Technical Report: Final Project DS 5110
Introduction to Data Management and Processing

Team Members:
Khoury College of Computer Sciences
Data Science Program
gomatumsreenivaasa.j@northeastern.edu
jajoo.j@northeastern.edu
chetlapalli.a@northeastern.edu

December 2024

Contents

1 Introduction

1.1 Overview

With the increasing deployment of IoT devices across industries, vast amounts of real-time sensor data is being generated. To make this data actionable, it is crucial to implement a robust data streaming pipeline that ensures data quality and detects anomalies.

1.2 Objective

To design and implement a real-time data streaming pipeline for IoT sensor data that integrates data quality monitoring and anomaly detection. The pipeline will ensure reliable and accurate data processing, enabling efficient decision-making in IoT applications.

1.3 Goals

Detecting anomalies in real time is crucial for timely intervention. By identifying unusual sensor data patterns as they occur, organizations can respond promptly to potential issues, minimizing risks and ensuring smooth operations.

Building a scalable real-time data pipeline is essential for efficiently processing continuous IoT sensor data. This ensures that large volumes of incoming information are handled seamlessly, supporting the demands of modern IoT ecosystems. Ensuring data quality and integrity is equally important. Monitoring data for completeness, consistency, and accuracy guarantees that insights derived from the data are reliable and actionable.

1.4 Scope

Real-time data ingestion involves developing modules to standardize incoming data formats, ensuring uniformity and compatibility across the system. To maintain data quality, automated scripts or tools can be implemented to monitor incoming data for completeness, consistency, and accuracy.

Validation checks and error-handling mechanisms should be incorporated to promptly address any data quality issues that arise. Additionally, integrating anomaly detection algorithms into the data pipeline enables real-time identification of irregular patterns or potential issues. To support growing data demands, the pipeline must be designed for scalability, capable of handling large volumes of data with low latency to ensure efficient and timely processing.

2 Literature Review

2.1 A Comprehensive Review of Machine Learning Approaches for Anomaly Detection in Smart Homes

(Authors: Md Motiur Rahman, Deepti Gupta, Smriti Bhatt, Shiva Shokouhmand and Miad Faezipour) This review evaluates machine learning algorithms for anomaly detection in smart homes, particularly in multi-resident settings. The study compares models such as Decision Trees, Naive Bayes, Gradient Boosting, and RNN-based approaches (LSTM, GRU). It tests these classifiers on the ARAS smart home dataset, generating 50,000 abnormal scenarios to assess model performance. The paper also discusses the impact of train-test split ratios and k-fold cross-validation on performance, providing insights into computational complexity and implementation challenges.

2.2 LSTM-Autoencoder-Based Anomaly Detection for Indoor Air Quality Time-Series Data

(Authors: Yuanyuan Wei, Julian Jang-Jaccard, Wen Xu, Fariza Sabrina, Seyit Camtepe, Mikael Boulic) This study tackles indoor air quality (IAQ) monitoring, focusing on CO2 anomaly detection in schools. Traditional methods like ARIMA and KNN face limitations, while AI models require multiple correlated data points. The proposed hybrid deep learning model combines Long-Short-Term Memory (LSTM) and Auto-Encoder (AE) for improved anomaly detection in time-series data. By extracting features, reconstructing data, and calculating reconstruction error, the system detects anomalies effectively. When tested in real-world CO2 datasets, the model achieved high precision (99. 50%) and precision (100%).

2.3 HomeGuardian: Detecting Anomaly Events in Smart Home Systems

(Authors: Dai Xuan, Mao Jian, Li Jiawei, Lin Qixiao, Liu Jianwei) This paper introduces HomeGuardian, a context-based approach for anomaly detection in smart home environments. The system leverages temporal and environmental contexts extracted from system logs to identify abnormal events. By integrating these hybrid contexts into a learning-based classifier, HomeGuardian enhances security by detecting unusual patterns indicative of security threats. The approach was tested on a dedicated testbed, showcasing its effectiveness in anomaly detection for smart homes.

3 Methodology

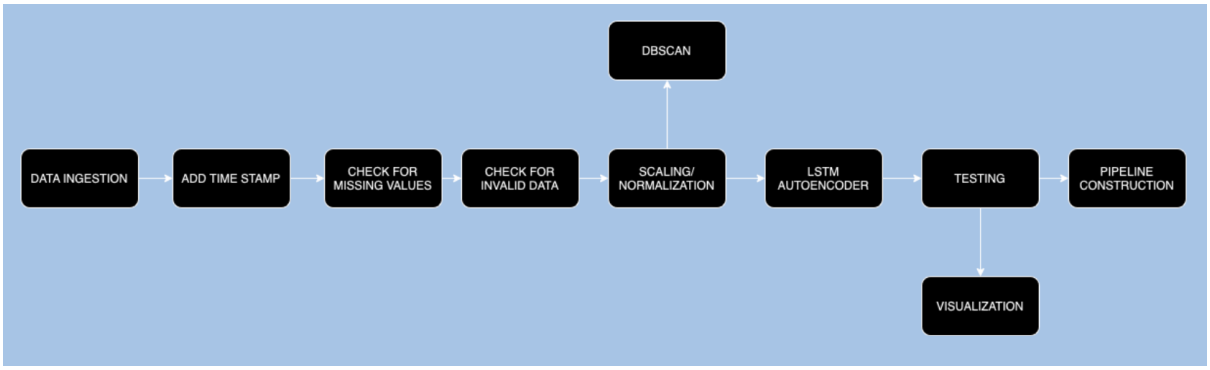

Figure 1: Project Workflow

3.1 Data Collection

The dataset used for the project is the ARAS Dataset (Activities of Daily Living Recognition Using Ambient Sensors) dataset, which includes data captured from two smart home environments equipped with various ambient sensors. The dataset focuses on activity recognition and behavioral analysis for two residents performing daily activities. Key details of data collection include:

- **Environment:** Two real apartments outfitted with sensors.
- **Duration:** Data recorded over 30 days.
- **Sensors:** Passive Infrared (PIR) motion sensors, door contact sensors, and appliance power sensors.
- **Annotations:** Labelled activities with precise start and end times.
- **Metadata:** Details about sensor deployment, including location in the apartments.

The data was sourced as a publicly available dataset, ensuring it met the project's requirements for anomaly detection and activity recognition. The extended duration and variety of activities provided a robust foundation for analysis.

3.2 Data Preprocessing

Preprocessing steps ensured the data was clean and ready for analysis

Cleaning
To ensure data consistency and reliability, several cleaning steps were performed. Corrupted or missing entries were removed to prevent inaccuracies in the analysis. Additionally, binary sensor values were standardized to 0 or 1, ensuring uniformity across the dataset.

Feature Engineering
Relevant features were extracted to enhance the model's ability to detect anomalies. These features included temporal attributes such as the hour of the day, the day of the week, and the week itself. Furthermore, sensor states were combined with these temporal features to identify patterns that contribute to anomalies, improving the model's contextual understanding.
Reshaping Data
To capture temporal dependencies in the data, sequences of data points were created. Sequence lengths were optimized during experimentation, with lengths such as 1, 300, 600, and 4000 being tested to determine the most effective configuration for anomaly detection.

Normalization
Sensor readings were normalized to bring all features to a comparable scale. This step was essential for facilitating efficient model training and ensuring that no single feature dominated the learning process due to differences in scale.

3.3 Analysis Techniques

The project employed the following methods and models for analysis:

LSTM Autoencoder

Anomaly detection was performed by utilizing a model that learns to reconstruct input sequences and calculates reconstruction errors. To ensure robust detection, threshold values were established based on the 95th percentile of reconstruction errors, allowing the system to effectively identify deviations indicative of anomalies.

Model parameters:

- Hidden state size: 256
- Optimal sequence length: 1
- Batch size: (to maximize GPU utilization).

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was used as a comparative model for anomaly detection.
Limitations:

- High computational cost (requiring over 50GB of RAM).
- Limited ability to handle temporal dependencies, resulting in fewer detected anomalies.

Threshold Analysis

Statistical methods were employed to calculate the mean squared errors between the original and reconstructed data, providing a measure of reconstruction accuracy. The thresholds for anomaly detection varied depending on sequence length and batch size, with lower thresholds reflecting better model performance and higher reconstruction fidelity.

Visualization

Reconstruction errors and thresholds were plotted to evaluate the model's effectiveness in detecting anomalies. This visualization helped assess how well the model distinguishes normal data from anomalous patterns. Additionally, anomalies were analyzed and visualized during specific activities, such as going out, sleeping, and studying, providing insights into the frequency of anomalies across different contexts.

Pipeline creation:

The pipeline begins with incoming sensor data, which undergoes a data quality check to ensure consis-

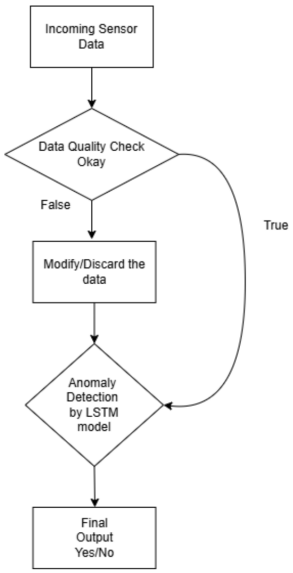

Figure 2: Pipeline Workflow

tency and accuracy. If the data passes the quality check , it proceeds directly to anomaly detection using an LSTM model. If the data fails the quality check , it is modified to maintain integrity before moving forward. The LSTM model analyzes the processed data for anomalies, and the final output indicates whether an anomaly is detected. This process ensures that only high-quality data is used for reliable anomaly detection.

4 Results

4.1 Reconstruction Errors

The LSTM autoencoder achieved excellent reconstruction performance, with thresholds set at the 95th percentile. Results for House A and House B demonstrate:

Low Reconstruction Errors

The model effectively reconstructed data with minimal errors. **Suitable Thresholds**

Most data points fell below the anomaly threshold, validating the model's performance.

Parameter	Threshold Value	Batch Size (BS)	House A
Sequence Length = 1	0.00016	2^{17}	Best performance
Sequence Length = 4000	17.743275	256	Poor performance
Sequence Length = 600	558.404052	1024	High errors, overfitting issues

Table 1: Reconstruction Errors for Different Sequence Lengths and Batch Sizes

4.2 Effects of Changes in Parameters

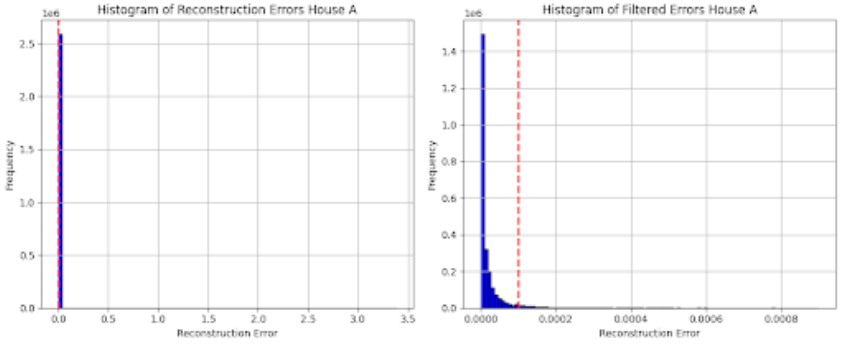

Figure 3: Seq_len = 1 and BS = 2^17

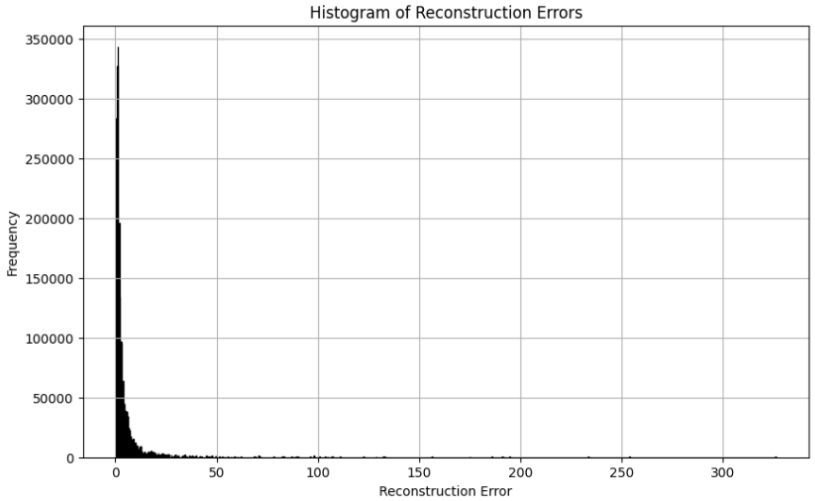

Figure 4: Seq_len = 600 and BS=1024

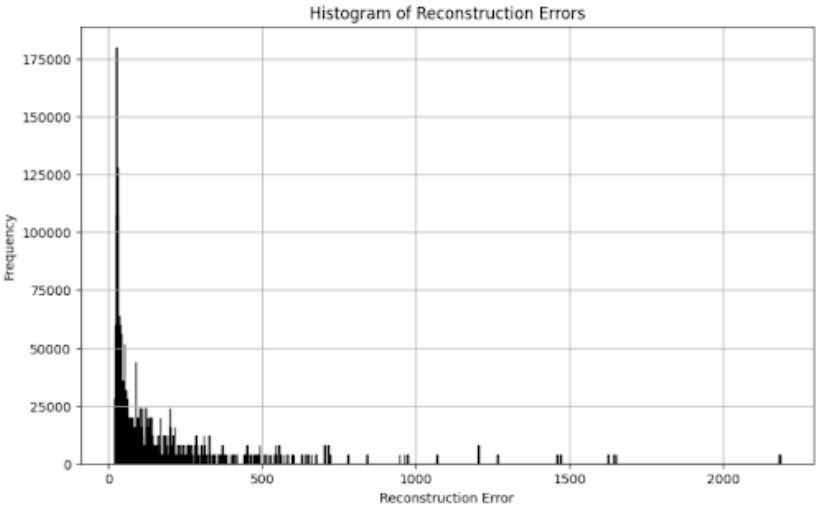

Figure 5: Seq_len = 4000 and BS = 256

4.3 Anomaly Detection

Maximum anomalies were detected during activities such as:
- Going out
- Sleeping
- Watching TV
- Studying
- Using the internet

See Figure 7 for more details.
The LSTM autoencoder detected significantly more anomalies compared to DBSCAN due to its ability to capture temporal dependencies across the dataset.

4.4 Comparative Model Performance

DBSCAN: Approximately 1,000 anomalies were identified by DBSCAN within a dataset containing 2.6 million rows, necessitating a day-by-day processing approach to effectively analyze the data. **LSTM Autoencoder:** The entire dataset was processed efficiently by the LSTM Autoencoder, enabling the identification of anomalies with improved accuracy and reduced computational cost.

5 Discussion

The results highlight the effectiveness of the LSTM autoencoder for anomaly detection in time-series data:

Threshold Determination:
Reconstruction error thresholds demonstrated high sensitivity to sequence length and batch size. Shorter sequences combined with larger batch sizes proved effective in minimizing overfitting and reducing noise, resulting in lower thresholds and improved accuracy in anomaly detection.

Model Selection:
DBSCAN proved useful for density-based clustering but was computationally expensive and less effective at capturing temporal dependencies in large datasets. In contrast, the LSTM autoencoder emerged as the superior model, offering a balance between computational efficiency and detection accuracy.

Implications for Smart Home Monitoring:
Effective anomaly detection plays a crucial role in enhancing home automation and elder care systems. By identifying patterns during specific activities, such as sleeping or watching TV, it provides actionable insights that support activity recognition and behavioral analysis, ultimately contributing to improved care and personalized automation.

Comparison with Literature:
The findings are consistent with prior research highlighting the significance of temporal context in activity recognition. Additionally, the use of LSTM autoencoders for anomaly detection has been validated in previous studies, showcasing their robustness and effectiveness in smart home applications.

6 Conclusion

In conclusion, the project's methodological choices and analytical techniques effectively addressed the challenges of anomaly detection in smart home environments. By utilizing hybrid LSTM-Autoencoder models, the project successfully identified subtle deviations in sensor data, ensuring reliable anomaly detection even in the presence of noise and missing data. The importance of data preprocessing, feature extraction, and model optimization was highlighted, with temporal and environmental context providing robustness in distinguishing normal activity from anomalies. This approach lays a strong foundation for future enhancements, such as incorporating additional sensors, refining model architectures, and adapting the system to more complex real-world environments. The project holds significant potential for practical applications in home automation, elderly care, and security, where real-time anomaly detection is crucial for safety and efficiency, offering a framework for smarter, more responsive living spaces.

7 References

[1] Motiur R., Deepti G., Smriti B., Shiva S. & Miad F. (2024) *A Comprehensive Review of Machine Learning Approaches for Anomaly Detection in Smart Homes: Experimental Analysis and Future Directions.* Wiley Online Library. https://www.researchgate.net/publication/379977751_A_Comprehensive_Review_of_Machine_Learning_Approaches_for_Anomaly_Detection_in_Smart_Homes_Experimental_Analysis_and_Future_Directions

[2] Wei, Y., Jang-Jaccard, J., Xu, W., Sabrina, F., Camtepe, S., & Boulic, M. (2023). *LSTM-Autoencoder-Based Anomaly Detection for Indoor Air Quality Time-Series Data.* IEEE Sensors. https://ieee-sensorsalert.org/articles/lstm-autoencoder-based-anomaly-detection-for-indoor-air-quality-time-series-data/

[3] Xuan, D., Jian, M., Jiawei, L., Qixiao, L., & Jianwei, L. (2022). *HomeGuardian: Detecting Anomaly Events in Smart Home Systems.* ResearchGate. https://www.researchgate.net/publication/361290448_HomeGuardian_Detecting_Anomaly_Events_in_Smart_Home_Systems

[4] Towards Data Science. (n.d.). *A Complete Guide to Anomaly Detection in Machine Learning.* Towards Data Science. https://towardsdatascience.com/anomaly-detection-techniques-in-machine-learning-with-python-d3e7b5bb9e18

[5] Analytics Vidhya. (n.d.). *Anomaly Detection in Data Science: A Comprehensive Overview.* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/anomaly-detection-in-data-science/

[6] Zhang, X., Wang, J., & Li, Y. (2021). *Survey on Deep Learning-based Anomaly Detection Techniques in Smart Home Systems.* IEEE Access. https://ieeexplore.ieee.org/document/9423452

[7] Machine Learning Mastery. (2020). *Anomaly Detection with Autoencoders in Keras.* Machine Learning Mastery. https://machinelearningmastery.com/autoencoders-for-anomaly-detection/

[8] Dataversity. (n.d.). *Top 5 Techniques for Anomaly Detection.* Dataversity. https://www.dataversity.net/top-5-techniques-for-anomaly-detection/

8 Appendix A: Code

8.1 LSTM Encoder

```
import torch
import torch.nn as nn

class LSTM_Autoencoder(nn.Module):
    def __init__(self, input_size, hidden_size, seq_len):
        super(LSTM_Autoencoder, self).__init__()
        self.encoder = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.decoder = nn.LSTM(hidden_size, hidden_size, batch_first=True)
        self.output_layer = nn.Linear(hidden_size, input_size)
        self.hidden_size = hidden_size
        self.seq_len = seq_len
```

```
    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_size).to(x.device)
        c0 = torch.zeros(1, x.size(0), self.hidden_size).to(x.device)

        encoded, (hn, cn) = self.encoder(x, (h0, c0))

        decoded, _ = self.decoder(encoded, (hn, cn))

        decoded = self.output_layer(decoded)

        return decoded
```

8.2 DBSCAN

```
for txt in sorted_txt_files:
    data = pd.read_csv(f"{base_path}/{txt}",sep=' ', header=None)
    sensor_data = data.iloc[:, :-2].values

    scaler = StandardScaler()
    sensor_data_scaled = scaler.fit_transform(sensor_data)

    pca = PCA(n_components=5)
    sensor_data_pca = pca.fit_transform(sensor_data_scaled)

    dbscan = DBSCAN(eps=0.4, min_samples=4)
    dbscan.fit(sensor_data_pca)
    labels = dbscan.labels_
    data['DBSCAN_Labels'] = labels
    final_df = pd.concat([final_df,data])
    print(f"{txt} done.",f" {final_df.shape}")
final_df


def pipeline(data_recs, data_columns,):
    for idx, data_rec in enumerate(data_recs):
        print(f"Processing record {idx + 1}...")

        data_rec = pd.DataFrame([data_rec], columns=data_columns)

        missing_values = data_rec.isnull().sum()
        if len(missing_values[missing_values > 0]):
            print(missing_values[missing_values > 0])
            data_rec.fillna(0, inplace=True)
        else:
            print('There are no missing values in the record')

        for i in range(1, 21):
            if data_rec.iloc[0, i] not in [0, 1]:
                print(f"Invalid value in column {i}. Assigned value: 0")
                data_rec.iloc[0, i] = 0

        for i in range(21, 23):
            if (data_rec.iloc[0, i] < 1) or (data_rec.iloc[0, i] > 27):
                print(f"Invalid value in column {i}. Assigned value: 1")
                data_rec.iloc[0, i] = 1

        print('Checked for invalid values and imputed the same')
        print('Using the LSTM Encoder, Standard scaler and Reconstruction Errors to check if the rec
```

```
if 'Unnamed: 0' in data_rec.columns:
    data_rec = data_rec.drop(['Unnamed: 0'], axis=1)

input_size = 25
hidden_size = 256
seq_len = 1

model_path = "./House A files/lstm_autoencoder_ha.pth"
scaler_path = "./House A files/standardscaler_ha.joblib"
reconstruction_error_path = "./House A files/reconstruction_errors_ha.npy"

device = torch.device('cpu')
model = LSTM_Autoencoder(input_size=input_size, hidden_size=hidden_size, seq_len=seq_len)

state_dict = torch.load(model_path, map_location=device)
if any(key.startswith("module.") for key in state_dict.keys()):
    state_dict = {key[len("module."):]: value for key, value in state_dict.items()}

try:
    model.load_state_dict(state_dict)
except RuntimeError as e:
    print(f"Error loading model: {e}")
    print("Ensure the model architecture matches the saved state_dict structure.")

model.eval()

reconstruction_errors = np.load(reconstruction_error_path)
threshold = np.percentile(reconstruction_errors, 95)

sc = load(scaler_path)
new_data = sc.transform(data_rec)
new_data_tensor = torch.tensor(new_data, dtype=torch.float32).unsqueeze(0).to(device)

with torch.no_grad():
    new_data_reconstructed = model(new_data_tensor)
    reconstruction_error = torch.mean((new_data_tensor - new_data_reconstructed) ** 2).item(

print("Error: ",reconstruction_error,"Threshold: ",threshold)
is_anomaly = reconstruction_error > threshold
if is_anomaly:
    print(f"Record {idx + 1}: Current Record is an anomaly\n")
else:
    print(f"Record {idx + 1}: Current Record is not an anomaly\n")
```

9 Appendix B: Additional Figures

9.1 Day Of The Week Analysis

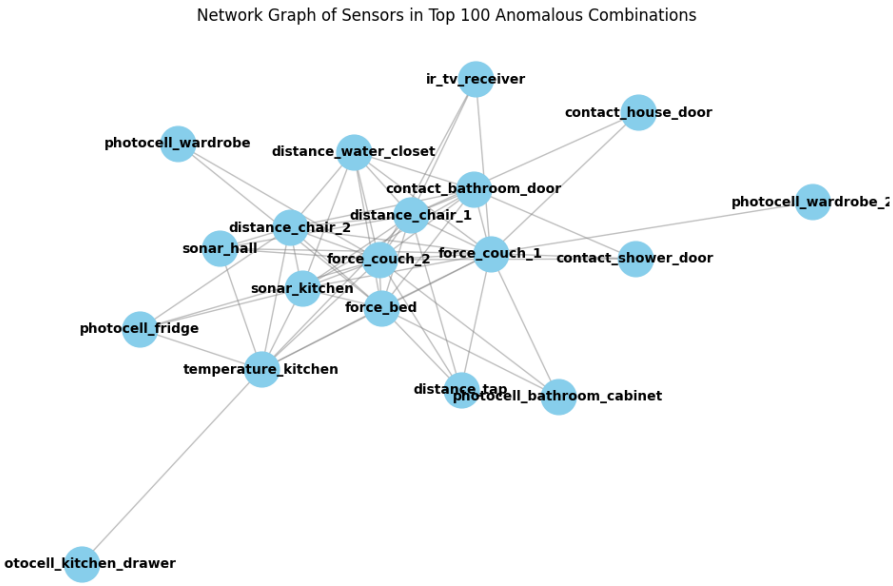

Figure 6: Total Anomalies By Day Of The Week

9.2 Correlation of Sensors with Anomalies

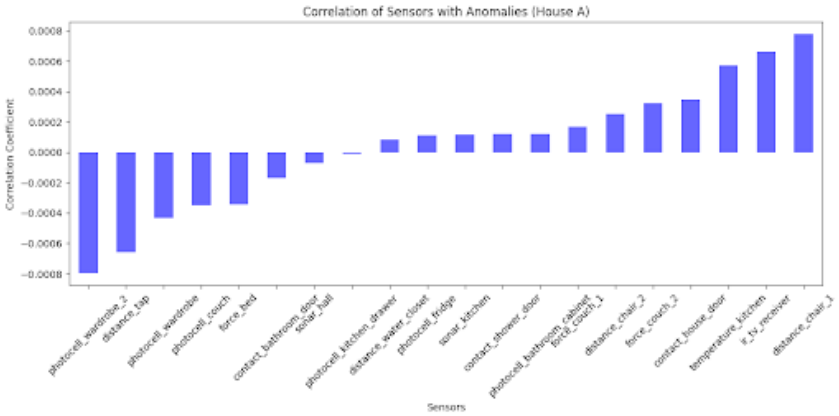

Figure 7: House A

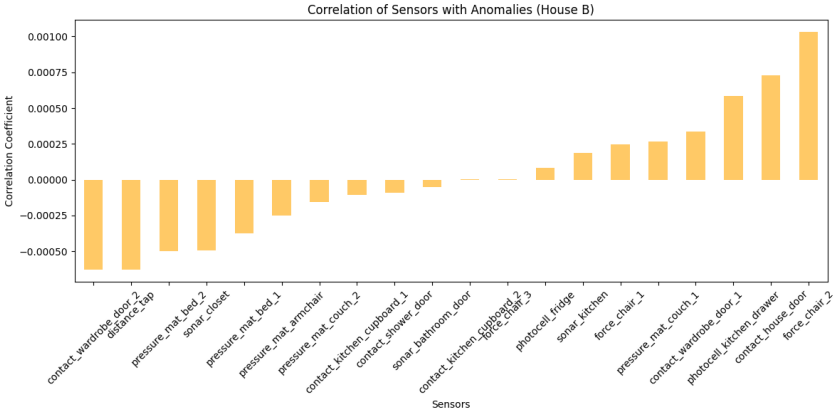

Figure 8: House B

9.3 Anomaly Count Corresponding to Resident Activities

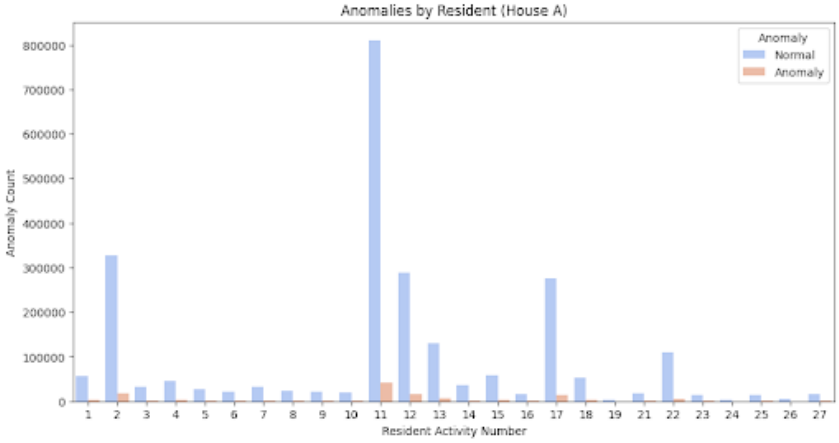

Figure 9: Anomaly Count Corresponding to Resident Activities

9.4 Network Graph of Sensors in Top 100 Anomalous Combinations

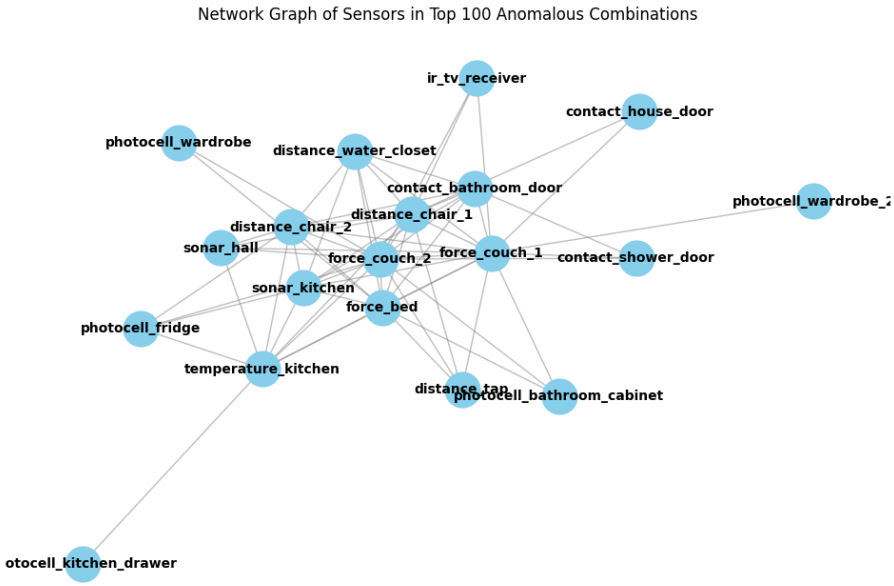

Figure 10: Network Graph of Sensors in Top 100 Anomalous Combinations