IoT Sensor Data Analysis and Cybersecurity Risk Assessment

1. Introduction

The rapid growth of **Internet of Things (IoT) networks** has enabled real-time monitoring of environmental conditions, industrial operations, and home automation. However, these systems remain vulnerable to cyber threats, particularly **Distributed Denial-of-Service (DDoS) attacks**, which can compromise data integrity and system availability.

This project focuses on: **Environmental Analysis** → Understanding unique sensor characteristics
Sensor Correlations → Identifying relationships between sensor readings **Seasonality Analysis** → Detecting trends and variations in environmental conditions
Daily Trends → Visualizing day-to-day sensor fluctuations
Cybersecurity Risk Assessment → Evaluating vulnerabilities to DDoS and other threats
Mitigation Strategies → Enhancing IoT security with **anomaly detection, IDS, and network hardening**

By integrating **data-driven insights** and **cyber risk assessment**, this study aims to optimize **IoT sensor deployments, improve security measures, and enhance data reliability**.

---

2. IoT Sensor Data Overview

The study analyzes data from **three IoT devices** deployed in different environments, each equipped with **seven sensors**:

Sensor Type	Description
Temperature	Measures ambient temperature in °C
Humidity	Monitors moisture levels in the air
CO (Carbon Monoxide)	Detects CO levels to assess air quality
LPG (Liquefied Petroleum Gas)	Monitors potential gas leaks
Smoke	Detects smoke particles indicating fire risks
Light	Measures light intensity
Motion	Detects movement in the monitored area

Each device provides **real-time readings**, allowing for in-depth environmental monitoring and security assessment.

---

3. Time Series Analysis and Anomaly Detection

3.1 Temperature Prediction and Anomaly Detection

This section performs **time series forecasting** and **anomaly detection** on IoT telemetry data:

- **Data Loading and Preprocessing:** Loads IoT telemetry data, sorts it by timestamp, and scales the relevant features (humidity, CO, LPG, smoke, and temperature).
- **Dataset Creation:** Uses a custom PyTorch dataset (`TimeSeriesDataset`) to prepare sequences of input features (excluding temperature) to predict the next temperature value.
- **Model Definition:** Implements a transformer-based model (`TransformerTimeSeries`) for temperature prediction.
- **Training and Prediction:** Trains the model using prepared data and predicts temperature values on the test set.
- **Inverse Transformation:** Converts predictions back to their original scale.
- **Visualization and Anomaly Detection:** Plots predictions vs. actual values and detects anomalies based on deviations beyond a threshold.

---

4. Correlation Analysis

4.1 Discovering Sensor Relationships

By analyzing relationships between sensor data, we can uncover dependencies:

- **Temperature & Humidity** → Warmer temperatures often lead to higher humidity levels.
- **CO & LPG** → Industrial areas may show simultaneous increases in **CO and LPG concentrations**.
- **Light & Motion** → Higher light intensity correlates with **increased motion detection** in occupied spaces.

Methods Used:
Pearson Correlation Coefficients → Measure linear relationships
Heatmaps → Visualize sensor interdependencies

---

5. Network Intrusion Detection System (NIDS) Analysis

This section evaluates network security threats by analyzing **network attack data**:

- **Data Loading (in chunks):** Reads large network traffic data in chunks to optimize memory usage.
- **Box Plots:** Visualizes distributions of numerical features to detect outliers.
- **Data Filtering:** Selects specific attacks (`mirai` and `gafgyt`) and limits samples to 2000 instances each.
- **Label Encoding:** Converts categorical features (`Attack` types) into numerical representations for ML models.
- **Regression Model (Linear Regression):** Attempts to predict attack types using linear regression (though the missing library prevents execution).
- **Pie Charts:** Displays category distributions of various attack types.
- **Heatmap (Attack vs. Sub-Attack):** Maps relationships between `Attack` and `Attack_SubType`.
- **Outlier Detection (Z-score):** Identifies outliers in network traffic using **Z-score thresholding**.
- **Visualization of Attack Labels:** Plots top attack sub-types based on occurrence frequency.

---

6. Cybersecurity Risk Assessment

6.1 Identifying Cybersecurity Risks in IoT Devices

IoT sensors are vulnerable to **DDoS attacks, data spoofing, and unauthorized access**. The following risks were assessed:

Cyber Threat	Impact on IoT Sensors
DDoS Attack	Overloads the network, causing **data delays and loss**
Data Spoofing	Attackers inject fake sensor data, affecting **system decisions**
Unauthorized Access	Hackers gain control over IoT devices, causing potential **shutdowns**

---

7. Mitigation Strategies for IoT Security

7.1 Defending Against DDoS and Other Attacks

Based on the cyber risk assessment, we propose the following **security enhancements**:

Mitigation Strategy	Implementation
Intrusion Detection System (IDS)	Detects unusual traffic spikes indicative of a **DDoS attack**
Anomaly Detection Models	Uses **ML-based algorithms** to identify irregular sensor readings

Edge Computing	Processes sensor data **locally** to reduce network dependency
Secure Authentication	Implements **strong encryption and authentication** to prevent unauthorized access

---

Data Processing Step

```
from google.colab import drive
drive.mount('/content/drive')

import numpy as np #
import pandas as pd #
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import autocorrelation_plot
import os


data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/AAI-530/iot_telemet
print(data.head())
```

```
Mounted at /content/drive
             ts             device        co   humidity  light       lpg  \
0  1.594512e+09  b8:27:eb:bf:9d:51  0.004956  51.000000  False  0.007651
1  1.594512e+09  00:0f:00:70:91:0a  0.002840  76.000000  False  0.005114
2  1.594512e+09  b8:27:eb:bf:9d:51  0.004976  50.900000  False  0.007673
3  1.594512e+09  1c:bf:ce:15:ec:4d  0.004403  76.800003   True  0.007023
4  1.594512e+09  b8:27:eb:bf:9d:51  0.004967  50.900000  False  0.007664

   motion     smoke       temp
0   False  0.020411  22.700000
1   False  0.013275  19.700001
2   False  0.020475  22.600000
3   False  0.018628  27.000000
4   False  0.020448  22.600000
```

```
print("\nMissing Values:")
print(data.isnull().sum())
```

```
Missing Values:
ts          0
device      0
co          0
humidity    0
light       0
lpg         0
motion      0
smoke       0
temp        0
dtype: int64
```

```
# Basic information about the dataset
print("\nDataset Info:")
data.info()
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 405184 entries, 0 to 405183
Data columns (total 9 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   ts        405184 non-null  float64
 1   device    405184 non-null  object
 2   co        405184 non-null  float64
 3   humidity  405184 non-null  float64
 4   light     405184 non-null  bool
 5   lpg       405184 non-null  float64
 6   motion    405184 non-null  bool
 7   smoke     405184 non-null  float64
 8   temp      405184 non-null  float64
dtypes: bool(2), float64(6), object(1)
memory usage: 22.4+ MB
```

'light' and 'motion' are transformed to 0 and 1.

```
# Transforming boolean columns 'light' and 'motion' into integers
data['light'] = data['light'].astype(int)
data['motion'] = data['motion'].astype(int)

# convert unix time to time of day
from datetime import datetime, timedelta
start = datetime(1970, 1, 1)  # Unix epoch start time
data['datetime'] = data.ts.apply(lambda x: start + timedelta(seconds=x))
data = data.drop('ts', axis=1)

# Convert the 'datetime' column to a datetime object, and make datetime column
data['datetime'] = pd.to_datetime(data['datetime'])
data.set_index('datetime', inplace=True)
data.head(5)
```

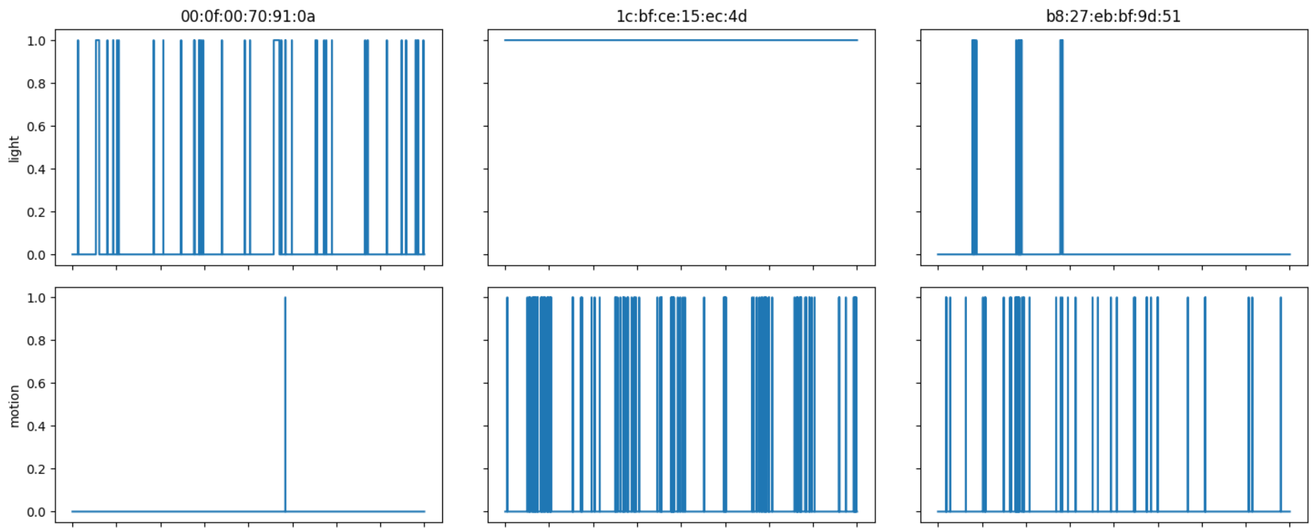
	device	co	humidity	light	lpg	motion	sm
datetime							
2020-07-12 00:01:34.385975	b8:27:eb:bf:9d:51	0.004956	51.000000	0	0.007651	0	0.020
2020-07-12 00:01:34.735568	00:0f:00:70:91:0a	0.002840	76.000000	0	0.005114	0	0.013
2020-07-12 00:01:38.073573	b8:27:eb:bf:9d:51	0.004976	50.900000	0	0.007673	0	0.020
2020-07-12	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
# Grouping data by 'device' and creating a separate DataFrame for each device
device_groups = data.groupby('device')

# Dictionary to store each device's DataFrame
device_df = {}

for device, group in device_groups:
    device_df[device] = group
```

Data Overview

```
def plot_device_sensors(device_df, undersample_rate=1):


    # Sensors in the desired order
    sensors = ['light', 'motion', 'temp', 'humidity', 'co', 'lpg', 'smoke']
```

```
    # Number of devices and sensors
    num_devices = len(device_df)
    num_sensors = len(sensors)

    # Create a figure with subplots
    fig, axes = plt.subplots(nrows=num_sensors, ncols=num_devices, figsize=(nun

    # Iterate through each device and sensor
    for j, (device_id, df) in enumerate(device_df.items()):
        # Undersample the data
        df_undersampled = df.iloc[::undersample_rate, :]

        for i, sensor in enumerate(sensors):
            # Plot each sensor in a separate subplot
            sns.lineplot(data=df_undersampled, x=df_undersampled.index, y=sensc
            axes[i, j].tick_params(axis='x', rotation=45)  # Rotate x-axis labe

            # Set x and y labels
            if j == 0:  # Only set y-axis label for the first column
                axes[i, j].set_ylabel(sensor)
            if i == num_sensors - 1:  # Only set x-axis label for the bottom rc
                axes[i, j].set_xlabel('Datetime')
            else:
                axes[i, j].set_xlabel('')

            # Set titles for the first row and first column
            if i == 0:
                axes[i, j].set_title(device_id)

    plt.tight_layout()
    plt.show()

plot_device_sensors(device_df, undersample_rate=2)
```

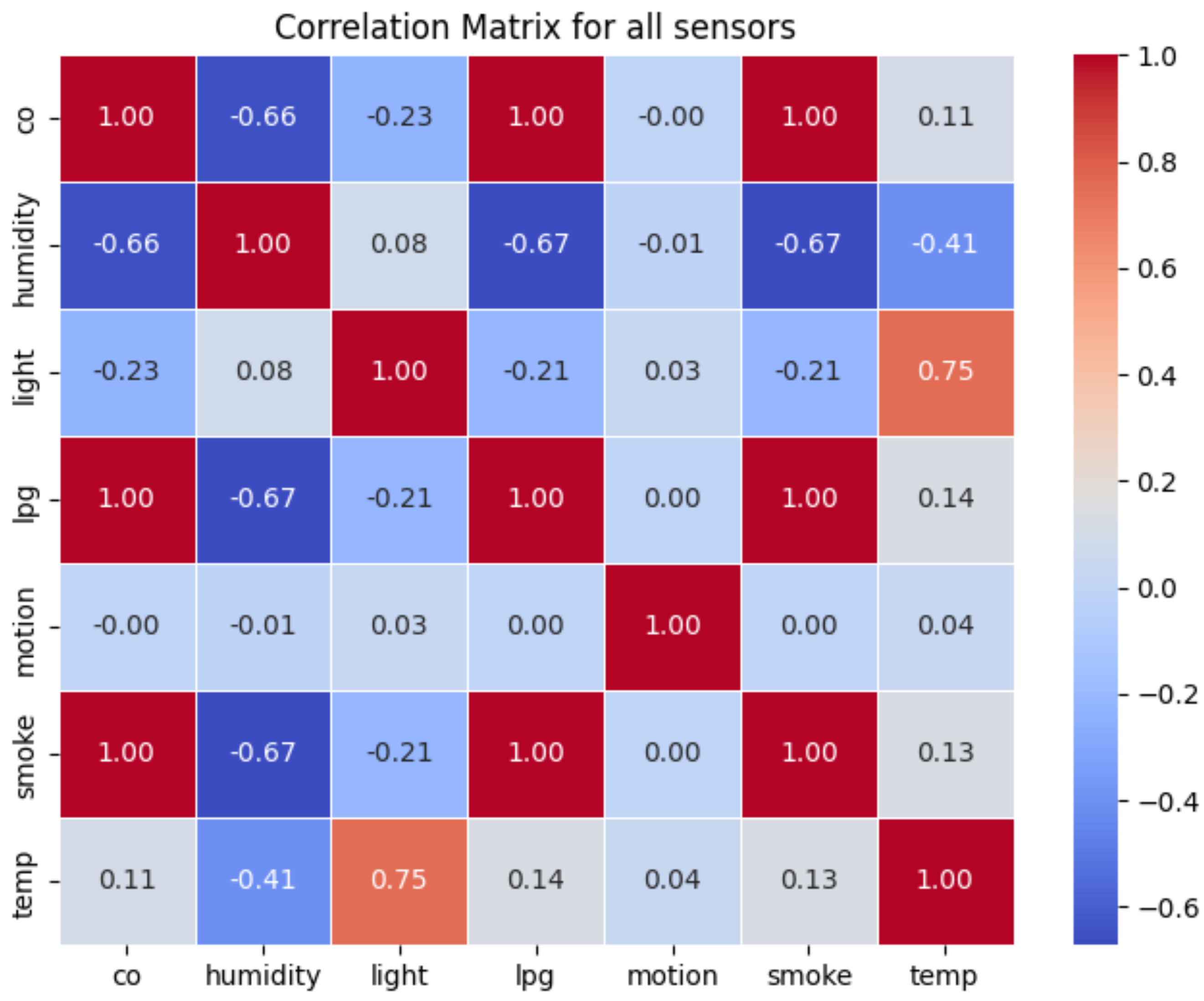



temp
humidity
co
lpg
smoke
Datetime
Datetime
Datetime
2020-07-12
2020-07-13
2020-07-14
2020-07-15
2020-07-16
2020-07-17
2020-07-18
2020-07-19
2020-07-20

Removing redundancies

CO, lpg, and smoke readout are highly correlated. correlation matrix to verify this observation.

```
# Drop the sensor column
corr_data = data.drop(['device'],axis=1)

# Compute the correlation matrix
corr_matrix = corr_data.corr()

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix for all sensors')
plt.show()
```



Removed lpg and smoke as they do not provide new information and kept only CO

```
# Iterating through the dictionary and removing 'lpg' and 'smoke' columns
for device_id, df in device_df.items():
    device_df[device_id] = df.drop(columns=['lpg', 'smoke'])

device_df
```

```
{'00:0f:00:70:91:0a':                                          device
co   humidity  light  \
 datetime
 2020-07-12 00:01:34.735568  00:0f:00:70:91:0a  0.002840  76.000000      0
 2020-07-12 00:01:46.869076  00:0f:00:70:91:0a  0.002938  76.000000      0
 2020-07-12 00:02:02.785732  00:0f:00:70:91:0a  0.002905  75.800003      0
 2020-07-12 00:02:11.476376  00:0f:00:70:91:0a  0.002938  75.800003      0
 2020-07-12 00:02:15.289086  00:0f:00:70:91:0a  0.002840  76.000000      0
 ...                                       ...       ...        ...    ...
 2020-07-20 00:03:16.329782  00:0f:00:70:91:0a  0.003745  75.300003      0
 2020-07-20 00:03:20.684223  00:0f:00:70:91:0a  0.003745  75.400002      0
 2020-07-20 00:03:25.039890  00:0f:00:70:91:0a  0.003745  75.400002      0
 2020-07-20 00:03:33.162015  00:0f:00:70:91:0a  0.003745  75.300003      0
 2020-07-20 00:03:36.979522  00:0f:00:70:91:0a  0.003745  75.300003      0

                             motion       temp
 datetime
 2020-07-12 00:01:34.735568       0  19.700001
 2020-07-12 00:01:46.869076       0  19.700001
 2020-07-12 00:02:02.785732       0  19.700001
 2020-07-12 00:02:11.476376       0  19.700001
 2020-07-12 00:02:15.289086       0  19.700001
 ...                            ...        ...
 2020-07-20 00:03:16.329782       0  19.200001
 2020-07-20 00:03:20.684223       0  19.200001
 2020-07-20 00:03:25.039890       0  19.200001
 2020-07-20 00:03:33.162015       0  19.200001
 2020-07-20 00:03:36.979522       0  19.200001

 [111815 rows x 6 columns],
 '1c:bf:ce:15:ec:4d':                                          device
co   humidity  light  \
 datetime
 2020-07-12 00:01:39.589146  1c:bf:ce:15:ec:4d  0.004403  76.800003      1
 2020-07-12 00:01:44.468411  1c:bf:ce:15:ec:4d  0.004391  77.900002      1
 2020-07-12 00:01:48.275382  1c:bf:ce:15:ec:4d  0.004345  77.900002      1
 2020-07-12 00:01:55.288543  1c:bf:ce:15:ec:4d  0.004383  78.000000      1
 2020-07-12 00:01:59.098014  1c:bf:ce:15:ec:4d  0.004451  78.000000      1
 ...                                       ...       ...        ...    ...
 2020-07-20 00:03:09.090696  1c:bf:ce:15:ec:4d  0.004524  75.900002      1
 2020-07-20 00:03:20.460079  1c:bf:ce:15:ec:4d  0.004532  75.900002      1
 2020-07-20 00:03:24.269880  1c:bf:ce:15:ec:4d  0.004532  75.900002      1
 2020-07-20 00:03:30.755704  1c:bf:ce:15:ec:4d  0.004553  75.800003      1
 2020-07-20 00:03:36.167959  1c:bf:ce:15:ec:4d  0.004540  75.699997      1

                             motion  temp
 datetime
```

```
2020-07-12 00:01:39.589146      0  27.0
2020-07-12 00:01:44.468411      0  27.0
2020-07-12 00:01:48.275382      0  27.0
2020-07-12 00:01:55.288543      0  27.0
2020-07-12 00:01:59.098014      0  27.0
...                           ...   ...
2020-07-20 00:03:09.090696      0  26.6
2020-07-20 00:03:20.460079      0  26.6
2020-07-20 00:03:24.269880      0  26.6
2020-07-20 00:03:30.755704      0  26.6
2020-07-20 00:03:36.167959      0  26.6
```

Step Determine sampling rates for performing analysis - seasonal and daily trend

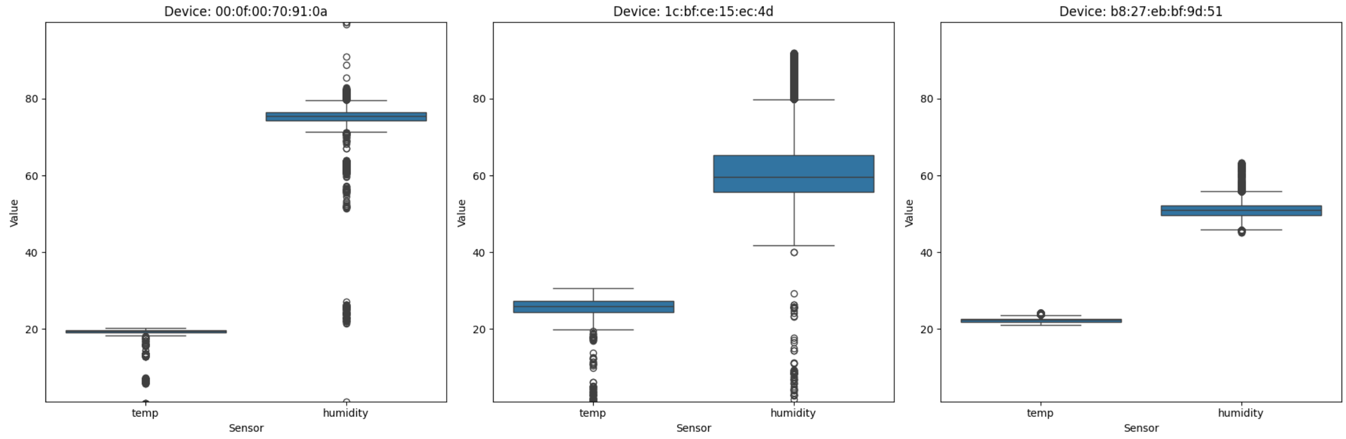
```
# Function that calculate the sampling rate
def calculate_sampling_stats(device_df):
    sampling_stats = {}

    for device_id, df in device_df.items():
        # Calculate time differences between consecutive data points
        time_diffs = df.index.to_series().diff().dropna()

        # Convert time differences to a consistent unit, e.g., seconds
        time_diffs_in_seconds = time_diffs.dt.total_seconds()

        # Calculate mean and standard deviation
        mean_sampling_rate = time_diffs_in_seconds.mean()
        std_sampling_rate = time_diffs_in_seconds.std()

        # Store in dictionary
        sampling_stats[device_id] = {'mean': mean_sampling_rate, 'std': std_sam

    return sampling_stats

# Calculate sampling stats for each device
device_sampling_stats = calculate_sampling_stats(device_df)
for device, stats in device_sampling_stats.items():
    print(f"Device {device} - Mean Sampling Rate: {stats['mean']}s, Std Dev: {s
```

```
Device 00:0f:00:70:91:0a - Mean Sampling Rate: 6.182787879460532s, Std Dev:
Device 1c:bf:ce:15:ec:4d - Mean Sampling Rate: 6.526965254047981s, Std Dev:
Device b8:27:eb:bf:9d:51 - Mean Sampling Rate: 3.6880388281568415s, Std Dev
```

Step Understand the Environmental Characteristics from the device sensors

Temperature and Humidity of each device is calculated.

```
def plot_sensor_boxplots(device_df, sensors=['temp', 'humidity']):
    """
    Plot sensors from each device with same y-axis scale.
    """

    num_devices = len(device_df)
    fig, axes = plt.subplots(nrows=1, ncols=num_devices, figsize=(num_devices *

    # Determine the global min and max values across all devices for each senso
    global_min = {sensor: float('inf') for sensor in sensors}
    global_max = {sensor: float('-inf') for sensor in sensors}

    for df in device_df.values():
        for sensor in sensors:
            global_min[sensor] = min(global_min[sensor], df[sensor].min())
            global_max[sensor] = max(global_max[sensor], df[sensor].max())

    # Plot the box plots
    for j, (device_id, df) in enumerate(device_df.items()):
        data_to_plot = df[sensors].melt(var_name='Sensor', value_name='Value')
        sns.boxplot(x='Sensor', y='Value', data=data_to_plot, ax=axes[j])
        axes[j].set_title(f'Device: {device_id}')

        # Set the same y-axis limits for each subplot
        for i, sensor in enumerate(sensors):
            axes[j].set_ylim([global_min[sensor], global_max[sensor]])

    plt.tight_layout()
    plt.show()

plot_sensor_boxplots(device_df)
```



Device Name - '00:0f:00:70:91:0a Device Name - '1c:bf:ce:15:ec:4d' Device Name - 'b8:27:eb:bf:9d:51'

- **Device Mac : '00:0f:00:70:91:0a (00)** This environment is cool and humid, indicative of a well-controlled indoor setting. Such conditions suggest a stable and consistent environmental control system.
- **Device '1c:bf:ce:15:ec:4d'** This environment is warm and moderately humid, with significant variation. The fluctuating readings indicate a less well-controlled environment, where temperature and humidity are subject to natural variations or human activity.
- **Device 'b8:27:eb:bf:9d:51' -** This environment is warm and dry, and it also appears to be well-controlled. The consistent readings suggest an stable environmental control, similar to Device 00, but with a different temperature and humifty settings.

Step Extract (Moving Average) Frequency of the sensors from Binary Data

motion and light sensor processing

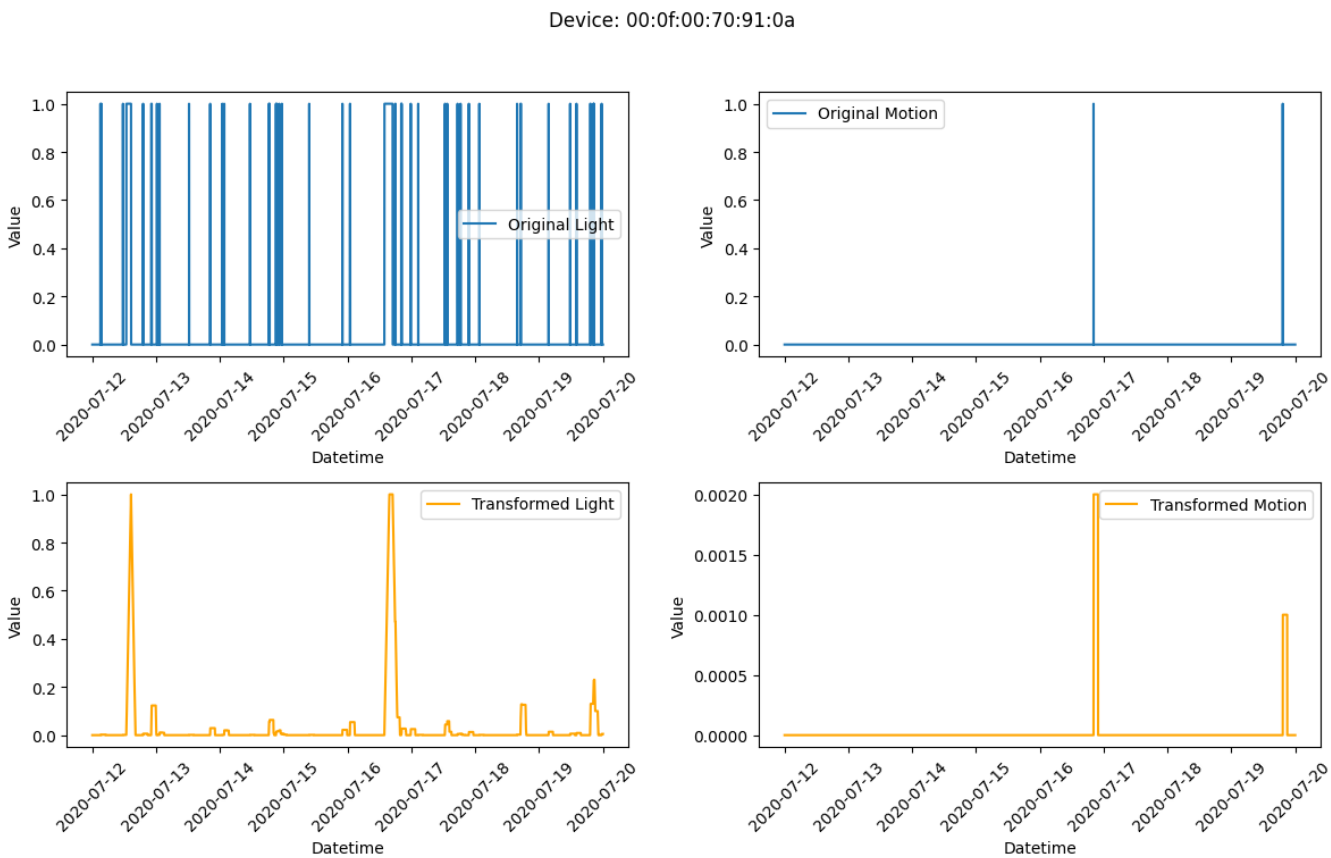
Given the binary nature of this data, which does not readily convey the intensity of sensor activity, rolling average is considered.

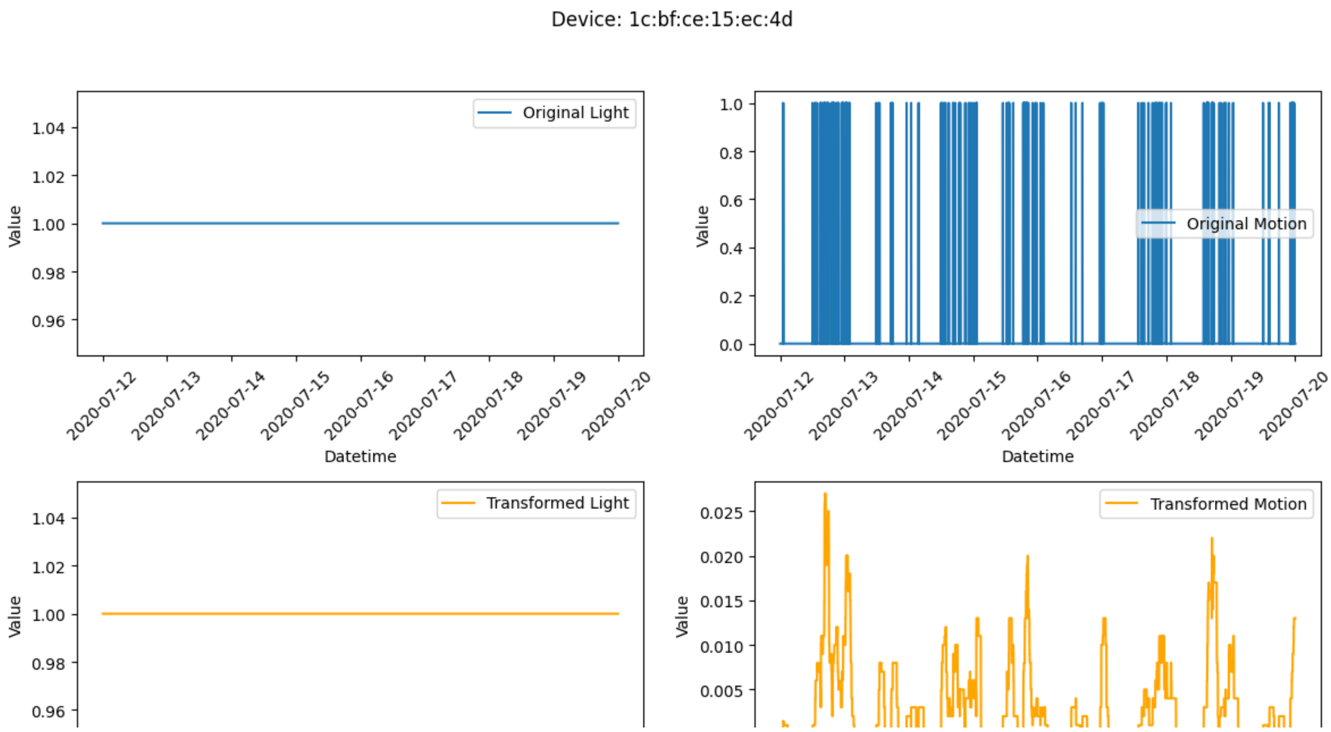
This approach will smooth out the data epochs, transforming the binary readouts into a more continuous and interpretable measure of sensor activity frequency.

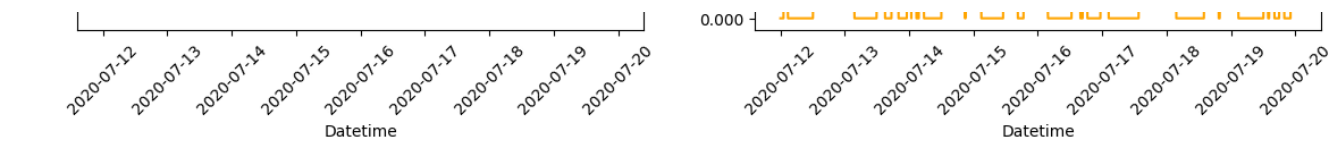
The rolling average (also known as the moving average) is a common technique used in time-series analysis to smooth out short-term fluctuations and highlight longer-term trends or cycles. For a time-series dataset

```
def transform_binary_to_frequency(df, window_size):

    df_transformed = df.rolling(window=window_size, min_periods=1).mean()
    return df_transformed

def plot_transformed_data(device_df, window_size):

    for device_id, df in device_df.items():
        # Transform binary signals
        df_transformed = transform_binary_to_frequency(df[['light', 'motion']],

        # Plotting
        fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
        fig.suptitle(f"Device: {device_id}")

        # Original Data
        axes[0, 0].plot(df.index, df['light'], label='Original Light')
        axes[0, 1].plot(df.index, df['motion'], label='Original Motion')

        # Transformed Data
        axes[1, 0].plot(df_transformed.index, df_transformed['light'], label='T
        axes[1, 1].plot(df_transformed.index, df_transformed['motion'], label='

        # Setting labels
        for i in range(2):
            for j in range(2):
                axes[i, j].set_xlabel('Datetime')
                axes[i, j].set_ylabel('Value')
```

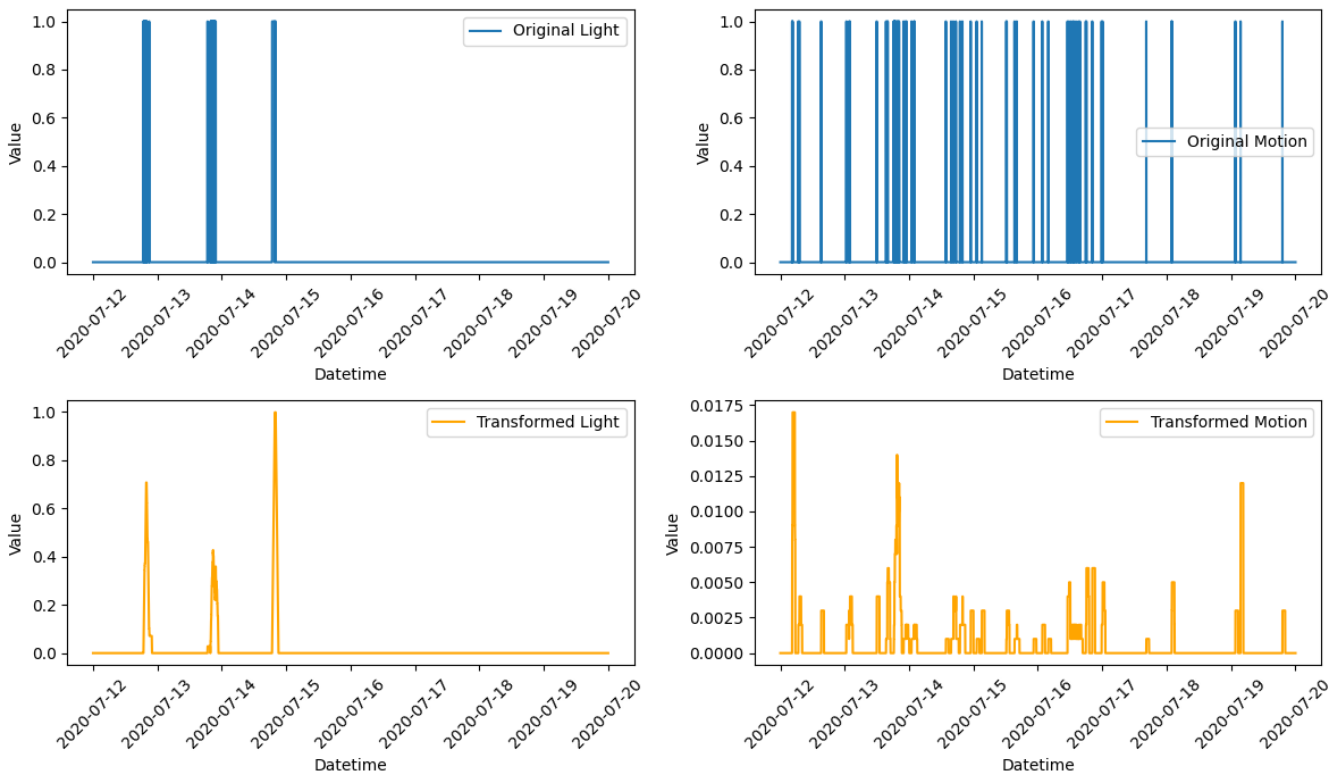
```
            axes[i, j].legend()
            axes[i, j].tick_params(axis='x', rotation=45)

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

window_size = 1000 # Adjustable until a sweet spot is found
plot_transformed_data(device_df, window_size)
```







Device: b8:27:eb:bf:9d:51



Step Autocorrelation Used for Detecting Seasonality information

The light and motion sensor data before and after applying a rolling average, transforming the data from binary to a continuous format. This transformed data is then reintegrated into the original grouped dataframe.

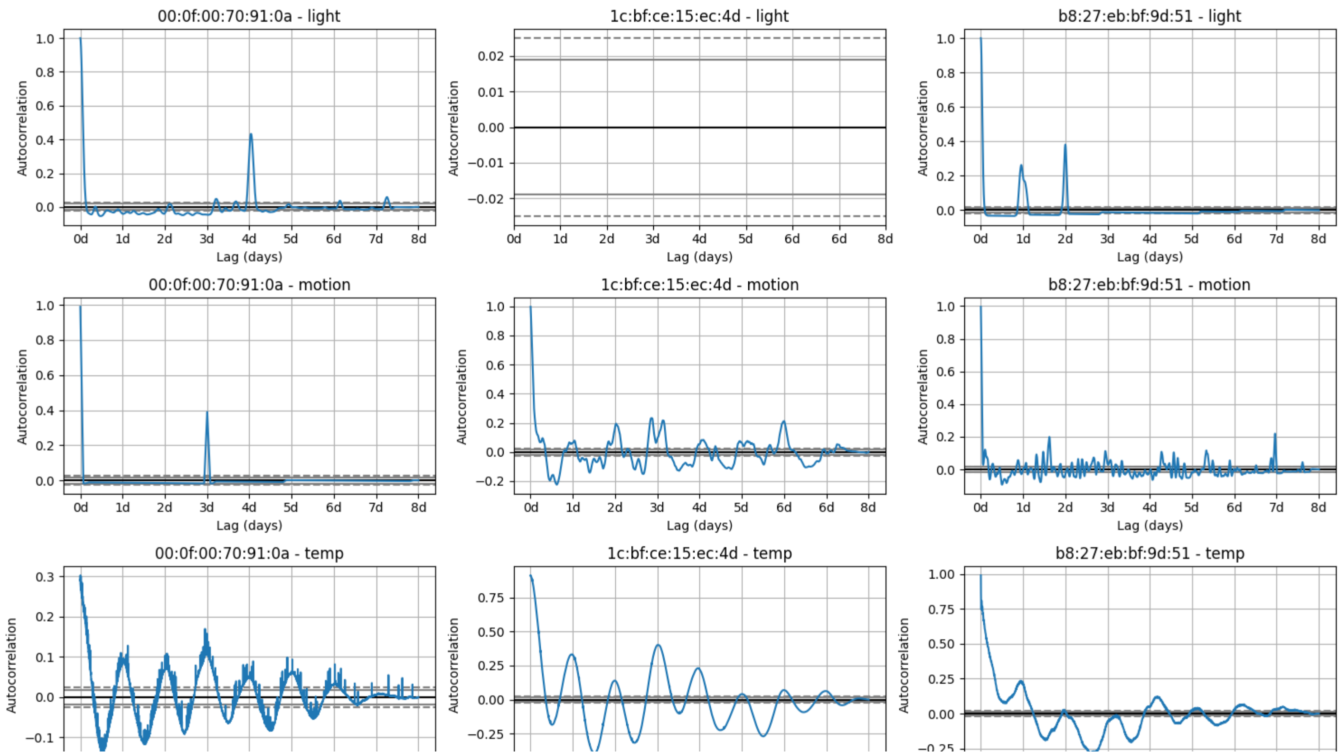
Next, we use the previously calculated sampling rates to generate autocorrelation plots so that we can observe the seasonal patterns within the dataset.

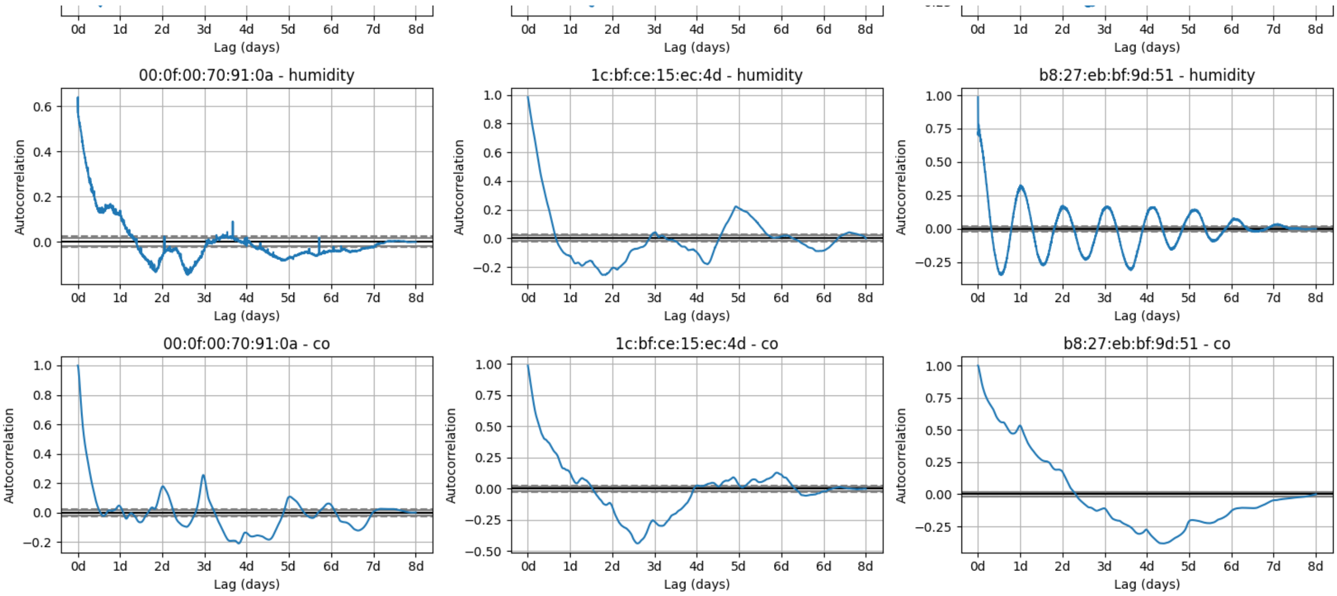
Autocorrelation is a statistical measure that describes the degree to which a time series (a sequence of data points collected over time) is correlated with itself at different time lags. In other words, it measures the relationship between a variable's current value and its past values. Autocorrelation is commonly used in time series analysis to identify patterns, trends, and seasonality in data.

```
def plot_autocorrelation(device_df, undersample_rate=1, sampling_rates=None):

    # Sensors in the desired order
    sensors = ['light', 'motion', 'temp', 'humidity', 'co']

    # Number of devices and sensors
    num_devices = len(device_df)
    num_sensors = len(sensors)

    # Seconds in a day
    seconds_in_day = 24 * 60 * 60

    # Create a figure with subplots
    fig, axes = plt.subplots(nrows=num_sensors, ncols=num_devices, figsize=(num

    # Iterate through each device and sensor
    for j, (device_id, df) in enumerate(device_df.items()):
        # Calculate number of samples in a day
        samples_per_day = seconds_in_day / (sampling_rates[device_id] * undersa

        # Undersample the data
        df_undersampled = df.iloc[::undersample_rate, :]

        for i, sensor in enumerate(sensors):
            # Create autocorrelation plot for each sensor
            autocorrelation_plot(df_undersampled[sensor], ax=axes[i][j])
```

```
            axes[i][j].set_title(f'{device_id} - {sensor}')
            axes[i][j].set_xlabel('Lag (days)')
            axes[i][j].set_ylabel('Autocorrelation')

            # Adjust x-axis to represent days
            max_lag = df_undersampled[sensor].shape[0]
            xticks = np.arange(0, max_lag, samples_per_day)
            xticklabels = [f"{int(lag/samples_per_day)}d" for lag in xticks]
            axes[i][j].set_xticks(xticks)
            axes[i][j].set_xticklabels(xticklabels)

    plt.tight_layout()
    plt.show()

# Apply the transformation to binary data
window_size = 1000
for device_id in device_df:
    device_df[device_id][['light', 'motion']] = transform_binary_to_frequency(d

# Sampling rates for each device
sampling_rates = {
    '00:0f:00:70:91:0a': 6.182787879460532,
    '1c:bf:ce:15:ec:4d': 6.526965254047981,
    'b8:27:eb:bf:9d:51': 3.6880388281568415
}

plot_autocorrelation(device_df, undersample_rate=10, sampling_rates=sampling_ra
```

```
/usr/local/lib/python3.11/dist-packages/pandas/plotting/_matplotlib/misc.py
  return ((data[: n - h] - mean) * (data[h:] - mean)).sum() / n / c0
```




00:0f:00:70:91:0a - humidity
1c:bf:ce:15:ec:4d - humidity
b8:27:eb:bf:9d:51 - humidity
00:0f:00:70:91:0a - co
1c:bf:ce:15:ec:4d - co
b8:27:eb:bf:9d:51 - co
Autocorrelation
Lag (days)
0d
1d
2d
3d
4d
5d
6d
7d
8d

Output

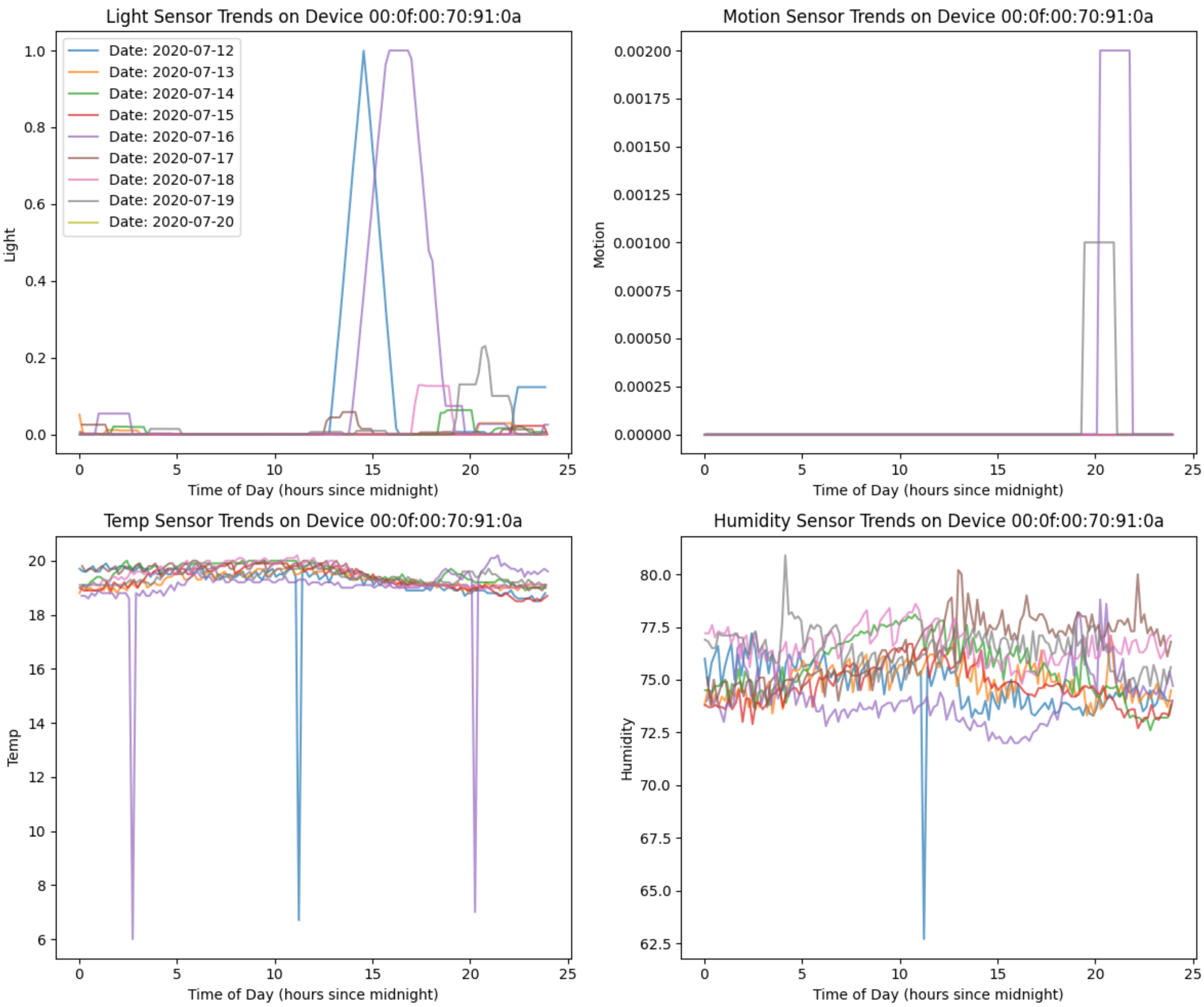
- All temperature sensors display daily seasonality, albeit with varying intensities.
- Device 1c-temperature stands out with the strongest seasonality, marked by high correlation factors and good signal-to-noise ratios. The pronounced seasonality in Device 1c coincides with daily human activity within its environment. Interestingly, this particular correlation pattern is not present in the other two devices.
- Regarding humidity, each environment demonstrates unique trends. In the Device 00 setting, there is an absence of noticeable daily seasonality. In contrast, Device 1c shows a vague bi-daily pattern, indicating more complex environmental dynamics. Device b8, on the other hand, experiences regular daily fluctuations in humidity.

Step Daily trends - Sensors information.

Convert time to hours for calculation and sample using moving average With the observation of a daily pattern, we analyze and compare the daily variations across all devices.

```
def convert_time_to_hours(time_obj):

    return time_obj.hour + time_obj.minute / 60 + time_obj.second / 3600

def plot_daily_trends_for_sensors(device_df, device_id, sensors, undersample_ra
    df = device_df[device_id]

    # Create a 2x2 subplot grid
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
    axes = axes.flatten()  # Flatten the axes array for easy indexing

    for i, sensor in enumerate(sensors):
        # Undersample and apply moving average
        df_resampled = df.iloc[::undersample_rate, :]
        df_smoothed = df_resampled[sensor].rolling(window=window_size, min_peri

        # Group by date and plot each day's data
        for date, group in df_smoothed.groupby(df_smoothed.index.date):
            # Convert index time to hours
            hours_since_midnight = [convert_time_to_hours(t) for t in group.inc
            axes[i].plot(hours_since_midnight, group, alpha=0.7, label=f'Date:
```

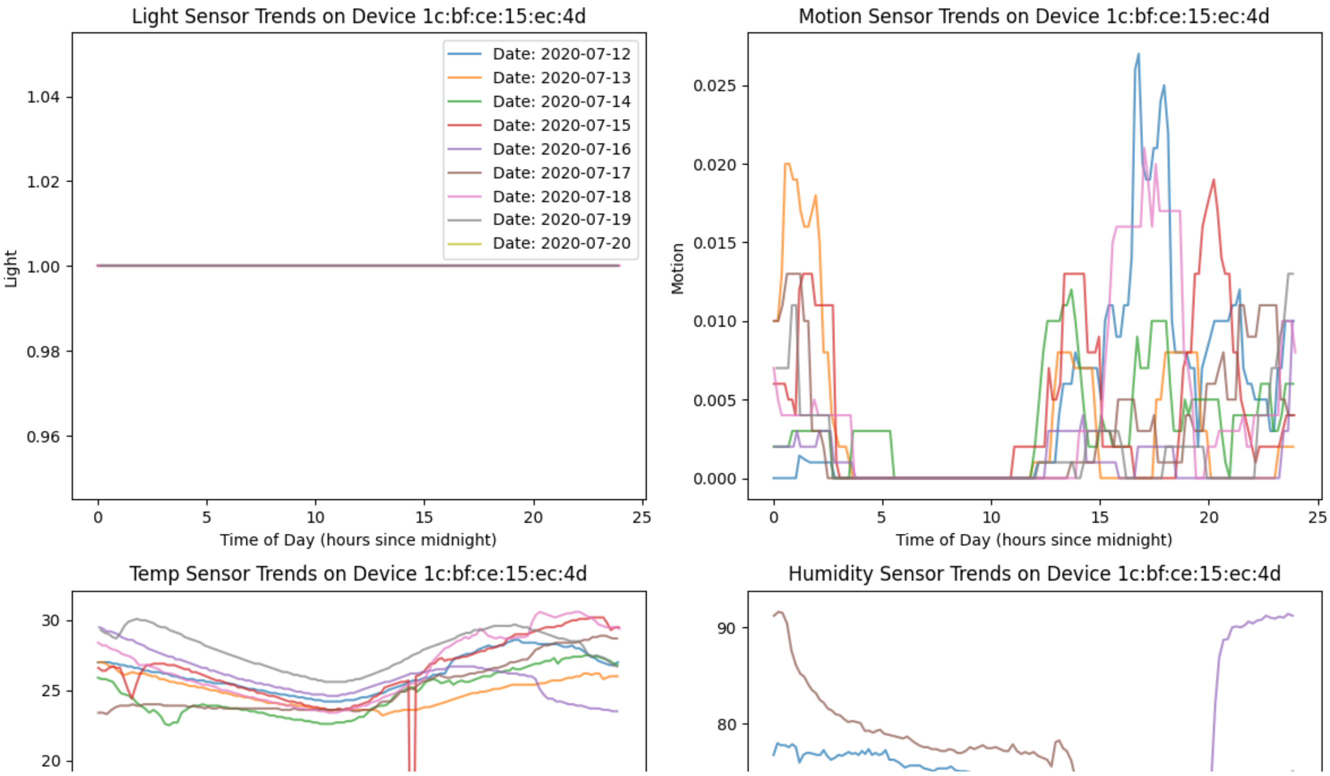
```
        axes[i].set_title(f'{sensor.capitalize()} Sensor Trends on Device {devi
        axes[i].set_xlabel('Time of Day (hours since midnight)')
        axes[i].set_ylabel(f'{sensor.capitalize()}')
        if i == 0:
            axes[i].legend()

    # Adjust layout and show plot
    fig.tight_layout()
    plt.show()

sensors = ['light', 'motion', 'temp', 'humidity']
plot_daily_trends_for_sensors(device_df, '00:0f:00:70:91:0a', sensors)
```

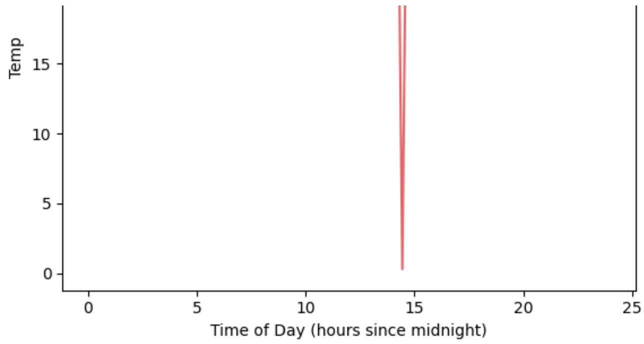


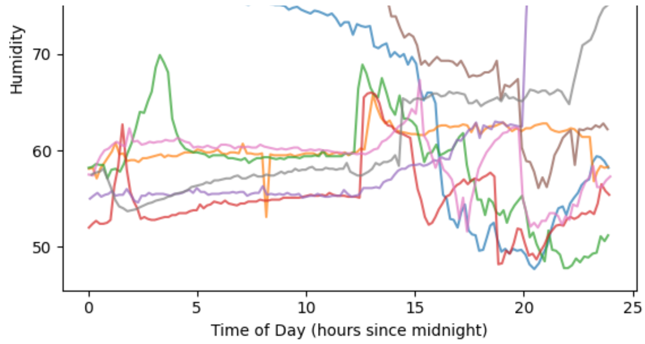
Output from Device 00 analysis

- Consistent with our previous observations, the temperature and humidity in the Device 00 environment appear to be well-controlled.
- Light and motion activities are rare and predominantly observed during afternoon and nighttime.

```
plot_daily_trends_for_sensors(device_df, '1c:bf:ce:15:ec:4d', sensors)
```



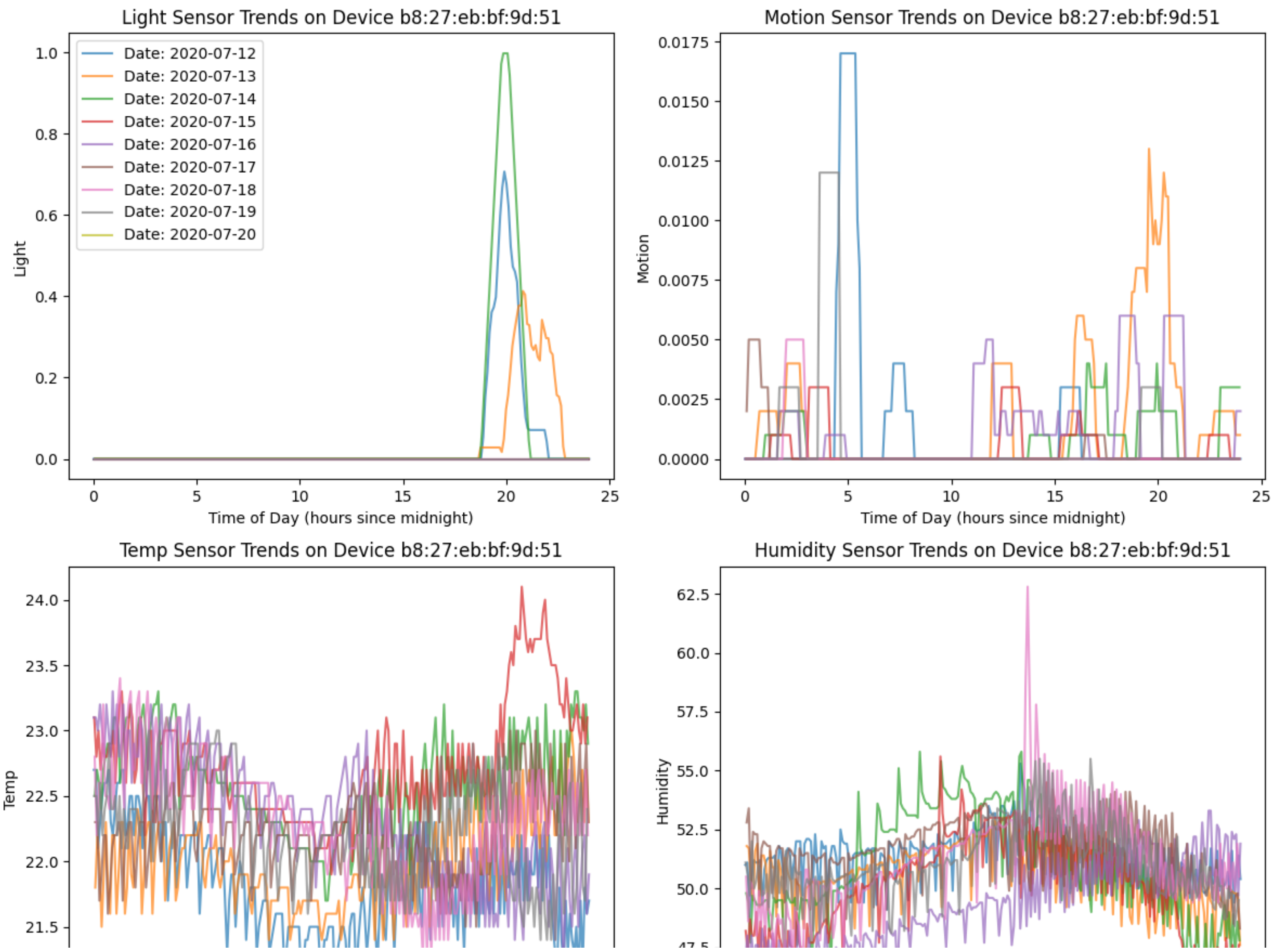

Temp
0
5
10
15
0
5
10
15
20
25
Time of Day (hours since midnight)

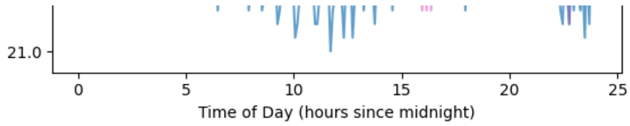

Humidity
50
60
70
0
5
10
15
20
25
Time of Day (hours since midnight)

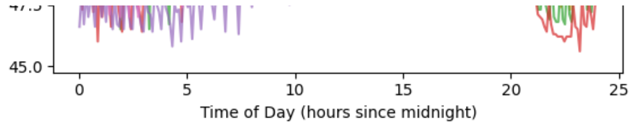
Output of Device 1c analysis

- Device 1c displays a distinct pattern compared to Device 00, with a noticeable daily fluctuations in temperature. This device records the coolest temperatures around noon, with an increase towards the night. This is a pattern opposite to natural temperature cycle. Moreover, human activity also concentrate on these periods. Therefore, such a pattern suggests a causal relationship between human activity and temperature dynamics within this environment, consistent with the seasonality observation.
- One can make assumption about this environment, say, a workshop in constant use. Another supporting evidence is the constant lighting condition.
- Additionally, the variability in humidity levels also suggests that the environment control here is less stringent, contrasting with the more stable environments observed in other devices. Based on these, we can imagine that the other two environments may be storage room with well-controlled environment and less frequent human activity.

```
plot_daily_trends_for_sensors(device_df, 'b8:27:eb:bf:9d:51', sensors)
```




21.0
0
5
10
15
20
25
Time of Day (hours since midnight)


45.0
0
5
10
15
20
25
Time of Day (hours since midnight)

Output of Device b8 analysis

- The light and motion sensor readings suggest that human activity tends to avoid the noon hours, instead concentrating during the afternoon and nighttime. The correlation between temperature and human activity is similar with the workshop area where Device 1c is located. However, the human activity is too random for any pattern to emerge in the autocorrelation plot.
- The environmental stability of Device b8 closely mirrors that of Device 00. A key distinction, however, lies in the high-frequency features in the data, in contrast to the high-intensity peaks observed in the other two devices. This difference is also presence in the humidity readings.
- The detailed sensor readout of Device b8, which is free of high-intensity noise and rich in periodic features, could be attributed to its sampling frequency being twice that of the other devices, enabling a finer resolution of data capture. Alternatively, the variation might also due to the devices being exposed to varying environmental influences, such as vibrations from human activities or mechanical operations. These factors could significantly affect the sensor outputs and need to be considered when interpreting the data.

Output Sensor IOT Results Intepretation

This data analysis project successfully interprete IoT sensor data to infer the environmental conditions and their correlation with human activity.

Environmental Control and Variation: The analysis revealed distinct environmental profiles for each device, with Device 00 and Device b8 exhibiting well-controlled temperature and humidity conditions resembling those of storage rooms, whereas Device 1c showed more variation resembling that of a work area with less environmental control.

Sensor Data Optimization: Correlation analysis led to the removal of redundant sensors (LPG and smoke), demonstrating the potential for more efficient sensor deployment and data collection strategies.

Advanced Data Processing Techniques: Techniques such as transforming binary data into continuous measures and autocorrelation analysis were effectively employed, enabling seasonal analysis that suggests the strong correlation between temperature variation and human activity.

The demonstrates the power of data analytics in extracting meaningful insights from IoT sensor data, providing a comprehensive understanding of indoor environmental dynamics and human interaction within these spaces.

Transformer Analysis for temperature prediction, training, fit and predicting and plotting

1. Time Series Analysis (Temperature Prediction and Anomaly Detection):

```
!pip install pytorch-lightning
```

```
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.w
   24.6/24.6 MB 14.4 MB/s eta 0:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64
   883.7/883.7 kB 40.9 MB/s eta 0:
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (6
   664.8/664.8 MB 2.6 MB/s eta 0:0
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (2
   211.5/211.5 MB 6.6 MB/s eta 0:0
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl
   56.3/56.3 MB 11.2 MB/s eta 0:00
```

```
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl
                                          127.9/127.9 MB 9.1 MB/s eta 0:0
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.w
                                          207.5/207.5 MB 5.5 MB/s eta 0:0
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.wh
                                          21.1/21.1 MB 76.9 MB/s eta 0:00
Downloading torchmetrics-1.6.1-py3-none-any.whl (927 kB)
                                          927.3/927.3 kB 57.5 MB/s eta 0:
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, n
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
```

```
import torch
import torch.nn as nn
```

```
import pytorch_lightning as pl
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import pandas as pd
import torch

import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from pytorch_lightning import LightningModule, Trainer
from pytorch_lightning.callbacks import EarlyStopping



from google.colab import drive
drive.mount('/content/drive')

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import autocorrelation_plot
import os

#
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/AAI-530/iot_telemet
print(data.head())
df = data


df = df.sort_values(by="ts")  # Ensure time ordering
df["ts"] = pd.to_datetime(df["ts"], unit='s')  # Convert timestamp

# Select relevant features for prediction
features = ["humidity", "co", "lpg", "smoke", "temp"]

# Normalize the data
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])

# Convert to numpy array
data = df[features].values

# Define dataset class for PyTorch Lightning
class TimeSeriesDataset(Dataset):
    def __init__(self, data, seq_length=10):
        self.data = data
        self.seq_length = seq_length
```

```
    def __len__(self):
        return len(self.data) - self.seq_length

    def __getitem__(self, idx):
        x = self.data[idx:idx + self.seq_length, :-1]  # Input features
        y = self.data[idx + self.seq_length, -1]  # Temperature target

        # Convert to PyTorch tensors and stack x
        x = torch.tensor(x, dtype=torch.float32)
        y = torch.tensor(y, dtype=torch.float32)

        return x, y

# Split dataset
seq_length = 10
train_size = int(0.8 * len(data))
train_dataset = TimeSeriesDataset(data[:train_size], seq_length)
test_dataset = TimeSeriesDataset(data[train_size:], seq_length)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Define Transformer Model
class TimeSeriesDataset(Dataset):
    def __init__(self, data, seq_length=10):
        self.data = data
        self.seq_length = seq_length

    def __len__(self):
        return len(self.data) - self.seq_length

    def __getitem__(self, idx):
        x = self.data[idx:idx + self.seq_length, :-1]  # Input features
        y = self.data[idx + self.seq_length, -1]  # Temperature target

        # Convert to PyTorch tensors and stack x
        x = torch.tensor(x, dtype=torch.float32)
        y = torch.tensor(y, dtype=torch.float32)

        return x, y  # Return x as a single tensor, and y as a tensor

# Split dataset
seq_length = 10
train_size = int(0.8 * len(data))
train_dataset = TimeSeriesDataset(data[:train_size], seq_length)
test_dataset = TimeSeriesDataset(data[train_size:], seq_length)
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Define Transformer Model
class TransformerTimeSeries(pl.LightningModule):
    def __init__(self, input_dim, d_model=64, nhead=4, num_layers=2, lr=1e-3):
        super().__init__()
        self.lr = lr
        self.encoder = nn.Linear(input_dim, d_model)
        self.transformer = nn.Transformer(
            d_model=d_model, nhead=nhead, num_encoder_layers=num_layers, num_de
        )
        self.decoder = nn.Linear(d_model, 1)
        self.loss_fn = nn.MSELoss()

    def forward(self, src):


        src = src.view(src.size(0), src.size(1), -1).float()  # Reshape and con

        src = self.encoder(src)
        output = self.transformer(src, src)
        return self.decoder(output[:, -1, :])  # Predict next time step


    def training_step(self, batch, batch_idx):
        x, y = batch
        y_pred = self(x).squeeze()
        loss = self.loss_fn(y_pred, y)
        self.log("train_loss", loss, prog_bar=True)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_pred = self(x).squeeze()
        loss = self.loss_fn(y_pred, y)
        self.log("val_loss", loss, prog_bar=True)

    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=self.lr)

# Instantiate the model
model = TransformerTimeSeries(input_dim=len(features) - 1)
```

```
Mounted at /content/drive
             ts             device        co   humidity  light       lpg  \
0  1.594512e+09  b8:27:eb:bf:9d:51  0.004956  51.000000  False  0.007651
1  1.594512e+09  00:0f:00:70:91:0a  0.002840  76.000000  False  0.005114
2  1.594512e+09  b8:27:eb:bf:9d:51  0.004976  50.900000  False  0.007673
3  1.594512e+09  1c:bf:ce:15:ec:4d  0.004403  76.800003   True  0.007023
4  1.594512e+09  b8:27:eb:bf:9d:51  0.004967  50.900000  False  0.007664

   motion     smoke       temp
0   False  0.020411  22.700000
1   False  0.013275  19.700001
2   False  0.020475  22.600000
3   False  0.018628  27.000000
4   False  0.020448  22.600000
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/transformer.py:379
  warnings.warn(
```

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
import pytorch_lightning as pl
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
import os

# Install necessary libraries
!pip install pytorch-lightning

# Mount Google Drive
drive.mount('/content/drive')

# Load the dataset
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/AAI-530/iot_telemet

# Preprocess the data
df = data.copy()
df = df.sort_values(by="ts")
df["ts"] = pd.to_datetime(df["ts"], unit='s')
features = ["humidity", "co", "lpg", "smoke", "temp"]
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])
data = df[features].values

# Define the dataset
class TimeSeriesDataset(Dataset):
```

```
    def __init__(self, data, seq_length=10):
        self.data = data
        self.seq_length = seq_length

    def __len__(self):
        return len(self.data) - self.seq_length

    def __getitem__(self, idx):
        x = self.data[idx:idx + self.seq_length, :-1]  # Input features only
        y = self.data[idx + self.seq_length, -1]  # Target (temperature)
        return torch.tensor(x, dtype=torch.float32), torch.tensor(y, dtype=torc

# Split the data
seq_length = 10
train_size = int(0.8 * len(data))
train_dataset = TimeSeriesDataset(data[:train_size], seq_length)
test_dataset = TimeSeriesDataset(data[train_size:], seq_length)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

# Define the model
class TransformerTimeSeries(pl.LightningModule):
    def __init__(self, input_dim, d_model=64, nhead=4, num_layers=2, lr=1e-3):
        super().__init__()
        self.save_hyperparameters()  # Save hyperparameters for easier loading
        self.encoder = nn.Linear(input_dim, d_model)
        self.transformer = nn.Transformer(d_model=d_model, nhead=nhead,
                                          num_encoder_layers=num_layers,
                                          num_decoder_layers=num_layers)
        self.decoder = nn.Linear(d_model, 1)
        self.loss_fn = nn.MSELoss()

    def forward(self, x):
        # Reshape and ensure float type within forward
        x = x.view(x.size(0), x.size(1), -1).float()
        x = self.encoder(x)
        output = self.transformer(x, x)  # Pass x as both encoder and decoder i
        return self.decoder(output[:, -1, :])  # Predict next time step

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_pred = self(x).squeeze()
        loss = self.loss_fn(y_pred, y)
        self.log("train_loss", loss, prog_bar=True)
        return loss

    def validation_step(self, batch, batch_idx):
```

```
        x, y = batch
        y_pred = self(x).squeeze()
        loss = self.loss_fn(y_pred, y)
        self.log("val_loss", loss, prog_bar=True)

    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=self.hparams.lr)

    def predict_step(self, batch, batch_idx, dataloader_idx=0):
        x, _ = batch  # Extract input features, ignore target
        return self(x)  # Directly call forward with input features

# Train the model
model = TransformerTimeSeries(input_dim=len(features) - 1)
trainer = pl.Trainer(max_epochs=1, accelerator="gpu" if torch.cuda.is_available
                     devices=1 if torch.cuda.is_available() else 0)
trainer.fit(model, train_loader)
```

```
Requirement already satisfied: pytorch-lightning in /usr/local/lib/python3.
Requirement already satisfied: torch>=2.1.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: tqdm>=4.57.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: PyYAML>=5.4 in /usr/local/lib/python3.11/dis
Requirement already satisfied: fsspec>=2022.5.0 in /usr/local/lib/python3.1
Requirement already satisfied: torchmetrics>=0.7.0 in /usr/local/lib/python
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/p
Requirement already satisfied: lightning-utilities>=0.10.0 in /usr/local/li
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/loc
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/l
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/loc
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/li
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/l
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/li
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/loc
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/p
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/loca
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/d
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/py
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.1
```

```
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.1
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.1
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11
Requirement already satisfied: idna>=2.0 in /usr/local/lib/python3.11/dist-
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/transformer.py:379
  warnings.warn(
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 T
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 H
/usr/local/lib/python3.11/dist-packages/pytorch_lightning/trainer/configura
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVIC
INFO:pytorch_lightning.callbacks.model_summary:
  | Name        | Type        | Params | Mode
-------------------------------------------------------
0 | encoder     | Linear      | 320    | train
1 | transformer | Transformer | 1.2 M  | train
2 | decoder     | Linear      | 65     | train
3 | loss_fn     | MSELoss     | 0      | train
-------------------------------------------------------
1.2 M     Trainable params
0         Non-trainable params
1.2 M     Total params
4.635     Total estimated model params size (MB)
58        Modules in train mode
0         Modules in eval mode
```

Epoch 0: 100% 10130/10130 [02:53<00:00, 58.28it/s, v_num=0, train_loss=0.00696]

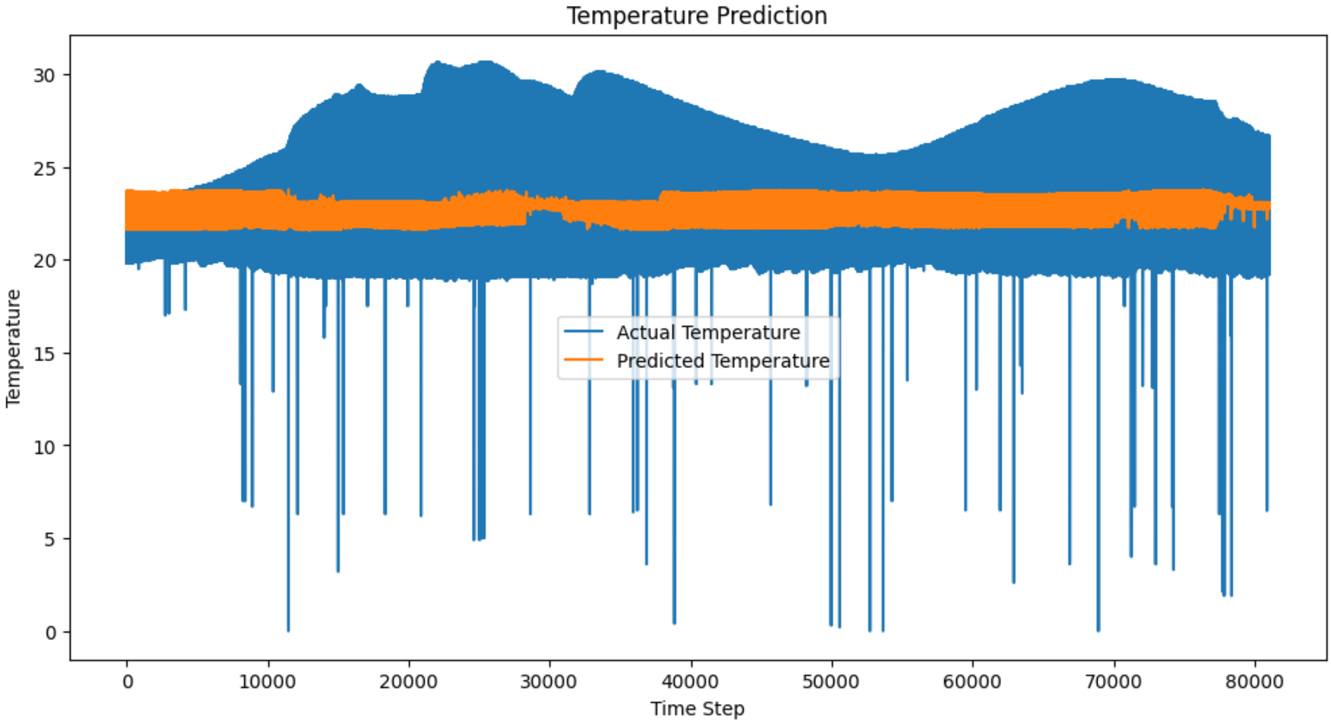
```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epoc
```

```
predictions = trainer.predict(model, test_loader)
predictions = torch.cat(predictions).cpu().numpy()  # Concatenate and move to (

# Create a new scaler for the target variable (temperature) only
temp_scaler = MinMaxScaler()
temp_scaler.min_, temp_scaler.scale_ = scaler.min_[-1], scaler.scale_[-1] # Ext

# Inverse transform to get actual temperature values
predicted_temps = temp_scaler.inverse_transform(predictions.reshape(-1, 1))  #
predicted_temps = predicted_temps.flatten() # Flatten to 1D

# Get actual temperatures using temp_scaler
actual_temps = temp_scaler.inverse_transform(data[train_size + seq_length:][:,
actual_temps = actual_temps.flatten() # Flatten to 1D

plt.figure(figsize=(12, 6))
```

```
plt.plot(actual_temps, label='Actual Temperature')
plt.plot(predicted_temps, label='Predicted Temperature')
plt.xlabel('Time Step')
plt.ylabel('Temperature')
plt.title('Temperature Prediction')
plt.legend()
plt.show()
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVIC
```

Predicting DataLoader 0: 100% 2533/2533 [00:18<00:00, 135.52it/s]

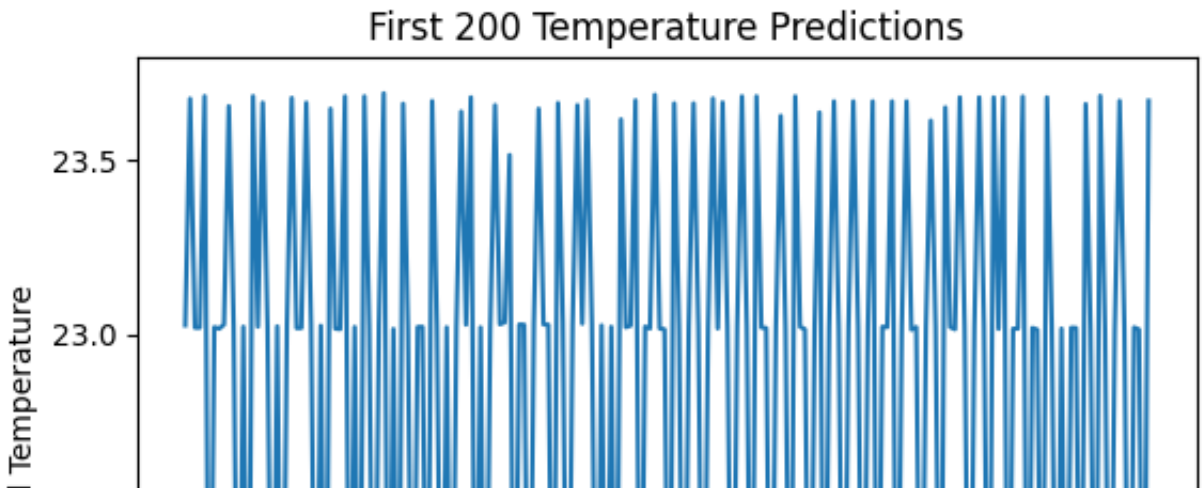


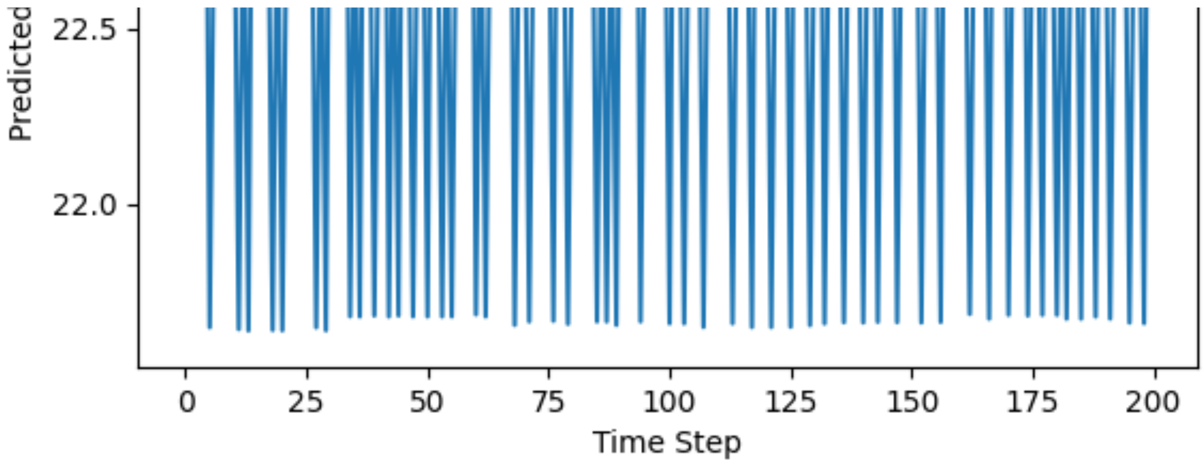
prediction of first 200 datapoints with transformer model

```
first_200_predictions = predicted_temps[:200]
```

```
# Print the predictions
print(first_200_predictions)

#
plt.plot(first_200_predictions)
plt.xlabel('Time Step')
plt.ylabel('Predicted Temperature')
plt.title('First 200 Temperature Predictions')
plt.show()
```

```
[23.02618  23.67337  23.019495 23.019495 23.680546 21.647133 23.020912
 23.016636 23.028347 23.65195  23.023289 21.641872 23.02216  21.63799
 23.680546 23.022314 23.661905 23.016926 21.639454 23.023155 21.638449
 23.023014 23.675638 23.01822  23.018644 23.661905 23.024147 21.646862
 23.02499  21.638578 23.64464  23.016897 23.015213 23.6801   21.678415
 23.021376 21.677847 23.6801   23.015213 21.680706 23.024576 23.688688
 21.677847 23.015654 21.680706 23.658907 23.013336 21.678108 23.0212
 23.021654 21.677979 23.66677  23.01826  21.677979 23.020494 21.677979
 23.01885  23.636597 23.027903 23.677408 21.6839   23.020325 21.677979
 23.016764 23.654821 23.029325 23.034904 23.512604 21.653843 23.028448
 23.027733 21.66374  23.028805 23.644669 23.028795 23.027784 21.665636
 23.660997 23.035519 21.656979 23.026197 23.654821 23.030941 23.668678
 23.028255 21.66374  23.025942 21.66374  23.021843 21.653843 23.614353
 23.020866 23.025476 23.668678 21.663836 23.022234 23.017302 23.684635
 23.017302 23.014622 21.658346 23.659939 23.012934 21.658346 23.020626
 23.659939 23.018879 21.64795  23.019575 23.674072 23.017237 23.662722
 23.018406 21.658346 23.019585 23.680174 23.02016  21.64769  23.680174
 23.02004  23.018316 21.64769  23.0193   23.624369 23.015448 21.64866
 23.680174 23.022305 23.0147   21.653185 23.021265 23.6341   21.658075
 23.016499 23.66586  23.013111 21.661497 23.019644 23.66586  23.018764
 21.66069  23.01859  23.66586  21.66257  23.022026 23.020847 23.66586
 21.661766 23.014864 23.66586  23.013794 23.020964 21.66069  23.019775
 23.610785 23.019775 21.662308 23.648436 23.021547 23.01492  23.677273
 23.018696 21.685163 23.017498 23.677273 23.019266 21.67156  23.677273
 23.015955 23.677273 21.682745 23.016317 23.016857 23.6796   21.680656
 23.018675 23.01358  21.682745 23.677273 23.019756 21.682745 23.016817
 21.67156  23.019165 23.018635 21.67156  23.658016 23.012426 21.6779
 23.681486 23.010283 21.671831 23.010227 23.667332 23.014112 21.660446
 23.020092 23.014112 21.65879  23.667332]
```





Detect Anomaly in temp prediction

Anomalies are detected by computing the average and standard deviation of the sensor data

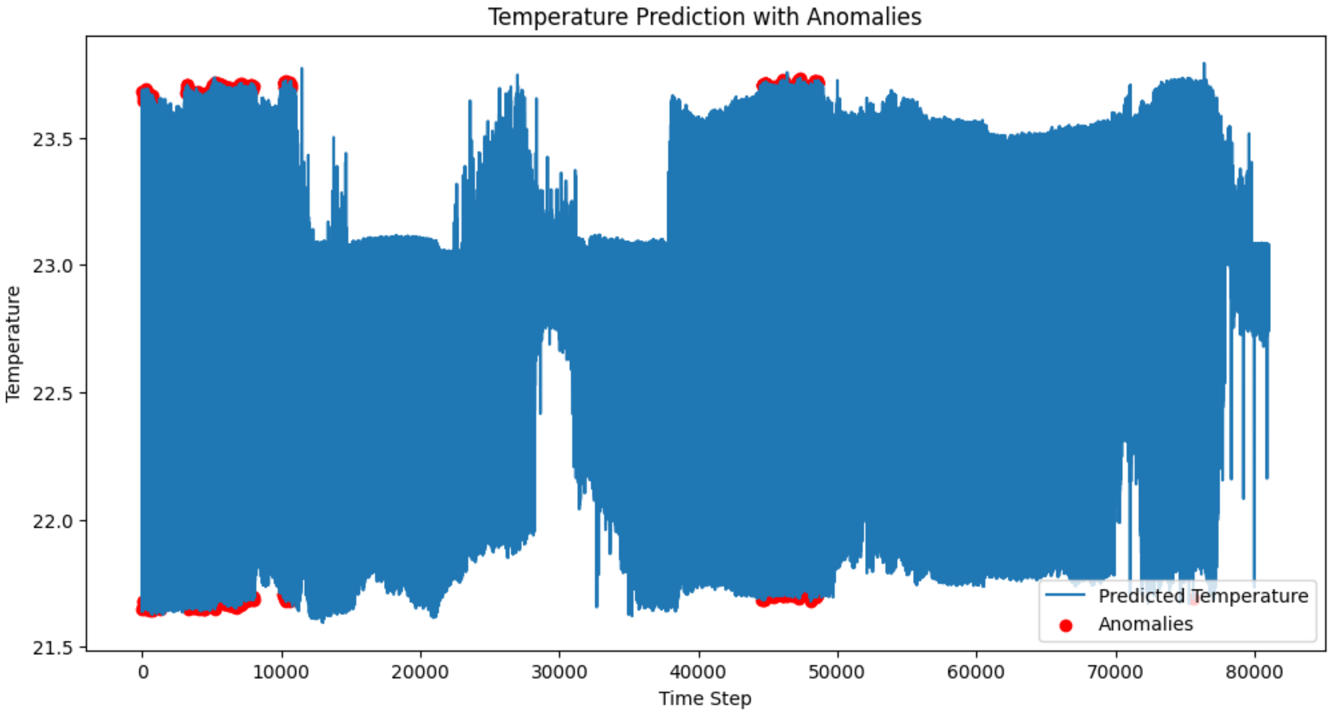
```
import numpy as np

threshold = 2  # Example threshold. Adjust this value based on your data.

# Calculate the absolute difference between consecutive predictions
differences = np.abs(np.diff(predicted_temps))

# Find indices where the difference exceeds the threshold
anomaly_indices = np.where(differences > threshold)[0]

# Print the indices of the anomalies
print("Anomaly indices:", anomaly_indices)

# You can also print the actual values that are considered anomalies:
print("Anomalous temperature values:", predicted_temps[anomaly_indices + 1]) #

# Plot the data with anomalies highlighted
plt.figure(figsize=(12, 6))
plt.plot(predicted_temps, label='Predicted Temperature')
plt.scatter(anomaly_indices + 1, predicted_temps[anomaly_indices + 1], color='r
plt.xlabel('Time Step')
plt.ylabel('Temperature')
plt.title('Temperature Prediction with Anomalies')
plt.legend()
plt.show()
```

```
Anomaly indices: [     4     13     29     33     36     41     93    100    117     1
   166    188    198    202    224    232    241    245    250    257    263    267
   273    281    285    288    292    296    299    307    316    319    327    341
   359    372    386    392    393    398    408    421    429    432    436    439
   480    499    530    537    548    556    564    570    586    595    662    736
   743    747    756    759    774   1356   3108   3127   3132   3141   3153   3177
  3182   3188   3206   3210   3215   3222   3228   3231   3235   3240   3247   3258
  3270   3274   3294   3372   3428   3436   3519   3574   3590   3613   3624   3635
  3684   3691   3705   3709   3729   3733   3745   3750   3756   3899   3909   3921
  3944   3970   3973   3976   3993   3999   4041   4069   4118   4126   4135   4142
  4145   4150   4155   4163   4168   4172   4177   4184   4187   4213   4362   4383
  4420   4423   4429   4481   4497   4509   4557   4602   4605   4613   4680   4816
  4827   4930   4934   4938   4941   4951   4982   4985   5000   5007   5011   5019
  5031   5036   5044   5049   5053   5061   5068   5085   5088   5097   5105   5112
  5124   5131   5135   5139   5142   5151   5160   5163   5167   5170   5176   5179
  5182   5195   5200   5211   5214   5219   5228   5233   5256   5261   5273   5277
  5284   5291   5294   5298   5307   5311   5314   5318   5323   5328   5337   5340
  5351   5354   5361   5365   5391   5401   5409   5412   5417   5426   5430   5448
  5471   5480   5483   5489   5494   5499   5521   5536   5539   5547   5551   5557
  5563   5566   5570   5577   5590   5598   5605   5615   5619   5622   5626   5639
  5650   5653   5658   5669   5672   5676   5683   5686   5690   5695   5700   5707
  5712   5715   5719   5724   5727   5731   5737   5740   5745   5752   5757   5761
  5774   5781   5790   5802   5805   5809   5812   5819   5834   5842   5845   5859
  5863   5870   5878   5882   5891   5901   5911   5915   5927   5934   5938   5953
  5957   5966   5982   6007   6010   6019   6024   6028   6033   6053   6058   6071
  6078   6081   6090   6093   6103   6111   6117   6120   6131   6140   6143   6152
  6159   6162   6169   6174   6177   6192   6208   6215   6219   6222   6228   6232
  6235   6248   6251   6260   6268   6277   6282   6291   6313   6320   6324   6327
  6336   6340   6343   6346   6350   6357   6372   6375   6379   6386   6391   6397
  6409   6424   6429   6433   6436   6441   6445   6453   6457   6467   6470   6479
  6487   6497   6507   6513   6516   6519   6522   6525   6529   6532   6540   6544
  6554   6558   6561   6572   6577   6581   6584   6591   6600   6604   6608   6620
  6628   6632   6645   6653   6656   6661   6668   6671   6677   6691   6694   6698
  6704   6723   6730   6738   6752   6757   6763   6769   6777   6785   6792   6796
  6799   6815   6819   6830   6834   6840   6843   6860   6870   6873   6876   6879
  6883   6890   6897   6910   6920   6923   6933   6938   6945   6951   6956   6963
  6967   6974   6987   6991   7004   7014   7019   7027   7035   7039   7042   7049
  7059   7062   7069   7080   7083   7089   7094   7097   7103   7112   7117   7123
  7138   7145   7151   7159   7166   7169   7174   7183   7193   7199   7202   7209
  7219   7225   7238   7273   7279   7282   7287   7291   7306   7312   7331   7334
  7366   7375   7384   7395   7402   7422   7427   7434   7438   7453   7457   7460
  7463   7473   7477   7493   7503   7524   7535   7541   7546   7562   7571   7587
  7593   7618   7625   7638   7644   7696   7748   7752   7755   7768   7774   7779
  7782   7799   7803   7806   7814   7819   7823   7826   7833   7838   7850   7860
  7867   7874   7879   7882   7897   7903   7907   7914   7919   7928   7950   7985
  7997   8050   8060  10150  10172  10181  10188  10193  10198  10208  10211  10222
 10230  10235  10239  10242  10251  10260  10268  10275  10280  10283  10287  10290
 10295  10298  10305  10310  10317  10324  10330  10335  10339  10347  10355  10362
 10366  10369  10373  10378  10389  10392  10395  10482  10506  10511  10514  10532
 10537  10540  10566  10573  10600  10613  10620  10627  10632  10642  10647  10651
 10671  10674  10681  10696  10701  10706  10713  10758  44561  44583  44603  44612
 44619  44622  44626  44629  44635  44643  44658  44665  44669  44685  44702  44737
 44746  44750  44754  44766  44769  44776  44779  44784  44791  44798  44802  44806
```

```
 44827 44843 44868 44982 45253 45358 45361 45380 45561 45565 45866 45871
 45892 45928 45934 45946 45955 45960 45966 45971 45977 45981 46020 46027
 46032 46038 46063 46070 46081 46086 46091 46098 46102 46106 46110 46113
 46125 46130 46133 46137 46338 46341 46347 46783 46800 47072 47097 47104
 47108 47122 47130 47148 47153 47156 47166 47169 47173 47176 47180 47191
 47198 47204 47209 47212 47218 47229 47243 47246 47252 47258 47265 47271
 47274 47277 47301 47310 47313 47316 47319 47323 47326 47333 47336 48059
 48068 48076 48080 48087 48093 48098 48106 48109 48121 48130 48134 48137
 48144 48355 48363 48370 48377 48396 48404 48407 48410 48415 48433 48442
 48446 48449 48455 48458 48463 48466 48470 48473 48478 48485 48489 48493
 48496 48508 48515 48518 48523 48529 48532 48537 48579 48584 48588 48593
 48598 48601 48651 75587]
Anomalous temperature values: [21.647133 23.680546 23.64464  21.678415 23.6
 23.659939 23.680174 23.680174 21.66257  21.661766 23.677273 23.681486
 23.667332 21.648792 23.689352 21.677721 21.67159  23.685402 21.67624
 23.678623 23.678623 21.66149  21.661291 23.673706 21.666851 21.676214
 23.684835 21.674614 21.674614 21.674738 21.661945 21.67138  23.665878
 21.659138 21.646687 23.658176 21.66498  23.68215  21.65187  21.65187
 21.65187  23.669395 23.663525 23.657213 23.657213 23.663525 23.671202
 23.665962 23.666288 21.661259 23.668259 21.650486 21.639635 21.65326
 21.652796 23.647205 21.660059 21.640833 23.665627 21.653667 23.655537
 23.648317 23.632654 21.648623 21.669321 21.667652 21.667652 21.661266
 23.676294 21.661894 23.697344 21.663013 21.685371 21.68617  23.690979
 21.684723 21.68145  23.703926 23.703926 23.703926 21.670818 21.68137
 23.688606 21.655987 21.649652 23.675442 23.679955 21.676004 23.675154
 23.674002 23.659603 23.648165 23.681265 23.671177 23.669952 21.650108
 23.66351  23.666828 21.6678   21.672709 21.652508 23.671864 21.652472
 21.673595 23.658886 21.65728  23.667334 23.681442 23.672327 23.66703
 21.666399 21.6551   23.67078  23.665464 21.655134 21.645205 23.65726
 21.65248  23.66911  21.65248  23.65726  21.656116 23.669876 21.668541
 23.669876 23.657875 23.657875 23.658932 21.65959  21.662241 23.666172
 21.649944 23.653265 21.646162 21.646055 23.6535   21.661118 23.653534
 21.65341  23.674778 23.658428 23.665623 21.661158 23.676561 23.671297
 21.662529 21.662529 23.676153 23.684187 23.688646 21.66336  21.661978
 21.662937 21.662937 23.700245 21.664495 21.664587 23.705935 21.675785
 21.66883  21.674398 21.66757  21.659842 23.683811 23.679226 23.700748
 23.708138 21.67124  21.6818   21.67124  21.684425 21.679815 21.679815
 21.689257 23.701586 23.701586 23.707243 23.707243 23.70885  21.67337
 21.67115  21.67115  21.67115  23.712288 23.716007 21.649666 23.70806
 23.713734 21.659266 21.659266 21.682146 21.687227 21.684755 23.704601
 21.679588 23.711868 23.709417 21.6929   21.693027 21.70623  21.6929
 23.704243 23.694654 23.694654 23.691021 23.700014 21.68236  21.687225
 23.712015 21.674986 23.707142 21.676168 21.676168 23.695705 23.69501
 23.698156 23.70964  23.702845 21.684141 23.703472 23.699379 21.686956
 21.68829  23.705046 21.688206 21.686747 21.68829  21.68829  21.679642
 23.704868 21.681068 21.678629 23.703289 23.705963 23.695312 23.70021
 21.68466  21.69092  23.703125 23.691437 23.697884 21.67893  23.703976
 21.680067 23.70405  23.702526 23.702526 23.69803  21.688292 23.704943
 21.688139 23.69803  21.688139 23.69803  21.686684 21.686272 23.691456
 21.686684 23.697573 23.693571 23.70058  21.680027 23.70228  23.699291
 23.699291 21.680035 21.681305 23.700994 21.681038 21.681139 21.683798
 23.701178 23.69961  23.702608 23.699675 23.698452 23.699675 21.682707
 21.670609 21.670807 23.697344 21.670893 23.694702 23.69592  21.69521
 21.682602 21.682257 21.68198  23.696848 23.691265 21.689041 21.671188
```

```
23.691734 21.671188 23.694883 23.6947   23.690517 23.69294  21.672552
21.665962 23.694496 23.696264 21.665962 21.677206 23.692835 23.69077
21.678148 21.678804 21.690216 21.688074 21.690062 23.69293  23.691408
23.691408 23.692577 21.66231  23.692312 23.692312 23.693604 23.692312
21.6806   23.693886 23.68793  23.693192 23.692076 23.687643 23.69081
23.686872 21.662603 21.662775 23.69393  23.692331 23.692331 23.692331
23.691277 23.691277 21.670929 21.670555 21.670929 21.670929 21.67096
21.670416 21.670872 21.67096  23.687704 21.67096  23.6911   23.689178
23.689178 21.668407 21.6668   23.674986 21.6668   21.6668   23.682941
23.686169 21.667294 21.66814  21.667189 21.673843 23.689413 21.673685
21.673534 21.673283 23.68756  23.67607  21.675608 21.675608 21.675608
23.685936 21.674784 21.675486 21.671404 23.684872 21.670977 23.687023
23.687023 21.658272 23.672487 21.658548 23.668697 23.679434 21.671051
23.677092 21.665524 23.680218 21.674805 21.668756 23.684961 23.683504
23.681692 23.686392 23.686392 21.66748  21.667574 23.692186 23.692186
21.651508 23.681053 21.663773 21.663773 21.663773 21.664675 21.660671
21.660671 21.66047  23.681479 21.672655 21.673637 23.691656 23.693634
23.692938 23.689589 21.678736 23.69371  23.69371  23.69371  23.691595
21.677156 23.693787 21.676977 21.677156 23.698954 23.698954 23.701532
23.695673 21.67896  21.675829 23.699547 23.700205 23.699547 21.68327
23.703924 21.671997 23.703924 21.663456 23.703445 21.663473 21.663473
21.669062 21.668816 21.669025 23.702774 23.702774 23.709936 23.70867
23.70403  21.670225 23.696993 21.667614 23.703304 21.667711 21.668201
21.667446 23.712463 23.697802 21.677307 21.677547 21.677547 21.68006
23.704777 23.698519 23.697256 21.683588 21.683588 21.683588 23.698067
23.698067 21.696358 23.699091 23.69722  23.695772 21.687275 21.688297
21.694437 21.69392  21.69443  21.694363 23.693674 23.693674 21.69245
21.68402  23.695042 23.695042 21.680618 21.680618 23.693933 23.695686
23.700397 23.699783 21.691206 23.698471 21.690598 21.690804 21.69108
23.691956 23.689713 21.682373 21.682613 23.693438 23.693438 23.69151
21.688753 21.683872 21.683996 21.683996 23.693491 21.692305 21.691109
21.690655 21.691109 23.692453 23.695831 21.697168 21.696106 23.701275
23.700619 23.700619 23.68981  21.687838 21.688332 21.688776 23.708933
21.689228 21.68456  23.705656 23.689785 21.688585 23.696686 23.696686
21.688772 21.687246 21.687246 23.69958  21.686918 23.68761  21.699484
21.683117 21.683092 23.6994   21.693495 21.702965 21.703764 21.708698
23.698309 23.713312 21.699389 23.697556 23.705902 23.714823 23.716297
23.716297 21.692284 21.691792 23.712685 23.714607 23.716434 23.714298
21.699255 23.717592 23.717592 23.719107 23.710775 23.719107 23.714434
23.712536 23.708338 23.723774 21.688398 23.709919 23.709919 23.708136
21.680048 23.703726 23.702879 23.706615 21.684618 23.706606 23.704346
23.704346 23.701204 23.699287 23.695236 21.690264 21.69195  23.704168
23.704168 23.704168 23.69818  23.69818  23.69757  23.71442  21.687103
21.680155 23.707466 21.681757 23.700867 23.697008 21.68374  21.689844
23.69425  21.71245  23.707693 21.691479 21.693226 21.689394 21.68647
23.70261  21.696892 23.70805  21.685604 21.69064  23.708317 23.70805
23.704683 21.690475 23.707869 23.707869 21.690475 23.705988 23.707294
21.686066 23.711489 21.686066 23.712765 21.688875 21.700926 23.708603
23.709618 23.711454 23.712227 23.712227 23.710468 21.69904  23.703096
23.711859 23.713491 23.701477 21.701002 21.701159 23.701061 21.696096
21.701225 23.700039 23.708162 21.7077   21.70878  23.712751 23.715595
23.713507 21.704138 21.704138 21.704138 23.710232 21.703136 21.703136
21.70015  21.70015  23.724455 21.701094 23.716507 21.715239 23.721302
21.709135 23.711077 23.716576 21.709135 23.719309 23.723373 23.725143
```

```
 23.719755 21.708652 23.714933 21.707628 21.700972 21.70191  23.706608
 21.698088 23.694439 21.698578 21.707726 23.714437 23.714437 21.705206
 21.708578 21.695276 23.717463 23.717463 23.717463 23.722301 21.707552
 21.707552 21.710447 21.705091 21.710447 23.723225 23.721369 23.721369
 21.694496 21.698774 23.72862  23.72862  21.6988   21.688797 21.69644
 21.716236 23.710808 23.711876 23.729717 23.731638 21.717068 21.711046
 21.708721 23.729717 21.720541 23.71835  23.71835  21.689465 21.68877
 21.681782 21.678238 21.68877  23.698427 21.698944 21.701773 21.701773
 23.70806  23.693628 23.693628 23.693628 21.686691 21.70657  21.696053
 21.695057 21.694925 23.722197 21.713594 23.719782 23.722197 21.706701
 23.712187 21.706701 23.712187 23.710993 21.701777 21.7007   21.701618
 21.701618 21.701618 23.71716  21.701618 23.70659  21.697866 23.708038
 23.709599 21.696651 23.713184 23.714418 23.711884 21.70526  23.716082
 21.709988 23.712067 23.715147 21.705578 21.714464 23.715147 23.721079
 21.721333 21.690163]
```



Detect CyberAttack Analysis - Linear Regression is used for training and predicting the attack, pie plot for attack details and box plots to know the statistical information

```
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from keras.models import Sequential
from keras.layers import Dense
import keras.activations,keras.optimizers,keras.losses
```

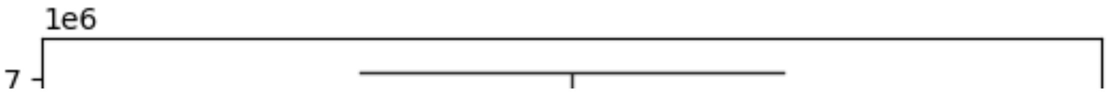
```
/usr/local/lib/python3.11/dist-packages/dask/dataframe/__init__.py:42: Futu
Dask dataframe query planning is disabled because dask-expr is not installe

You can install it with `pip install dask[dataframe]` or `conda install das
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
```

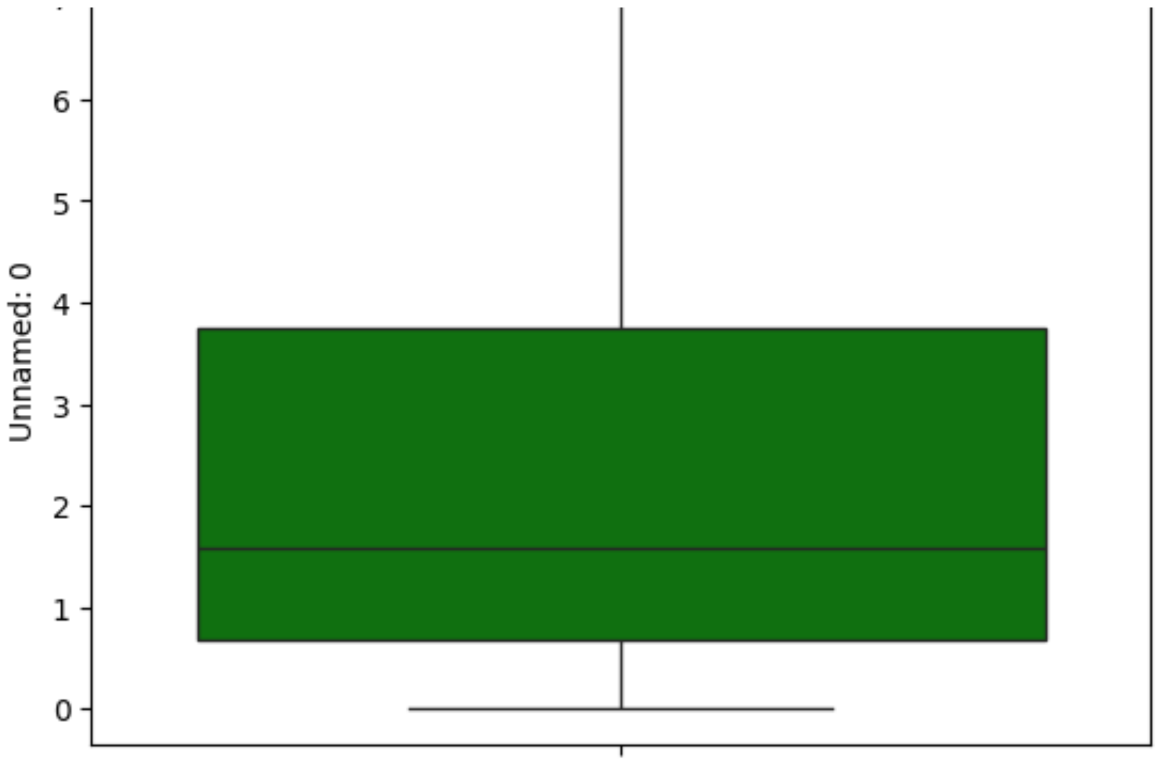
```
import seaborn as sn

from google.colab import drive
drive.mount('/content/drive')

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import autocorrelation_plot
```

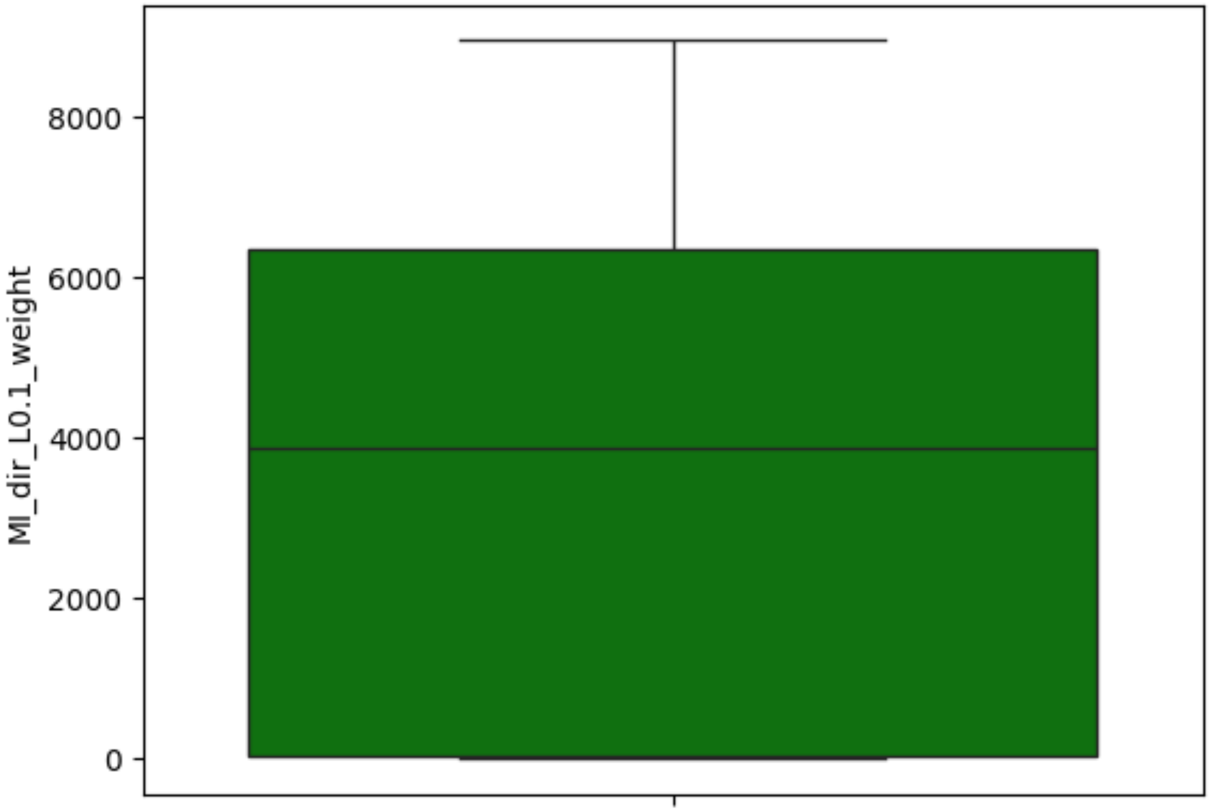
```
import os

#
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/AAI-530/BotNeTIoT-L
print(data.head())
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
   Unnamed: 0  MI_dir_L0.1_weight  MI_dir_L0.1_mean  MI_dir_L0.1_variance
0           0            1.000000         98.000000          0.000000e+00
1           1            1.931640         98.000000          1.818989e-12
2           2            2.904273         86.981750          2.311822e+02
3           3            3.902546         83.655268          2.040614e+02
4           4            4.902545         81.685828          1.775746e+02

   H_L0.1_weight  H_L0.1_mean  H_L0.1_variance  HH_L0.1_weight  HH_L0.1_mea
0       1.000000    98.000000     0.000000e+00         1.00000          98.
1       1.931640    98.000000     1.818989e-12         1.93164          98.
2       2.904273    86.981750     2.311822e+02         1.00000          66.
3       3.902546    83.655268     2.040614e+02         1.00000          74.
4       4.902545    81.685828     1.775746e+02         2.00000          74.

    HH_L0.1_std  ...  HH_jit_L0.1_mean  HH_jit_L0.1_variance  \
0  0.000000e+00  ...      1.505914e+09          0.000000e+00
1  1.348699e-06  ...      7.263102e+08          5.662344e+17
2  0.000000e+00  ...      1.505914e+09          0.000000e+00
3  0.000000e+00  ...      1.505914e+09          0.000000e+00
4  9.536743e-07  ...      7.529571e+08          5.669445e+17

   HpHp_L0.1_weight  HpHp_L0.1_mean  HpHp_L0.1_std  HpHp_L0.1_magnitude  \
0           1.00000            98.0       0.000000            98.000000
1           1.93164            98.0       0.000001           138.592929
2           1.00000            66.0       0.000000           114.856432
3           1.00000            74.0       0.000000            74.000000
4           1.00000            74.0       0.000000            74.000000

   HpHp_L0.1_radius  HpHp_L0.1_covariance  HpHp_L0.1_pcc  label
0      0.000000e+00                   0.0            0.0      0
1      1.818989e-12                   0.0            0.0      0
2      0.000000e+00                   0.0            0.0      0
3      0.000000e+00                   0.0            0.0      0
4      0.000000e+00                   0.0            0.0      0

[5 rows x 25 columns]
```

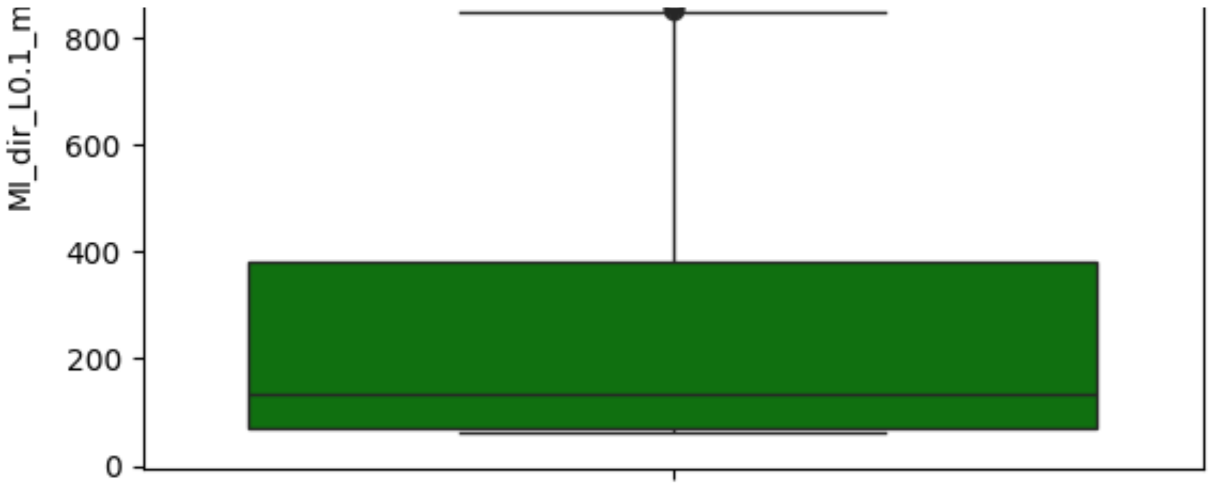
```
import seaborn as sn
for i in data.select_dtypes(include='number').columns.values:
    sn.boxplot(data[i],color='green')
    plt.show()
```

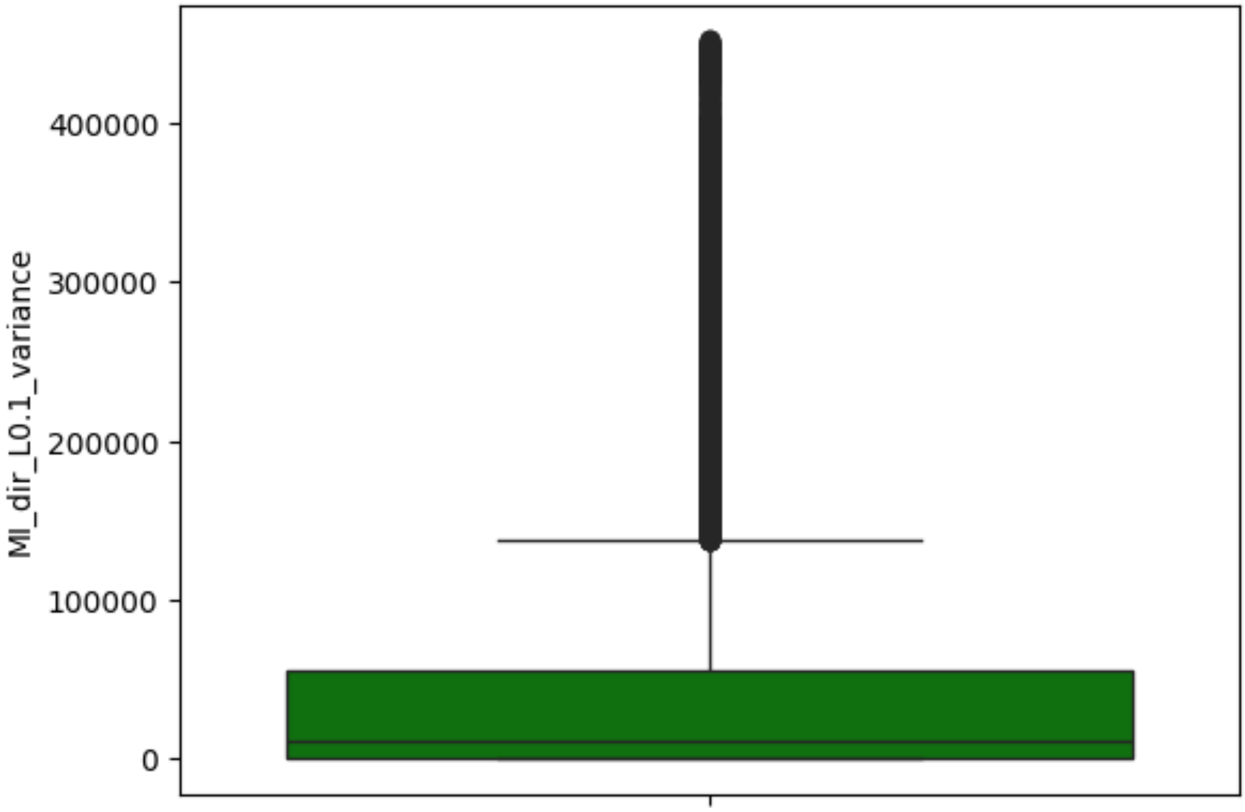


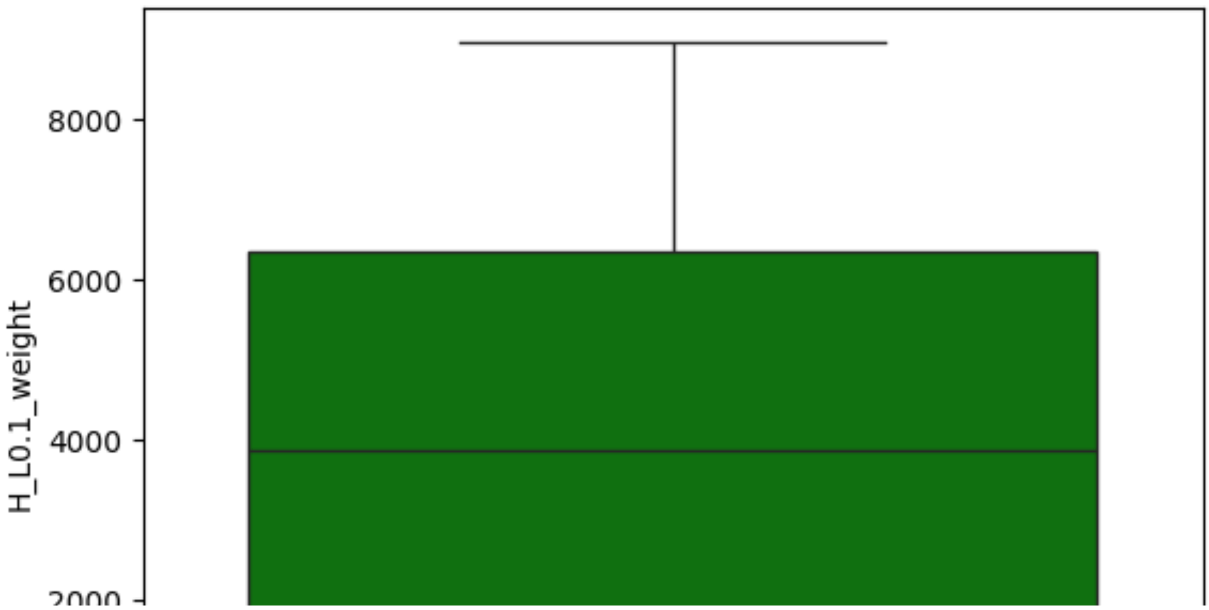

6
5
4
3
2
1
0
Unnamed: 0

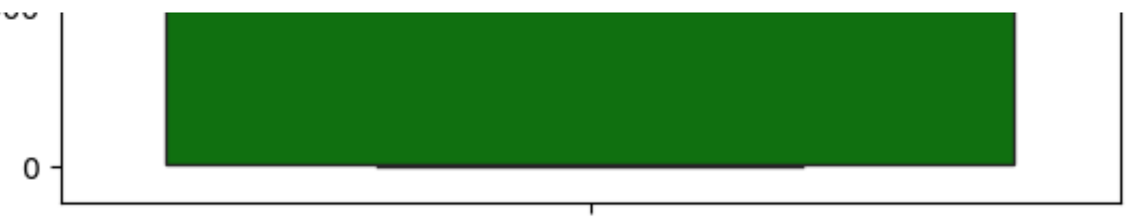

8000
6000
4000
2000
0
MI_dir_L0.1_weight

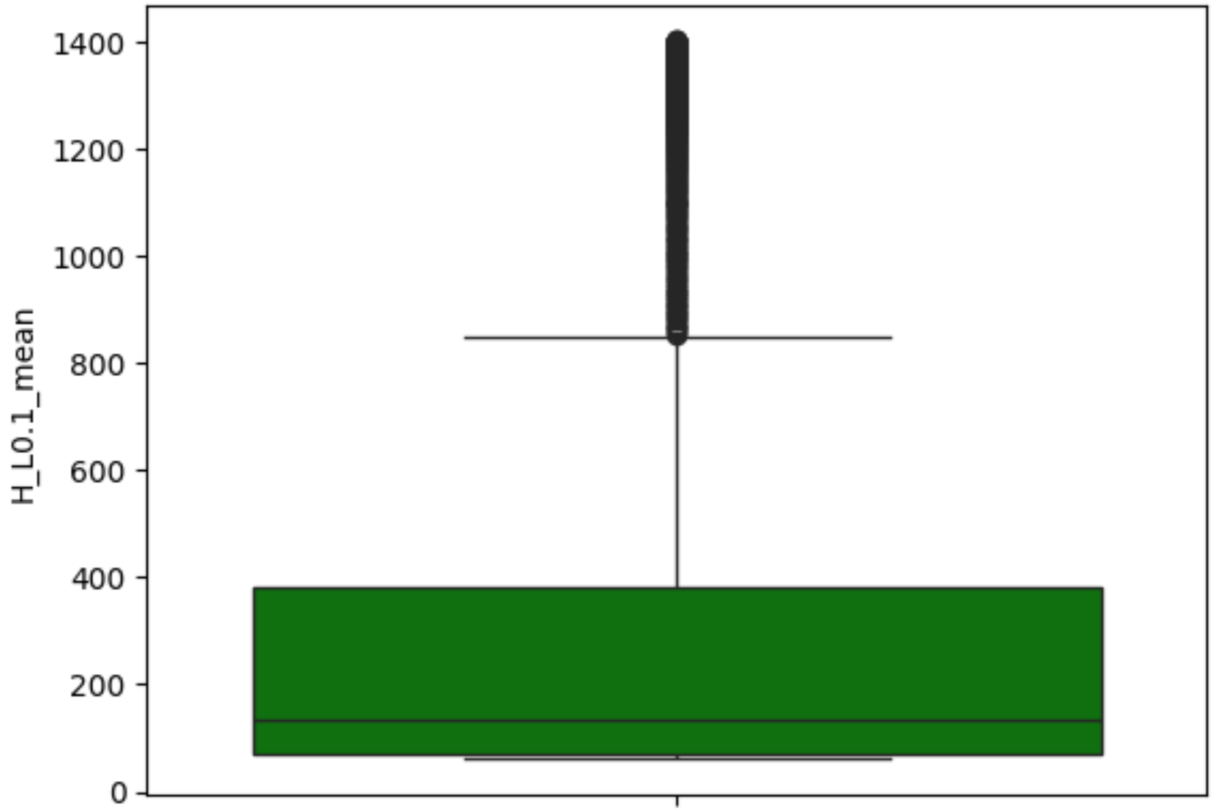

1400
1200
1000

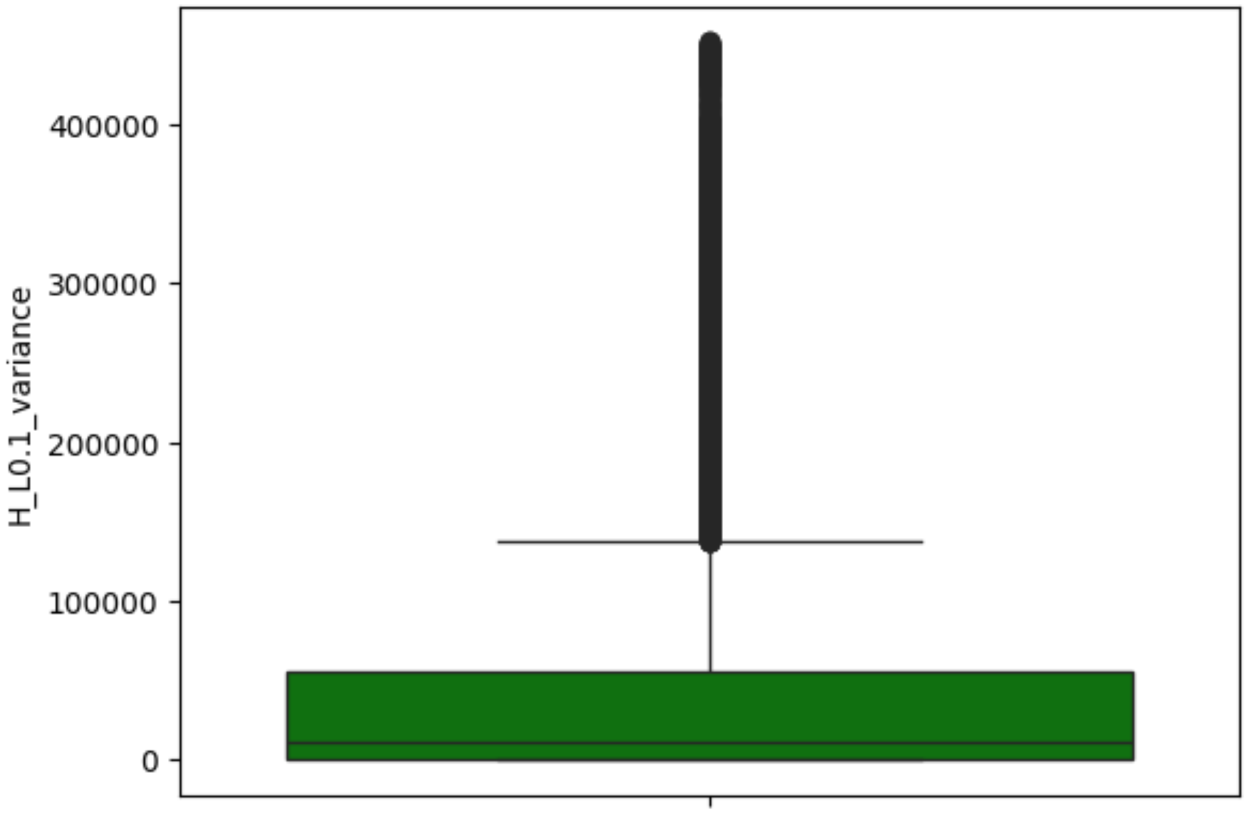

MI_dir_L0.1_m
800
600
400
200
0

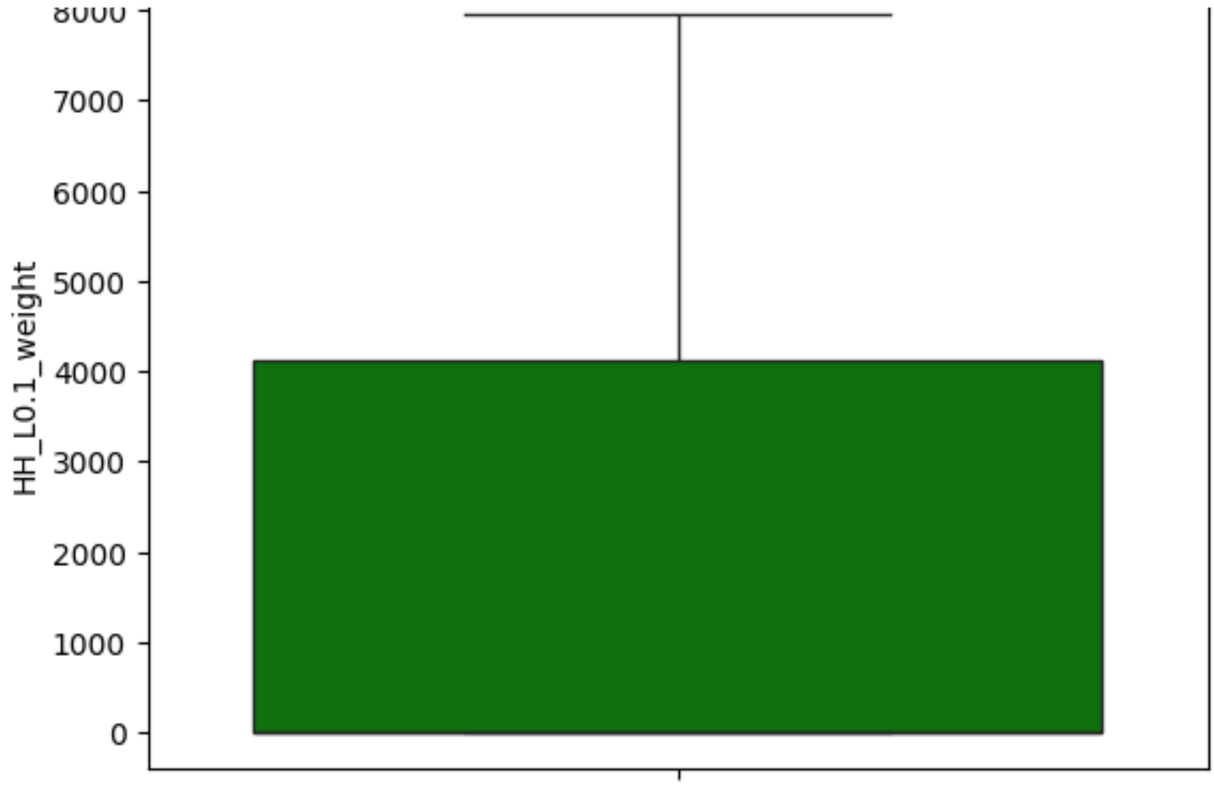

MI_dir_L0.1_variance
400000
300000
200000
100000
0

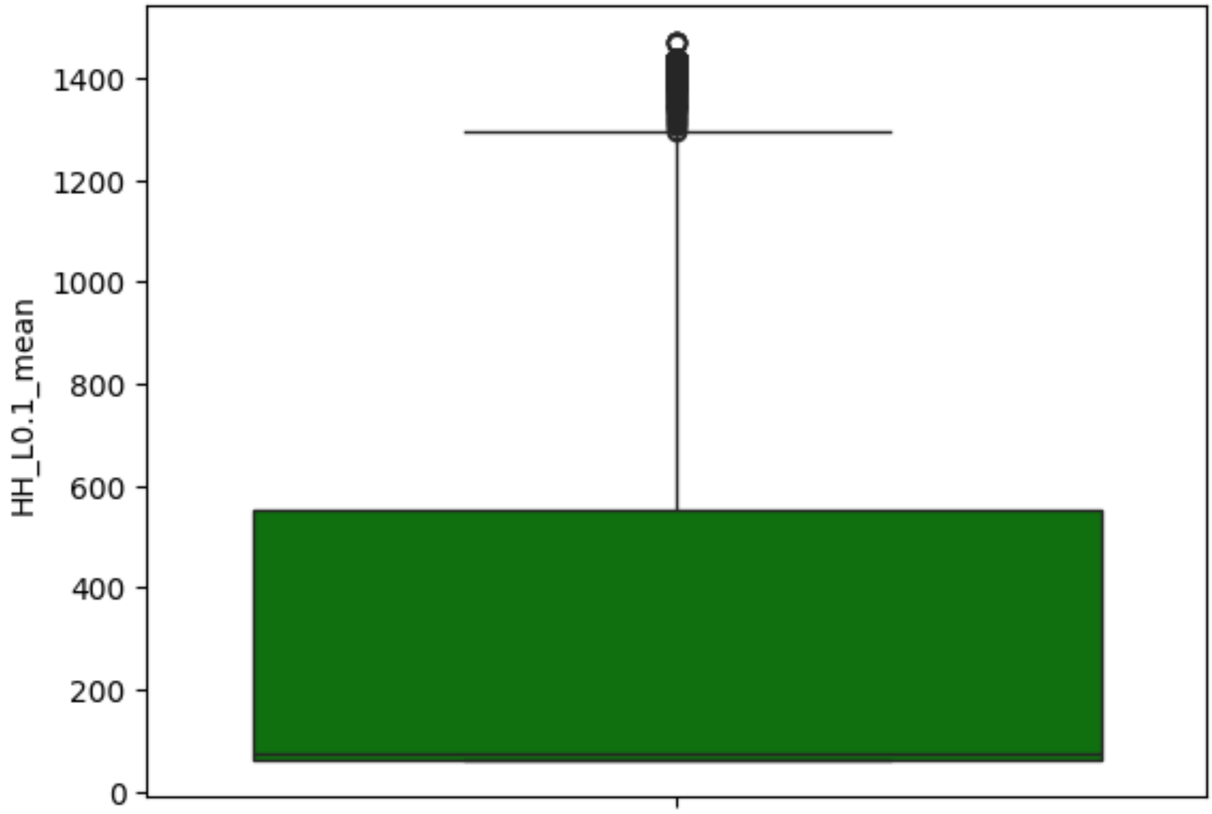

H_L0.1_weight
8000
6000
4000

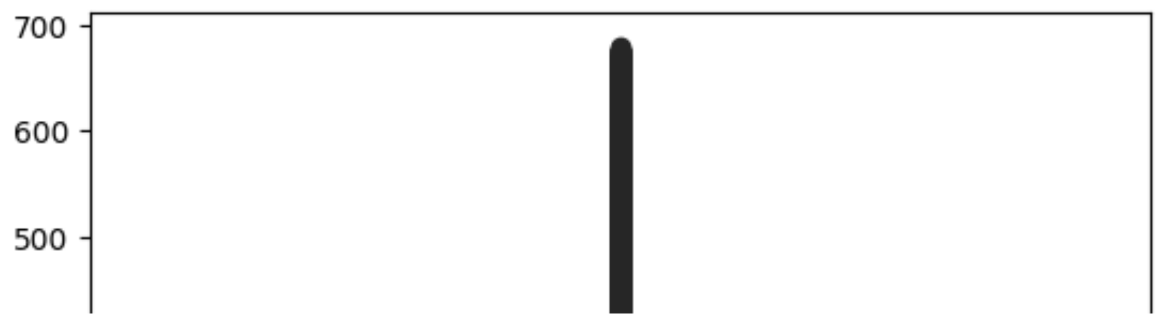

0

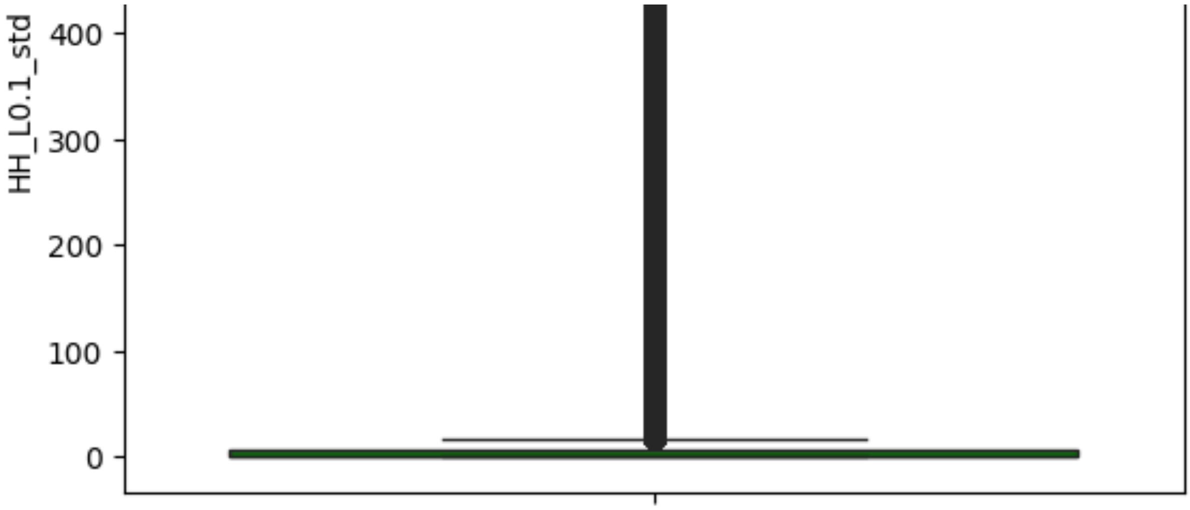

H_L0.1_mean
1400
1200
1000
800
600
400
200
0

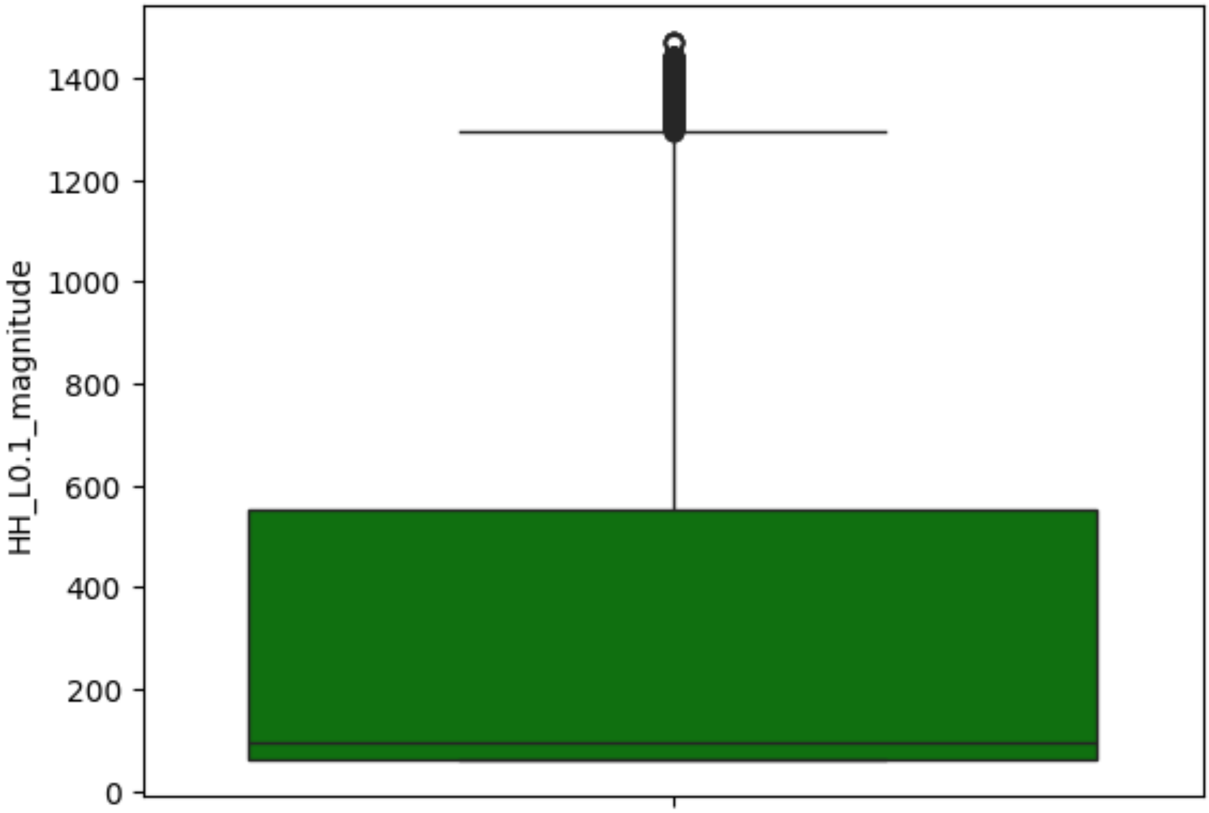

H_L0.1_variance
400000
300000
200000
100000
0

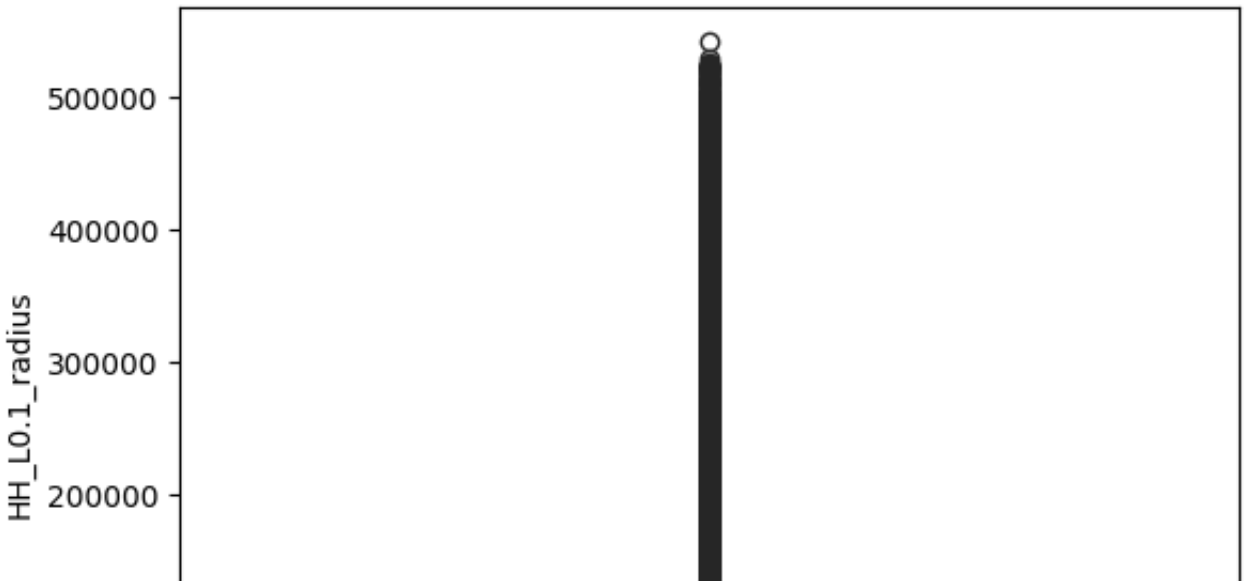

HH_L0.1_weight
8000
7000
6000
5000
4000
3000
2000
1000
0


HH_L0.1_mean
1400
1200
1000
800
600
400
200
0

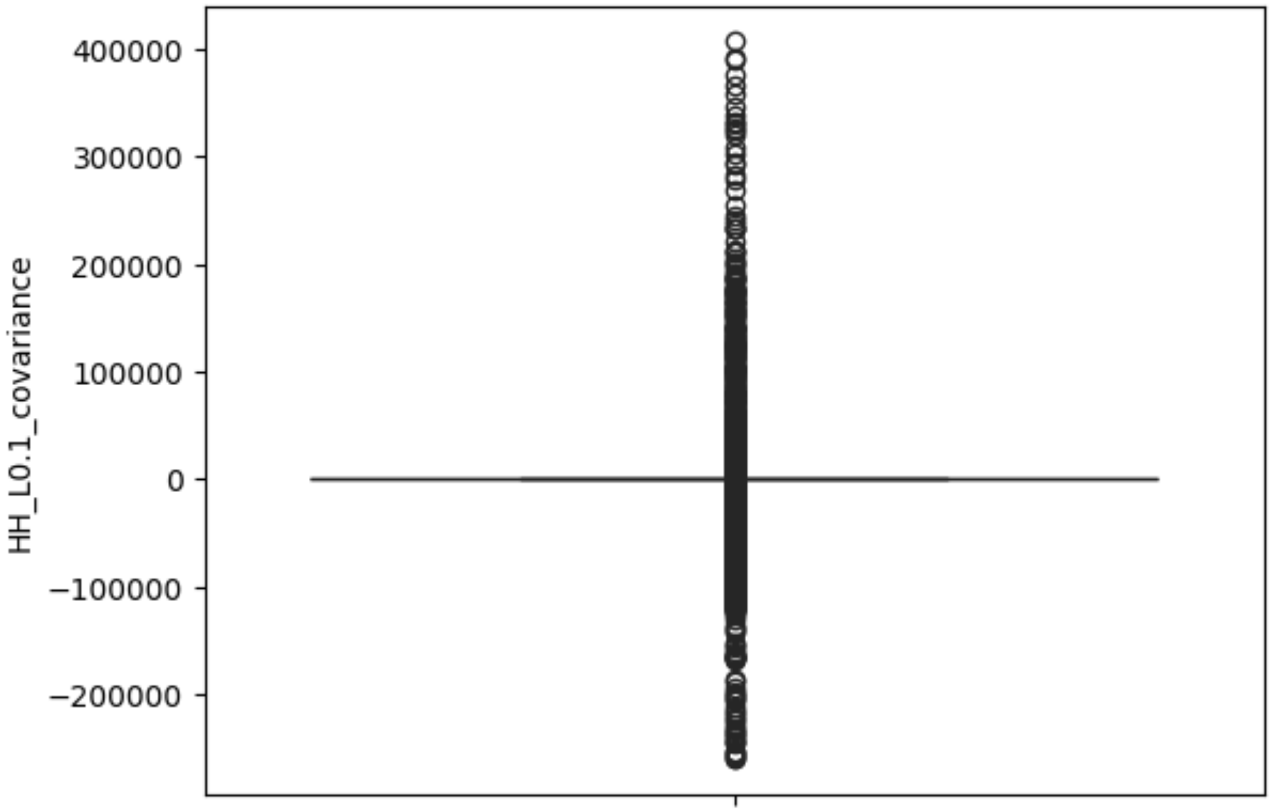

700
600
500

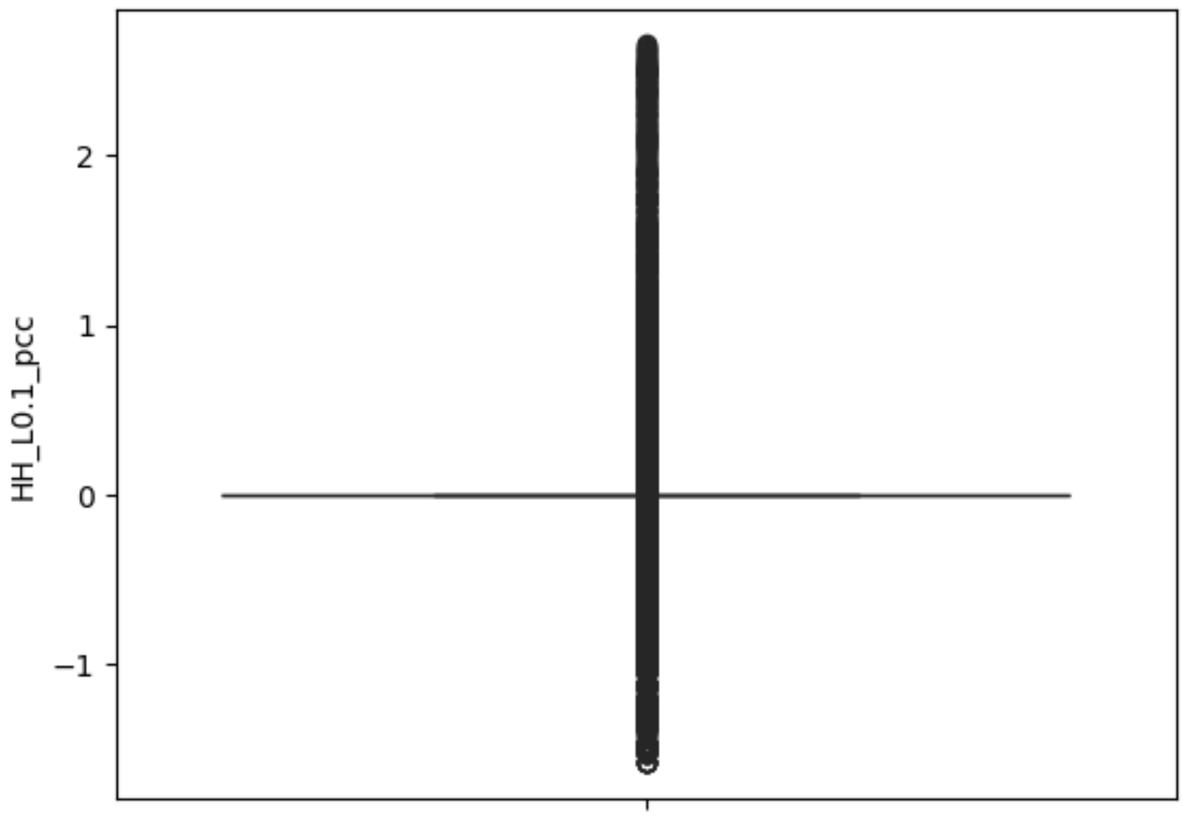

HH_L0.1_std
400
300
200
100
0

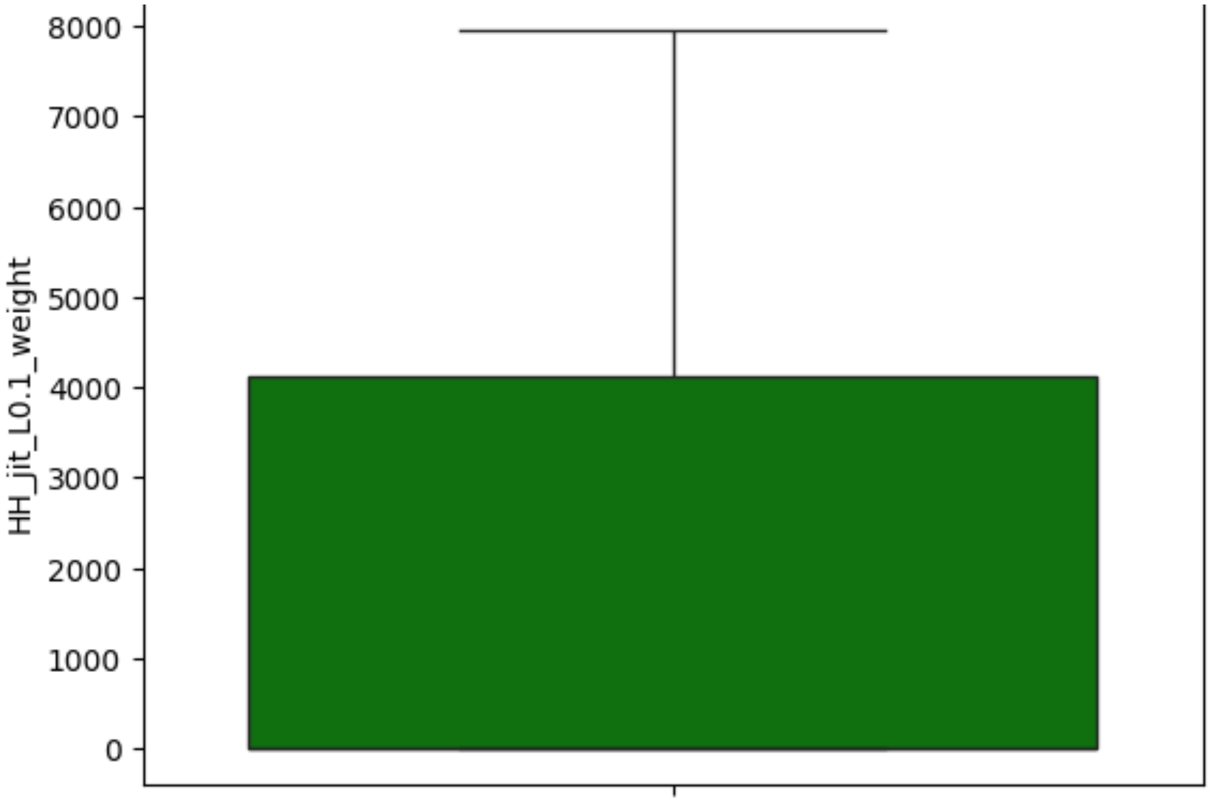

HH_L0.1_magnitude
1400
1200
1000
800
600
400
200
0

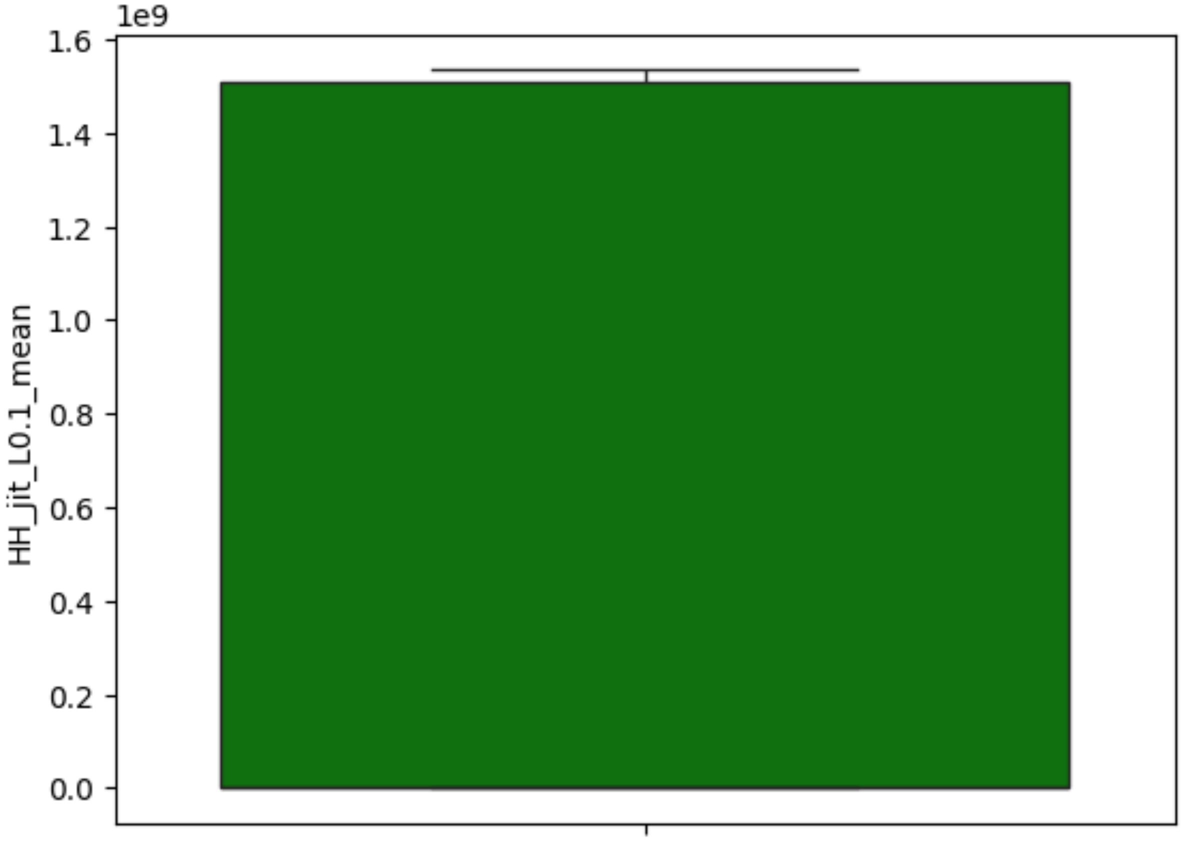

HH_L0.1_radius
500000
400000
300000
200000


100000
0

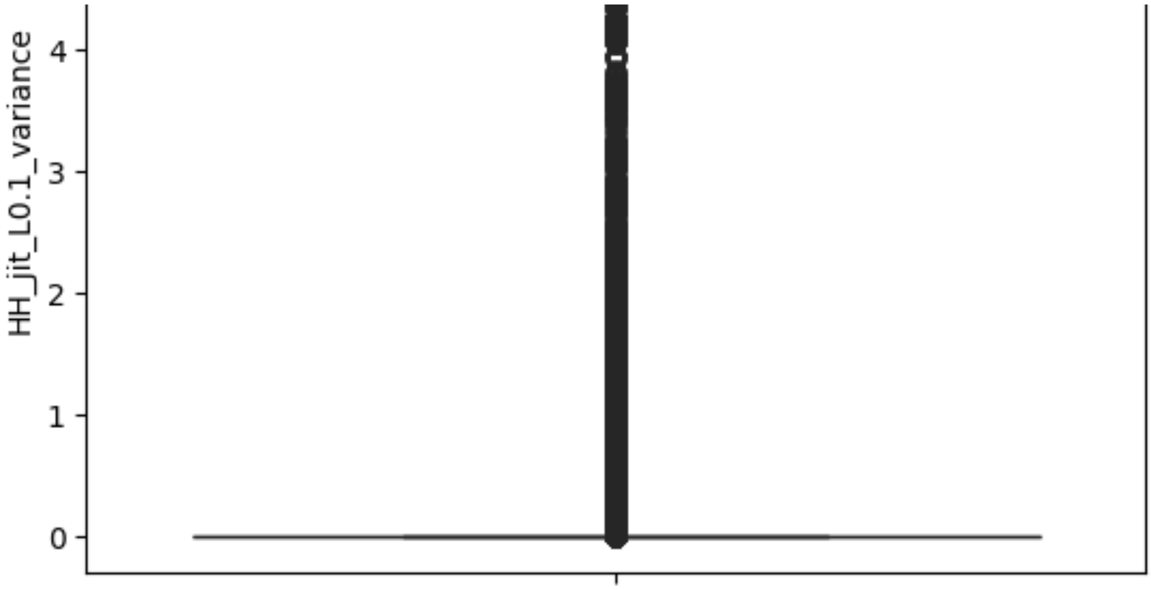

400000
300000
200000
100000
0
−100000
−200000
HH_L0.1_covariance

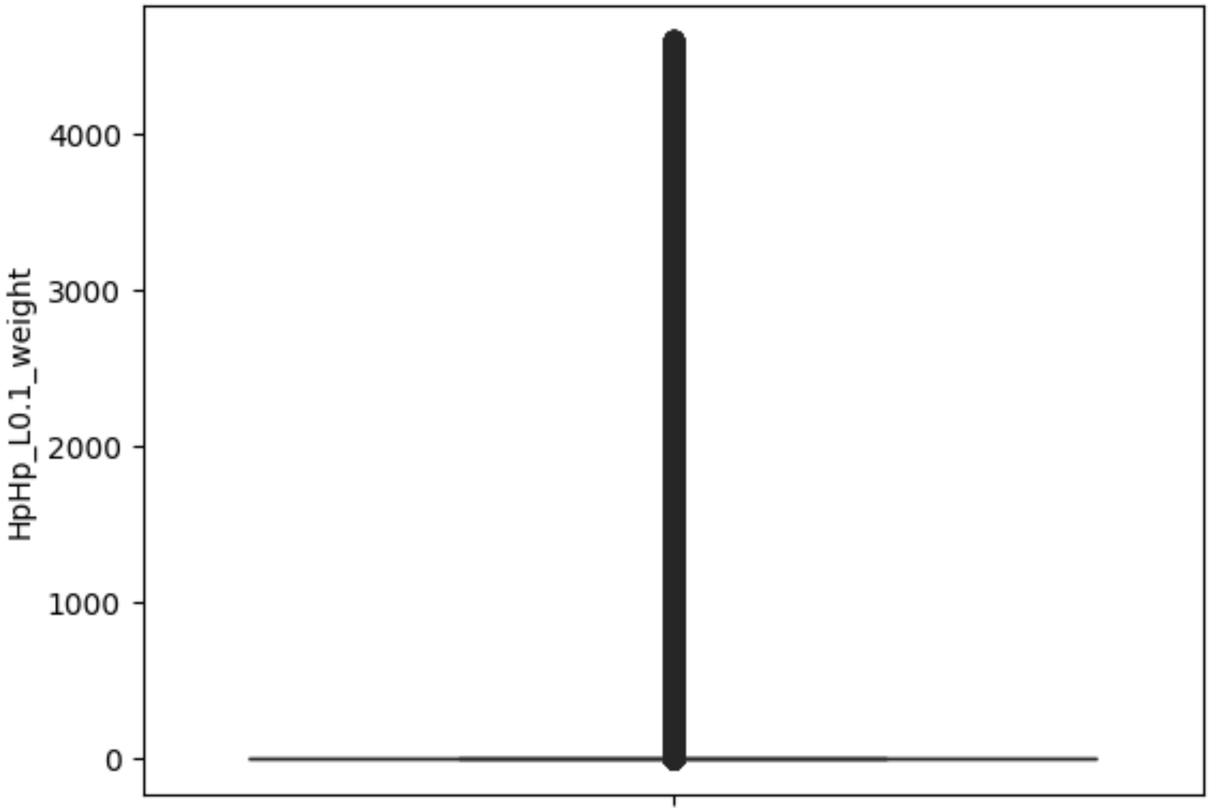

2
1
0
−1
HH_L0.1_pcc

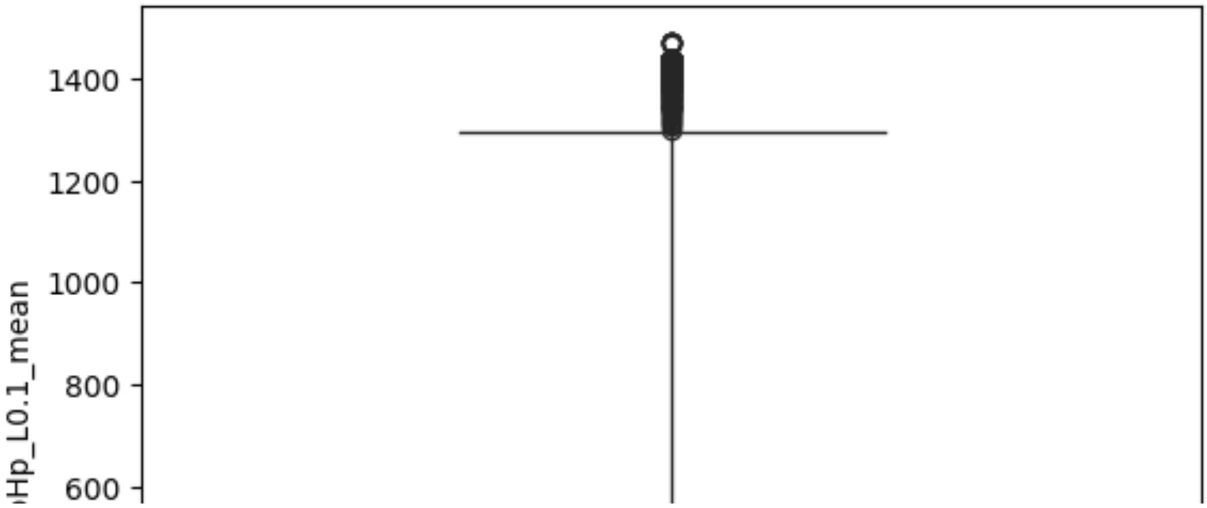

HH_jit_L0.1_weight
8000
7000
6000
5000
4000
3000
2000
1000
0

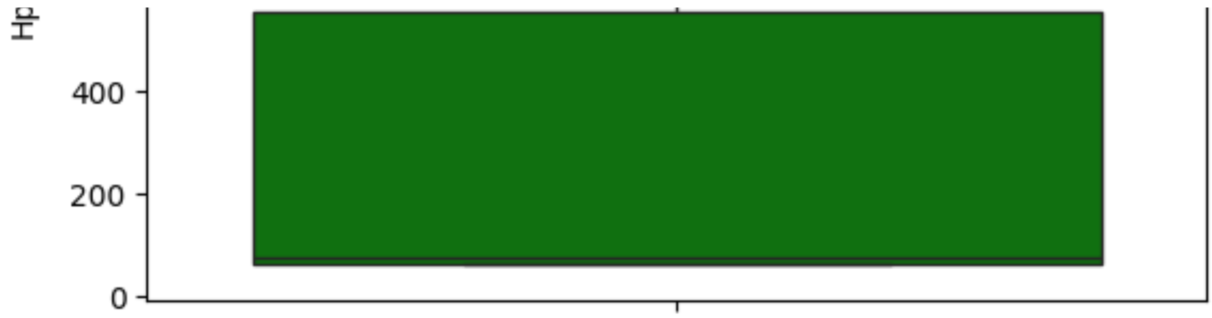

1e9
HH_jit_L0.1_mean
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0

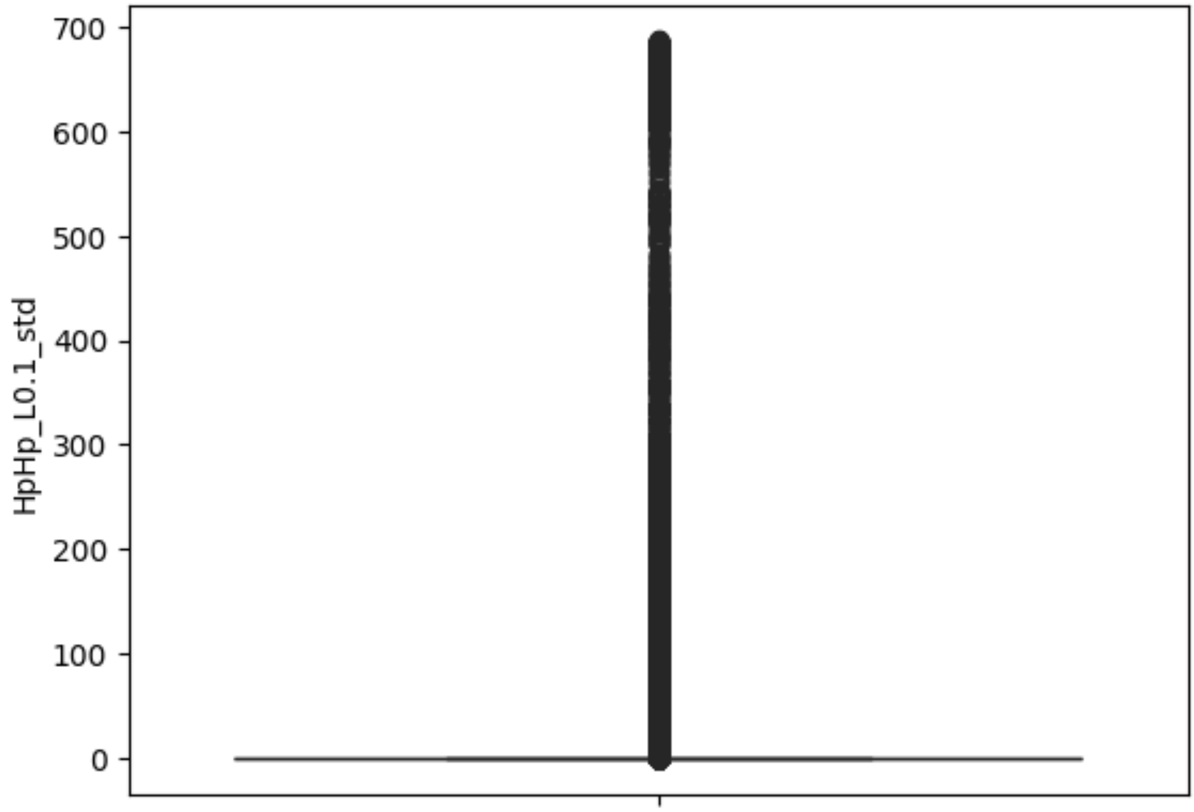

1e17
6
5

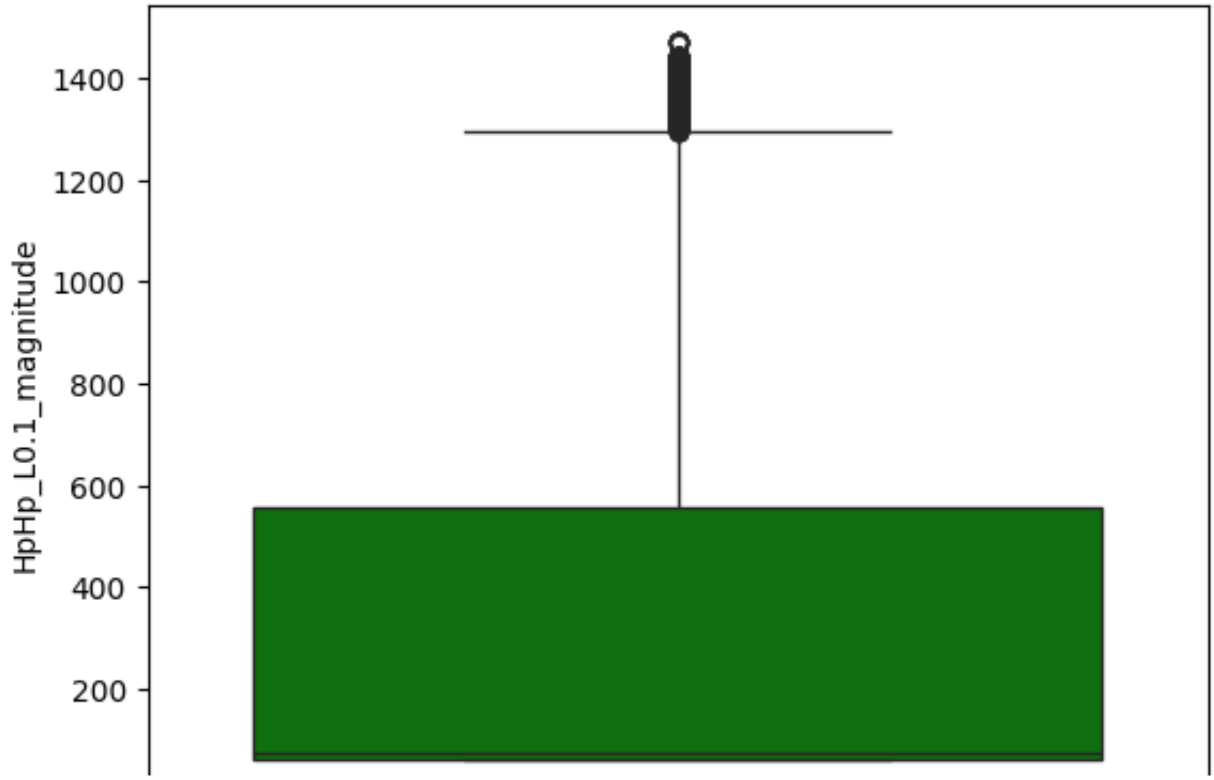

HH_jit_L0.1_variance
4
3
2
1
0


HpHp_L0.1_weight
4000
3000
2000
1000
0

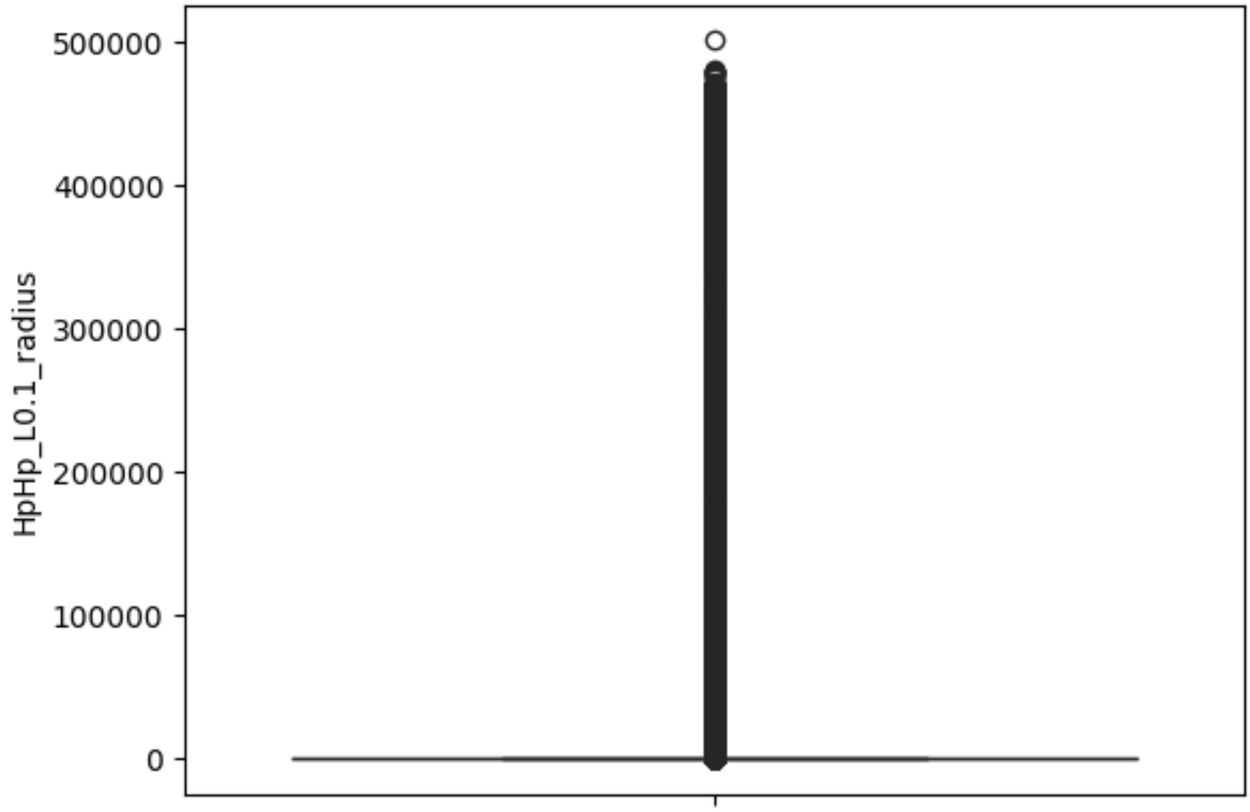

1400
1200
1000
800
600

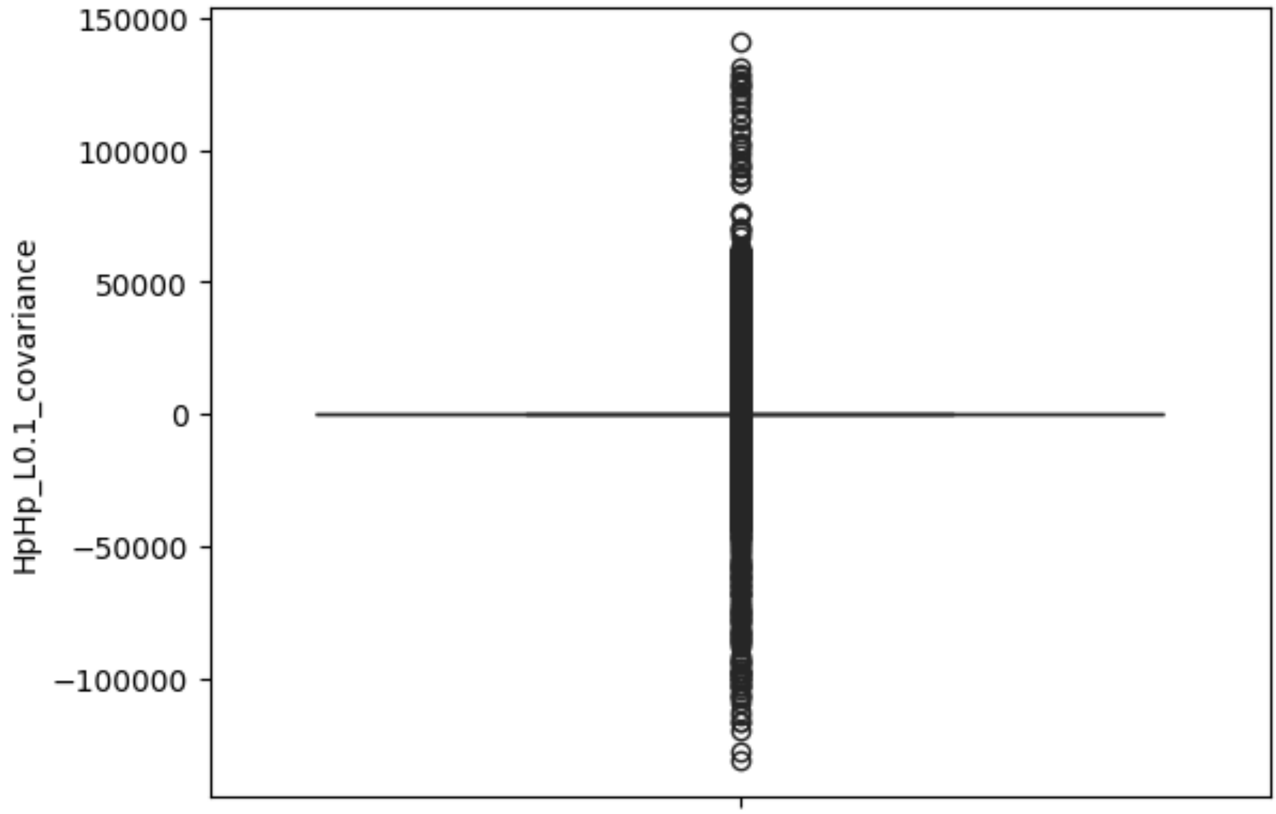

Hp
400
200
0


HpHp_L0.1_std
700
600
500
400
300
200
100
0

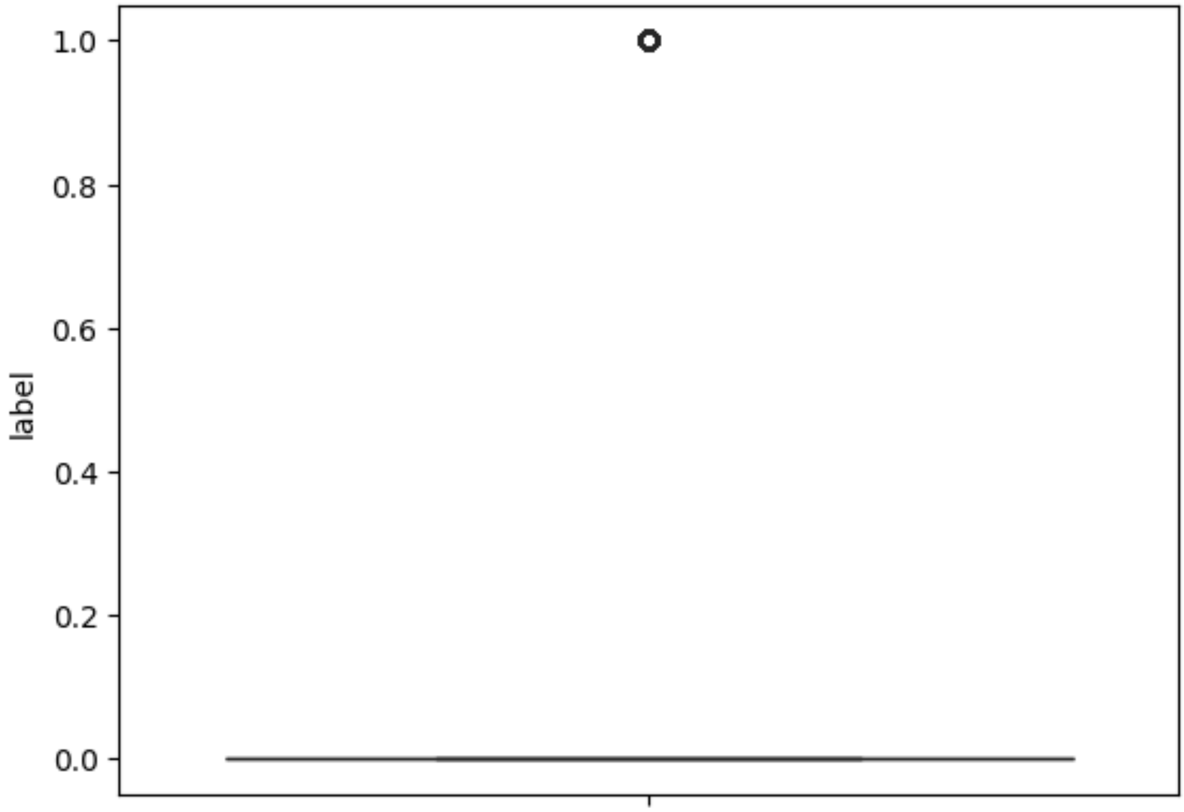

HpHp_L0.1_magnitude
1400
1200
1000
800
600
400
200


0


HpHp_L0.1_radius
500000
400000
300000
200000
100000
0


HpHp_L0.1_covariance
150000
100000
50000
0
−50000
−100000


2




```
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
```

```
from lightgbm import LGBMClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from keras.models import Sequential
from keras.layers import Dense
import keras.activations,keras.optimizers,keras.losses
from google.colab import drive
import numpy as np
import pandas as pd
import seaborn as sns
from pandas.plotting import autocorrelation_plot
import os


drive.mount('/content/drive')



data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/AAI-530/BoTNeTIoT-L0
chunk=data.get_chunk(10)
print(chunk)
chunk = chunk.dropna()

print(chunk)
data=chunk
```

```
0          1.000000        98.000000       0.000000e+00        1.00000
1          1.931640        98.000000       1.818989e-12        1.93164
2          2.904273        86.981750       2.311822e+02        2.90427
3          3.902546        83.655268       2.040614e+02        3.90254
4          4.902545        81.685828       1.775746e+02        4.90254
5          5.902539        80.383706       1.558026e+02        5.90253
6        613.818538        74.095096       2.659110e+00      613.81853
7        614.778927        74.094941       2.654800e+00      614.77892
8        615.618170        74.094787       2.650502e+00      615.61817
9        616.596022        74.094633       2.646218e+00      616.59602

   H_L0.1_mean  H_L0.1_variance  HH_L0.1_weight  HH_L0.1_mean   HH_L0.1_std
0    98.000000     0.000000e+00        1.000000          98.0  0.000000e+00
1    98.000000     1.818989e-12        1.931640          98.0  1.348699e-06
2    86.981750     2.311822e+02        1.000000          66.0  0.000000e+00
3    83.655268     2.040614e+02        1.000000          74.0  0.000000e+00
4    81.685828     1.775746e+02        2.000000          74.0  9.536743e-07
5    80.383706     1.558026e+02        2.999997          74.0  9.536743e-07
6    74.095096     2.659110e+00      610.152839          74.0  3.814697e-06
7    74.094941     2.654800e+00      611.113465          74.0  3.814697e-06
8    74.094787     2.650502e+00      611.953666          74.0  3.568323e-06
9    74.094633     2.646218e+00      612.931650          74.0  3.693565e-06
```

```
   HH_L0.1_magnitude  ...  HpHp_L0.1_mean  HpHp_L0.1_std  HpHp_L0.1_magnitu
0          98.000000  ...              98       0.000000            98.0000
1         138.592929  ...              98       0.000001           138.5929
2         114.856432  ...              66       0.000000           114.8564
3          74.000000  ...              74       0.000000            74.0000
4          74.000000  ...              74       0.000000            74.0000
5          74.000000  ...              74       0.000000            74.0000
6          95.268043  ...              74       0.000000            74.0000
7          95.268043  ...              74       0.000000            74.0000
8          95.268043  ...              74       0.000000            74.0000
9          95.268043  ...              74       0.000000            74.0000

   HpHp_L0.1_radius  HpHp_L0.1_covariance  HpHp_L0.1_pcc       Device_Name
0      0.000000e+00                     0              0  Danmini_Doorbell
1      1.818989e-12                     0              0  Danmini_Doorbell
2      0.000000e+00                     0              0  Danmini_Doorbell
3      0.000000e+00                     0              0  Danmini_Doorbell
4      0.000000e+00                     0              0  Danmini_Doorbell
5      0.000000e+00                     0              0  Danmini_Doorbell
6      0.000000e+00                     0              0  Danmini_Doorbell
7      0.000000e+00                     0              0  Danmini_Doorbell
8      0.000000e+00                     0              0  Danmini_Doorbell
9      0.000000e+00                     0              0  Danmini_Doorbell

   Attack  Attack_subType  label
0  gafgyt           combo      0
1  gafgyt           combo      0
2  gafgyt           combo      0
3  gafgyt           combo      0
4  gafgyt           combo      0
5  gafgyt           combo      0
6  gafgyt           combo      0
7  gafgyt           combo      0
8  gafgyt           combo      0
9  gafgyt           combo      0
```

```
import pandas as pd

print(data['Attack'].unique())
```

```
['gafgyt']
```

```
chunk['Attack']
```

	Attack
0	gafgyt
1	gafgyt
2	gafgyt
3	gafgyt
4	gafgyt
5	gafgyt
6	gafgyt
7	gafgyt
8	gafgyt
9	gafgyt

dtype: object

```
import pandas as pd


mirai_df = pd.DataFrame()
graft_df = pd.DataFrame()

for chunk in pd.read_csv('/content/drive/My Drive/Colab Notebooks/AAI-530/BoTNe
    mirai_chunk = chunk[chunk['Attack'] == 'mirai']
    graft_chunk = chunk[chunk['Attack'] == 'gafgyt']

    mirai_df = pd.concat([mirai_df, mirai_chunk])
    graft_df = pd.concat([graft_df, graft_chunk])

    if len(mirai_df) >= 2000 and len(graft_df) >= 2000:
        break

mirai_df = mirai_df.head(2000)
graft_df = graft_df.head(2000)

# Combine the dataframes if needed
combined_df = pd.concat([mirai_df, graft_df])

combined_df
```

	MI_dir_L0.1_weight	MI_dir_L0.1_mean	MI_dir_L0.1_variance	H_L0.1_w
316650	1.000000	566.000000	0.000000e+00	1.(
316651	1.999932	566.000000	1.746230e-10	1.9
316652	2.999171	566.000000	0.000000e+00	2.9
316653	3.999171	566.000000	0.000000e+00	3.9
316654	4.998261	566.000000	0.000000e+00	4.9
...	...	...	...	
1995	2227.972510	74.048593	1.246264e+00	2227.9
1996	2228.971884	74.048571	1.245706e+00	2228.9
1997	2229.796221	74.048549	1.245148e+00	2229.7
1998	2230.795742	74.048528	1.244591e+00	2230.7
1999	2231.787705	74.048506	1.244035e+00	2231.7

4000 rows × 27 columns

Linear Regression for predicting the attack

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
#Import LabelEncoder here as well to make it available within the else block.
from sklearn.preprocessing import LabelEncoder
from google.colab import drive
drive.mount('/content/drive')


file_path = '/content/drive/My Drive/Colab Notebooks/AAI-530/BoTNeTIoT-L01-v2.c

#
total_rows = sum(1 for _ in open(file_path))
half_point = total_rows // 2
df = pd.read_csv(file_path, skiprows=half_point, engine='python')

# Handle potential errors during file reading.
```

```
try:
    df = pd.read_csv(file_path, skiprows=range(1,half_point+1), engine='python'
except pd.errors.ParserError:
    print("Error: Could not parse the CSV file. Check the file format and try a
except FileNotFoundError:
    print(f"Error: File '{file_path}' not found.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
else:
    #Preprocess the data
    df = df.dropna() # Remove rows with missing values


    # save the Attack rolumn
    df['Attack_name']=df['Attack']
    le = LabelEncoder()

    df['Attack'] = le.fit_transform(df['Attack'])

    # Prepare data for linear regression
    X = df.drop('Attack', axis=1)
    y = df['Attack']

    #Convert non-numeric columns to numeric representation.
    for col in X.columns:
        if X[col].dtype == 'object':
            X[col] = le.fit_transform(X[col])

    X = X.select_dtypes(include=np.number) #Only include numeric columns
    y = y.astype(int) #Convert to integers

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra

    # Train a Linear Regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions
    y_pred = model.predict(X_test)

    # Evaluate the model
    mse = mean_squared_error(y_test, y_pred)
    print(f"Mean Squared Error: {mse}")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
Mean Squared Error: 0.5236330182518893
```

```
print(df['Attack_name'].unique())
```

```
['mirai' 'gafgyt' 'Normal']
```

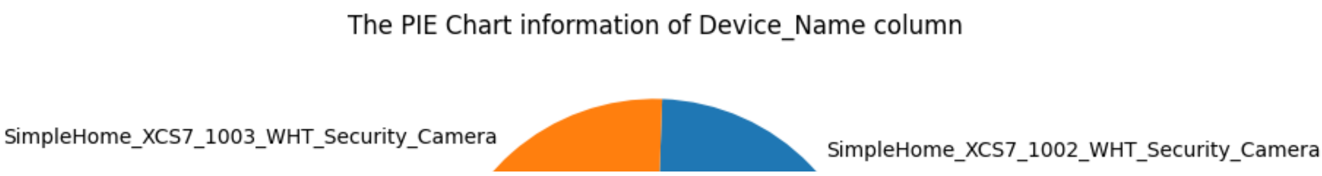
```
print(df['Attack'])
df['Attack'].value_counts()
```

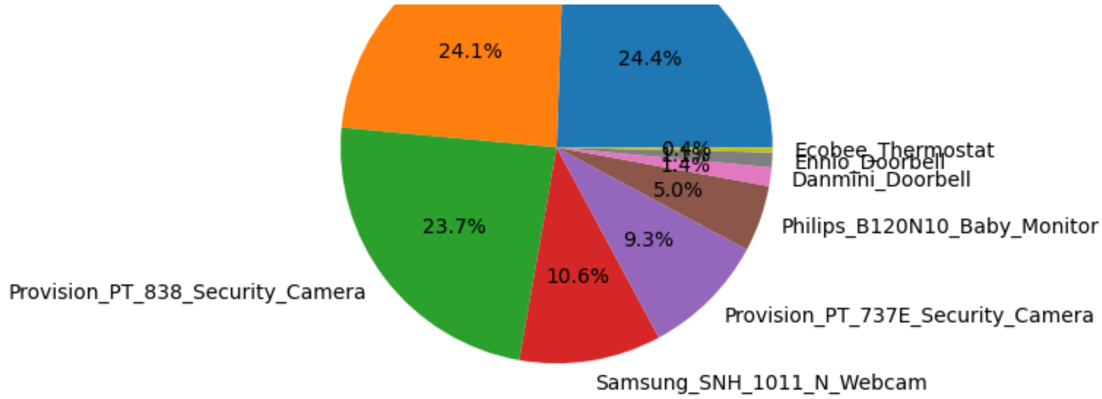
```
0          2
1          2
2          2
3          2
4          2
          ..
3531298    0
3531299    0
3531300    0
3531301    0
3531302    0
Name: Attack, Length: 3531303, dtype: int64
```

Attack	count
2	1723598
1	1251773
0	555932

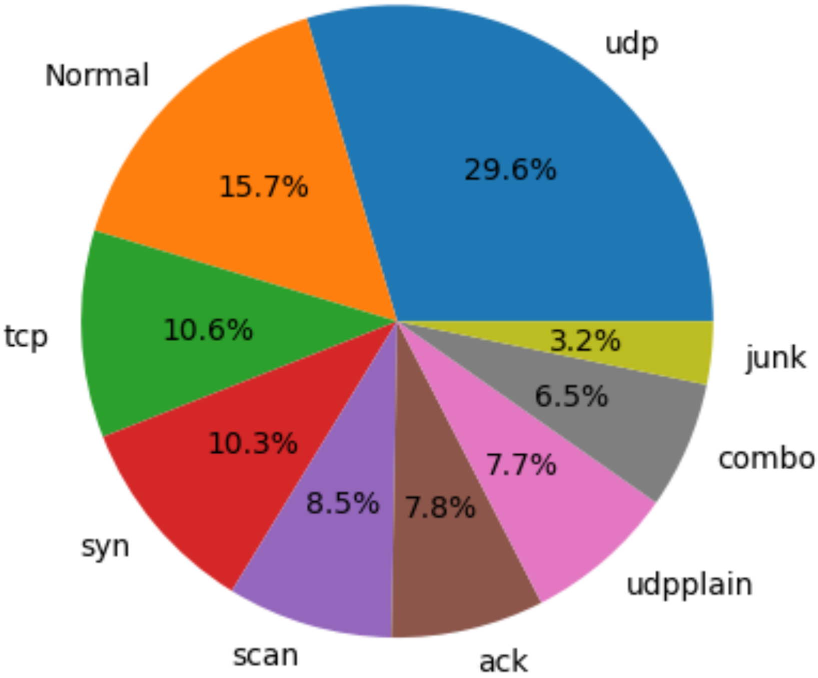
dtype: int64

```
import matplotlib.pyplot as plt

data = df
for i in data.select_dtypes(include='object').columns.values:
    if len(data[i].value_counts()) <=10:
        val=data[i].value_counts().values
        index=data[i].value_counts().index
        plt.pie(val,labels=index,autopct='%1.1f%%')
        plt.title(f'The PIE Chart information of {i} column')
        plt.show()
```

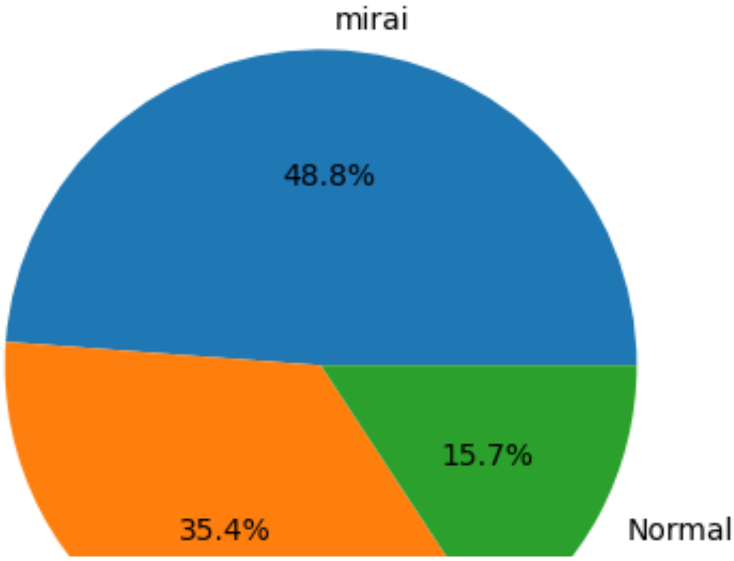




The PIE Chart information of Attack_subType column



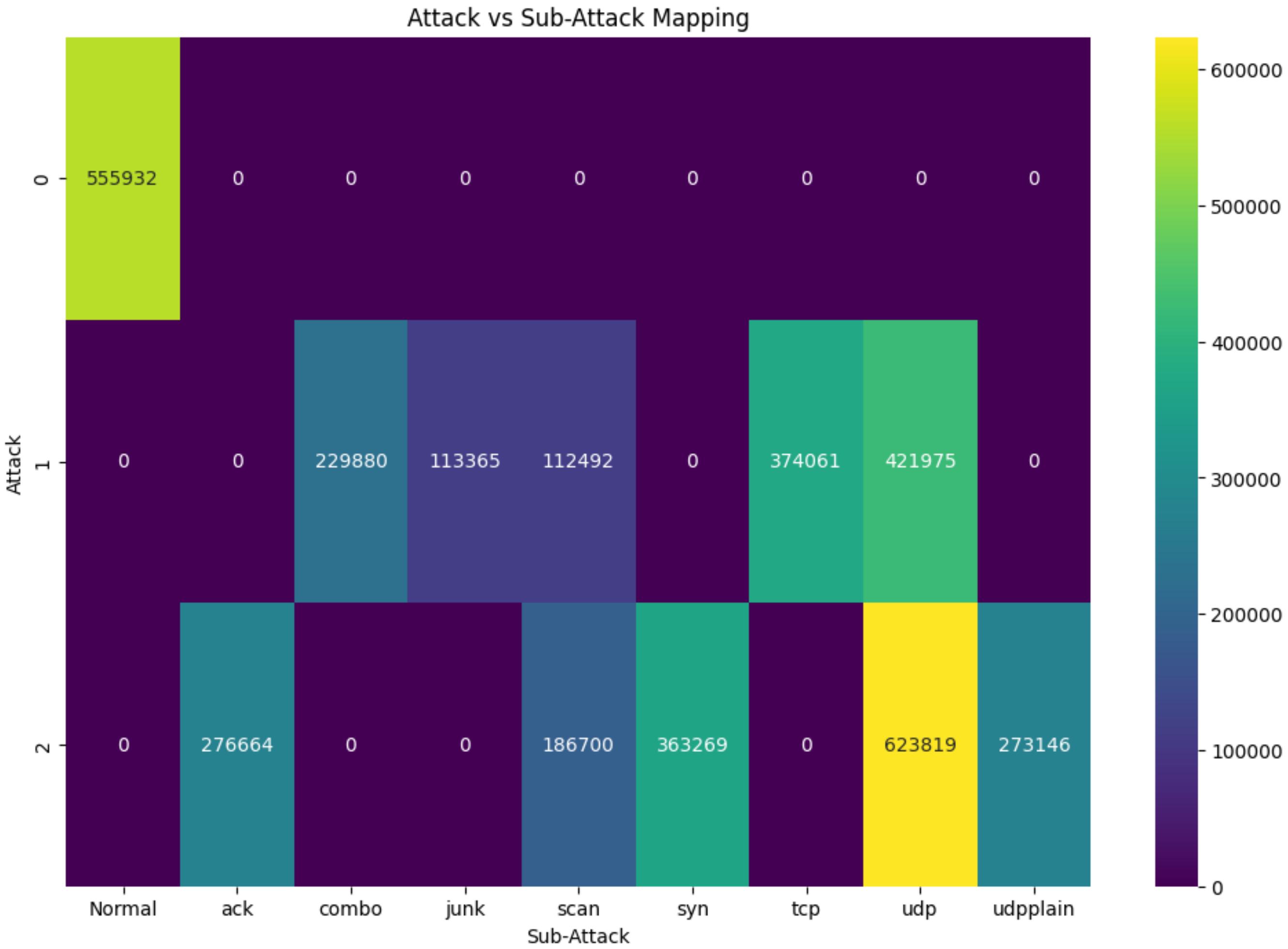
The PIE Chart information of Attack_name column



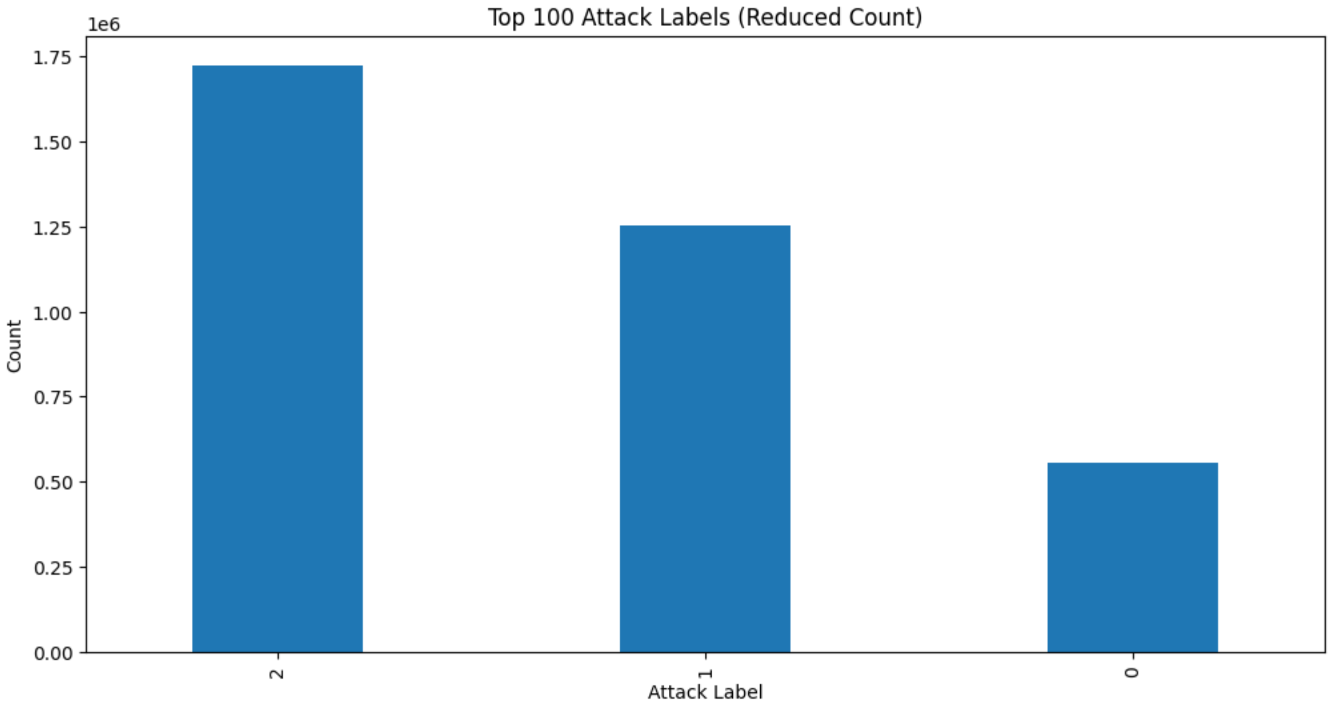

```
for i in data.select_dtypes(include='object').columns.values:
    print(data[i].value_counts())
    print("--------------------------------")
```

```
Device_Name
SimpleHome_XCS7_1002_WHT_Security_Camera    863056
SimpleHome_XCS7_1003_WHT_Security_Camera    850826
Provision_PT_838_Security_Camera            836891
Samsung_SNH_1011_N_Webcam                   375222
Provision_PT_737E_Security_Camera           328307
Philips_B120N10_Baby_Monitor                175240
Danmini_Doorbell                             49548
Ennio_Doorbell                               39100
Ecobee_Thermostat                            13113
Name: count, dtype: int64
--------------------------------
Attack_subType
udp         1045794
Normal       555932
tcp          374061
syn          363269
scan         299192
ack          276664
udpplain     273146
combo        229880
junk         113365
Name: count, dtype: int64
--------------------------------
Attack_name
mirai     1723598
gafgyt    1251773
Normal     555932
Name: count, dtype: int64
--------------------------------
```

```
import matplotlib.pyplot as plt
import seaborn as sns


attack_subattack_counts = pd.crosstab(df['Attack'], df['Attack_subType'])

# Plot the heatmap
plt.figure(figsize=(12, 8))  # Adjust figure size as needed
sns.heatmap(attack_subattack_counts, annot=True, fmt="d", cmap="viridis")
plt.title("Attack vs Sub-Attack Mapping")
plt.xlabel("Sub-Attack")
plt.ylabel("Attack")
plt.show()
```


Attack vs Sub-Attack Mapping
Attack
0
1
2
Sub-Attack
Normal
ack
combo
junk
scan
syn
tcp
udp
udpplain
555932
0
229880
113365
112492
374061
421975
276664
186700
363269
623819
273146
600000
500000
400000
300000
200000
100000

```
attack_counts = df['Attack'].value_counts().head(100)
plt.figure(figsize=(12, 6))
attack_counts.plot(kind='bar', width=0.4)  # Reduce bar width
plt.title('Top 100 Attack Labels (Reduced Count)')
plt.xlabel('Attack Label')
plt.ylabel('Count')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.show()
```




```
print(df['Attack'].value_counts())
```

```
Attack
2    1723598
1    1251773
0     555932
Name: count, dtype: int64
```

```
lab=LabelEncoder()
for i in data.select_dtypes(include='object').columns.values:
    data[i]=lab.fit_transform(data[i])


x={}
X=[]
for i in data.columns.values:
    data['z-scores']=(data[i]-data[i].mean())/(data[i].std())
    outliers=np.abs(data['z-scores'] > 3).sum()
    x[i]=outliers



for keys,values in x.items():
    if values>0:
        X.append(keys)
print(x)
```

```
{'MI_dir_L0.1_weight': 0, 'MI_dir_L0.1_mean': 894, 'MI_dir_L0.1_variance':
```

```
x=[]

thresh=2
for i in data[X].columns.values:
    upper=data[i].mean()+thresh*data[i].std()
    lower=data[i].mean()-thresh*data[i].std()
    data2=data[(data[i]>lower)&(data[i]<upper)]

print(len(data))
print(data)
```

```
3531303
         MI_dir_L0.1_weight  MI_dir_L0.1_mean  MI_dir_L0.1_variance  \
0               3102.162512         67.503270             48.964091
1               3102.892660         67.505364             48.961909
2               3103.892660         67.507456             48.959720
3               3104.892454         67.509547             48.957524
4               3105.238049         67.511637             48.955319
...                     ...               ...                   ...
3531298            2.937269        217.763487          17706.823640
3531299            1.730254        282.630543          10545.887900
3531300            2.730251        299.980395           7204.116620
3531301            2.882414        216.723647          17753.083150
3531302            2.032574        154.377267          13032.487600

         H_L0.1_weight  H_L0.1_mean  H_L0.1_variance  HH_L0.1_weight  \
0          3102.162512    67.503270        48.964091     1653.072952
1          3102.892660    67.505364        48.961909     1653.929154
```

```
2          3103.892660   67.507456        48.959720     1654.929154
3          3104.892454   67.509547        48.957524     1655.929044
4          3105.238049   67.511637        48.955319     1656.580031
...                ...         ...              ...             ...
3531298       2.937269  217.763487     17706.823640        1.220882
3531299       1.730254  282.630543     10545.887900        1.213342
3531300       2.730251  299.980395      7204.116620        1.213352
3531301       2.882414  216.723647     17753.083150        1.209274
3531302       2.032574  154.377267     13032.487600        1.299681

         HH_L0.1_mean   HH_L0.1_std  HH_L0.1_magnitude  ...  \
0           73.979998  5.287936e-01          73.979998  ...
1           73.980010  5.286340e-01          73.980010  ...
2           73.980023  5.284745e-01          73.980023  ...
3           73.980035  5.283151e-01          73.980035  ...
4           73.980047  5.281558e-01          73.980047  ...
...               ...           ...                ...  ...
3531298     60.000000  9.540000e-07          84.852814  ...
3531299    330.000000  5.390000e-06         431.490440  ...
3531300    330.000000  6.610000e-06         431.490440  ...
3531301     60.000000  6.740000e-07          84.852814  ...
3531302    145.339354  1.010891e+02         195.783485  ...

         HpHp_L0.1_magnitude  HpHp_L0.1_radius  HpHp_L0.1_covariance  \
0                  74.000000      0.000000e+00          0.000000e+00
1                  74.000000      0.000000e+00          0.000000e+00
2                  74.000000      0.000000e+00          0.000000e+00
3                  74.000000      0.000000e+00          0.000000e+00
4                  74.000000      0.000000e+00          0.000000e+00
...                      ...               ...                   ...
3531298            84.852814      1.290000e-12          1.720000e-29
3531299           431.490440      2.910000e-11          7.390000e-83
3531300           431.490440      4.370000e-11          1.560000e-81
3531301            84.852814      4.550000e-13          8.910000e-30
3531302           195.783485      1.218303e+04          1.917443e+03

         HpHp_L0.1_pcc  Device_Name  Attack  Attack_subType  label  \
0         0.000000e+00            4       2               5      0
1         0.000000e+00            4       2               5      0
2         0.000000e+00            4       2               5      0
3         0.000000e+00            4       2               5      0
4         0.000000e+00            4       2               5      0
```

```
data['Attack']
```

	Attack
0	2
1	2
2	2
3	2
4	2
...	...
3531298	0
3531299	0
3531300	0
3531301	0
3531302	0

3531303 rows × 1 columns

dtype: int64

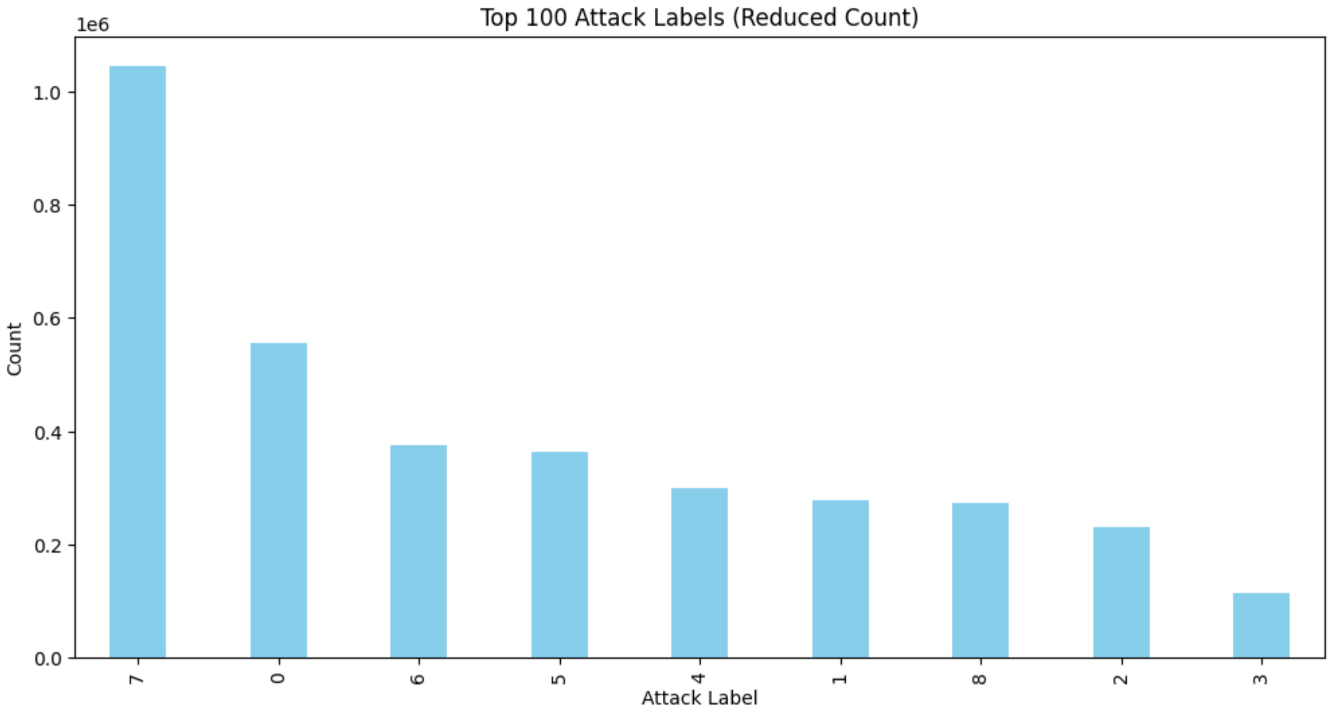
```
print(data['Attack'].value_counts())
```

```
Attack
2    1723598
1    1251773
0     555932
Name: count, dtype: int64
```

```
attack_mapping = {
    'mirai': 2,
    'gafgyt': 1,
    'Normal': 0
}
```

```
df['Attack_name1'] = df['Attack'].map(attack_mapping)
```

```
# Plot the top 100 attack labels with reduced count and bar width, with customi
attack_counts = df['Attack_subType'].value_counts().head(100)
plt.figure(figsize=(12, 6))
attack_counts.plot(kind='bar', width=0.4, color='skyblue')  # Set the color to
plt.title('Top 100 Attack Labels (Reduced Count)')
plt.xlabel('Attack Label')
plt.ylabel('Count')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.show()
```



This code performs time series analysis and anomaly detection on IoT telemetry data, followed by an analysis of network attack data. Let's break down the key parts:

1. Time Series Analysis (Temperature Prediction and Anomaly Detection):

- **Data Loading and Preprocessing:** Loads IoT telemetry data, sorts it by timestamp, and scales the relevant features (humidity, CO, LPG, smoke, and temperature).

- **Dataset Creation:** Creates a custom PyTorch dataset (`TimeSeriesDataset`) to prepare the data for a transformer model. It creates sequences of input features (excluding temperature) to predict the next temperature value.
- **Model Definition:** Defines a transformer-based model (`TransformerTimeSeries`) for time series prediction using PyTorch Lightning.
- **Training and Prediction:** Trains the model using the prepared dataset and then predicts temperature values on the test set.
- **Inverse Transformation:** Inverse transforms the scaled predictions and actual temperature values back to their original scale.
- **Visualization and Anomaly Detection:** Plots the predicted and actual temperatures and then identifies anomalies by analyzing the differences between consecutive predictions. A threshold is applied to detect significant deviations. Anomalies are then highlighted on the plot.

2. Network Intrusion Detection System (NIDS) Analysis:

- **Data Loading (in chunks):** Reads a large CSV file containing network traffic data in smaller chunks to avoid memory issues.
- **Box Plots:** Creates box plots for numerical features in the dataset to visualize their distributions and identify potential outliers.
- **Data Filtering:** Selects specific attacks ('mirai' and 'gafgyt') and limits their count to 2000 each for analysis.
- **Label Encoding:** Converts categorical features (e.g. 'Attack' types) into numerical representations for Machine Learning algorithms.
- **Regression Model (Linear Regression):** The code attempts to perform linear regression to predict the "Attack" type. The model's performance is evaluated using the Mean Squared Error (MSE).
- **Pie Charts:** Generates pie charts for categorical features with a small number of unique values to show the distribution of different categories.
- **Heatmap (Attack vs. Sub-Attack):** Visualizes the relationship between 'Attack' and 'Attack_SubType' using a heatmap.
- **Outlier Detection (Z-score):** Detects outliers in the dataset using z-score calculations and thresholds. It calculates z-scores for each column and then removes observations based on a criteria (z-score >3).
- **Visualization of Attack Labels:** Plots the top attack sub-types showing the number of occurences of each sub-type.

Overall:

The code combines time series forecasting with network intrusion detection system analysis. It uses different visualization tools such as line plots, box plots, pie charts, heat maps and bar charts to gain insights from the respective data. The time series analysis part is well-defined and complete, but the NIDS section could use some improvements. Also, the code includes several data loading and processing steps that could be streamlined.

8. Conclusion

This study demonstrates how **IoT sensor data analysis** provides valuable insights into **environmental monitoring, sensor correlations, and security risks**.

By integrating **machine learning, real-time monitoring, and cybersecurity defenses**, IoT systems can achieve **greater resilience and reliability** in real-world deployments.

---

9. References

1. **Zhang, X., & Li, J. (2021)**. "Anomaly Detection in IoT Sensor Networks Using Machine Learning." *IEEE Transactions on Industrial Informatics, 17(5), 3256-3267.*
2. **Tang, W., et al. (2020)**. "Cybersecurity Threats in IoT: Detecting DDoS Attacks on Sensor Networks." *ACM IoT Security Journal, 8(3), 112-124.*
3. **Hochreiter, S., & Schmidhuber, J. (1997)**. "Long Short-Term Memory." *Neural Computation, 9(8), 1735-1780.*