Task 1: Sensor Data Parsing and Separation

Code

```
import os
import math
import csv

# Set the path for the input log file
logfile_path = r'C:\Users\Asus\PycharmProjects\Assignment1\logfile_2024_09_23_11_58_27.txt'

# Create directories to store the text and CSV files
txt_output_dir = 'sensor_data_txt'
csv_output_dir = 'sensor_data_csv'
os.makedirs(txt_output_dir, exist_ok=True)
os.makedirs(csv_output_dir, exist_ok=True)

# Dictionary to hold file handles for text files based on sensor prefixes
sensor_txt_files = {}

# Function to compute the magnitude from x, y, and z components
def calculate_magnitude(x, y, z):
    return math.sqrt(x ** 2 + y ** 2 + z ** 2)

# Dictionary to hold CSV writers and their file handles for each sensor type
sensor_csv_files = {}
csv_headers = ['AppTimestamp(s)', 'SensorTimestamp(s)', 'X', 'Y', 'Z', 'Accuracy',
'Magnitude']

try:
    # Open the log file to read data
    with open(logfile_path, 'r') as log_file:
        # Process the file line by line
        for line in log_file:
            # Skip lines starting with '%' as they are comments or empty
            if line.startswith('%') or not line.strip():
                continue

            # Determine the sensor type from the first four characters
            sensor_type = line[:4]

            # Set the output file names based on the sensor type
            if sensor_type == 'ACCE':
                file_name = 'Accelerometer.txt'
                csv_file_name = 'Accelerometer.csv'
            elif sensor_type == 'GYRO':
                file_name = 'Gyroscope.txt'
                csv_file_name = 'Gyroscope.csv'
            elif sensor_type == 'MAGN':
                file_name = 'Magnetometer.txt'
    csv_file_name = 'Magnetometer.csv'
elif sensor_type == 'PRES':
    file_name = 'Pressure.txt'  # No magnitude needed
elif sensor_type == 'LIGH':
    file_name = 'Light.txt'  # No magnitude needed
elif sensor_type == 'PROX':
    file_name = 'Proximity.txt'
elif sensor_type == 'HUMI':
    file_name = 'Humidity.txt'
elif sensor_type == 'TEMP':
    file_name = 'Temperature.txt'
elif sensor_type == 'AHRS':
    file_name = 'Orientation.txt'
    csv_file_name = 'Orientation.csv'
elif sensor_type == 'GNSS':
    file_name = 'GPS.txt'
elif sensor_type == 'WIFI':
    file_name = 'WiFi.txt'
elif sensor_type == 'BLUE':
    file_name = 'Bluetooth.txt'
elif sensor_type == 'SOUN':
    file_name = 'Sound.txt'
elif sensor_type == 'RFID':
    file_name = 'RFID.txt'
elif sensor_type == 'IMUX':
    file_name = 'IMU_XSens.txt'
    csv_file_name = 'IMU_XSens.csv'
elif sensor_type == 'IMUL':
    file_name = 'IMU_LPMS.txt'
    csv_file_name = 'IMU_LPMS.csv'
else:
    # Ignore any unsupported sensor types
    continue

# Construct the full path for the text output file
full_txt_file_path = os.path.join(txt_output_dir, file_name)

# Write to the corresponding text file
if full_txt_file_path not in sensor_txt_files:
    sensor_txt_files[full_txt_file_path] = open(full_txt_file_path, 'w')
sensor_txt_files[full_txt_file_path].write(line)

# For sensors with X, Y, Z values, compute the magnitude and write to CSV
if sensor_type in ['ACCE', 'GYRO', 'MAGN']:
    # Split the line into data fields (assumed to be semi-colon separated)
    data = line.strip().split(';')

    try:
        app_timestamp = data[1]
        sensor_timestamp = data[2]
        x = float(data[3])
        y = float(data[4])
        z = float(data[5])
```

```
                    accuracy = data[6] if len(data) > 6 else None

                    # Calculate the magnitude
                    magnitude = calculate_magnitude(x, y, z)

                    # Write the data to the corresponding CSV file
                    full_csv_file_path = os.path.join(csv_output_dir, csv_file_name)

                    if full_csv_file_path not in sensor_csv_files:
                        csv_file = open(full_csv_file_path, 'w', newline='')
                        writer = csv.writer(csv_file)
                        writer.writerow(csv_headers)
                        sensor_csv_files[full_csv_file_path] = (csv_file, writer)  # Store the
file and writer

                    # Log the data in CSV format
                    sensor_csv_files[full_csv_file_path][1].writerow(
                        [app_timestamp, sensor_timestamp, x, y, z, accuracy, magnitude])

                except ValueError:
                    # Handle any parsing errors gracefully
                    continue

finally:
    # Close all text and CSV files
    for file in sensor_txt_files.values():
        file.close()
    for csv_file, _ in sensor_csv_files.values():
        csv_file.close()  # Close the CSV file instead of the writer
```

Code Description

Importing Libraries: The code begins by importing the following libraries: csv to interact with CSV files, math to do mathematical computations, and os to handle files.

Setting Up File Paths: This sets the location of the sensor data-containing input log file. Two directories are created, one for text files and the other for CSV files. It won't create these folders again if they already exist.

Defining Functions and Variables: To manage text files for various sensors, a dictionary named sensor_txt_files is made. To determine a vector's magnitude (or size) from its x, y, and z components, a function called calculate_magnitude is defined. To manage the CSV files and the writers that go with them, sensor_csv_files is another dictionary built. Additionally, a set of headers for the CSV files is ready.

Reading the Log File: After the log file has been opened, the code reads each line by line. Lines that begin with % (comments) or are blank are skipped.

Sensor Type Identification: To identify the type of sensor, the code looks up the first four characters of each line. The output text and CSV file names are determined by the type of sensor.

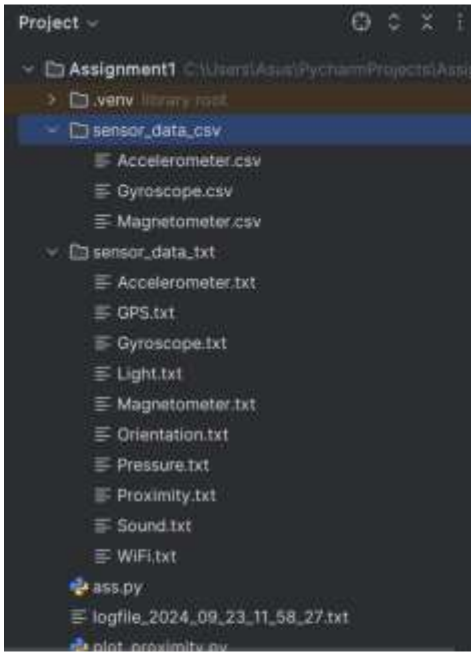
Writing Data to Text Files: It creates the complete path for the text output file for every line and writes that line to the relevant text file.

Managing Sensors that Have X, Y, and Z Values: The algorithm divides the line into separate data points for sensor types (such as accelerometers, gyroscopes, and magnetometers) that provide x, y, and z data. Time stamps, x, y, and z values, as well as precision, are extracted from the data. The calculate_magnitude function is then used by the code to determine the magnitude of the sensor data.

Writing Data to CSV Files: The whole path for the CSV output file is created. It creates the file and writes the headers if it doesn't already exist. Lastly, it uploads the extracted information to the relevant CSV file, along with the magnitude that was computed.

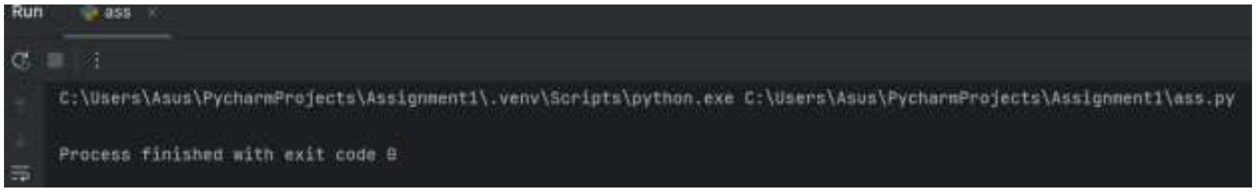
Closing Files: To prevent data loss or file corruption, the code ensures that all opened text and CSV files are correctly closed after processing every line.

<u>Screenshots of Csv files:</u>



```
Project
Assignment1 C:\Users\Asus\PycharmProjects\Ass
  .venv library root
  sensor_data_csv
    Accelerometer.csv
    Gyroscope.csv
    Magnetometer.csv
  sensor_data_txt
    Accelerometer.txt
    GPS.txt
    Gyroscope.txt
    Light.txt
    Magnetometer.txt
    Orientation.txt
    Pressure.txt
    Proximity.txt
    Sound.txt
    WiFi.txt
  ass.py
  logfile_2024_09_23_11_58_27.txt
  plot_proximity.py
```

<u>Output</u>

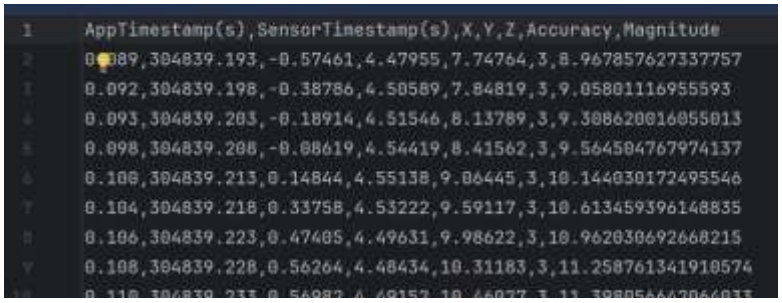


```
C:\Users\Asus\PycharmProjects\Assignment1\.venv\Scripts\python.exe C:\Users\Asus\PycharmProjects\Assignment1\ass.py

Process finished with exit code 0
```

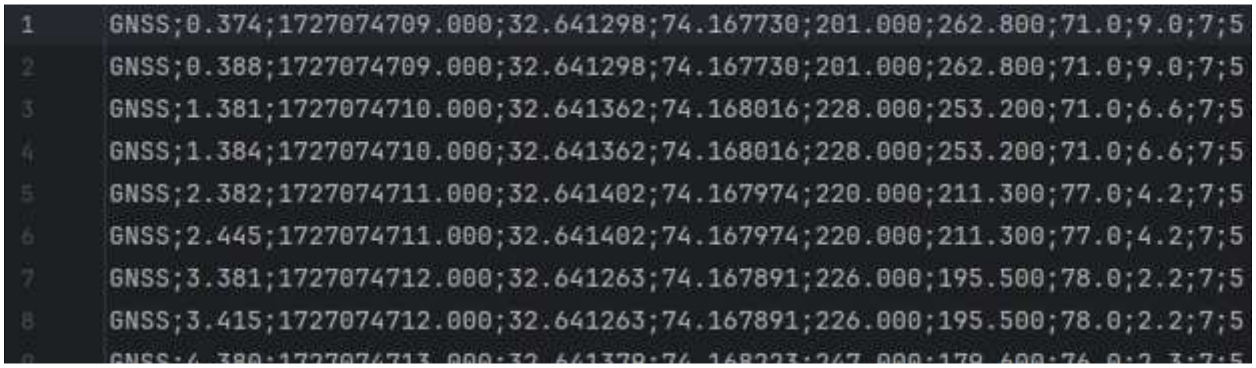
<u>Files data</u>

1. Accelerometer



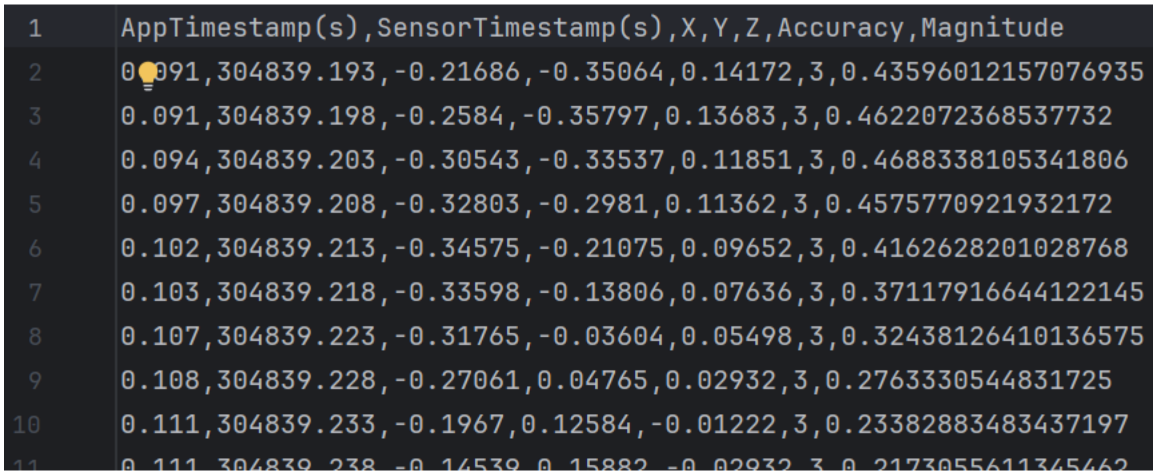
```
AppTimestamp(s),SensorTimestamp(s),X,Y,Z,Accuracy,Magnitude
0.089,304839.193,-0.57461,4.47955,7.74764,3,8.967857627337757
0.092,304839.198,-0.38786,4.50589,7.84819,3,9.05801116955593
0.093,304839.203,-0.18914,4.51546,8.13789,3,9.308620016055013
0.098,304839.208,-0.08619,4.54419,8.41562,3,9.564504767974137
0.100,304839.213,0.14844,4.55138,9.06445,3,10.144030172495546
0.104,304839.218,0.33758,4.53222,9.59117,3,10.613459396148835
0.106,304839.223,0.47405,4.49631,9.98622,3,10.962030692668215
0.108,304839.228,0.56264,4.48434,10.31183,3,11.258761341910574
0.110,304839.233,0.56982,4.49157,10.46077,3,11.398056647064033
```

2. GPS



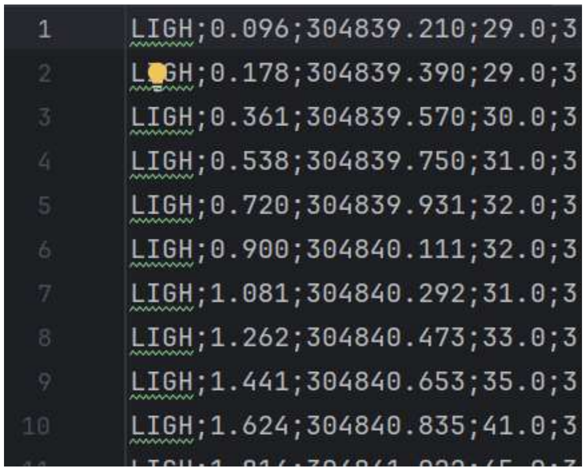
```
GNSS;0.374;1727074709.000;32.641298;74.167730;201.000;262.800;71.0;9.0;7;5
GNSS;0.388;1727074709.000;32.641298;74.167730;201.000;262.800;71.0;9.0;7;5
GNSS;1.381;1727074710.000;32.641362;74.168016;228.000;253.200;71.0;6.6;7;5
GNSS;1.384;1727074710.000;32.641362;74.168016;228.000;253.200;71.0;6.6;7;5
GNSS;2.382;1727074711.000;32.641402;74.167974;220.000;211.300;77.0;4.2;7;5
GNSS;2.445;1727074711.000;32.641402;74.167974;220.000;211.300;77.0;4.2;7;5
GNSS;3.381;1727074712.000;32.641263;74.167891;226.000;195.500;78.0;2.2;7;5
GNSS;3.415;1727074712.000;32.641263;74.167891;226.000;195.500;78.0;2.2;7;5
GNSS;4.380;1727074713.000;32.641378;74.168223;247.000;178.600;76.0;2.3;7;5
```

3. GYROSCOPE



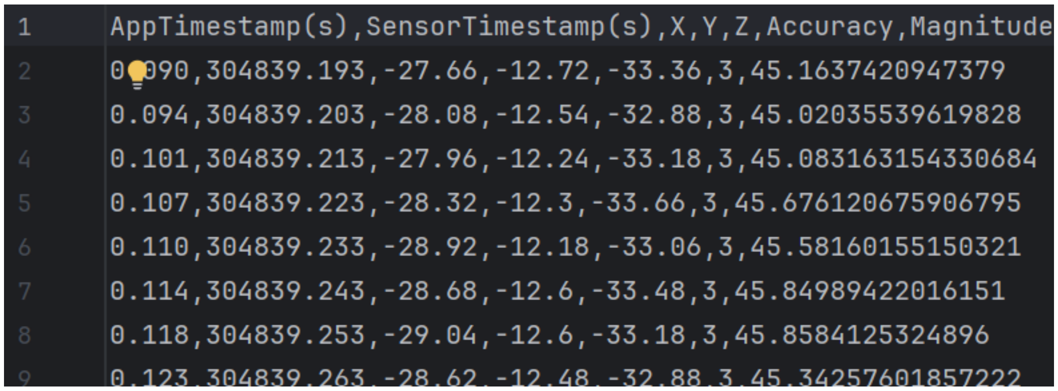
```
AppTimestamp(s),SensorTimestamp(s),X,Y,Z,Accuracy,Magnitude
0.091,304839.193,-0.21686,-0.35064,0.14172,3,0.43596012157076935
0.091,304839.198,-0.2584,-0.35797,0.13683,3,0.4622072368537732
0.094,304839.203,-0.30543,-0.33537,0.11851,3,0.4688338105341806
0.097,304839.208,-0.32803,-0.2981,0.11362,3,0.4575770921932172
0.102,304839.213,-0.34575,-0.21075,0.09652,3,0.4162628201028768
0.103,304839.218,-0.33598,-0.13806,0.07636,3,0.37117916644122145
0.107,304839.223,-0.31765,-0.03604,0.05498,3,0.32438126410136575
0.108,304839.228,-0.27061,0.04765,0.02932,3,0.2763330544831725
0.111,304839.233,-0.1967,0.12584,-0.01222,3,0.233828834834437197
0.111,304839.238,-0.14539,0.15882,-0.02932,3,0.2173055611345462
```

4. LIGHT



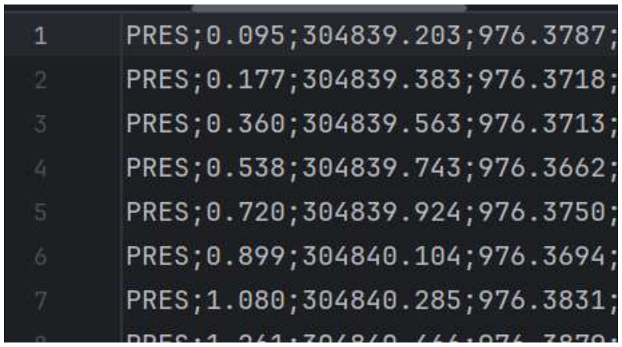
```
LIGH;0.096;304839.210;29.0;3
LIGH;0.178;304839.390;29.0;3
LIGH;0.361;304839.570;30.0;3
LIGH;0.538;304839.750;31.0;3
LIGH;0.720;304839.931;32.0;3
LIGH;0.900;304840.111;32.0;3
LIGH;1.081;304840.292;31.0;3
LIGH;1.262;304840.473;33.0;3
LIGH;1.441;304840.653;35.0;3
LIGH;1.624;304840.835;41.0;3
```

5. MAGNETOMETER



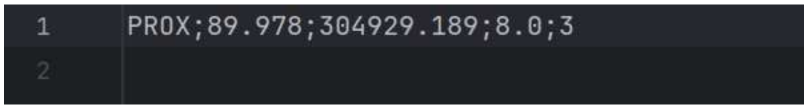
```
AppTimestamp(s),SensorTimestamp(s),X,Y,Z,Accuracy,Magnitude
0.090,304839.193,-27.66,-12.72,-33.36,3,45.1637420947379
0.094,304839.203,-28.08,-12.54,-32.88,3,45.02035539619828
0.101,304839.213,-27.96,-12.24,-33.18,3,45.083163154330684
0.107,304839.223,-28.32,-12.3,-33.66,3,45.676120675906795
0.110,304839.233,-28.92,-12.18,-33.06,3,45.58160155150321
0.114,304839.243,-28.68,-12.6,-33.48,3,45.84989422016151
0.118,304839.253,-29.04,-12.6,-33.18,3,45.8584125324896
0.123,304839.263,-28.62,-12.48,-32.88,3,45.34257601857222
```

6. PRESSURE



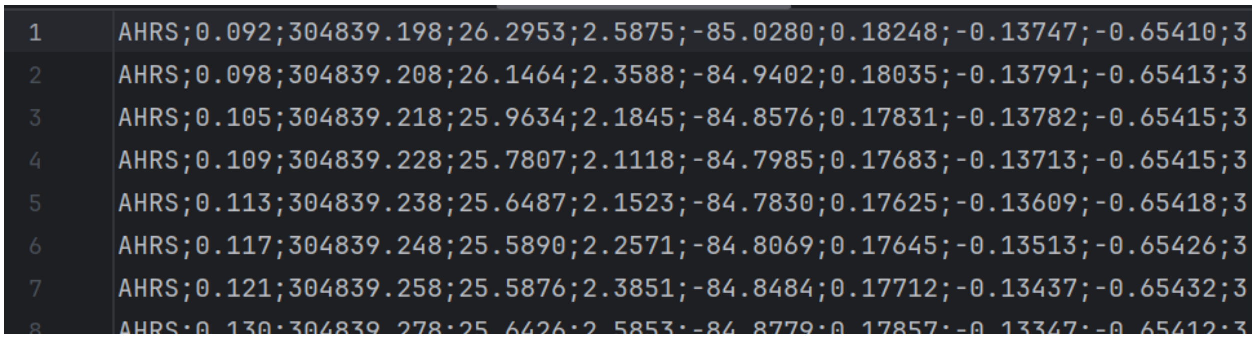
```
PRES;0.095;304839.203;976.3787;
PRES;0.177;304839.383;976.3718;
PRES;0.360;304839.563;976.3713;
PRES;0.538;304839.743;976.3662;
PRES;0.720;304839.924;976.3750;
PRES;0.899;304840.104;976.3694;
PRES;1.080;304840.285;976.3831;
```

7. PROXIMETER



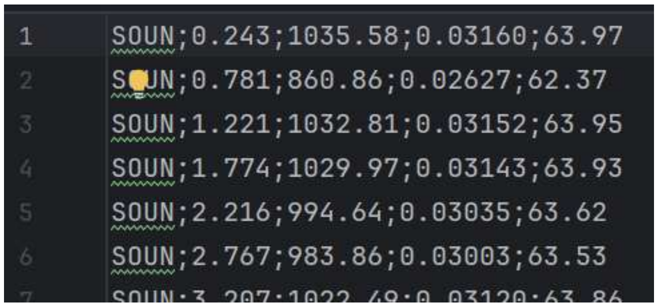
```
PROX;89.978;304929.189;8.0;3
```

8. ORIENTATION



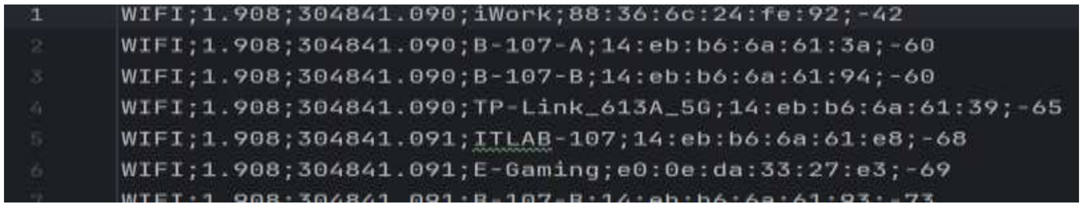
```
AHRS;0.092;304839.198;26.2953;2.5875;-85.0280;0.18248;-0.13747;-0.65410;3
AHRS;0.098;304839.208;26.1464;2.3588;-84.9402;0.18035;-0.13791;-0.65413;3
AHRS;0.105;304839.218;25.9634;2.1845;-84.8576;0.17831;-0.13782;-0.65415;3
AHRS;0.109;304839.228;25.7807;2.1118;-84.7985;0.17683;-0.13713;-0.65415;3
AHRS;0.113;304839.238;25.6487;2.1523;-84.7830;0.17625;-0.13609;-0.65418;3
AHRS;0.117;304839.248;25.5890;2.2571;-84.8069;0.17645;-0.13513;-0.65426;3
AHRS;0.121;304839.258;25.5876;2.3851;-84.8484;0.17712;-0.13437;-0.65432;3
AHRS;0.130;304839.278;25.6426;2.5853;-84.8779;0.17857;-0.13347;-0.65412;3
```

9. SOUND



```
SOUN;0.243;1035.58;0.03160;63.97
SOUN;0.781;860.86;0.02627;62.37
SOUN;1.221;1032.81;0.03152;63.95
SOUN;1.774;1029.97;0.03143;63.93
SOUN;2.216;994.64;0.03035;63.62
SOUN;2.767;983.86;0.03003;63.53
SOUN;3.207;1022.49;0.03120;63.86
```

10. WIFI



```
WIFI;1.908;304841.090;iWork;88:36:6c:24:fe:92;-42
WIFI;1.908;304841.090;B-107-A;14:eb:b6:6a:61:3a;-60
WIFI;1.908;304841.090;B-107-B;14:eb:b6:6a:61:94;-60
WIFI;1.908;304841.090;TP-Link_613A_5G;14:eb:b6:6a:61:39;-65
WIFI;1.908;304841.091;ITLAB-107;14:eb:b6:6a:61:e8;-68
WIFI;1.908;304841.091;E-Gaming;e0:0e:da:33:27:e3;-69
WIFI;1.908;304841.091;B-107-B;14:eb:b6:6a:61:93;-73
```

Task 2: Sensor Data Analysis and Visualization

Code

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

# Load data from CSV files
df_proximity = pd.read_csv('proximity_sensor_data.csv')
df_gesture = pd.read_csv('proximity_with_gesture_data.csv')
df_screen = pd.read_csv('screen_proximity_data.csv')
df_detection = pd.read_csv('object_detection_data.csv')

# Convert 'Timestamp' column to datetime for proper plotting
df_proximity['Timestamp'] = pd.to_datetime(df_proximity['Timestamp'])
df_gesture['Timestamp'] = pd.to_datetime(df_gesture['Timestamp'])
df_screen['Timestamp'] = pd.to_datetime(df_screen['Timestamp'])
df_detection['Timestamp'] = pd.to_datetime(df_detection['Timestamp'])

# --- 1. Proximity Sensor Data Over Time ---
plt.figure(figsize=(10, 6))
plt.plot(df_proximity['Timestamp'], df_proximity['Proximity Value'], label='Proximity Value',
color='b', marker='o')
plt.xlabel('Time', fontsize=12)
plt.ylabel('Proximity Value', fontsize=12)
plt.title('Proximity Sensor Data Over Time', fontsize=14)
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# --- 2. Proximity with Gesture Recognition Over Time ---
plt.figure(figsize=(10, 6))
for gesture in df_gesture['Gesture Type'].unique():
    subset = df_gesture[df_gesture['Gesture Type'] == gesture]
    plt.plot(subset['Timestamp'], subset['Proximity Value'], marker='o', label=gesture)
plt.xticks(rotation=45)
plt.xlabel('Time', fontsize=12)
plt.ylabel('Proximity Value', fontsize=12)
plt.title('Proximity with Gesture Recognition Over Time', fontsize=14)
plt.legend(title='Gesture Type')
plt.grid(True)
plt.tight_layout()
plt.show()

# --- 3. Screen On/Off Transitions and Proximity Over Time ---
plt.figure(figsize=(10, 6))
colors = {'Off': 'red', 'On': 'green'}
for status in df_screen['Screen On/Off'].unique():
    subset = df_screen[df_screen['Screen On/Off'] == status]
    plt.plot(subset['Timestamp'], subset['Proximity Value'], marker='o', color=colors[status],
label=status)
```

```
plt.xticks(rotation=45)
plt.xlabel('Time', fontsize=12)
plt.ylabel('Proximity Value', fontsize=12)
plt.title('Screen On/Off Status with Proximity Over Time', fontsize=14)
plt.legend(title='Screen On/Off')
plt.grid(True)
plt.tight_layout()
plt.show()

# --- 4. Object Detection (Scatter Plot) ---
plt.figure(figsize=(10, 6))
for obj in df_detection['Object Detected'].unique():
    subset = df_detection[df_detection['Object Detected'] == obj]
    plt.scatter(subset['Timestamp'], subset['Proximity Value'], label=obj, s=100)
plt.xticks(rotation=45)
plt.xlabel('Time', fontsize=12)
plt.ylabel('Proximity Value', fontsize=12)
plt.title('Proximity Values with Object Detection Over Time', fontsize=14)
plt.grid(True)
plt.tight_layout()
plt.legend(title='Object Detected')
plt.show()
```

CODE EXPLANATION

Four CSV files containing sensor data are read by this Python script, which then creates four graphs to show various elements of proximity sensor data over time.

1. Loading Information

Four CSV files are read by the script to get data:

- priciency_sensor_data.csv (for values of the proximity sensor over time)
- Proximity with gesture data can be found in the fileproximity_with_gesture_data.csv.
- screen_proximity_data.csv (which includes screen on/off status information)
- object_detection_data.csv (for object detection status-associated proximity data)

To plot the timestamp on the x-axis, each file's timestamp column is transformed to a datetime format.

2. Plotting Proximity Sensor Data Over Time: The proximity sensor data are displayed over time in the first graph. On the x-axis, the timestamp is plotted, and on the y-axis, the proximity value. The data points have been marked with blue circles (o).

3. Plotting Proximity with Gesture detection: Values for proximity are plotted alongside gesture detection in the second graph. It depicts each gesture independently using distinct markers after classifying the data according to Gesture Type (e.g., "Swipe Left," "Tap").

4. Plotting Proximity and Screen On/Off Transitions: The third graph displays the proximity values according to the screen's state of "On" or "Off." "Off" and "On" are represented by

various colours, with red denoting "Off" and green denoting "On."
5. Plotting Proximity and Object Detection: The fourth graph is a scatter plot that displays proximity values in the presence or absence of objects. The labels "Yes" (object identified) and "No" (no item detected) are used to indicate points.

ANALYSIS:

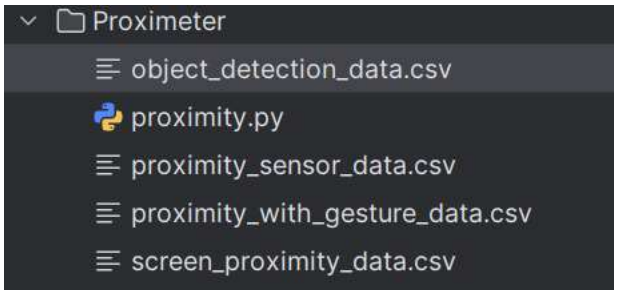
An extensive time-series visualisation of sensor data is provided by this code. Every graph has been meticulously designed to showcase distinct correlations between the several sensor outputs (such as proximity, gesture, screen on/off, and object detection) and time. While matplotlib offers flexible graphing, pandas facilitates simple data manipulation.
The following insights are offered by the visualisations:
1. Proximity Sensor Trends: Comprehending the temporal fluctuations in proximity values.

2. Gesture Impact on Proximity: Investigating the relationship between variations in proximity sensor readings and various gestures.

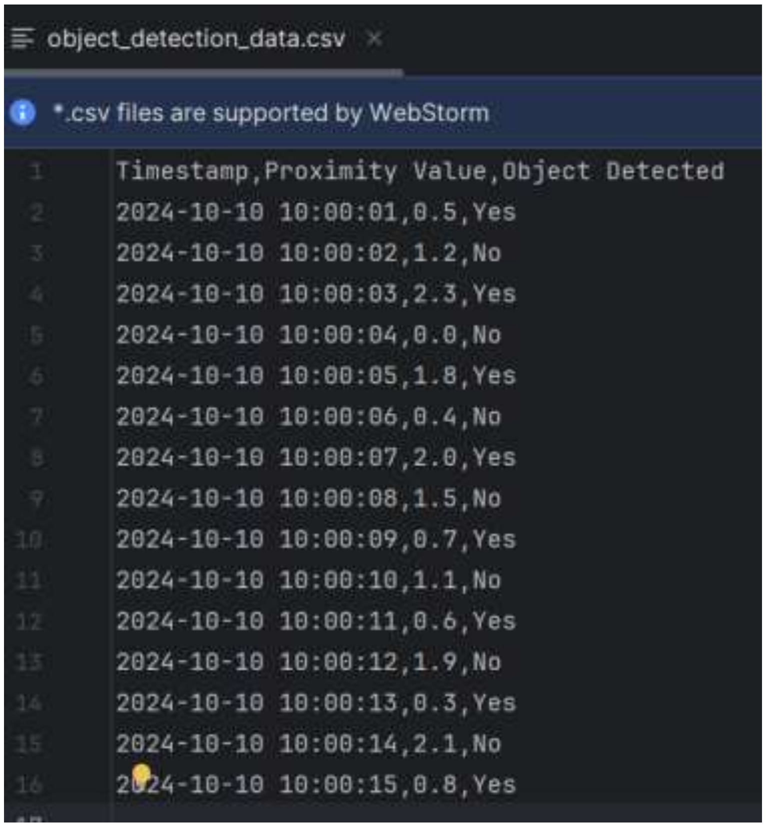
3. Screen and Proximity Interaction: Determining how proximity values and screen states (on/off) relate to one another.

4. Proximity and Object Detection: Finding trends in the data from proximity sensors and object detection events.

SCREENSHOTS:

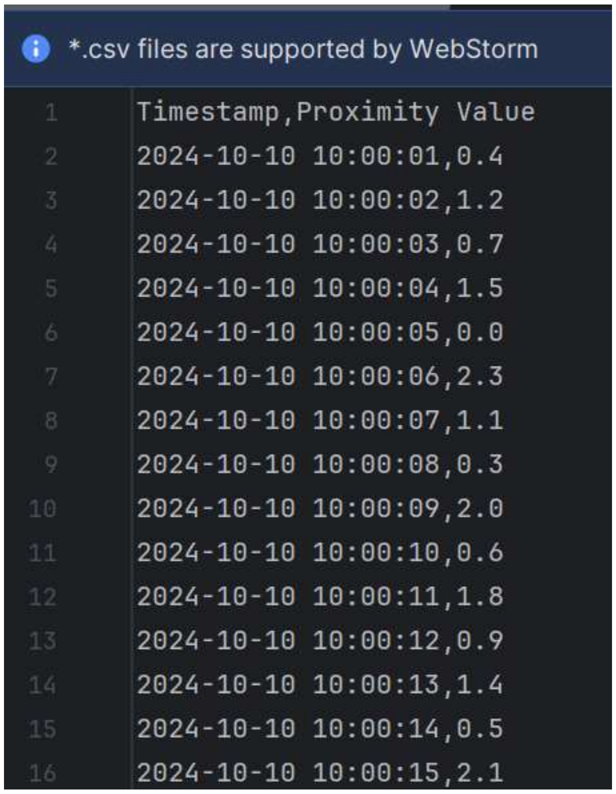


```
Proximeter
    object_detection_data.csv
    proximity.py
    proximity_sensor_data.csv
    proximity_with_gesture_data.csv
    screen_proximity_data.csv
```

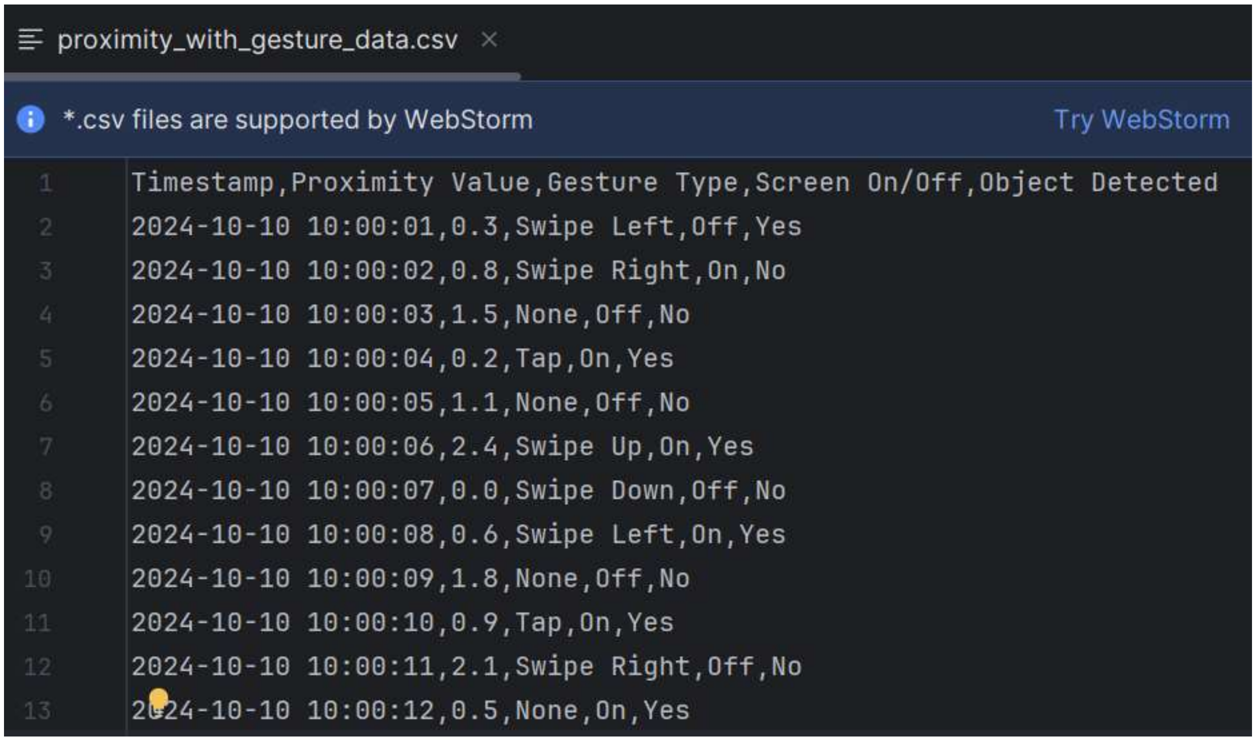

object_detection_data.csv

*.csv files are supported by WebStorm

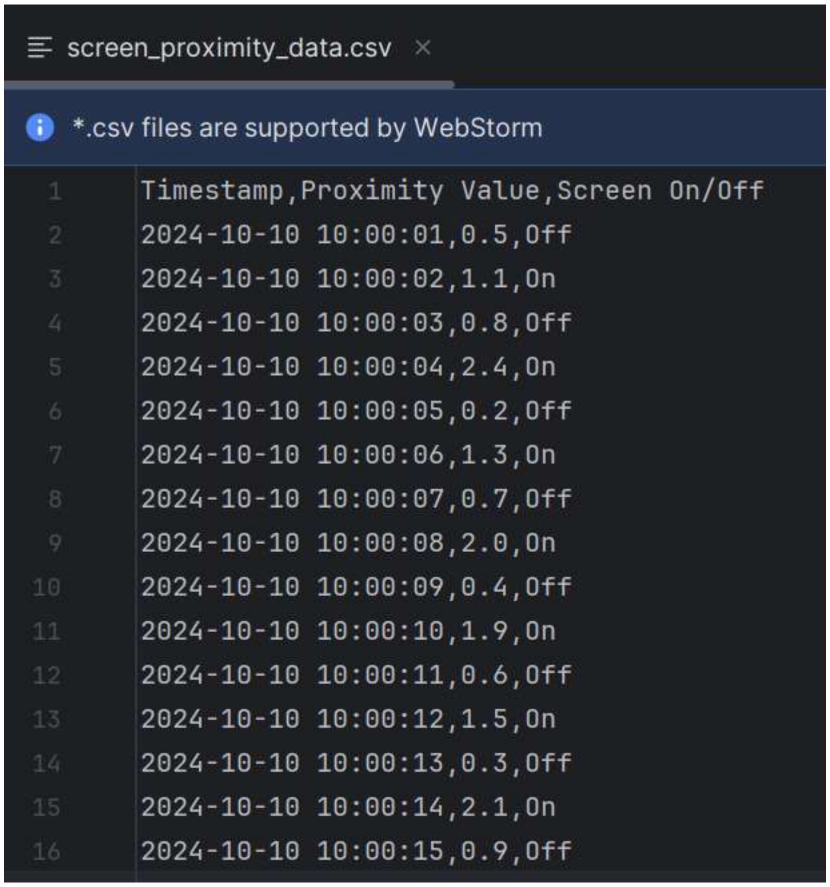
```
Timestamp,Proximity Value,Object Detected
2024-10-10 10:00:01,0.5,Yes
2024-10-10 10:00:02,1.2,No
2024-10-10 10:00:03,2.3,Yes
2024-10-10 10:00:04,0.0,No
2024-10-10 10:00:05,1.8,Yes
2024-10-10 10:00:06,0.4,No
2024-10-10 10:00:07,2.0,Yes
2024-10-10 10:00:08,1.5,No
2024-10-10 10:00:09,0.7,Yes
2024-10-10 10:00:10,1.1,No
2024-10-10 10:00:11,0.6,Yes
2024-10-10 10:00:12,1.9,No
2024-10-10 10:00:13,0.3,Yes
2024-10-10 10:00:14,2.1,No
2024-10-10 10:00:15,0.8,Yes
```


*.csv files are supported by WebStorm

```
Timestamp,Proximity Value
2024-10-10 10:00:01,0.4
2024-10-10 10:00:02,1.2
2024-10-10 10:00:03,0.7
2024-10-10 10:00:04,1.5
2024-10-10 10:00:05,0.0
2024-10-10 10:00:06,2.3
2024-10-10 10:00:07,1.1
2024-10-10 10:00:08,0.3
2024-10-10 10:00:09,2.0
2024-10-10 10:00:10,0.6
2024-10-10 10:00:11,1.8
2024-10-10 10:00:12,0.9
2024-10-10 10:00:13,1.4
2024-10-10 10:00:14,0.5
2024-10-10 10:00:15,2.1
```

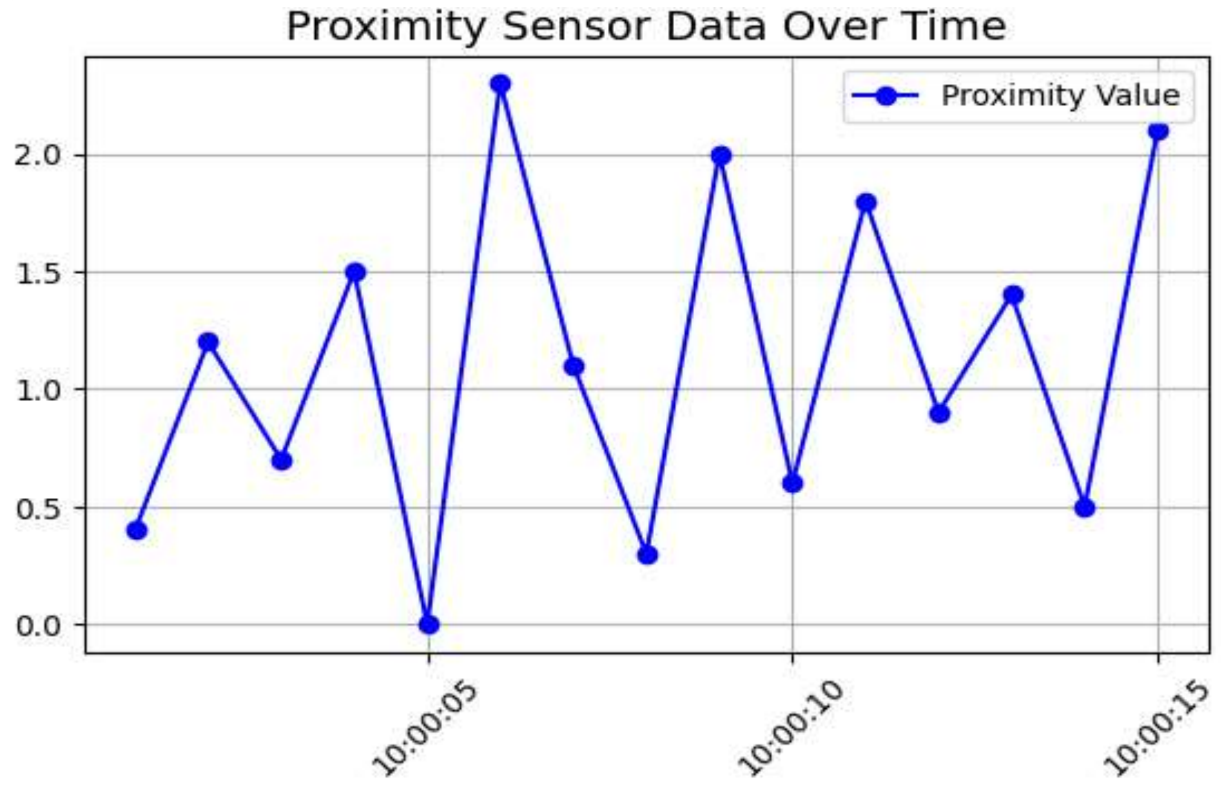

proximity_with_gesture_data.csv
*.csv files are supported by WebStorm
Try WebStorm

```
Timestamp,Proximity Value,Gesture Type,Screen On/Off,Object Detected
2024-10-10 10:00:01,0.3,Swipe Left,Off,Yes
2024-10-10 10:00:02,0.8,Swipe Right,On,No
2024-10-10 10:00:03,1.5,None,Off,No
2024-10-10 10:00:04,0.2,Tap,On,Yes
2024-10-10 10:00:05,1.1,None,Off,No
2024-10-10 10:00:06,2.4,Swipe Up,On,Yes
2024-10-10 10:00:07,0.0,Swipe Down,Off,No
2024-10-10 10:00:08,0.6,Swipe Left,On,Yes
2024-10-10 10:00:09,1.8,None,Off,No
2024-10-10 10:00:10,0.9,Tap,On,Yes
2024-10-10 10:00:11,2.1,Swipe Right,Off,No
2024-10-10 10:00:12,0.5,None,On,Yes
```


screen_proximity_data.csv
*.csv files are supported by WebStorm

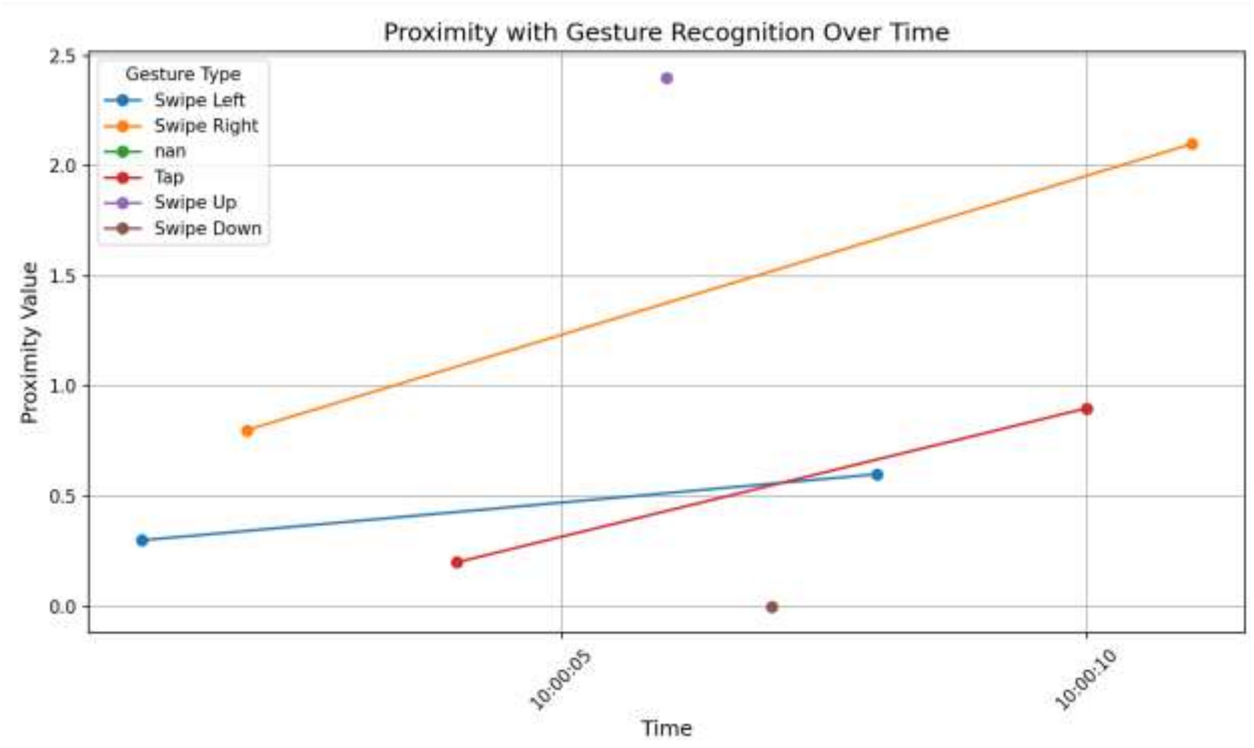
```
Timestamp,Proximity Value,Screen On/Off
2024-10-10 10:00:01,0.5,Off
2024-10-10 10:00:02,1.1,On
2024-10-10 10:00:03,0.8,Off
2024-10-10 10:00:04,2.4,On
2024-10-10 10:00:05,0.2,Off
2024-10-10 10:00:06,1.3,On
2024-10-10 10:00:07,0.7,Off
2024-10-10 10:00:08,2.0,On
2024-10-10 10:00:09,0.4,Off
2024-10-10 10:00:10,1.9,On
2024-10-10 10:00:11,0.6,Off
2024-10-10 10:00:12,1.5,On
2024-10-10 10:00:13,0.3,Off
2024-10-10 10:00:14,2.1,On
2024-10-10 10:00:15,0.9,Off
```

GRAPHS



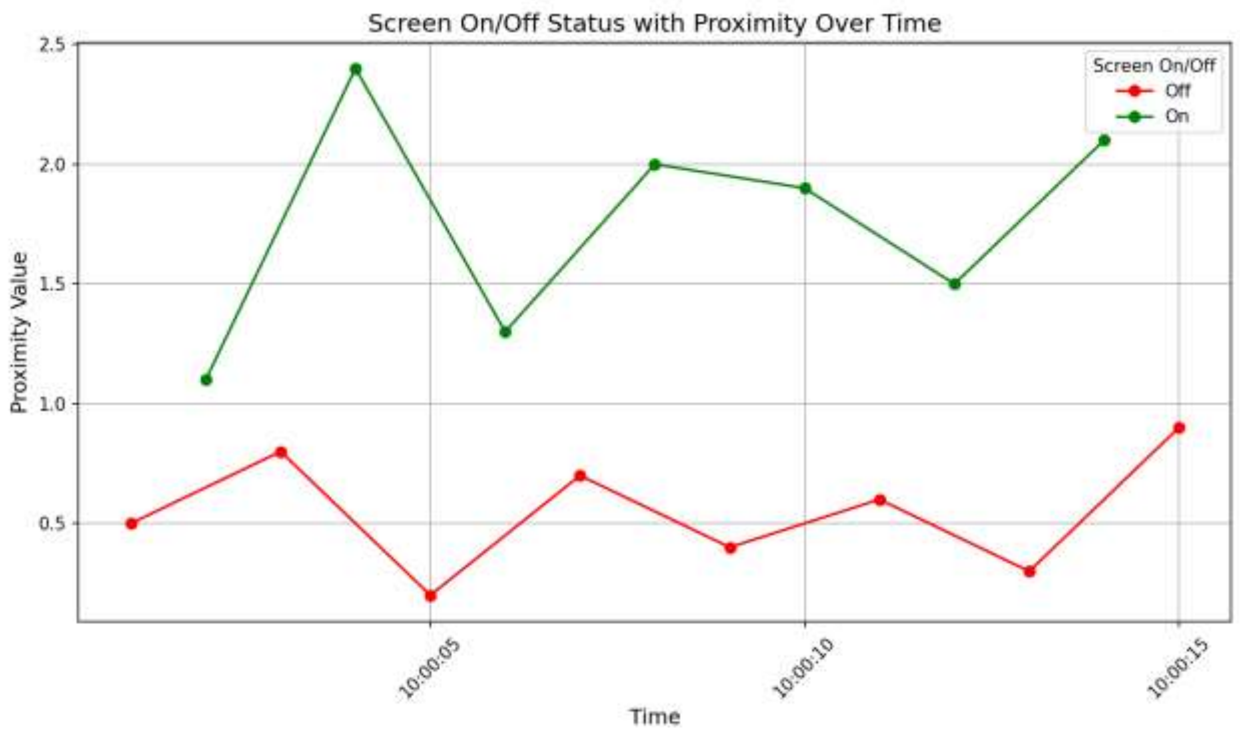
Explanation of graph:

Plotting the closeness values on the y-axis against time on the x-axis is what this line graph does.
Trends: The variation in closeness values over time is visible. For example, the proximity value may increase with item proximity and decrease with object removal or movement away.
High points and Low points: Peaks (greater proximity values) represent the times when the sensor senses anything nearby. For instance, you might spot a peak around 2024-10-10 10:00:03, which would indicate that an object was really close at that moment. Troughs indicate instances where either the distance was larger or no object was spotted.
General Patterns: Repetitive patterns, such as proximity spikes occurring on a regular basis, may indicate frequent interactions with an object.



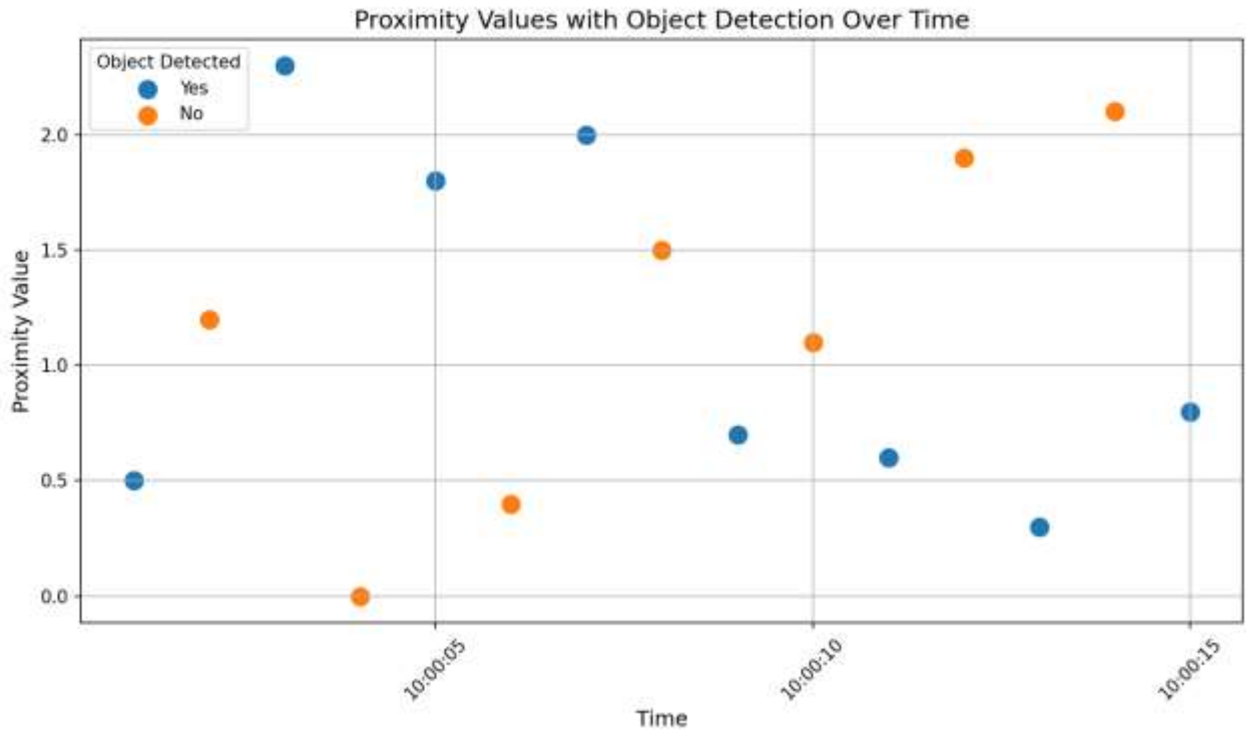
Explanation of graph:

The closeness values are shown on this line graph, and the different gesture kinds are indicated by distinct colours.
A distinct gesture is associated with each colour. This lets you see how specific movements might correspond with proximity readings. For instance, you may see that the proximity value tends to be larger during "Swipe Left" movements, suggesting that gestures are made when an object is closer. The proximity values can range dramatically across various movements, showing how a user's interaction with a device can be influenced by an object's distance from them. User behaviour patterns can be uncovered by observing the timing of gestures in relation to proximity. Gestures that are made often when the proximity value is low may indicate that people are interacting closely with an object.



Explanation of graph:

This graph displays time on the x-axis and proximity values on the y-axis, coloured according to whether the screen is on or off.
The graph shows the difference between the on (blue) and off (orange) states of the screen.
One way to determine whether the device performs differently when the screen is active compared to inactive is to analyse the proximity values during various screen states. For example, if the proximity numbers are high when the screen is off, it could mean that the sensor is still picking up items when it's not in use.
It may be a sign that people tend to keep their devices farther away when they are not using them if you see that proximity values decrease when the screen is off.



Explanation of graph:

The y-axis of this scatter plot shows proximity values, and the x-axis indicates whether or not an object was detected (Yes/No).
Status of Object Detection: Each point's colour (such as "Yes" or "No") indicates if an object was spotted at that specific moment.
When the concentration of "Yes" dots is high at lower proximity values, it indicates that the sensor is good at detecting objects that are in close vicinity. It suggests that objects are not recognised at greater distances if there are a lot of "No" detections at increasing proximity levels.
The dispersion of the spots can provide information on detecting powers. For instance, a detection range restriction may be indicated if several objects are spotted at lower proximity values but few at greater distances.

Pycharm File Link : ASSIGNMENT1

https://drive.google.com/file/d/1ki-kNWx4HRGdULXZTsYF8Zfwbo9Gk-2m/view?usp=sharing