MACHINE LEARNING FOR TEMPERATURE PREDICTION

Introduction

According to WHO, environmental factors contribute to 23% of all deaths worldwide and 36% of all deaths among children 0-14 years old. Monitoring environmental conditions can highlight trends that can help improve the safety of the ecosystem.

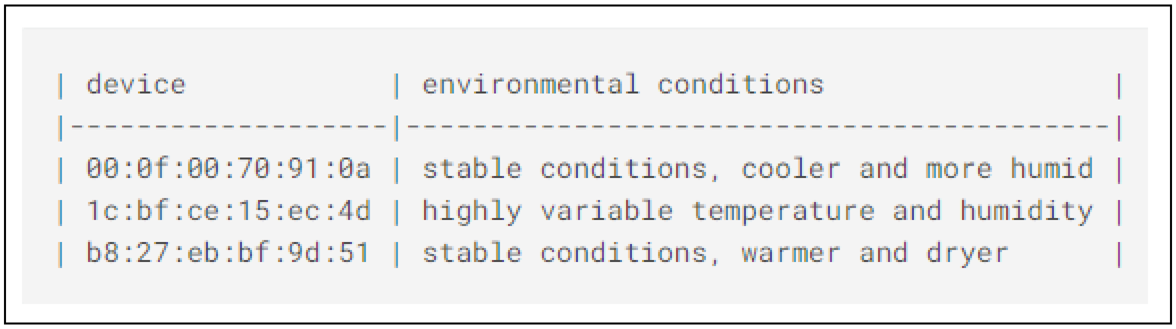
Environmental sensors can provide a variety of information on important environmental parameters, such as temperature, humidity, carbon monoxide(CO) and more.
By using telemetry, monitoring these parameters can be done remotely and in real time.

Artificial Intelligence can play a huge role in decreasing environmental harms by harnessing the loads of data from environmental sensors.

The proximity of a person to the IoT device could affect the parameters in various ways. This project aims at building a machine learning model to predict motion near an IoT device used for environmental sensing. This model will be deployed into production using Kubeflow.

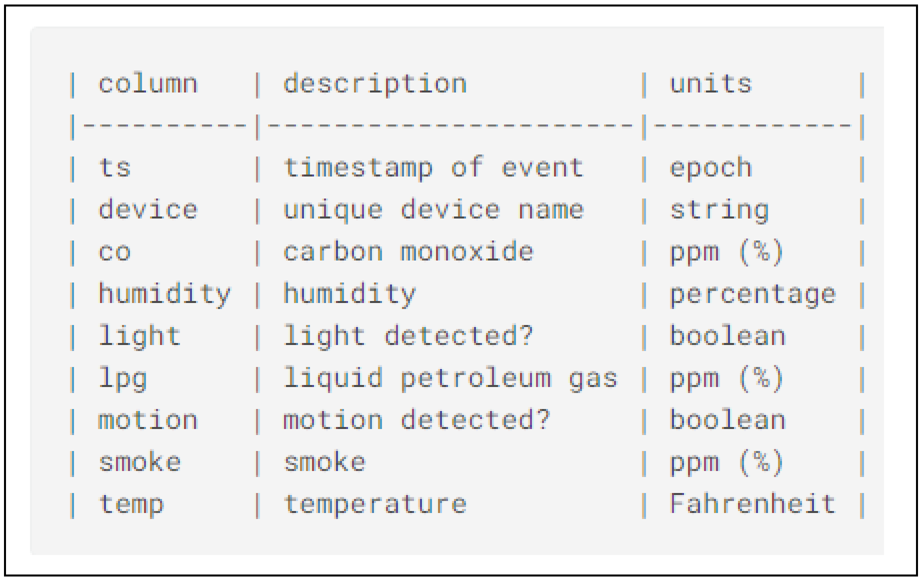
Dataset

The Environmental sensor telemetry dataset for this project was taken from kaggle. The data was generated from a series of three identical, custom-built, breadboard-based sensor arrays. Each array was connected to a Raspberry Pi device. Each of the three IoT devices was placed in a physical location with varied environmental conditions as shown below:



device	environmental conditions
00:0f:00:70:91:0a	stable conditions, cooler and more humid
1c:bf:ce:15:ec:4d	highly variable temperature and humidity
b8:27:eb:bf:9d:51	stable conditions, warmer and dryer

Each IoT device collected a total of seven different readings from the three sensors on a regular interval. Sensor readings include temperature, humidity, carbon monoxide (CO), liquid petroleum gas (LPG), smoke, light, and motion. The data spans the period from 07/12/2020 00:00:00 UTC – 07/19/2020 23:59:59 UTC. The dataset contains a total of 405,184 rows and 9 columns of data.

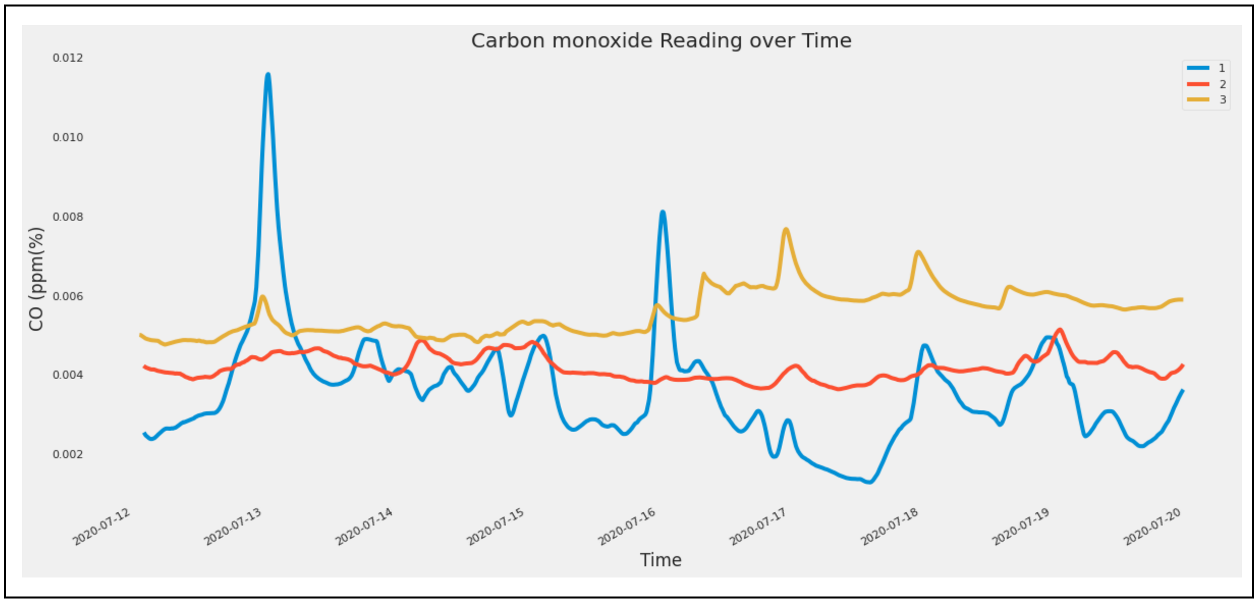


column	description	units
ts	timestamp of event	epoch
device	unique device name	string
co	carbon monoxide	ppm (%)
humidity	humidity	percentage
light	light detected?	boolean
lpg	liquid petroleum gas	ppm (%)
motion	motion detected?	boolean
smoke	smoke	ppm (%)
temp	temperature	Fahrenheit

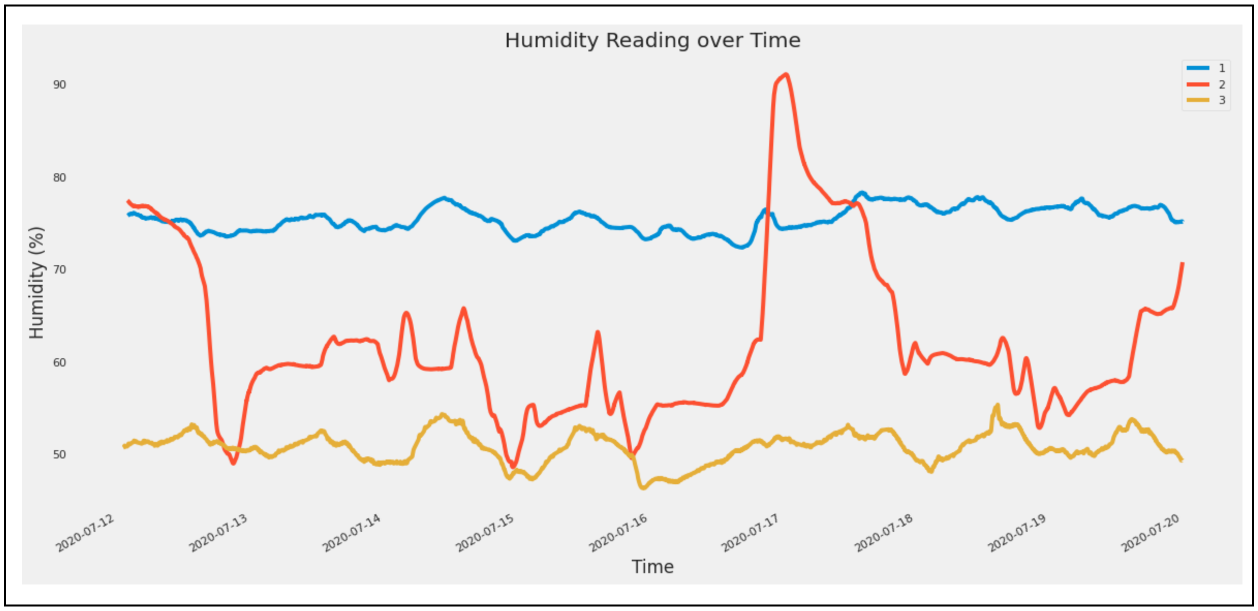
Exploratory Data Analysis

We plotted time series graphs on a 1 hour rolling window to compare readings across all 3 devices. For ease of typing, we renamed the devices thus:

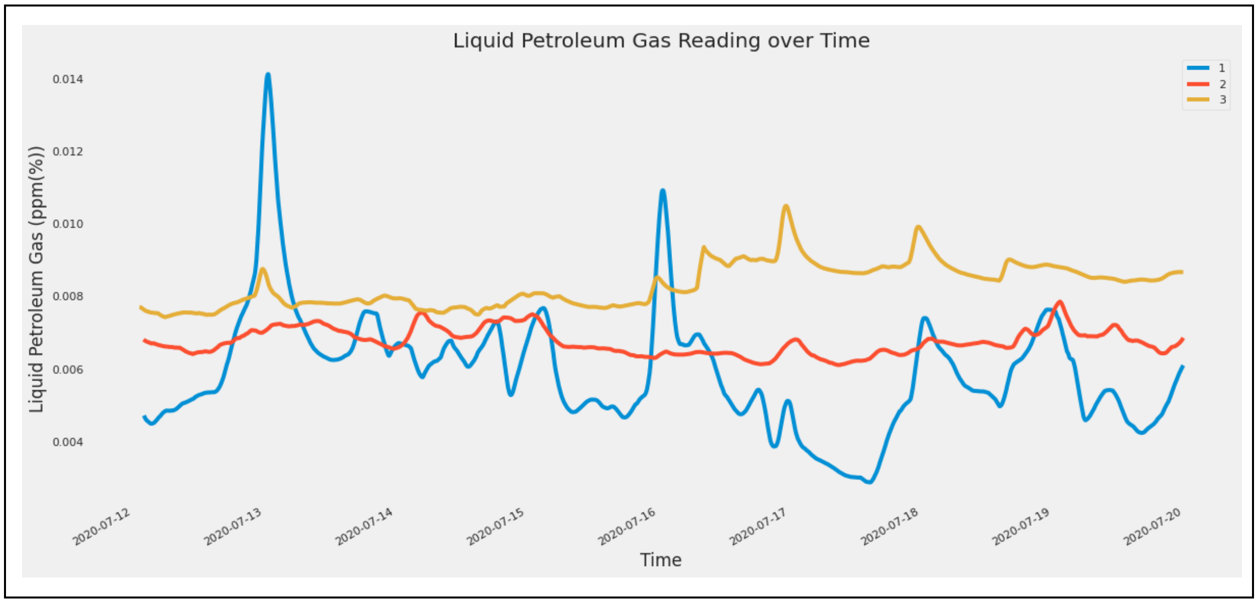
00:0f:00:70:91:0a	1
1c:bf:ce:15:ec:4d	2
b8:27:eb:bf:9d:51	3



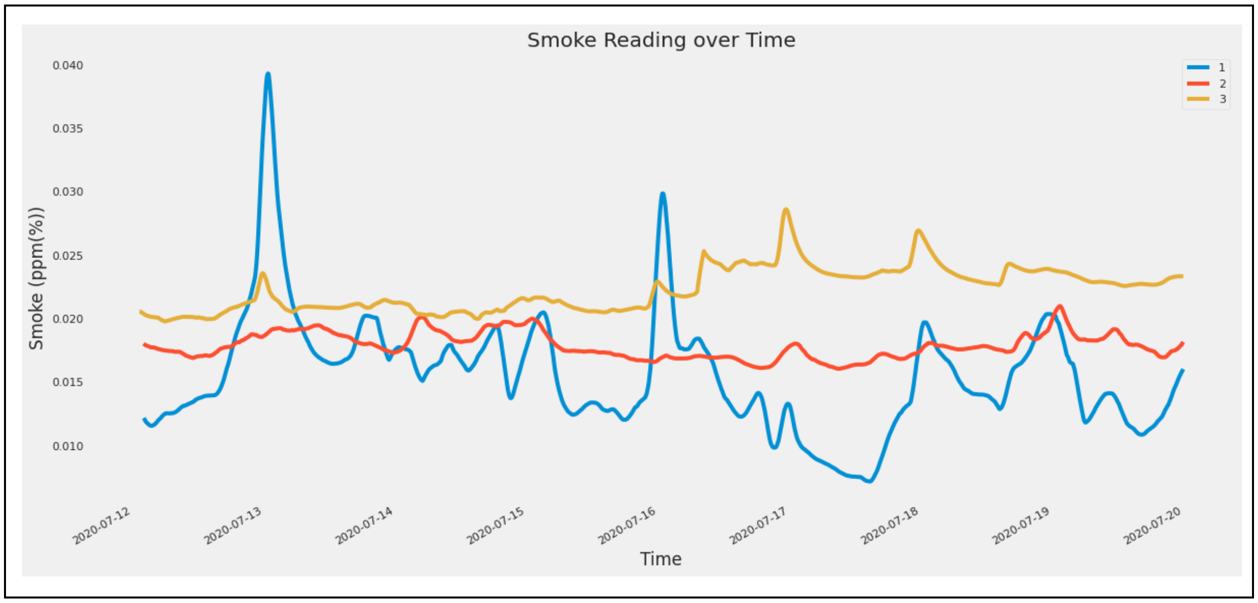
CO readings are correlated across all 3 devices even though there are spikes on the 13th and 16th on device 1



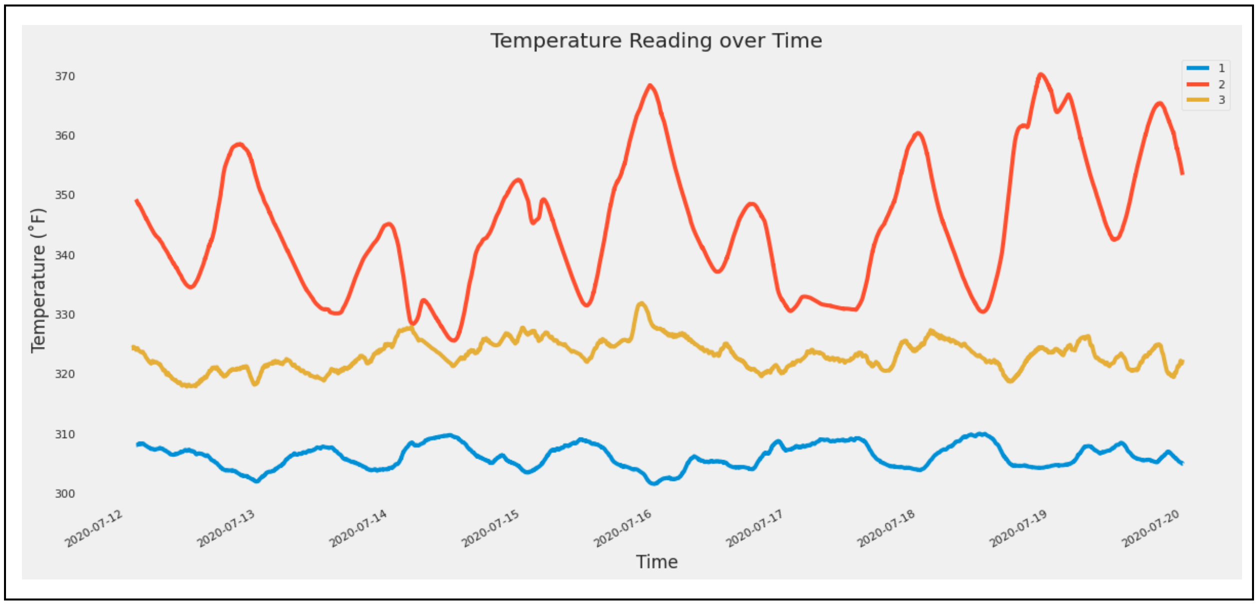
Humidity readings are more stable on devices 1 and 3. The spikes in device 2 is not surprising as humidity in that environment was highly variable. The high humidity recorded on device 1 is also expected as the environment was cooler and more humid.



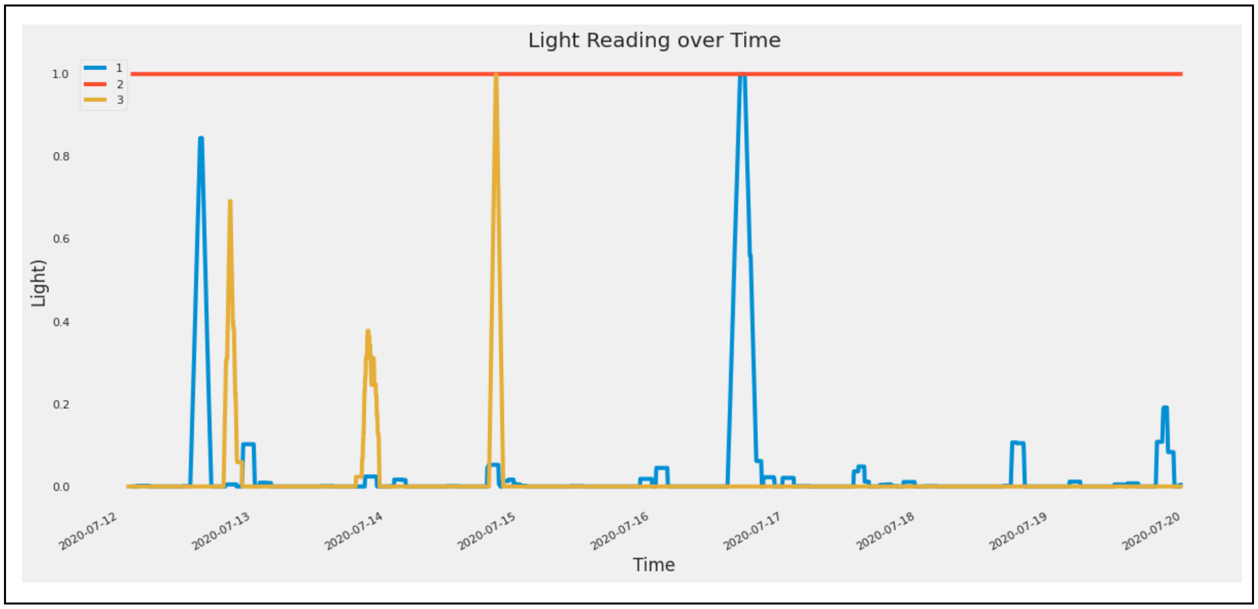
Lpg readings are correlated on all devices with more spikes and falls on device 1



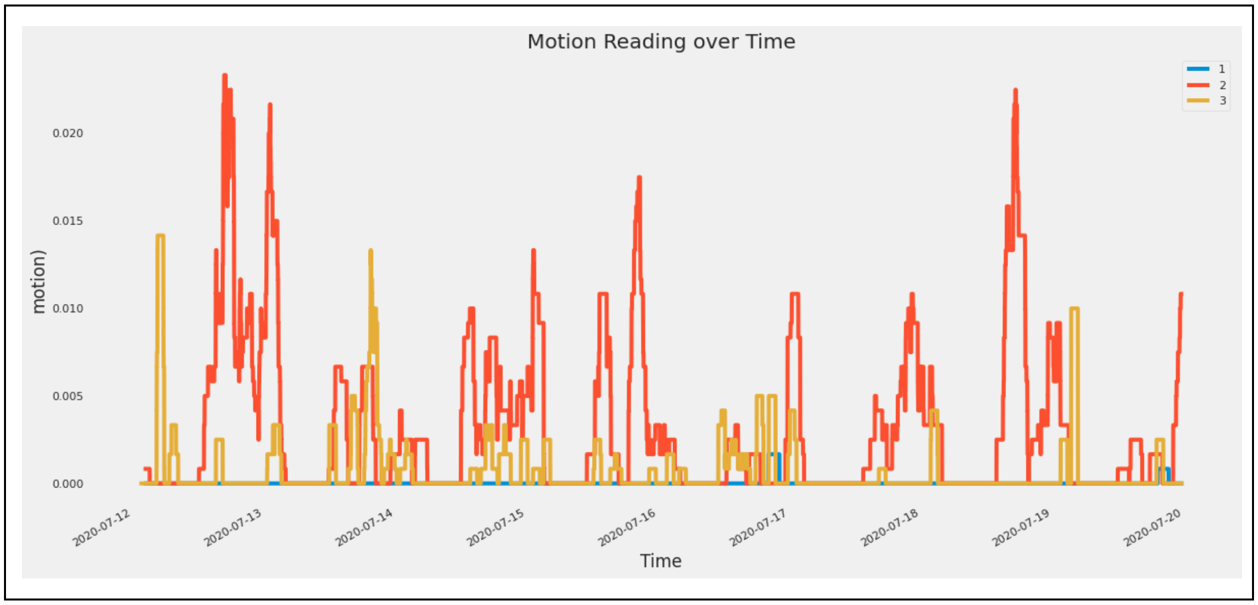
Smoke levels are correlated on all devices with more variability on device 1



The temperature oscillations on device 2 is expected as there was high temperature variability in that environment

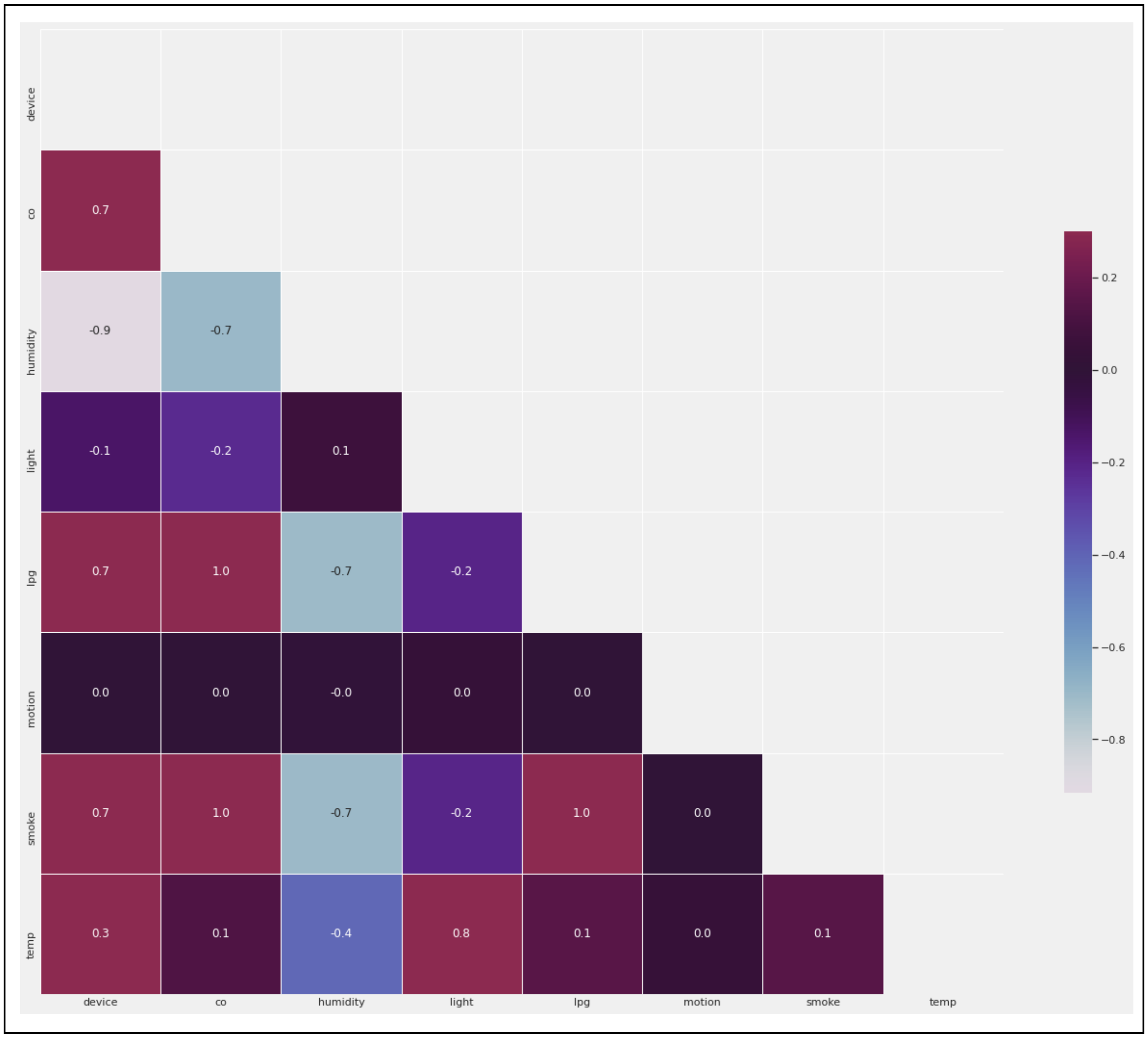


Light readings were continuously “True” on device 2

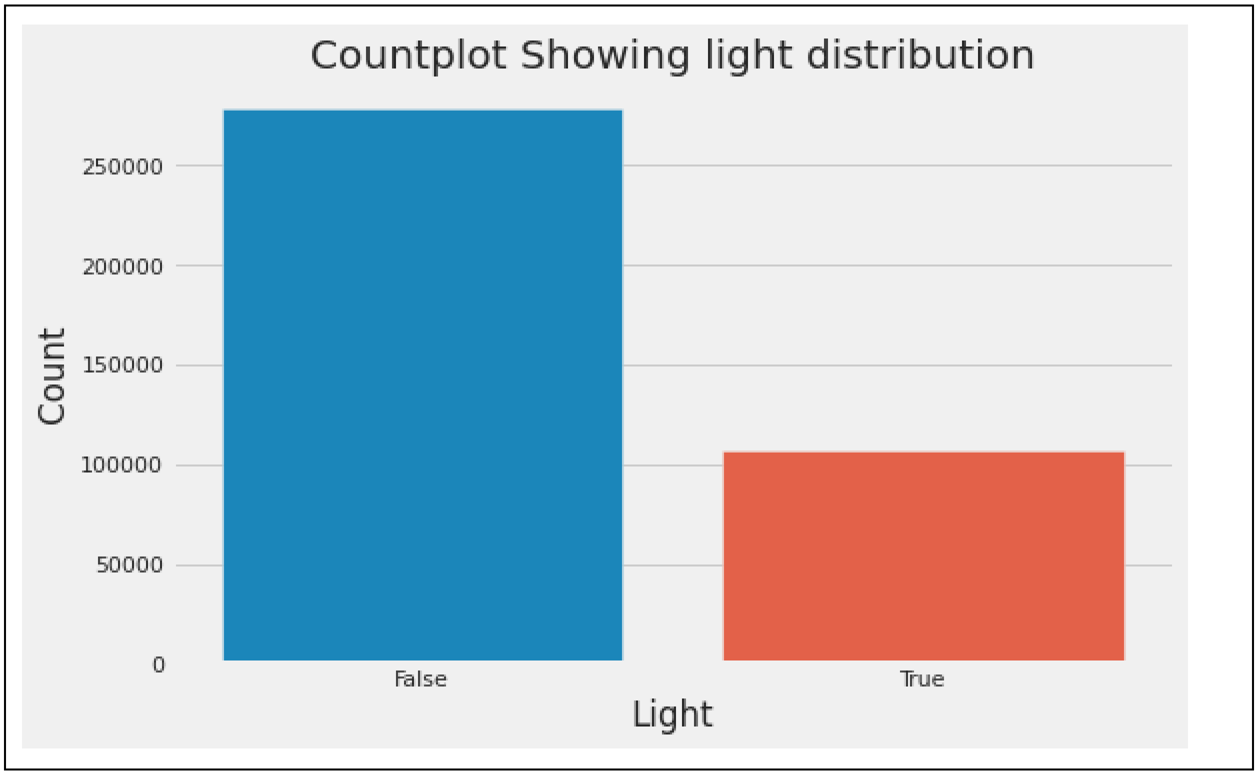


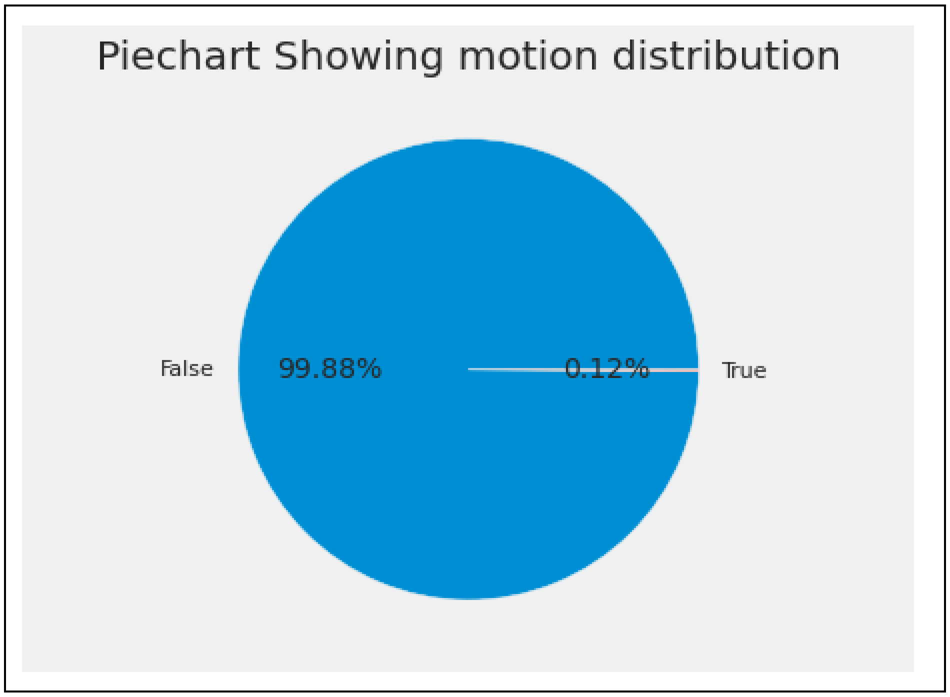
There is almost no noticeable movements around device 1

Correlation plot shows a high correlation between all air quality features (smoke, lpg and CO)



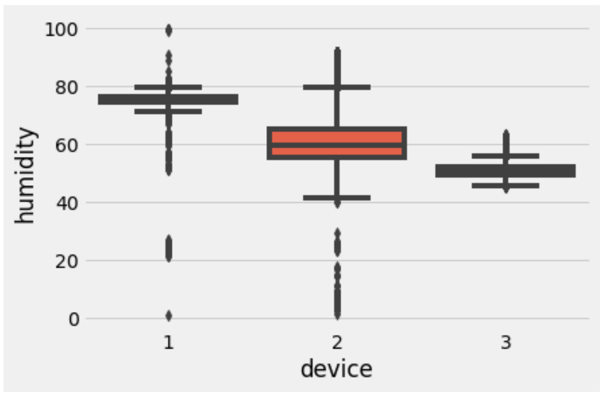
The distribution plots below reveal a high imbalance in light and especially in motion readings:

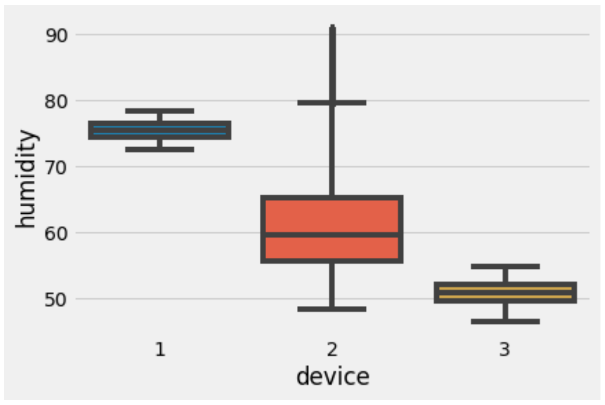

Countplot Showing light distribution
Count
250000
200000
150000
100000
50000
0
False
True
Light

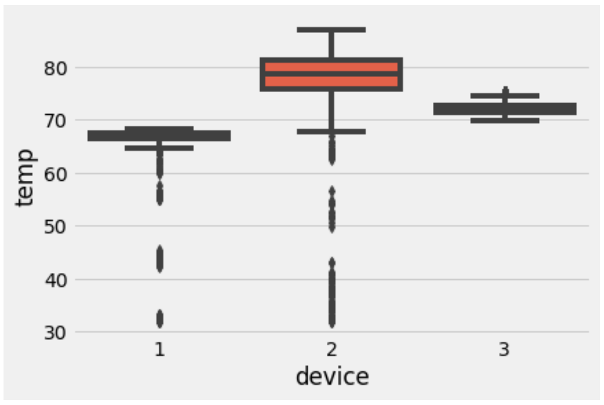

Piechart Showing motion distribution
False
99.88%
0.12%
True

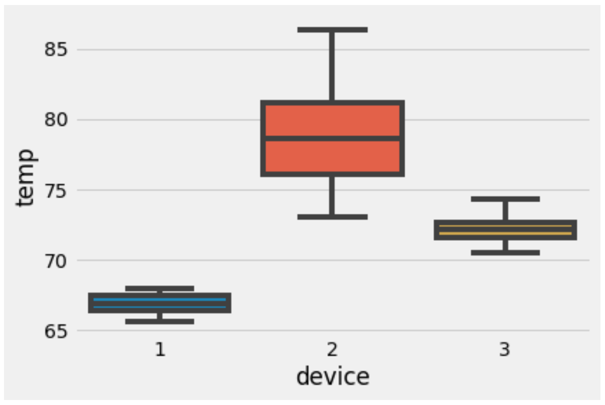
Data Preprocessing and Modeling

Data preprocessing steps taken include: feature scaling, reshaping the input data to a 3 dimensional format to fit into the multivariate time series model and removing outliers from humidity and temperature columns









We trained our data using two models, a Fully Convolutional Neural Network implemented here. The code is shown below:

```
def make_model(input_shape):
    input_layer = keras.layers.Input(input_shape)

    conv1 = keras.layers.Conv1D(filters=64, kernel_size=3, padding="same")(input_layer)
    conv1 = keras.layers.BatchNormalization()(conv1)
    conv1 = keras.layers.ReLU()(conv1)

    conv2 = keras.layers.Conv1D(filters=64, kernel_size=3, padding="same")(conv1)
    conv2 = keras.layers.BatchNormalization()(conv2)
    conv2 = keras.layers.ReLU()(conv2)

    conv3 = keras.layers.Conv1D(filters=64, kernel_size=3, padding="same")(conv2)
    conv3 = keras.layers.BatchNormalization()(conv3)
    conv3 = keras.layers.ReLU()(conv3)

    gap = keras.layers.GlobalAveragePooling1D()(conv3)

    output_layer = keras.layers.Dense(num_classes, activation="softmax")(gap)

    return keras.models.Model(inputs=input_layer, outputs=output_layer)


model = make_model(input_shape=X_train.shape[1:])
keras.utils.plot_model(model, show_shapes=True)
```