About Experimental Data

This dataset is generated from a multi-sensor Arduino/IoT experimental setup designed to simulate a real-world embedded electronics system. In this experiment, an Arduino-based microcontroller is connected to multiple sensors to continuously monitor environmental conditions, motion, electrical power parameters, and wireless communication quality over time. The system records data at regular time intervals, producing 500 observations that represent how the system behaves during operation. Such experiments are commonly used in electronics engineering, IoT system design, sensor fusion, and data analysis training.

The data includes environmental sensors such as temperature, humidity, atmospheric pressure, light intensity, and gas concentration to study surroundings and safety conditions. Motion sensors like accelerometer and gyroscope measure linear acceleration and rotational movement, helping analyze vibration, orientation, or motion events. Electrical parameters such as voltage and current are recorded to evaluate power consumption and energy efficiency of the system. Additionally, Wi-Fi RSSI (signal strength) data is captured to understand wireless connectivity performance, which is crucial for IoT applications. Overall, this dataset represents a complete cyber-physical system experiment, where hardware sensor data is collected and later analyzed using Python to extract insights, detect anomalies, and support intelligent decision-making.

Introduction

This notebook analyzes Arduino-based IoT sensor data using Python. The goal is to explore sensor behavior, perform time-series analysis, and generate meaningful visualizations for learning and documentation.

Uploading the Dataset

The dataset is uploaded into Google Colab to enable analysis using Python data analysis libraries.

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving arduino_complex__data.xlsx to arduino_complex__data.xlsx

Importing Libraries

Pandas, NumPy, and Matplotlib are used for data analysis, numerical computation, and visualization.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Loading the Dataset

1. The sensor data is loaded into a Pandas DataFrame for structured analysis and processing.
2. The first few rows are displayed to understand the dataset structure and sensor readings.

```
df = pd.read_excel("arduino_complex__data.xlsx")
df.head()
```

Show hidden output

Dataset Information

1. This step checks data types, column details, and the presence of missing values.
2. Basic statistics are generated to understand the distribution and range of sensor values.

```
df.info()
df.describe()
```

Show hidden output

Timestamp Conversion

The timestamp column is converted to datetime format to enable time-series analysis.

```
df['timestamp'] = pd.to_datetime(df['timestamp'])
```

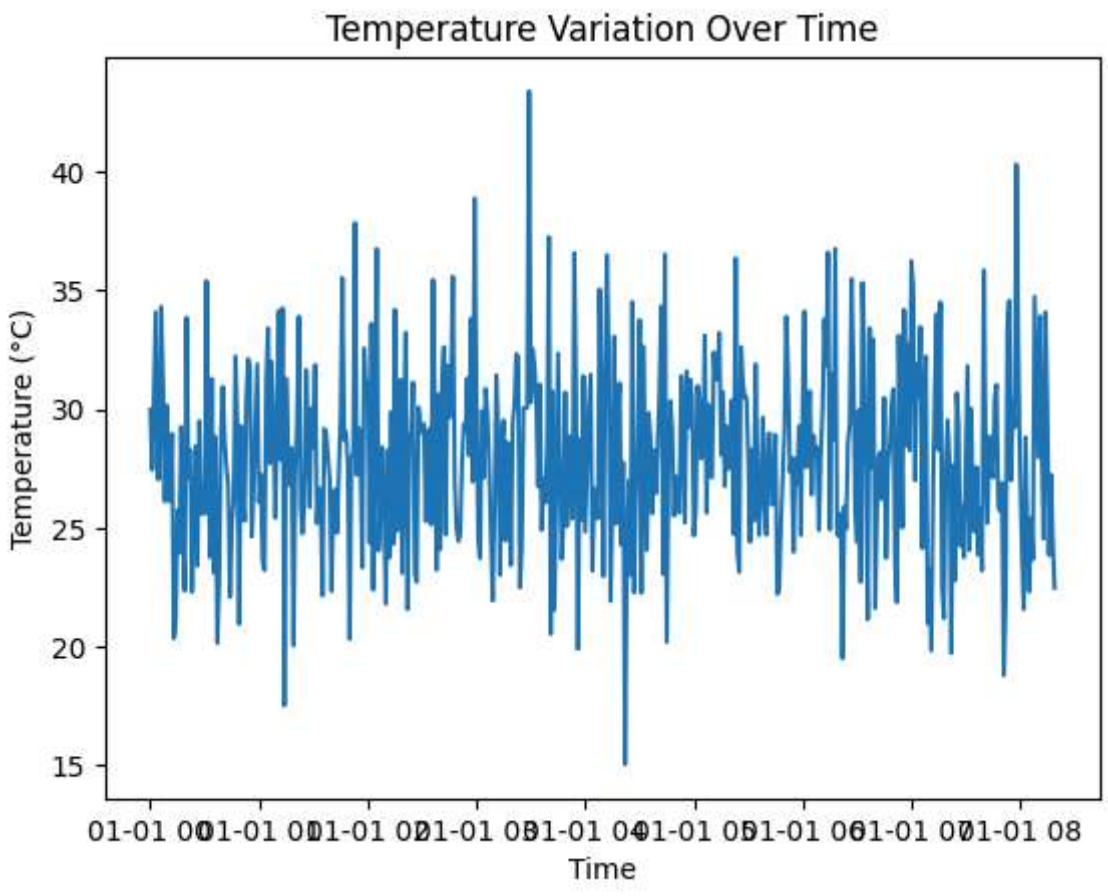
Temperature Trend Analysis

This plot shows temperature variation over time and helps identify abnormal fluctuations.

Insights

Temperature shows gradual variation over time, indicating stable environmental conditions without extreme fluctuations.

```
plt.figure()
plt.plot(df['timestamp'], df['temperature_c'])
plt.xlabel("Time")
plt.ylabel("Temperature (°C)")
plt.title("Temperature Variation Over Time")
plt.show()
```



Correlation Analysis

Correlation is calculated to understand relationships between environmental sensor readings.

Insights

Temperature and humidity show moderate correlation, while gas concentration appears mostly independent, which is expected in real-world environments.

```
correlation = df[['temperature_c','humidity_percent','gas_ppm','pressure_hpa']]
correlation
```

	temperature_c	humidity_percent	gas_ppm	pressure_hpa
temperature_c	1.000000	-0.010711	0.021351	0.005454
humidity_percent	-0.010711	1.000000	0.002664	0.001471
gas_ppm	0.021351	0.002664	1.000000	0.025878
pressure_hpa	0.005454	0.001471	0.025878	1.000000

Power Consumption Calculation

Power consumption is calculated using voltage and current values to analyze system behavior.

```
df['power_w'] = df['voltage_v'] * (df['current_ma'] / 1000)
```

Power Consumption Trend

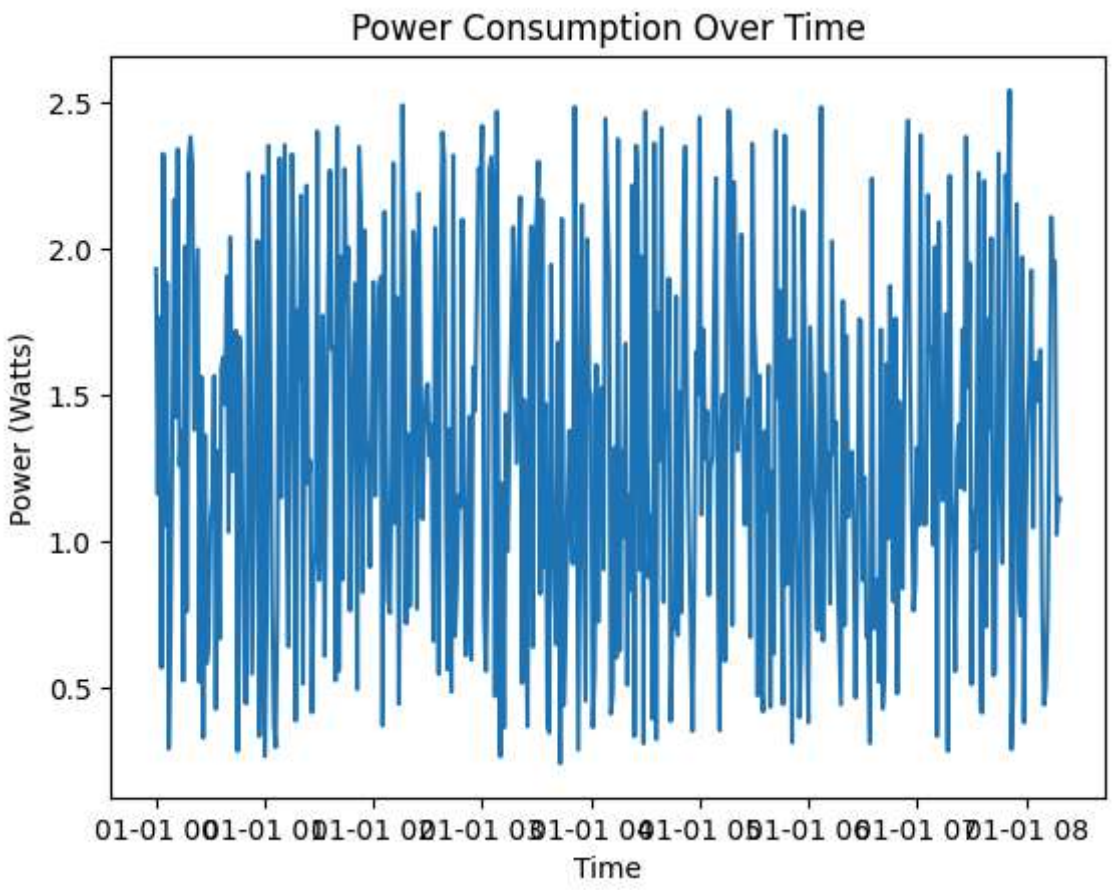
This visualization shows power usage over time and helps detect abnormal power spikes.

Insight:

Power consumption remains relatively stable, indicating efficient energy usage and no abnormal current spikes.

```
plt.figure()
plt.plot(df['timestamp'], df['power_w'])
```

```
plt.xlabel("Time")
plt.ylabel("Power (Watts)")
plt.title("Power Consumption Over Time")
plt.show()
```



Motion Detection

Acceleration magnitude is computed to detect motion or vibration events.

```
df['accel_magnitude'] = np.sqrt(
    df['accel_x']**2 +
    df['accel_y']**2 +
    df['accel_z']**2
)
```

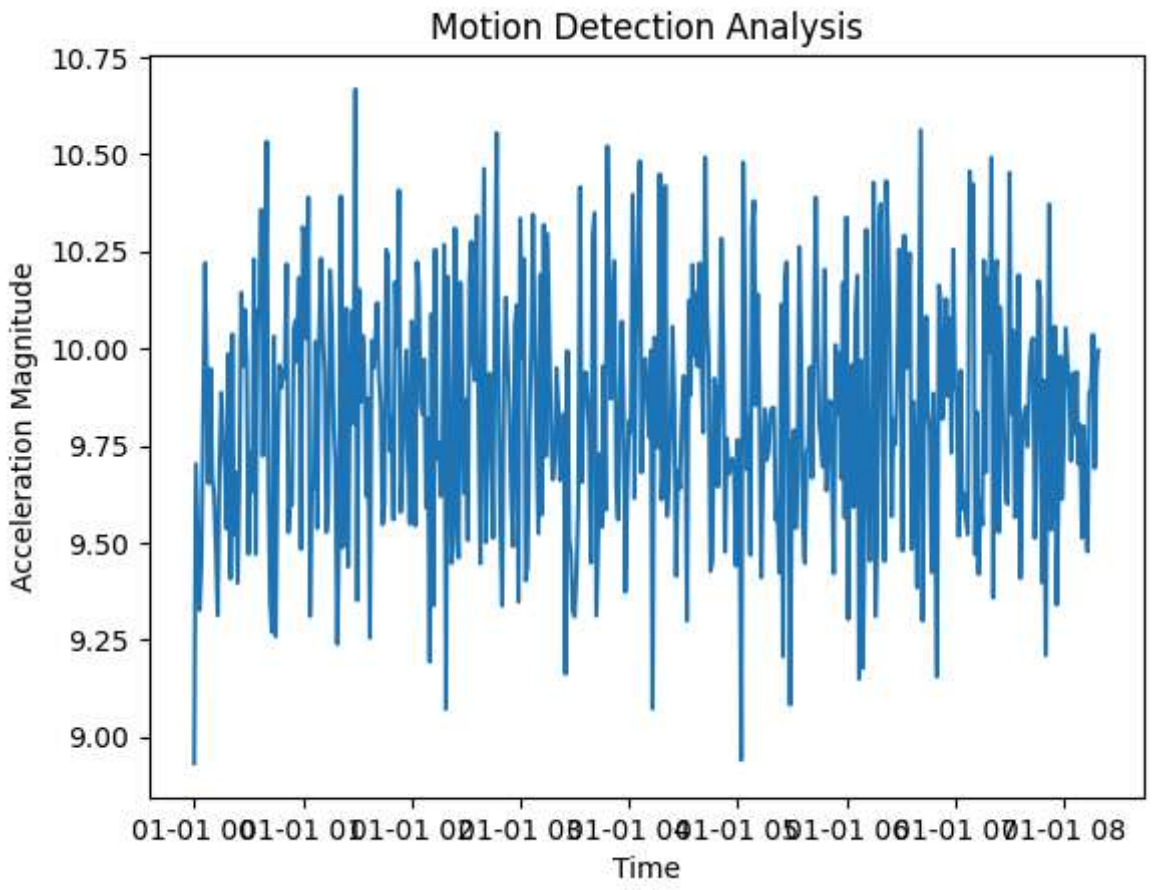
Motion Visualization

The acceleration magnitude is plotted over time to identify movement patterns.

Insight:

Sudden peaks in acceleration magnitude indicate movement or vibration events detected by the sensor.

```
plt.figure()
plt.plot(df['timestamp'], df['accel_magnitude'])
plt.xlabel("Time")
plt.ylabel("Acceleration Magnitude")
plt.title("Motion Detection Analysis")
plt.show()
```



Gas Sensor Anomaly Detection

A statistical threshold is used to detect abnormally high gas sensor readings.

Insights:

Gas concentration anomalies were detected using a statistical threshold, which can be useful for early gas leakage or pollution alerts in IoT systems.

```
threshold = df['gas_ppm'].mean() + 2 * df['gas_ppm'].std()
anomalies = df[df['gas_ppm'] > threshold]
```

```
anomalies[['timestamp', 'gas_ppm']]
```

timestamp	gas_ppm

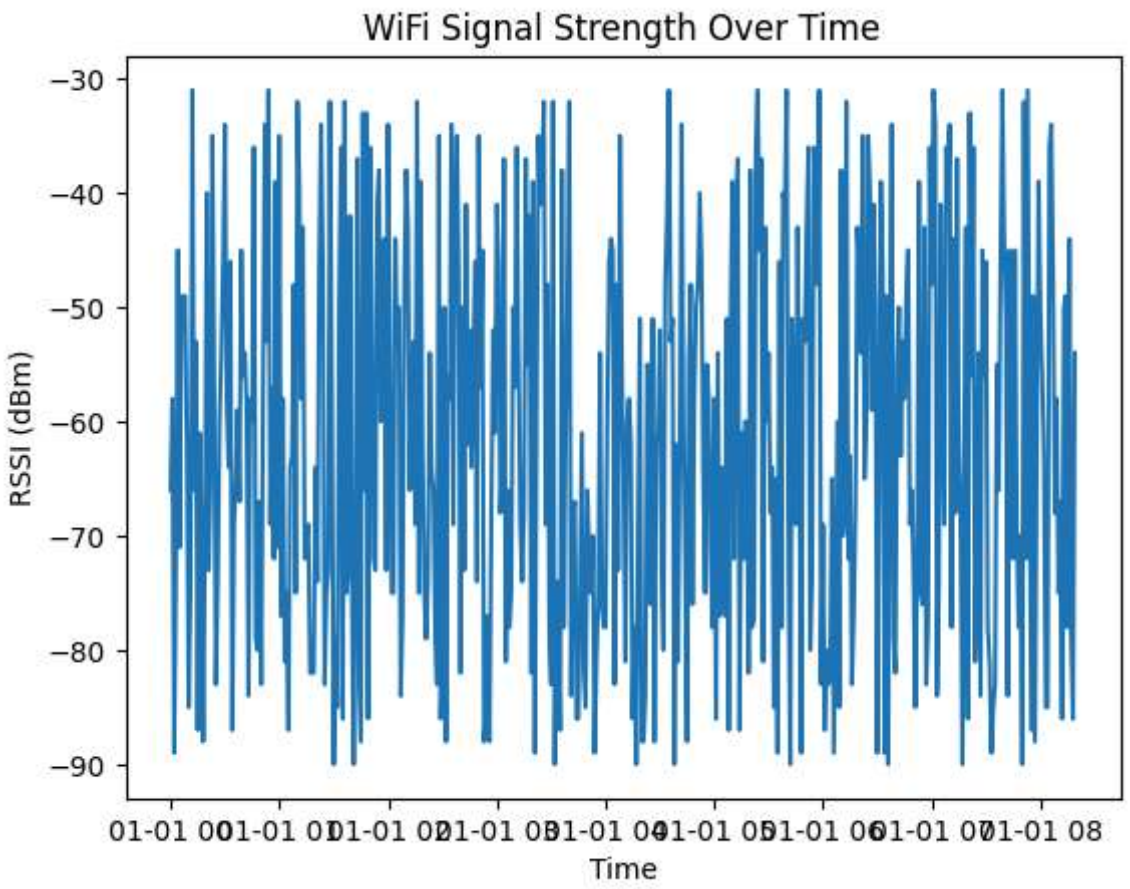
WiFi Signal Analysis

This plot evaluates WiFi signal strength and network stability over time.

Insight:

WiFi signal strength fluctuates within an acceptable range, indicating stable network connectivity for IoT communication.

```
plt.figure()
plt.plot(df['timestamp'], df['wifi_rssi_dbm'])
plt.xlabel("Time")
plt.ylabel("RSSI (dBm)")
plt.title("WiFi Signal Strength Over Time")
plt.show()
```



Saving Processed Data

The processed dataset is saved for future use and documentation.

```
df.to_csv("processed_arduino_data.csv", index=False)
```

Final Summary

This notebook demonstrates a complete workflow for analyzing Arduino sensor data using Python.

The data analysis task was successfully completed by our team member, Shweta.