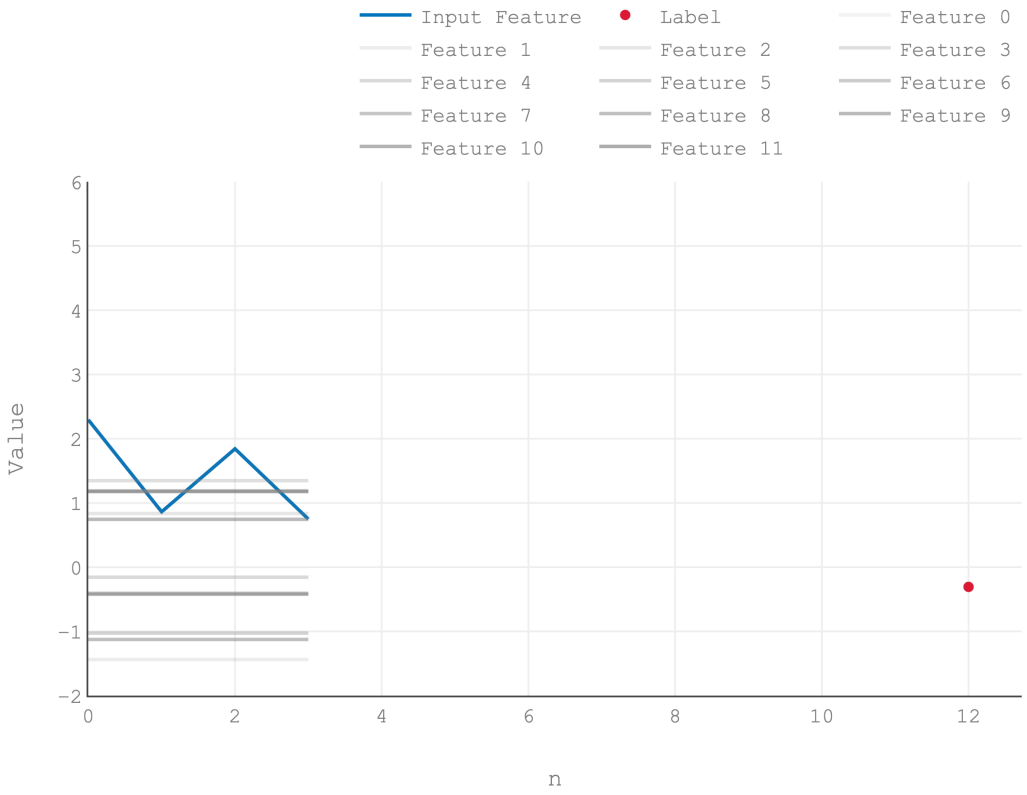


Figure 7: Examples of data quality issues in the urban observatory

1.4.1 Model Training

LSTMs (long-short term memory units) and GRUs (gated recurrent units) are well regarded in scientific literature for their ability to capture long-term dependencies in sequential data and have therefore been chosen for this research. The LSTM (Figure 10) is a more complex model than the GRU (Figure 9) and has more parameters. Both models are types of recurrent neural networks (Figure 8) that have been designed to overcome the vanishing/exploding gradients problem. The basic idea behind RNNs is that the neural network is updated with each new recurrent sequence. A standard RNN consists of layers of neurons that are connected to each other. The output of each neuron is fed back into the network at the next time step. This allows the network to learn the temporal dependencies in the data. The problem with standard RNNs is that they have difficulty learning long-term dependencies. The LSTM and GRU models have been designed to overcome this issue through the addition of gates and memory cells (additional neurons that store information about the sequence). The LSTM has three gates (input, output, and forget) and the GRU has two gates (reset and update).

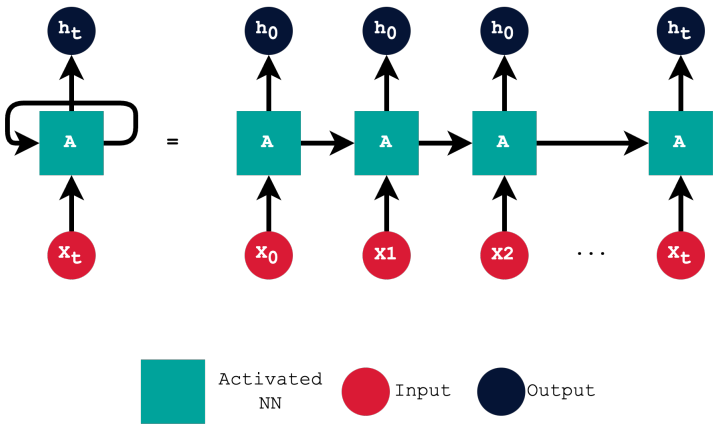


Figure 8: Recurrent neural network (RNN) architecture

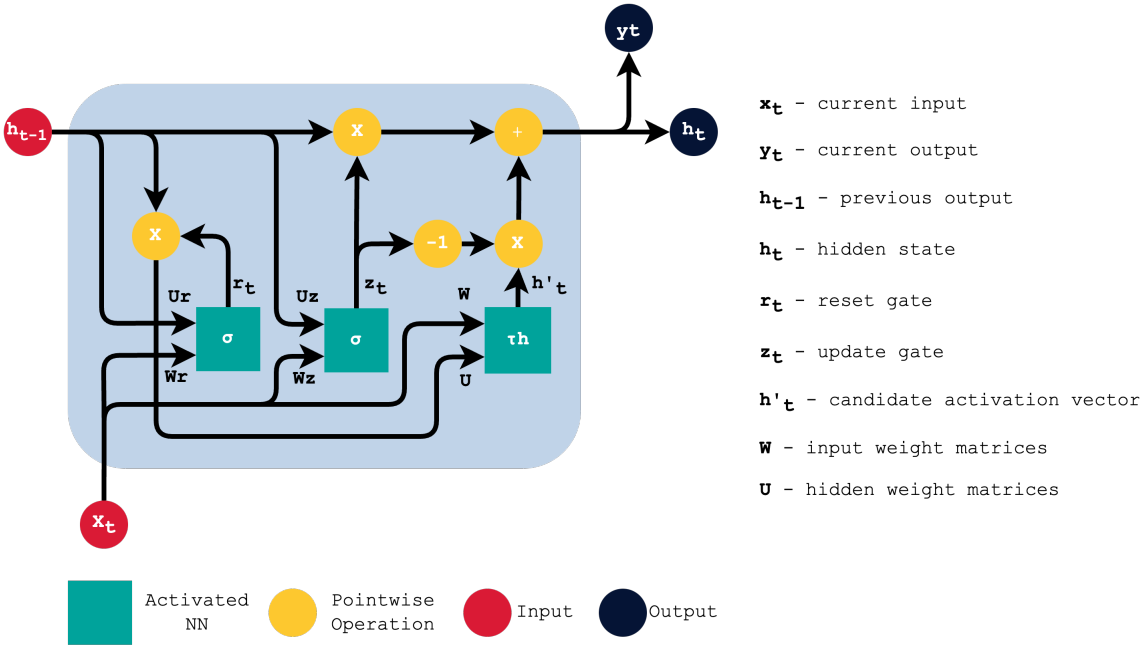


Figure 9: Gated recurrent unit (GRU) architecture

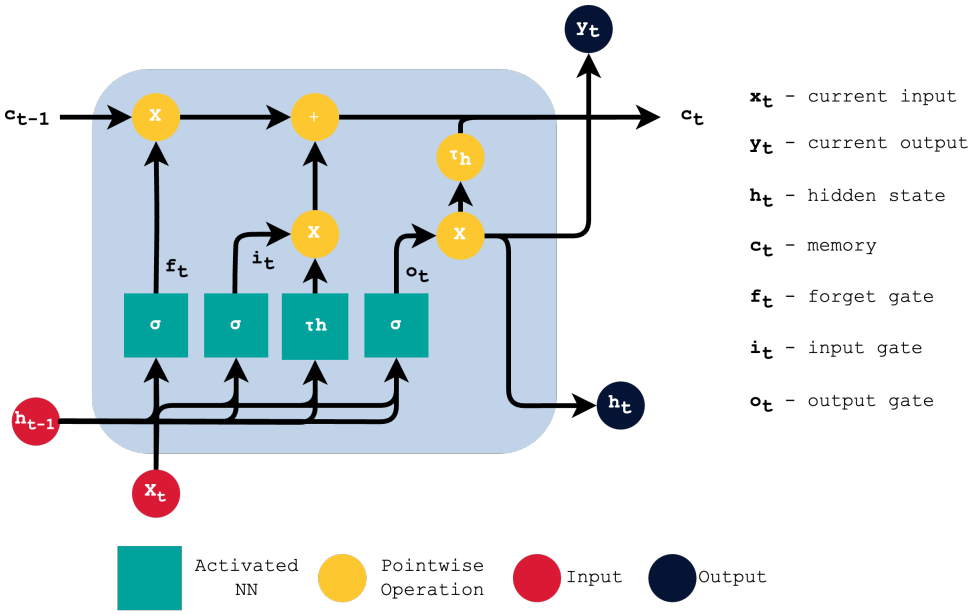


Figure 10: Long short-term memory (LSTM) architecture

A number of hyper-parameters have been used in training the models. An overview and explanation is provided here, with a full list of the implemented hyper-parameters and their values provided in table 1. Some values that could be considered hyper-parameters but will not be tuned in this research are listed in the preset hyper-parameters section below. The hyper-parameters that will be tuned are split into two categories: data hyper-parameters and model hyper-parameters. The data hyper-parameters are those that affect the structure of the data used in the model training process. The model hyper-parameters are those that affect the architecture of the model itself.

Preset hyper-parameters (data):

- **Length of time-series** - the number of time steps available for training the model. This varies depending on the sensor and the data quality issues present in the data. Whilst this could be restricted to a fixed number of data points, the aim of the experiment is to train the model on as much data as possible in order to understand the effect of data quality on the model performance. Therefore a fixed time period is used instead.
- **Optimiser** - whilst experimenting with different optimisers (e.g. Adam, RMSprop, SGD with momentum) can lead to different convergence behaviours, Adam will be used as the preset optimiser for this project to avoid high dimensionality during hyperparameter tuning.
- **Activation function** - LSTMs and GRUs have their own internal activation functions typically using a combination of sigmoid and tanh. As the required outputs of the models are continuous, no activation function is used on the output layer.
- **Loss function** - Mean squared error (MSE) will be used as the preset loss function

for this project. This is a common loss function for regression problems is being used to avoid high dimensionality during hyperparameter tuning.

Tuneable hyper-parameters (data):

- **Window size** - the number of time steps used in each training window. A large window size can cause the model to overfit the data. A small window size can cause the model to under-fit the data.
- **Batch size** - the number of samples used in each iteration of training. A large batch size can cause the model to converge too quickly and miss the global minimum. A small batch size can cause the model to take too long to converge.
- **Horizon** - the number of time steps into the future the model is trained to predict. A large horizon is more difficult to predict and can lead to overfitting. A small horizon is much easier and often leads to under-fitting (the model learns to predict the last data point in the sequence).

Tuneable hyper-parameters (model):

- **Number of hidden units** - this determines the capacity and complexity of the model. More units can capture more complex patterns in the data, but may lead to overfitting.
- **Number of layers** - deeper networks can learn more abstract representations but are harder to train and more prone to overfitting.
- **Epochs** - complete passes through the training data. Too few epochs generally lead to under-fitting, and too many can result in overfitting.
- **Learning rate** - this controls how quickly the model parameters are updated. during training. It's crucial for convergence and avoiding local optima. A high learning rate can cause the model to miss the global minimum. A low learning rate can cause the model to take too long to converge.
- **Dropout rate** - this is a regularisation technique that can help to prevent overfitting, the dropout rate determines how many units are randomly 'dropped' during training.

1.4.2 Model Evaluation

To evaluate the model a range of metrics will be used. The primary metric will be the root mean squared error (RMSE) shown in equation 2. RMSE measures the difference between the predicted and actual values and is more sensitive to outliers than the mean absolute error (MAE). Mean absolute percentage error (MAPE) shown in equation 3 and R-squared (R^2) metric shown in equation 4 will also be used. R^2 measures the proportion of the variance in the dependent variable that is predictable from the independent variable(s).

- Model evaluation - what metrics are used?

Equation 2 Root mean squared error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2}$$

Equation 3 Mean absolute percentage error (MAPE)

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{3}$$

Equation 4 R-squared (R^2)

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4}$$

1.5 Research Implementation

The research will be implemented using Python and various libraries such as Pandas, NumPy, and PyTorch. This system has been developed in a modular fashion to allow for easy integration of new sensors, data sources, models, and data transformation steps. A full description of the implementation methodology is provided in the package documentation which can be found here.

An overview of each stage is provided below:

1.5.1 Machine Learning Pipeline Development

A list of hyper-parameters used in the machine learning pipeline is shown in Table 1. The ranges shown reflect the values that were used in the hyperparameter tuning process.

Parameter	Suggestion Type	Value Range/Options
window_size	suggest_int	2 to 10
horizon	suggest_int	1 to 12
batch_size	suggest_categorical	[32, 64, 128]
lr	suggest_loguniform	1e-5 to 1e-1
epochs	suggest_int	5 to 50
model_type	suggest_categorical	["lstm", "gru"]
hidden_dim	suggest_int	32 to 256
num_layers	suggest_int	1 to 3
dropout	suggest_uniform	0.0 to 0.5

Table 1: Hyper-parameter tuning configuration

To tune the models hyper-parameters, the Optuna library was used in conjunction with MLflow for experiment tracking. The hyper-parameter tuning process is shown in Figure 11. The process involves involves running the pipeline multiple times with different hyper-parameters and recording the results. As the pipeline outputs as many models as there are sensors, the hyper-parameter tuning takes the average performance of all the models (an alternative approach would be to tune each sensor/model separately). R-squared was

used as the metric for hyper-parameter tuning (this needs to switched to RMSE, MAPE, Loss weighted 1.0, 0.2, 0.1 and potentially normalised first).

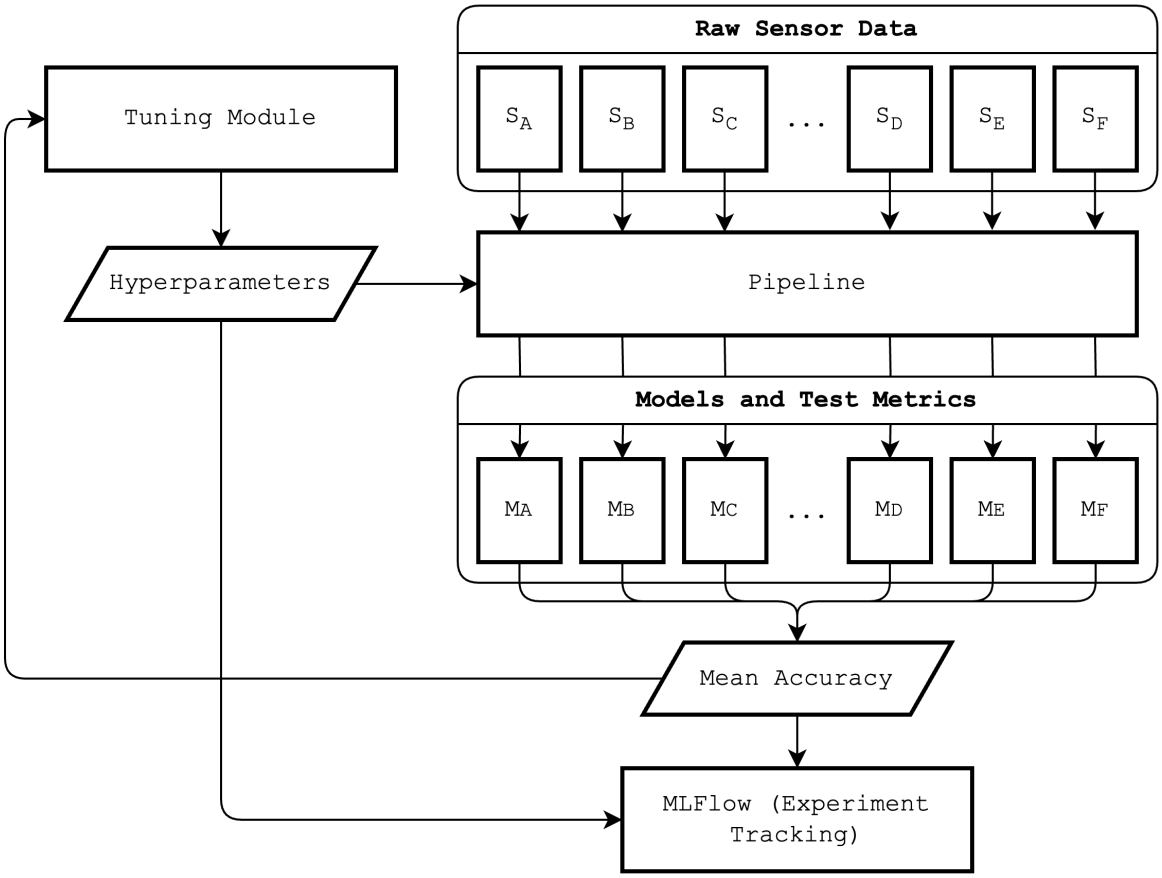


Figure 11: Hyper-parameter tuning process

1.5.2 Real-Time Monitoring System

- System architecture - how is this designed - open source - reproducible - executable?
- Event detection - how is this implemented?
- Dashboard development - what is the purpose and use of the dashboard?

1.5.3 Data Collection

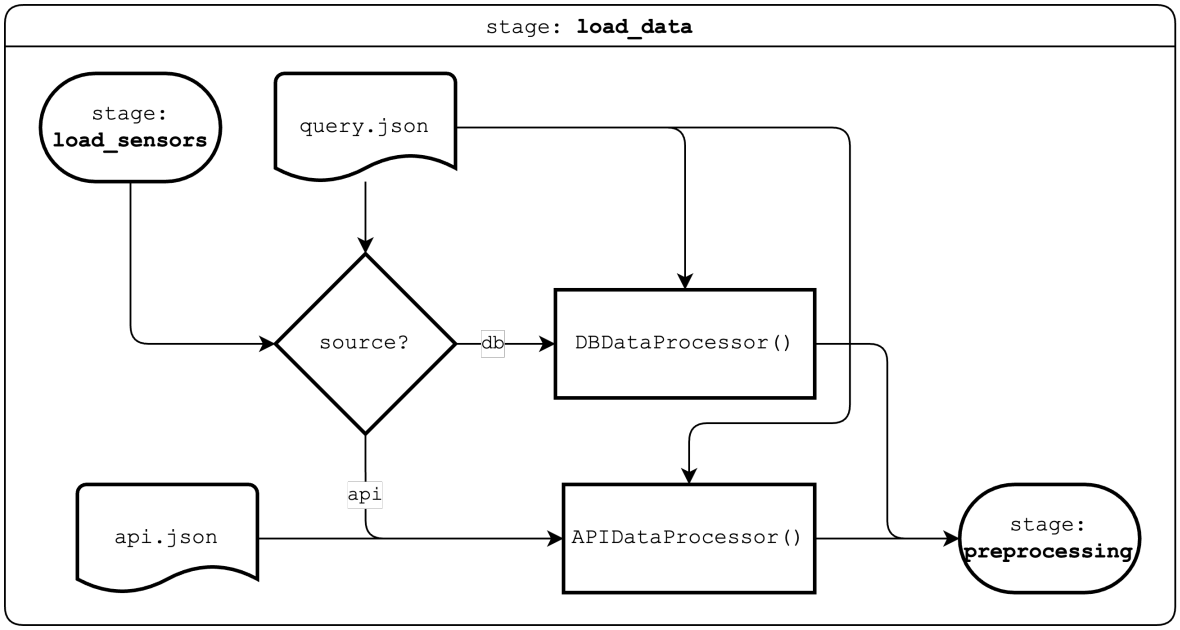


Figure 12: Loading data into the pipeline

1.5.4 Data Preprocessing

(Diagram for Data Preprocessing)

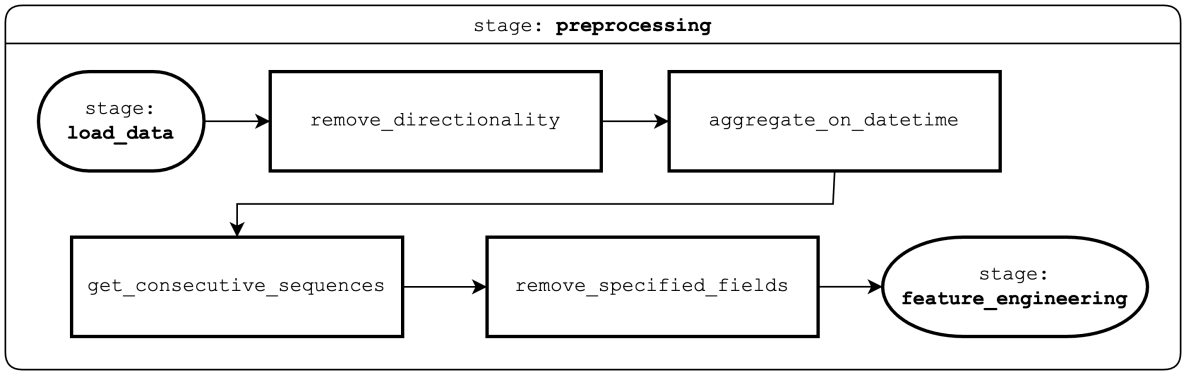


Figure 13: Data preprocessing stage

[H]

1.5.5 Feature Engineering

(Diagram for Feature Engineering)

The algorithm runs in O(N) runtime which is optimal for this task as every element in the list needs to be inspected at least once.

Algorithm 1 Find and process consecutive sequences

Require: DataFrame df with time series data
Ensure: DataFrame with consecutive sequences and assigned sequence numbers

function ProcessConsecutiveSequences(df)
    $sequences \leftarrow \emptyset$
    $current\_sequence \leftarrow \emptyset$
    $\Delta t_{min} \leftarrow$ minimum time delta between timestamps
    $w \leftarrow$ window size
    $h \leftarrow$ horizon
    for each row r in df **do**
        if $current\_sequence = \emptyset$ **or** time difference between r and last row of $current\_sequence = \Delta t_{min}$ **then**
            Add r to $current\_sequence$
        else
            if length of $current\_sequence > w + h$ **then**
                Add $current\_sequence$ to $sequences$
            end if
            $current\_sequence \leftarrow \{r\}$
        end if
    end for
    if length of $current\_sequence > w + h$ **then**
        Add $current\_sequence$ to $sequences$
    end if
    $result \leftarrow \emptyset$
    for $i \leftarrow 1$ to $|sequences|$ **do**
        Assign sequence number i to all rows in $sequences[i]$
        Add $sequences[i]$ to $result$
    end for
    return $result$
end function

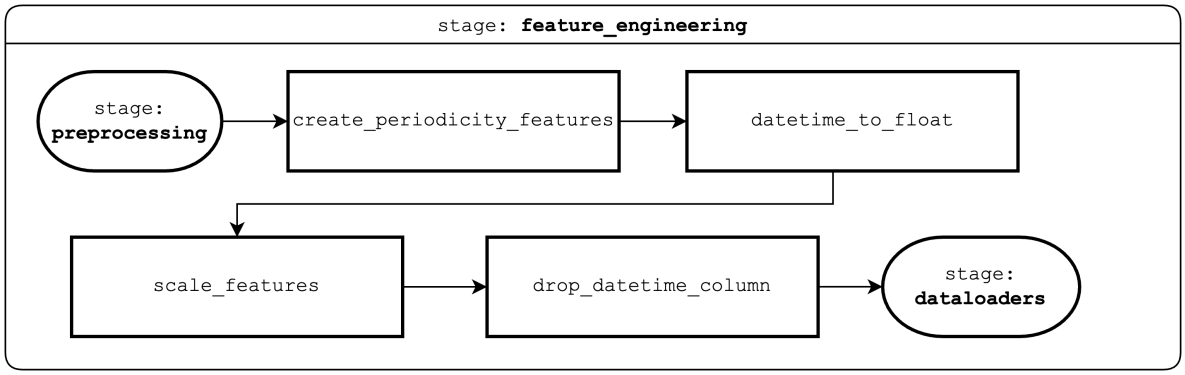


Figure 14: Feature engineering stage

1.5.6 Machine Learning Pipeline Development

(Diagram for Machine Learning Pipeline Development)

1.5.7 Real-Time Monitoring System

(Diagram for Real-Time Monitoring System)

1.5.8 Validation and Testing

(Diagram for Validation and Testing)

1.5.9 MLOPs

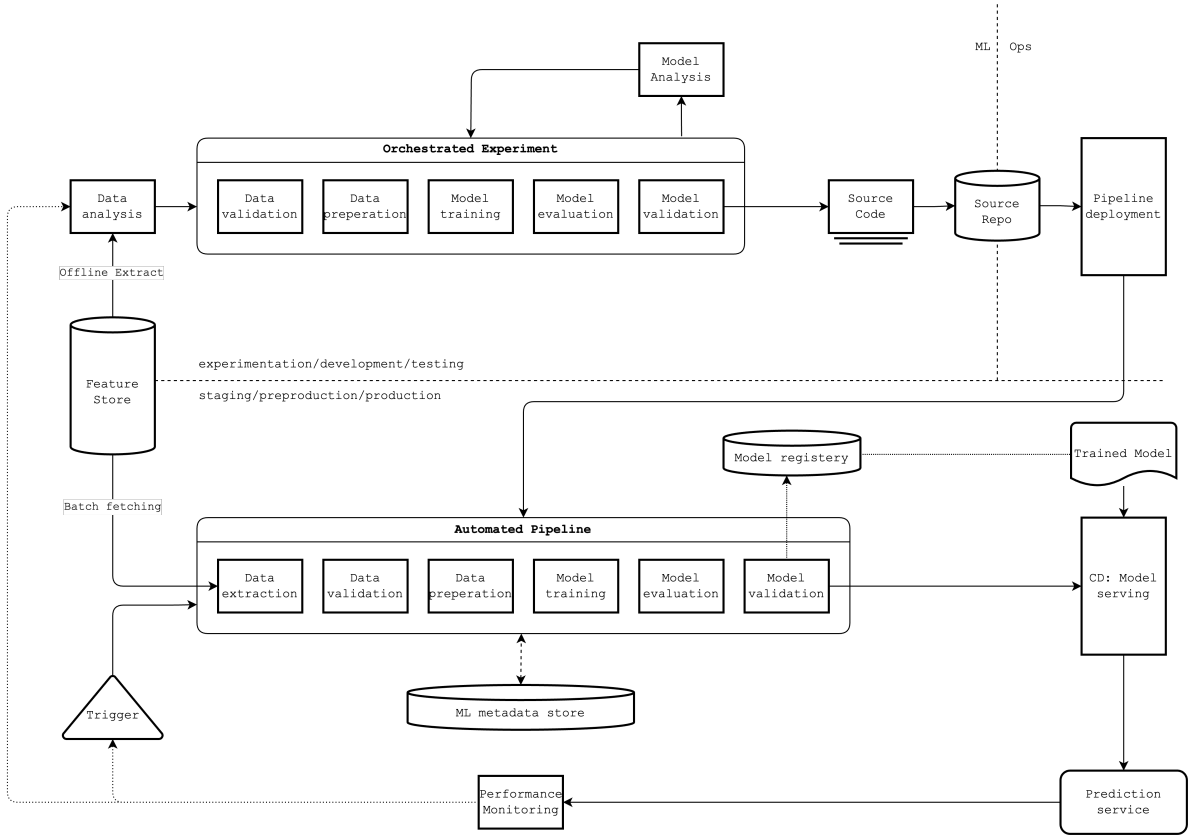


Figure 15: MLOPs pipeline (based on **?**)

1.5.10 Dashboard Development

(Diagram for Dashboard Development)

For the development of the dashboard, a combination of Flask and Dash was chosen. This decision was based on several factors:

1. Flask as the Web Framework: Flask is a lightweight, flexible, and extensible Python web framework. It provides a solid foundation for web applications without imposing a rigid structure, allowing for customisation and scalability.
2. Dash for Interactive Visualisations: Dash, built on top of Flask, Plotly.js, and React.js, offers a powerful toolset for building analytical web applications. It provides a Python interface for creating reactive, web-based visualisations without the need for extensive JavaScript knowledge. This aligns well with the data-centric nature of this project and the preference for Python in the scientific community.
3. Integration Capabilities: The combination of Flask and Dash allows for seamless integration with the existing Python-based machine learning pipeline. This integration ensures that real-time data from the sensors and outputs from the predictive models can be efficiently channeled into the dashboard.

4. Customisability and Extensibility: Both Flask and Dash offer high levels of customisability. This is crucial for a research project where requirements may evolve, and new visualisations or features may need to be added as the project progresses.
5. Performance: For real-time data visualisation, performance is critical. Both Flask and Dash are known for their efficiency in handling data streams and updating visualisations, making them suitable for monitoring live sensor data.

The dashboard incorporates several key features designed to effectively communicate data quality issues:

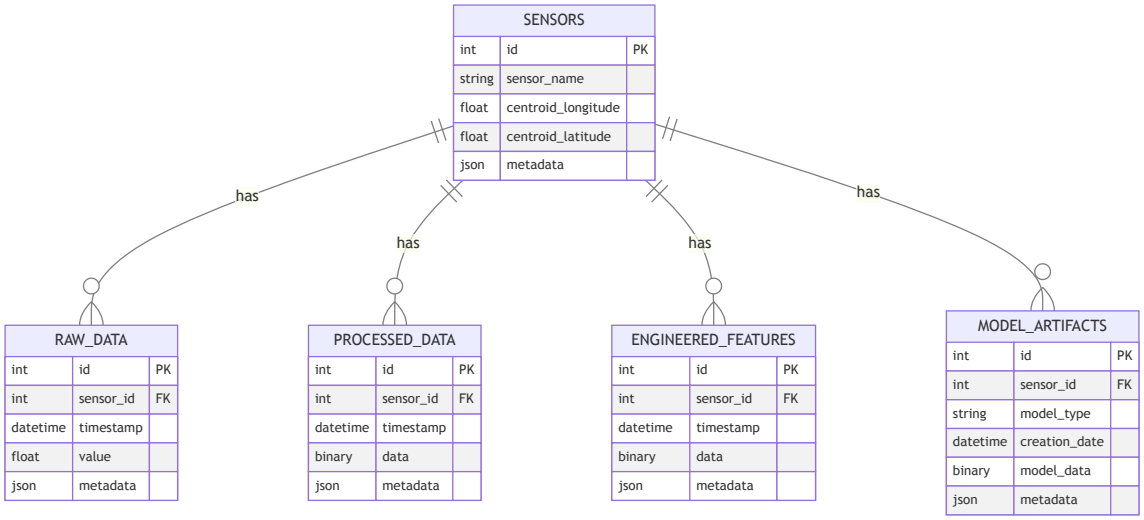
1. Real-time Data Visualisation: Live plots showing incoming sensor data, allowing for immediate detection of anomalies or data quality issues.
2. Quality Dimension Indicators: Visual representations of the various data quality dimensions identified in the literature review, such as completeness, accuracy, and timeliness.
3. Alert System: An alert mechanism to notify users of significant deviations in data quality or potential sensor malfunctions.
4. Historical Data View: The ability to view historical data and trends, enabling users to contextualise current data quality issues.
5. Sensor Network Overview: A map-based interface showing the geographical distribution of sensors and their current status. Predictive Analytics Display: Visualisation of the outputs from the machine learning models, including predictions and confidence intervals.

How does the dashboard link back to the data quality dimensions defined in the literature review?

1.6 Expected Outcomes

What are the three or four key messages that I want to convey?

- An understanding of the data quality issues in urban observatories
- A pathway for developing a machine learning pipeline for automated data quality monitoring in smart cities



2 Results (Plan)

2.1 Assessing Stream Quality

2.2 Predictive Modelling

What quality of data leads to the machine learning models underperforming? How can we monitor whether the data quality is 'good enough' to be incorporated into the next level of the pipeline (spatial component)?

To answer this question we will compare the performance of the models trained on the different sets of sensor data. We will plot a combined performance metric against various data quality metrics to determine the relationship between the two.

How well do the models perform at predicting the next value in the sequence?

1. Show range of different results from the models.
2. Analyse the affect of completeness of the data on the performance of the models.
3. Try to understand the relationship between the different data quality metrics and the performance of the models.
4. How does the performance of the model change as the predictive horizon increases?
5. Does the model learn the more complex patterns better when the horizon is increased?
6. Is this reflected in the hyperparameter tuning (if it is then I would expect the optimal hyper-parameters for number of layers and number of units to increase as the horizon increases)?

2.3 Anomaly Detection

How well do the models perform at detecting anomalies in the data?

1. What are the limitations of the models in detecting anomalies?
2. Is there a trade-off between the number of anomalies detected and the number of false positives?

How well do the models perform at detecting anomalies in the data? What are the limitations of the models in detecting anomalies? Is there a trade-off between the number of anomalies detected and the number of false positives?

To answer this question we will evaluate the performance of the models on the test set. We will plot the number of anomalies detected against the number of false positives to determine the trade-off between the two. This may require some manual labelling of the data to determine the true number of anomalies. Or we could inject synthetic anomalies into the data to determine the performance of the models.

The types of anomalies that we would be looking for are: point anomalies, sequence anomalies and contextual anomalies.

Point anomalies are those which are significantly different from the rest of the data. For example, a sensor reading that is much higher or lower than the rest of the data.

Sequence anomalies are those which are significantly different from the rest of the data in a sequence. For example, a sequence of sensor readings that are much higher or lower than the rest of the data.

Contextual anomalies are those which are significantly different from the rest of the data in a specific context. For example, a sensor reading that is much higher or lower than the rest of the data in a specific location.

2.4 Scalability

How well do the models perform at scale?

1. Timings for training the models on different sizes of data.

Include bibliography

References