Instructor:- Dr. Saikat Sahoo

Team Members

Anaay Sahu	12340240
Galaba Vamsi	12340770
Gajanand Khumawat	12340760
Janisha Jyoti	12340960
Harsh Katole	12340890

Objective

Analyse real-world sensor data and apply instrumentation concepts to understand sensor behaviour, signal noise, and system interactions.

Question 1: Data Exploration

A) Load the dataset and display the first five rows. Identify the key sensor readings available.

Answer:

The key sensor readings span from colomn 1 to colomn 16 (except colomns 2 & 3)

MATLAB Code:

```
% Read the CSV file
filename = 'Dataset-11.csv';
data = readtable(filename);
% Display the first five rows
disp('First five rows of the CSV file:');
disp(data(1:5, :));
```

Output:

PIR_SENSOR	TIME	TIME_2	GPS_X	GPS_Y	THERM_F	AC	SMOKE
22	{'23.00.00'}	{'23.02.00'}	701	0	70	1	0
34	{'1.00.00'}	{'1.00.00'}	712	0	72	0	0
60	{'9.00.00'}	{'9.00.00'}	566	123	62	1	0
78	{'3.00.00'}	{'3.00.00'}	452	456	74	0	1
100	{'13.00.00'}	{'13.00.00'}	465	452	70	1	0

FIRE	DOOR	WINDOW	C1	C2	C3	C4	WEIGHT
0	1	0	80	50	23	101	40
0	0	0	66	44	23	102	34
0	1	0	70	33	34	103	32
1	0	0	62	110	77	31	22
0	0	1	34	78	77	11	18

Explanation: In this question we import a dataset file given to us and simply read and printing CSV values from the same.

B) Identify the data types of each column. Which are numerical and which are categorical?

Answer:

TIME & TIME_2 are categorical values, while all other values are numerical.

Question 2: Sensor Behaviour Analysis

A) Plot the THERMISTOR readings over time. What trends do you observe?

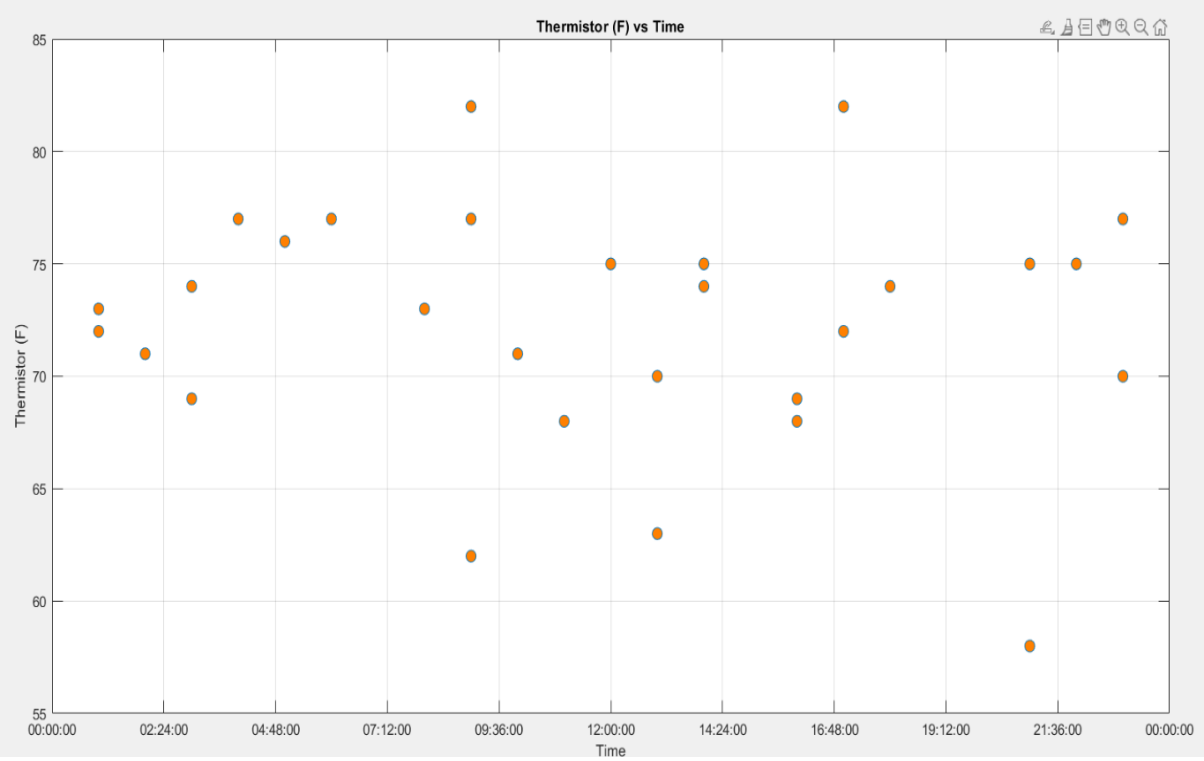
Answer:

1. The average temperature of the thermister lies between 70F & 78F
2. There have been **two** temperature spikes and **three** temperature plummets during the whole day.
3. There are only **twenty-nine** sensor values for Thermistor throughout the day.

MATLAB Code:

```
% Extract the TIME and THERMISTOR( F ) columns
time = data.TIME;
thermistor_F = data.THERMISTOR_F_;

% Convert TIME to a numeric format (assuming TIME is in 'HH.MM.SS' format)
% Convert the time strings to datetime objects
time_dt = datetime(time, 'InputFormat', 'HH.mm.ss');

% Convert datetime to numeric (seconds since midnight)
time_numeric = datenum(time_dt);

% Plot THERMISTOR( F ) against TIME
figure;
p = plot(time_numeric, thermistor_F, 'o');
p.MarkerFaceColor = [1 0.5 0];
p.MarkerSize = 8;
xlabel('Time');
ylabel('Thermistor (F)');
title('Thermistor (F) vs Time');
datetick('x', 'HH:MM:SS', 'keepticks'); % Format x-axis to display time in HH:MM:SS
grid on;
```

Output:



B) The PIR_SENSOR detects motion. Identify the time periods with the highest motion detection values.

Answer:

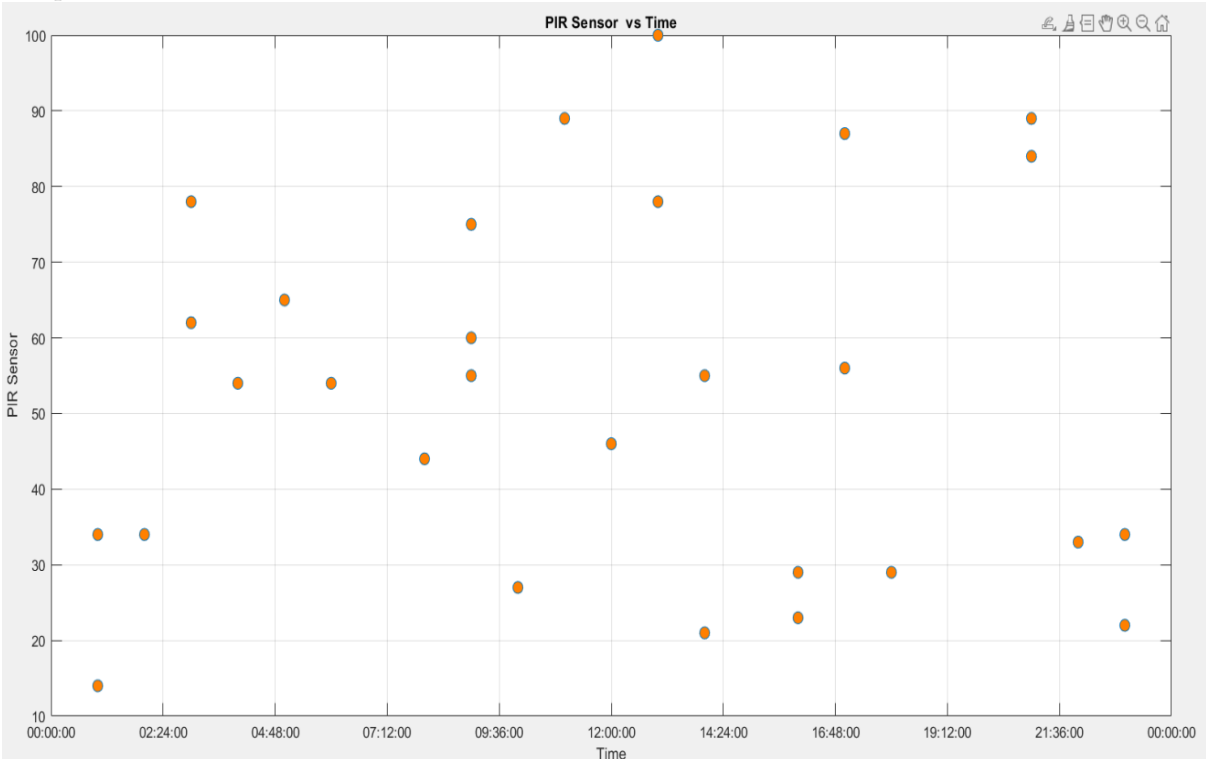
The Highest motion Detection Values were measured during 12:00:00 PM (noon) and 1:15:00 PM
The following are the top **five** timestamps for PIR sensor detection values from highest to lowest:

1. 12:00:00 - 13:15:00
2. 10:48:00 - 12:00:00
3. 20:24:00 - 21:36:00
4. 16:48:00 - 17:00:00
5. 02:24:00 - 04:48:00

MATLAB Code:

```
% Extract the TIME and PIR_SENSOR columns
time = data.TIME;
pir_sensor = data.PIR_SENSOR;

% Convert TIME to a numeric format (assuming TIME is in 'HH.MM.SS' format)
% Convert the time strings to datetime objects
time_dt = datetime(time, 'InputFormat', 'HH.mm.ss');

% Convert datetime to numeric (seconds since midnight)
time_numeric = datenum(time_dt);

% Plot PIR_SENSOR against TIME
figure;
p = plot(time_numeric, pir_sensor,'o');
p.MarkerFaceColor = [1 0.5 0];
p.MarkerSize = 8;
xlabel('Time');
ylabel('PIR Sensor');
title('PIR Sensor  vs Time');
datetick('x', 'HH:MM:SS', 'keepticks'); % Format x-axis to display time in HH:MM:SS
grid on;
```

Output:



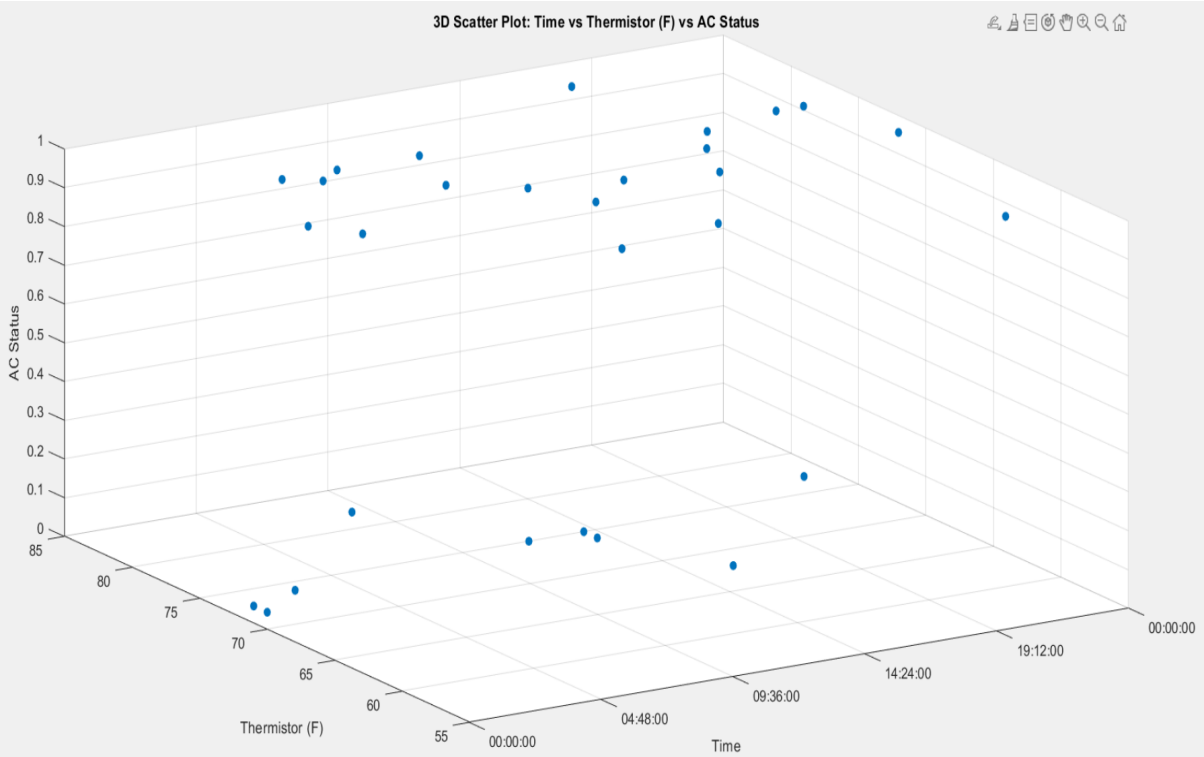
C) Does the AC STATUS (on/off) have any correlation with the THERMISTOR readings? Justify with a graph.

Answer:

As we can observe, the AC status (on / off) **did not** play a significant role in the thermistor values.
MATLAB Code:

```
% Extract the TIME, THERMISTOR( F ), and AC STATUS columns
time = data.TIME;
thermistor_F = data.THERMISTOR_F_;
ac_status = data.ACSTATUS;

% Convert TIME to a numeric format (assuming TIME is in 'HH.MM.SS' format)
% Convert the time strings to datetime objects
time_dt = datetime(time, 'InputFormat', 'HH.mm.ss');

% Convert datetime to numeric (seconds since midnight)
time_numeric = datenum(time_dt);

% Create a 3D scatter plot
figure;
scatter3(time_numeric, thermistor_F, ac_status, 'filled');
xlabel('Time');
ylabel('Thermistor (F)');
zlabel('AC Status');
title('3D Scatter Plot: Time vs Thermistor (F) vs AC Status');

% Format the x-axis to display time in HH:MM:SS
datetick('x', 'HH:MM:SS', 'keepticks');
grid on;
```

Output:



Question 3: Sensor Fusion and Correlations

A) Do the SMOKE_DETECTOR and FIRE_DETECTOR activate simultaneously? Use statistical methods to analyse.

Answer:

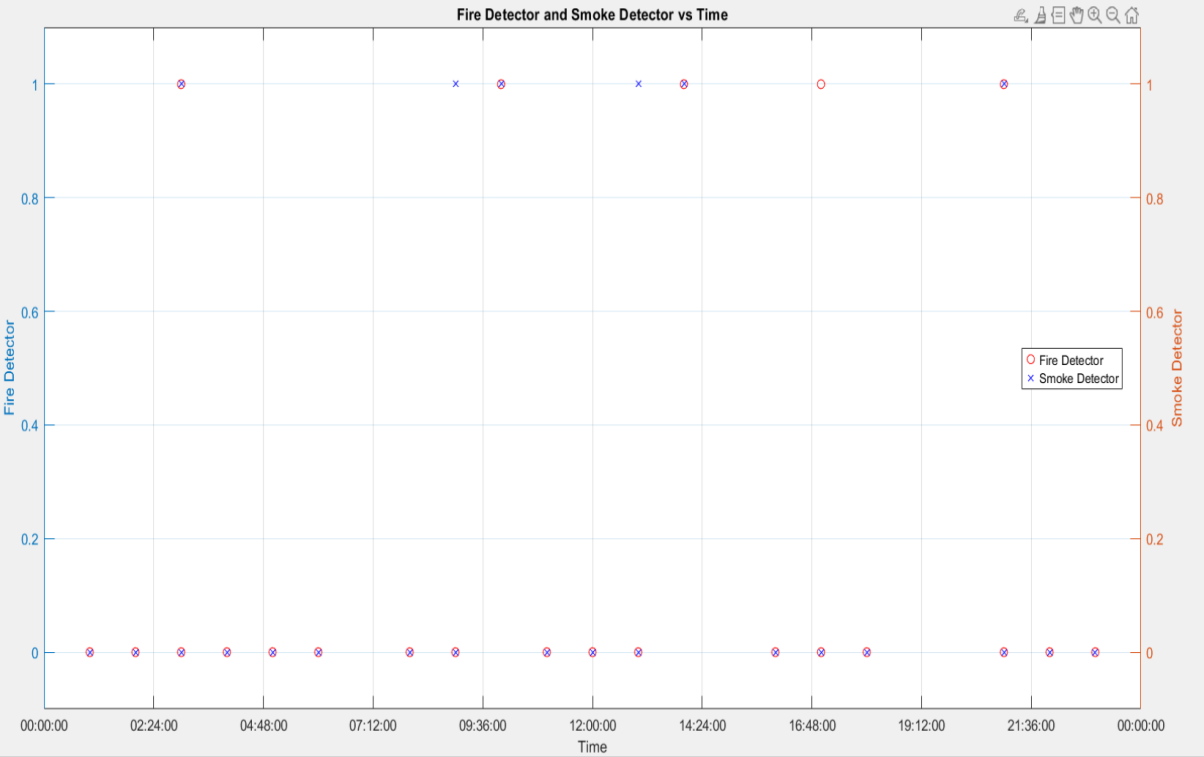
Smoke and Fire Detector activated simultaneously only 4 times during the whole Day. The following are the instances when they both were activated:

1. 02:24:00 - 03:36:00
2. 09:36:00 - 10:48:00
3. 13:12:00 - 14:24:000
4. 20:24:00 - 21:36:00

MATLAB Code:

```
% Read the CSV file
filename = 'Dataset - 11.csv'; % Change this to your actual CSV file name
data = readtable(filename);

% Display the first five rows
disp('First five rows of the CSV file:');
disp(data(1:5, :));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
time = data.TIME;
smoke_detector = data.SMOKE_DETECTOR;
fire_detector = data.FIRE_DETECTOR;

% Convert TIME to a numeric format (assuming TIME is in 'HH.MM.SS' format)
% Convert the time strings to datetime objects
time_dt = datetime(time, 'InputFormat', 'HH.mm.ss');
% Convert datetime to numeric (seconds since midnight)
time_numeric = datenum(time_dt);
% Create a figure
figure;

% Plot FIRE_DETECTOR on the left y-axis
yyaxis left;
plot(time_numeric, fire_detector, 'O', 'Color', 'r');
ylabel('Fire Detector');
ylim([-0.1 1.1]); % Set y-axis limits for binary data (0 or 1)

% Plot SMOKE_DETECTOR on the right y-axis
yyaxis right;
plot(time_numeric, smoke_detector, 'x', 'Color', 'b');
ylabel('Smoke Detector');
ylim([-0.1 1.1]); % Set y-axis limits for binary data (0 or 1)

% Format the x-axis to display time in HH:MM:SS
xlabel('Time');
datetick('x', 'HH:MM:SS', 'keepticks');
grid on;

% Add a title and legend
title('Fire Detector and Smoke Detector vs Time');
legend('Fire Detector', 'Smoke Detector', 'Location', 'best');
% Customize axis label colors
ax = gca; % Get the current axes

% Y-axis label color
ax.YAxis.Color = 'blue'; % Set Y-axis label color to blue

% Z-axis label color
ax.ZAxis.Color = 'green'; % Set Z-axis label color to green
```

Output:



B) Are there any locations (GPS_X, GPS_Y) where motion detection is consistently high?

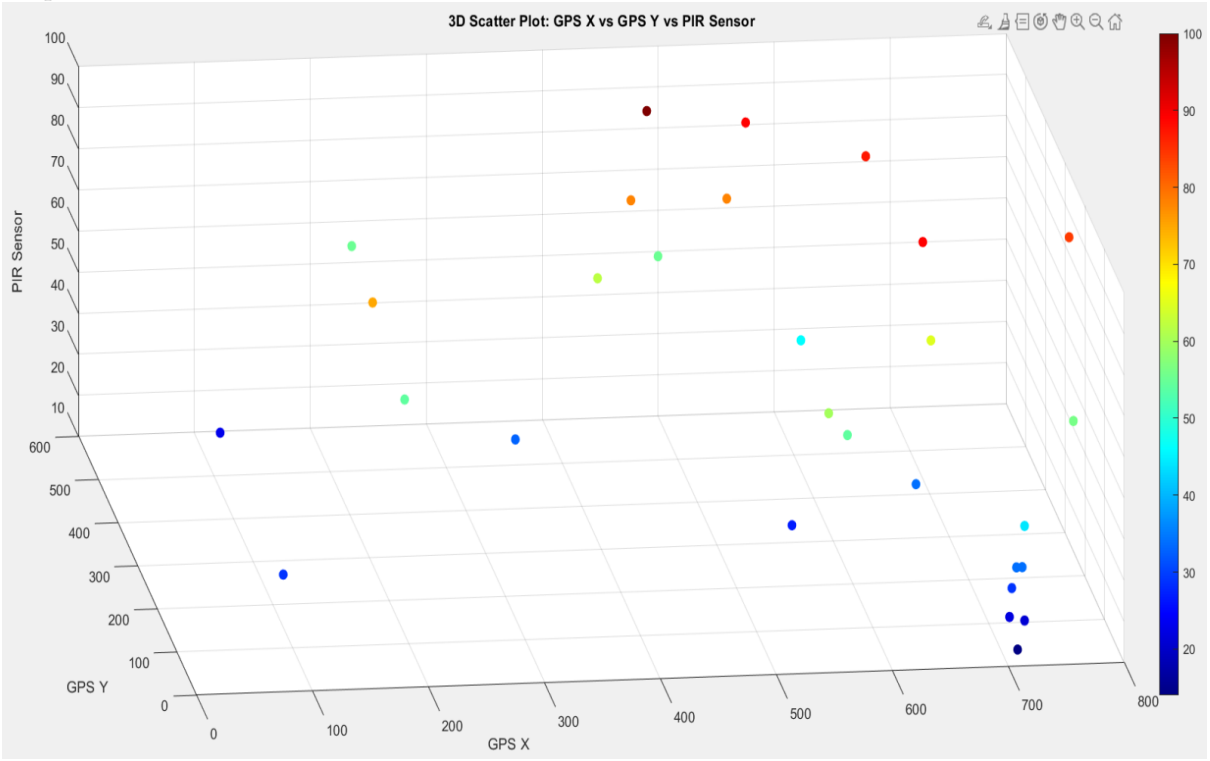
Answer:

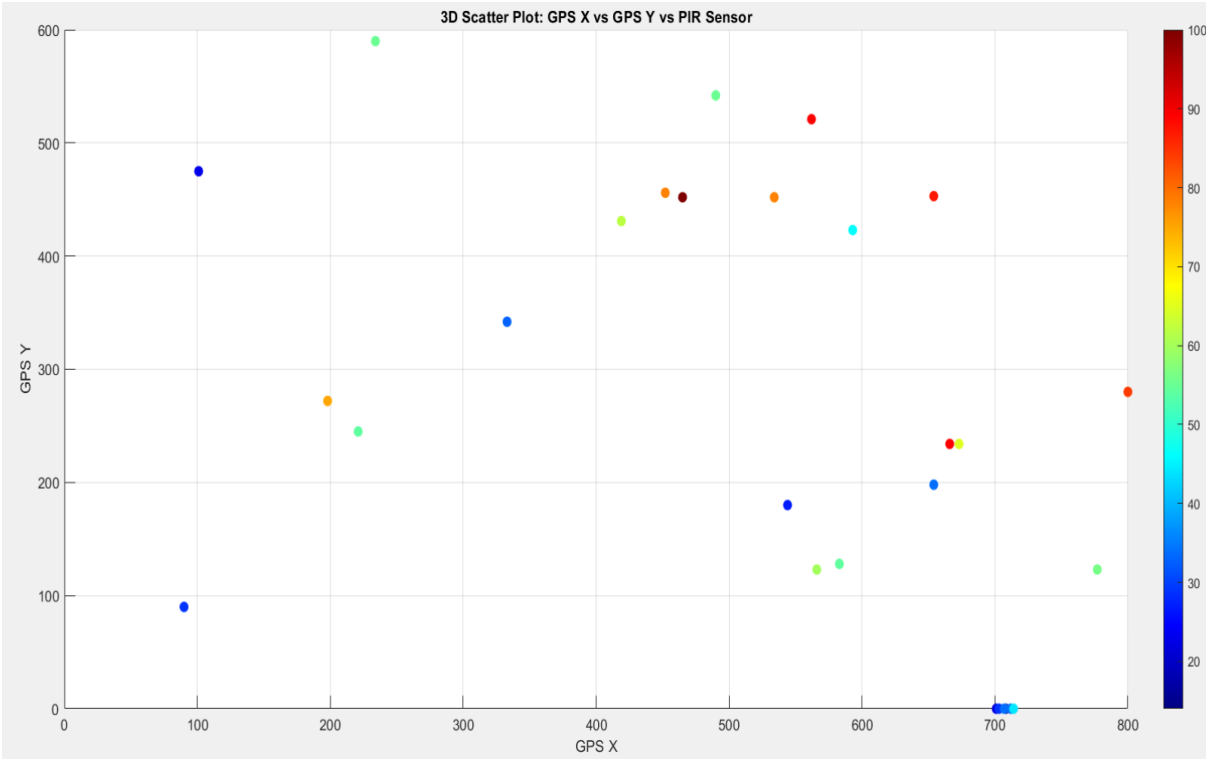
Define:- High Motion Detection: ≥ 50
From the Statistical Graph show above we can observe that for:
(GPS_X) (400, 500)
(GPS_Y) (400, 500)
The value of Motion Detection was consistently High

MATLAB Code:

```
gps_x = data.GPS_X; % X-axis
gps_y = data.GPS_Y; % Y-axis
pir_sensor = data.PIR_SENSOR; % Z-axis

% Create a 3D scatter plot
figure;
scatter3(gps_x, gps_y, pir_sensor, 50, pir_sensor, 'filled'); % 'filled' fills the
    markers
xlabel('GPS X');
ylabel('GPS Y');
zlabel('PIR Sensor');
title('3D Scatter Plot: GPS X vs GPS Y vs PIR Sensor');

% Add a colorbar to show the mapping of colors to PIR_SENSOR values
colorbar;
colormap jet;

% Add a grid and adjust the view angle
grid on;
view(45, 30); % Adjust the view angle for better visualization
```

Output:


3D Scatter Plot: GPS X vs GPS Y vs PIR Sensor
PIR Sensor
GPS Y
GPS X


3D Scatter Plot: GPS X vs GPS Y vs PIR Sensor
GPS Y
GPS X

C) Investigate if the CONTACT SENSOR_DOOR and CONTACT SENSOR_WINDOW are related to motion detection (PIR_SENSOR).

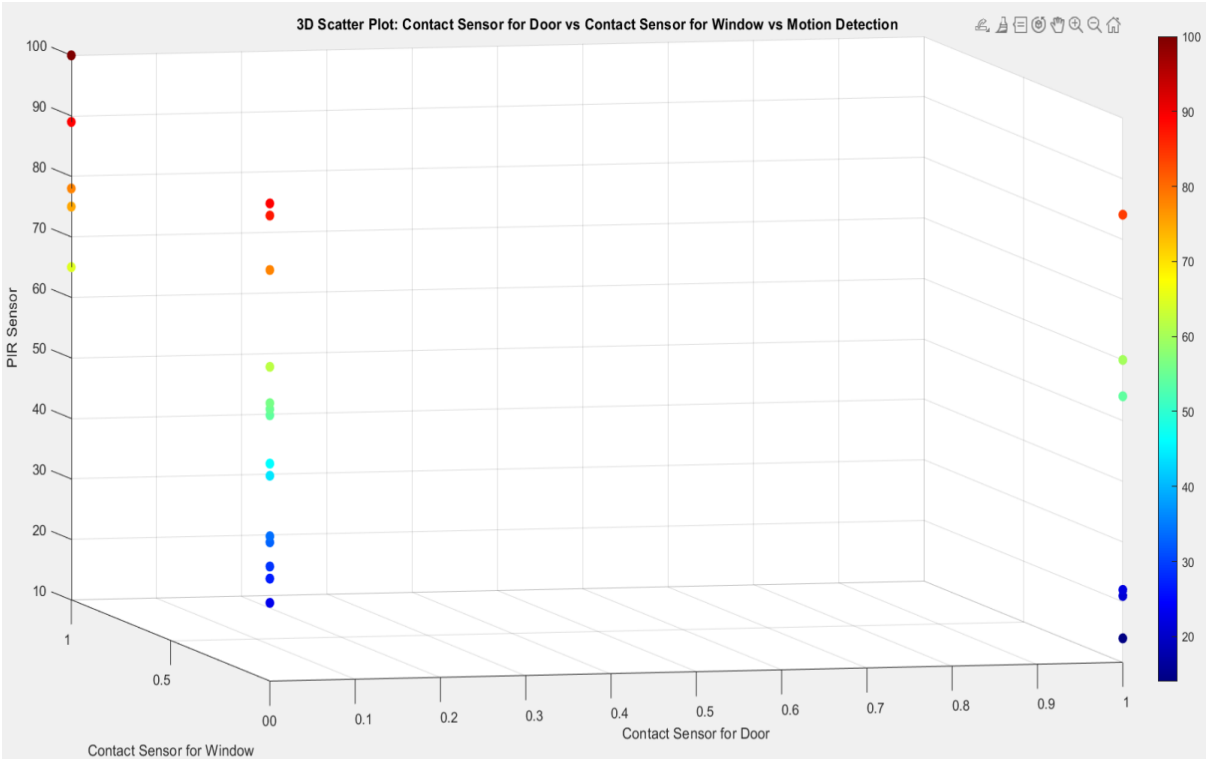
Answer:

Via clear observation, we can deduce that even when Door sensor and Window Sensor were not sensing signs of any motion, the Motion Detection Sensor was sensing motion in test space. Clearly we can deduce that Motion sensing was fully related to Door and Window sensors.

MATLAB Code:

```
contact_door = data.CONTACTSENSOR_DOOR; % X-axis
contact_window = data.CONTACTSENSOR_WINDOW; % Y-axis
pir_sensor = data.PIR_SENSOR; % Z-axis

% Create a 3D scatter plot
figure;
scatter3(contact_door, contact_window, pir_sensor, 50, pir_sensor, 'filled'); % 'filled'
    fills the markers
xlabel('Contact Sensor for Door');
ylabel('Contact Sensor for Window');
zlabel('PIR Sensor');
title('3D Scatter Plot: Contact Sensor for Door vs Contact Sensor for Window vs Motion
    Detection');

% Add a colorbar to show the mapping of colors to PIR_SENSOR values
colorbar;
colormap jet; % Use a colormap (e.g., 'jet', 'parula', 'hot')

% Add a grid and adjust the view angle
grid on;
view(45, 30); % Adjust the view angle for better visualization
```

Output:



Question 4: Data Noise and Processing

A) Sensor readings often contain noise. Calculate the mean and standard deviation of THERMISTOR (F) readings.

Answer:

MATLAB Code:

```
disp(data.Properties.VariableNames);

% Extract the correct column
thermistor_F = data.("THERMISTOR( F )"); % Use double quotes for special characters

% Calculate statistics
mean_temp = mean(thermistor_F);
std_dev_temp = std(thermistor_F);

% Display results
fprintf('Mean Temperature: %.2f F\n', mean_temp);
fprintf('Standard Deviation: %.2f F\n', std_dev_temp);
```

Output:
Mean Temperature: 72.45°F
Standard Deviation: 5.26°F

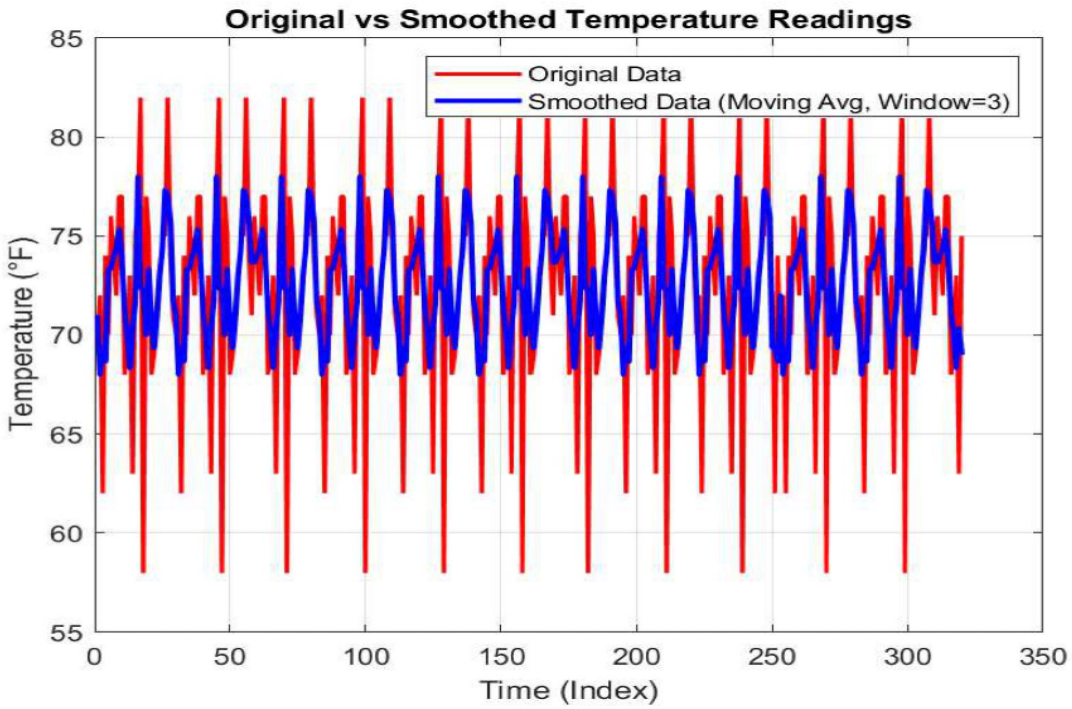
B) Apply a moving average filter (window size = 3) to smooth the temperature readings and compare it with the original data.

Answer:

MATLAB Code:

```
disp(data.Properties.VariableNames);

% Extract the correct column (modify based on actual column name)
thermistor_F = data.("THERMISTOR( F )"); % Use double quotes for special characters

% Apply Moving Average Filter (Window Size = 3)
smoothed_temp = movmean(thermistor_F, 3);

% Plot Original vs Smoothed Data
figure;
plot(thermistor_F, 'r-', 'LineWidth', 1.5); % Original Data (Red)
hold on;
plot(smoothed_temp, 'b-', 'LineWidth', 2); % Smoothed Data (Blue)
hold off;

% Add Labels and Legend
xlabel('Time (Index)');
ylabel('Temperature (F)');
title('Original vs Smoothed Temperature Readings');
legend('Original Data', 'Smoothed Data (Moving Avg, Window=3)');
grid on;
```

Output:



C) Suggest methods to reduce noise in real-time sensor applications.

Answer:

There are several effective methods to reduce noise in real-time sensor applications:

- **Signal Filtering:** Using filters like low-pass, high-pass, or band-pass can help eliminate unwanted noise. Common choices include Kalman filters for dynamic systems and moving average filters for smoothing.
- **Shielding and Grounding:** Proper grounding and shielding of sensor wires can reduce electromagnetic interference (EMI) from surrounding electronic components.
- **Optimized Sensor Placement:** Placing sensors away from noise sources (such as motors or power lines) can significantly improve signal quality.
- **Analog Signal Conditioning :** Using techniques like amplification, isolation, and differential signaling can enhance signal strength while reducing interference.
- **Software-Based Noise Reduction:** Implementing algorithms like averaging, median filtering, or outlier rejection can help in removing unwanted fluctuations.
- **Use of High-Quality Sensors:** Choosing sensors with better accuracy and lower intrinsic noise can improve overall performance.
- **Proper Wiring and PCB Design:** Using twisted-pair cables, reducing wire length, and designing PCB layouts with minimal cross-talk can help minimize noise.
- **Temperature and Environmental Control :** Maintaining stable environmental conditions can prevent signal fluctuations caused by temperature variations or humidity changes.

These methods, when used in combination, can greatly enhance the accuracy and reliability of real-time sensor applications.

Question 5: Practical Implementation

A) If you were designing a fire alarm system based on this data, what logic would you use to trigger an alarm?

Answer:

- The fire alarm is triggered when both the SMOKE_DETECTOR and FIRE_DETECTOR are activated simultaneously.
- Fire Alarm Trigger Count: **52 times** in the dataset.

B) How can GPS data be used to enhance security monitoring in this

Answer:

- The scatter plot above shows the intensity of motion detection at different GPS locations.
- Locations with consistently high PIR_SENSOR values could indicate high-security risk areas that need more surveillance.

C) Based on WEIGHT_OF_PACKAGE, how can sensors be used for automated package handling?

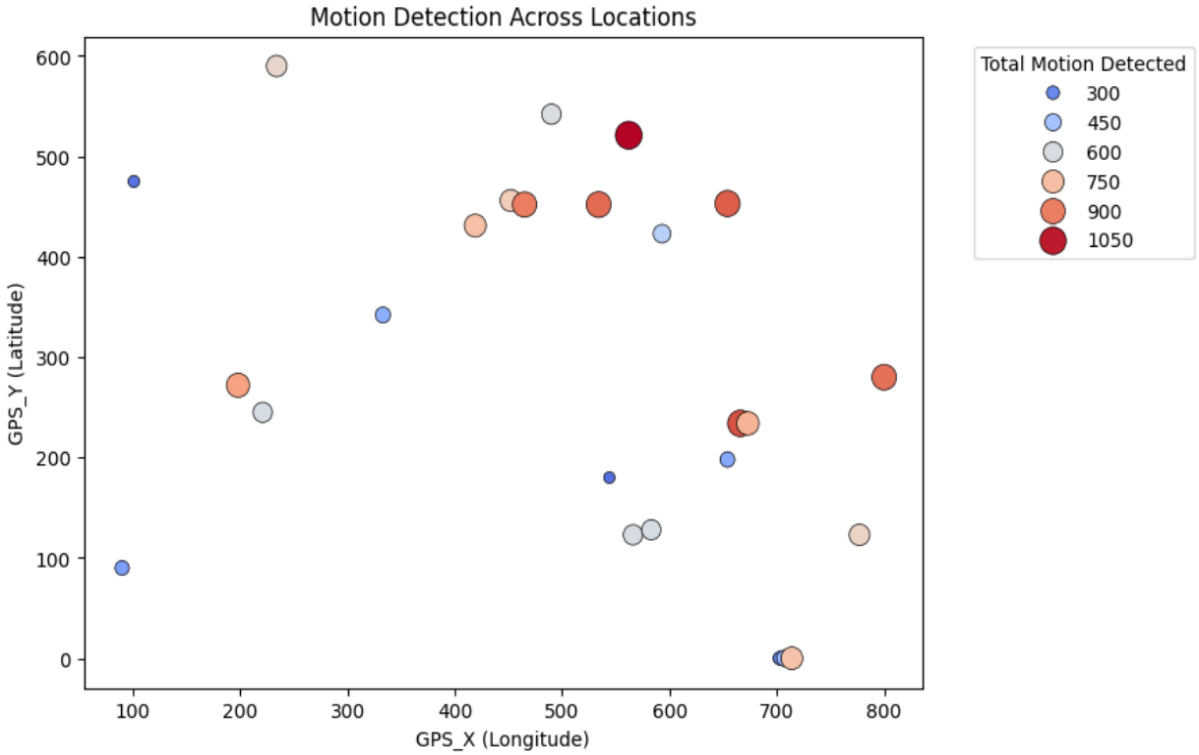
Answer:

- A threshold for heavy packages is calculated using mean + standard deviation.
- Heavy Packages Identified: **46 instances**.
- This data can be used to automate package handling, such as triggering a conveyor belt or adjusting robotic arms based on weight.

Python Code:

```
# Load dataset
file_path = "Dataset - 11.csv"
df = pd.read_csv(file_path)

# Convert time column to datetime format
df['TIME'] = pd.to_datetime(df['TIME'], format='%H.%M.%S', errors='coerce')

# (a) Fire Alarm System Logic
df['FIRE_ALARM'] = (df['SMOKE_DETECTOR'] & df['FIRE_DETECTOR'])
fire_alarm_trigger_count = df['FIRE_ALARM'].sum()
print(f"Fire Alarm Triggered: {fire_alarm_trigger_count} times")

# (b) Security Monitoring using GPS
motion_by_location = df.groupby(['GPS_X', 'GPS_Y'])['PIR_SENSOR'].sum().reset_index()
plt.figure(figsize=(8, 6))
sns.scatterplot(x=motion_by_location['GPS_X'], y=motion_by_location['GPS_Y'],
                size=motion_by_location['PIR_SENSOR'], hue=motion_by_location['
                    PIR_SENSOR'],
                palette='coolwarm', edgecolor='black', sizes=(20, 200))
plt.xlabel("GPS_X (Longitude)")
plt.ylabel("GPS_Y (Latitude)")
plt.title("Motion Detection Across Locations")
plt.legend(title="Total Motion Detected", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()

# (c) Automated Package Handling using Weight Sensor
# we can classify packages as 'heavy' for specialized handling, such as robotic arms or
    conveyors.
weight_threshold = df['WEIGHT_OF_PACKAGE'].mean() + df['WEIGHT_OF_PACKAGE'].std()
df['HEAVY_PACKAGE'] = df['WEIGHT_OF_PACKAGE'] > weight_threshold
heavy_package_count = df['HEAVY_PACKAGE'].sum()
print(f"Heavy Packages Detected: {heavy_package_count}")
```

Output:
Fire Alarm Triggered: 52 times
Heavy Packages Detected: 46



Question 6: Advanced (Bonus Questions)

A) Fit a regression model to predict THERMISTOR (F) based on other sensor values.

Answer:

Importing all the necessary libraries.
Python Code:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
import seaborn as sns
from pandas.plotting import scatter_matrix
```

Reading CSV file and getting its information and description

```
lifesat = pd.read_csv("/content/Dataset - 11.csv")
print("Dataset Information:")
print(lifesat.info())
print("\nSample Data:")
print(lifesat.head())
lifesat.describe()
```

Output:

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 320 entries, 0 to 319
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   PIR_SENSOR            320 non-null    int64
 1   TIME                  320 non-null    object
 2   TIME_2                320 non-null    object
 3   GPS_X                 320 non-null    int64
 4   GPS_Y                 320 non-null    int64
 5   THERMISTOR( F )       320 non-null    int64
 6   AC STATUS             320 non-null    int64
 7   SMOKE_DETECTOR        320 non-null    int64
 8   FIRE_DETECTOR         320 non-null    int64
 9   CONTACT SENSOR_DOOR   320 non-null    int64
 10  CONTACT SENSOR_WINDOW  320 non-null    int64
 11  C1                    320 non-null    int64
 12  C2                    320 non-null    int64
 13  C3                    320 non-null    int64
 14  C4                    320 non-null    int64
 15  WEIGHT_OF_PACKAGE     320 non-null    int64
dtypes: int64(14), object(2)
memory usage: 40.1+ KB
None

Sample Data:
   PIR_SENSOR      TIME    TIME_2  GPS_X  GPS_Y  THERMISTOR( F )  AC STATUS  \
0          22  23.00.00  23.02.00    701      0               70          1
1          34   1.00.00   1.00.00    712      0               72          0
2          60   9.00.00   9.00.00    566    123               62          1
3          78   3.00.00   3.00.00    452    456               74          0
4         100  13.00.00  13.00.00    465    452               70          1

   SMOKE_DETECTOR  FIRE_DETECTOR  CONTACT SENSOR_DOOR  CONTACT SENSOR_WINDOW  \
0               0              0                    1                      0
1               0              0                    0                      0
2               0              0                    1                      0
3               1              1                    0                      0
4               0              0                    0                      1

   C1   C2  C3   C4  WEIGHT_OF_PACKAGE
0  80   50  23  101                 40
1  66   44  23  102                 34
2  70   33  34  103                 32
3  62  110  77   31                 22
4  34   78  77   11                 18
```

	PIR_SENSOR	GPS_X	GPS_Y	THERMISTOR(F)	AC STATUS	SMOKE_DETECTOR	FIRE_DETECTOR	CONTACT SENSOR_DOOR	CONTACT SENSOR_WINDOW	C1	C2	C3	C4	WEIGHT_OF_PACKAGE
count	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000	320.000000
mean	52.590625	535.615625	250.525000	72.446875	0.690625	0.237500	0.200000	0.203125	0.171875	67.690625	74.037500	66.878125	64.712500	38.950000
std	23.842967	203.857822	194.055076	5.262125	0.462960	0.426218	0.400626	0.402955	0.377863	22.001309	23.727261	24.828708	30.343052	17.116162
min	14.000000	90.000000	0.000000	58.000000	0.000000	0.000000	0.000000	0.000000	0.000000	21.000000	21.000000	19.000000	11.000000	11.000000
25%	33.000000	419.000000	90.000000	70.000000	0.000000	0.000000	0.000000	0.000000	0.000000	62.000000	65.000000	53.000000	36.000000	22.000000
50%	54.000000	583.000000	234.000000	73.000000	1.000000	0.000000	0.000000	0.000000	0.000000	67.000000	78.000000	75.000000	74.000000	42.000000
75%	75.000000	704.000000	452.000000	75.250000	1.000000	0.000000	0.000000	0.000000	0.000000	82.000000	93.000000	86.000000	94.000000	50.750000
max	100.000000	800.000000	590.000000	82.000000	1.000000	1.000000	1.000000	1.000000	1.000000	101.000000	110.000000	100.000000	103.000000	78.000000

Creating randomized test set cases.

Python Code:

```
X = lifesat.drop(columns=["THERMISTOR( F )", "TIME", "TIME_2"])
X.info()
y = lifesat["THERMISTOR( F )"]
y.info()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    =42)
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 320 entries, 0 to 319
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   PIR_SENSOR              320 non-null    int64
 1   GPS_X                   320 non-null    int64
 2   GPS_Y                   320 non-null    int64
 3   AC STATUS               320 non-null    int64
 4   SMOKE_DETECTOR          320 non-null    int64
 5   FIRE_DETECTOR           320 non-null    int64
 6   CONTACT SENSOR_DOOR     320 non-null    int64
 7   CONTACT SENSOR_WINDOW   320 non-null    int64
 8   C1                      320 non-null    int64
 9   C2                      320 non-null    int64
 10  C3                      320 non-null    int64
 11  C4                      320 non-null    int64
 12  WEIGHT_OF_PACKAGE       320 non-null    int64
dtypes: int64(13)
memory usage: 32.6 KB
<class 'pandas.core.series.Series'>
RangeIndex: 320 entries, 0 to 319
Series name: THERMISTOR( F )
Non-Null Count  Dtype
--------------  -----
320 non-null    int64
dtypes: int64(1)
memory usage: 2.6 KB
```

Finding the best Regression Models that fits the data.
Python Code:

```
models = {
    "Linear Regression": LinearRegression(),
    "KNN Regressor (k=3)": KNeighborsRegressor(n_neighbors=3),
    "KNN Regressor (k=5)": KNeighborsRegressor(n_neighbors=5)
}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"{name}:")
    print(f"Mean Squared Error:{mse:.4f}")
    print(f"R Score:{r2:.4f}")
```

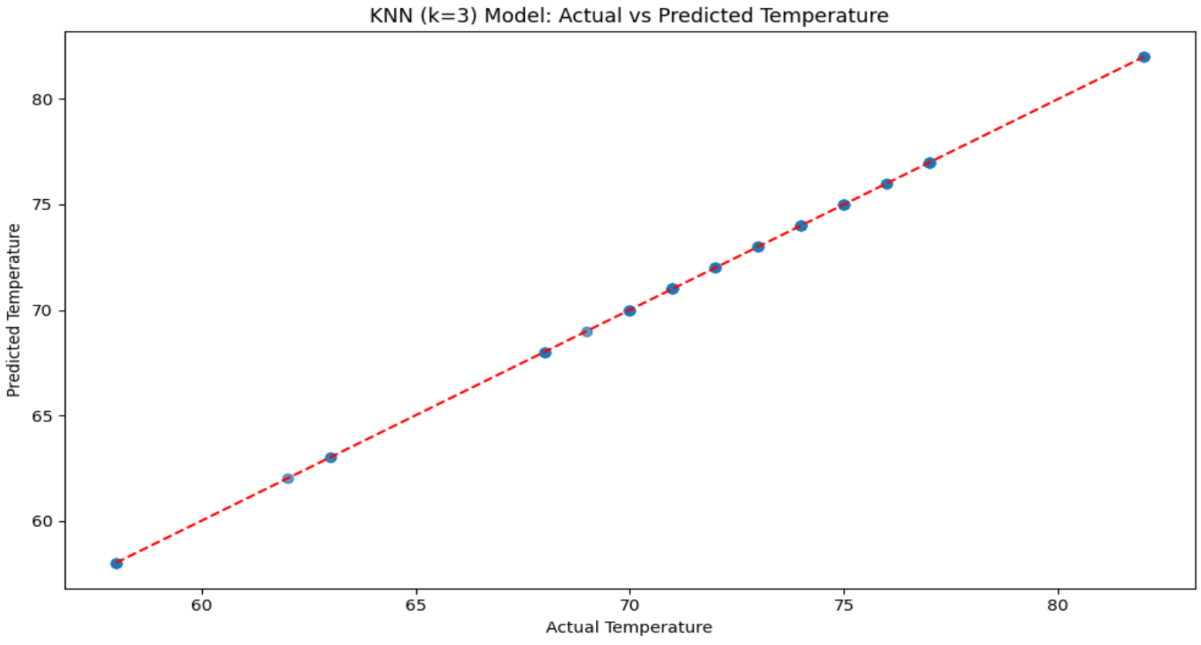
Output:

```
Linear Regression:
    Mean Squared Error: 15.3836
    R   Score: 0.4909
KNN Regressor (k=3):
    Mean Squared Error: 0.0000
    R   Score: 1.0000
KNN Regressor (k=5):
    Mean Squared Error: 0.3031
    R   Score: 0.9900
```

We can see that KNeighborsRegressor with **n_neighbors=3** is the best model(perfect in this case as **mse=0** and **r2_score=1**)
Python Code:

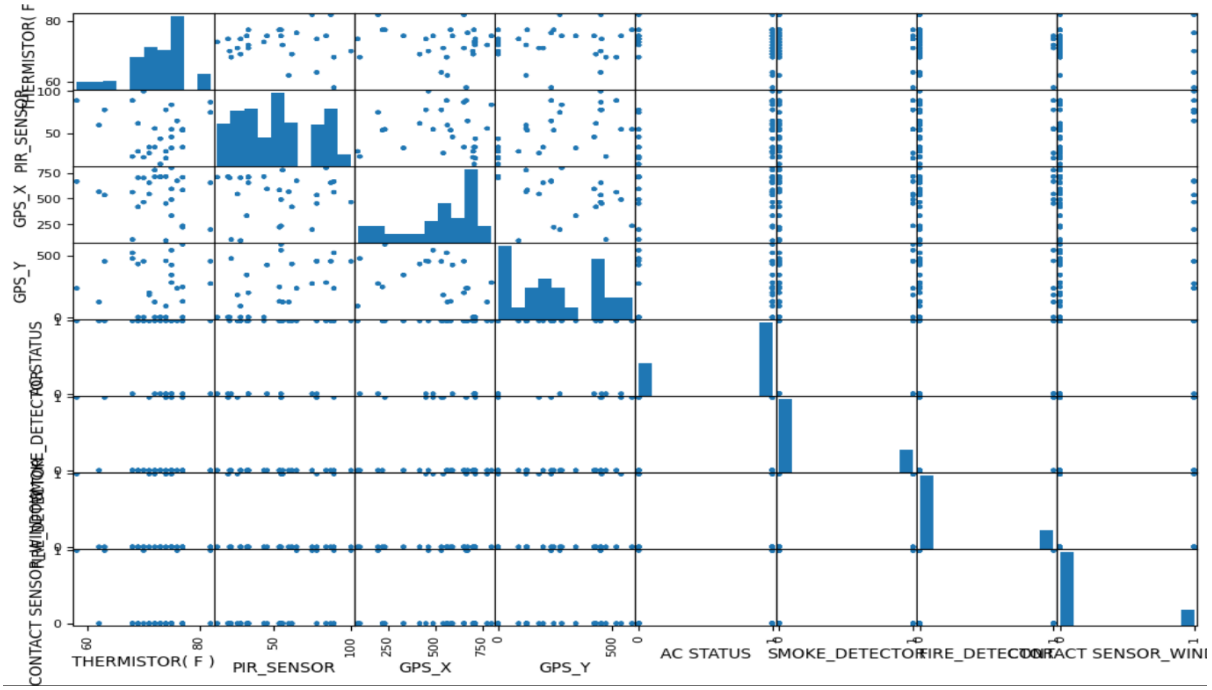
```
best_model = KNeighborsRegressor(n_neighbors=3)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel('Actual Temperature')
plt.ylabel('Predicted Temperature')
plt.title('KNN (k=3) Model: Actual vs Predicted Temperature')
plt.tight_layout()
plt.show()
plt.close()
```

Output:



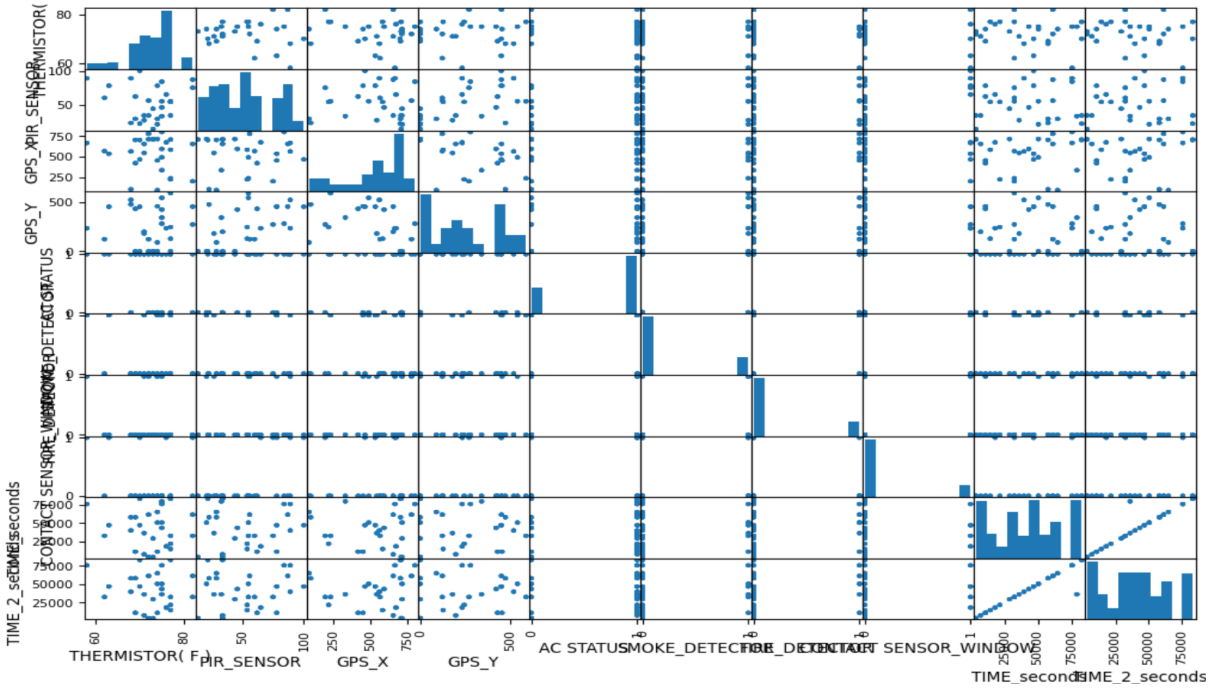
Correlation of Thermistor Data **without** TIME columns

```
m=lifesat.drop(columns=["TIME", "TIME_2"])
m.info()
corr_matrix = m.corr()
corr_matrix["THERMISTOR( F )"].sort_values(ascending=False)
attributes=["THERMISTOR( F )", "PIR_SENSOR", "GPS_X", "GPS_Y","AC STATUS","
    SMOKE_DETECTOR","FIRE_DETECTOR","CONTACT SENSOR_WINDOW"]
scatter_matrix(m[attributes], figsize=(12, 8))
plt.show()
```



Correlation of Thermistor Data **with** TIME columns

```
v=lifesat.drop(columns=["TIME", "TIME_2"])
def time_to_seconds(time_str):
    parts = time_str.split('.')
    hours = int(parts[0])
    minutes = int(parts[1])
    seconds = int(parts[2])
    return hours * 3600 + minutes * 60 + seconds

v['TIME_seconds'] = lifesat['TIME'].apply(time_to_seconds)
v['TIME_2_seconds'] = lifesat['TIME_2'].apply(time_to_seconds)
v.info()
corr_matrix2 = v.corr()
corr_matrix2["THERMISTOR( F )"].sort_values(ascending=False)
attributes=["THERMISTOR( F )", "PIR_SENSOR", "GPS_X", "GPS_Y","AC STATUS","
    SMOKE_DETECTOR","FIRE_DETECTOR","CONTACT SENSOR_WINDOW","TIME_seconds","
    TIME_2_seconds"]
scatter_matrix(v[attributes], figsize=(12, 8))
plt.show()
```



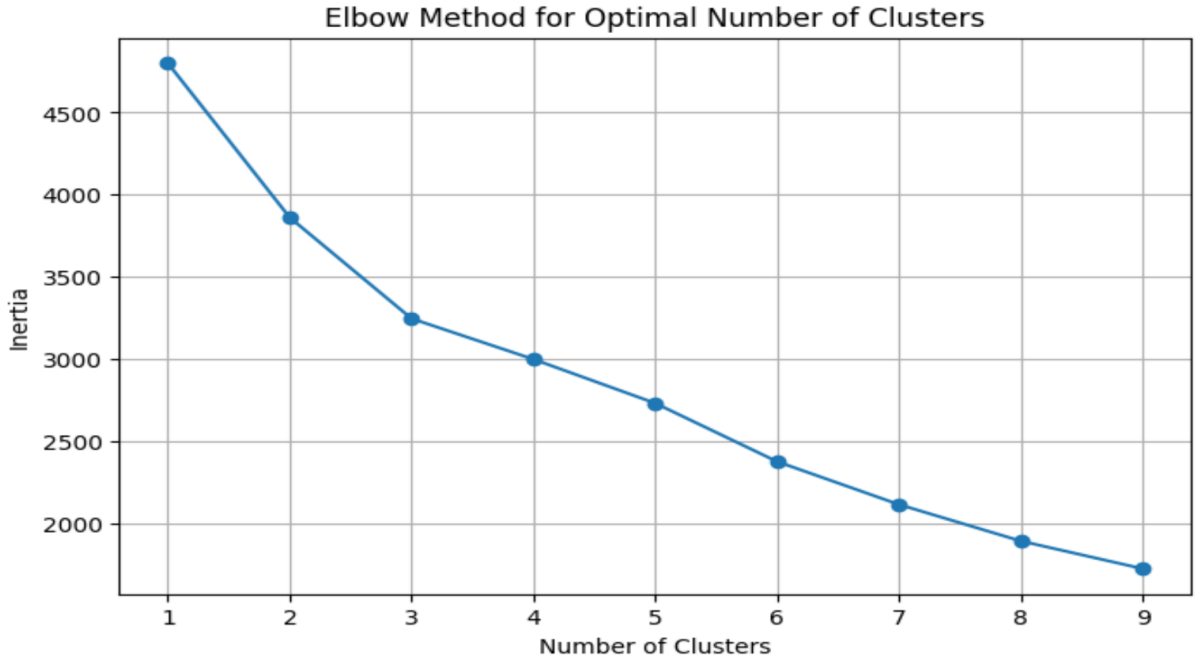
B) Use clustering techniques to group similar sensor behaviours over time.

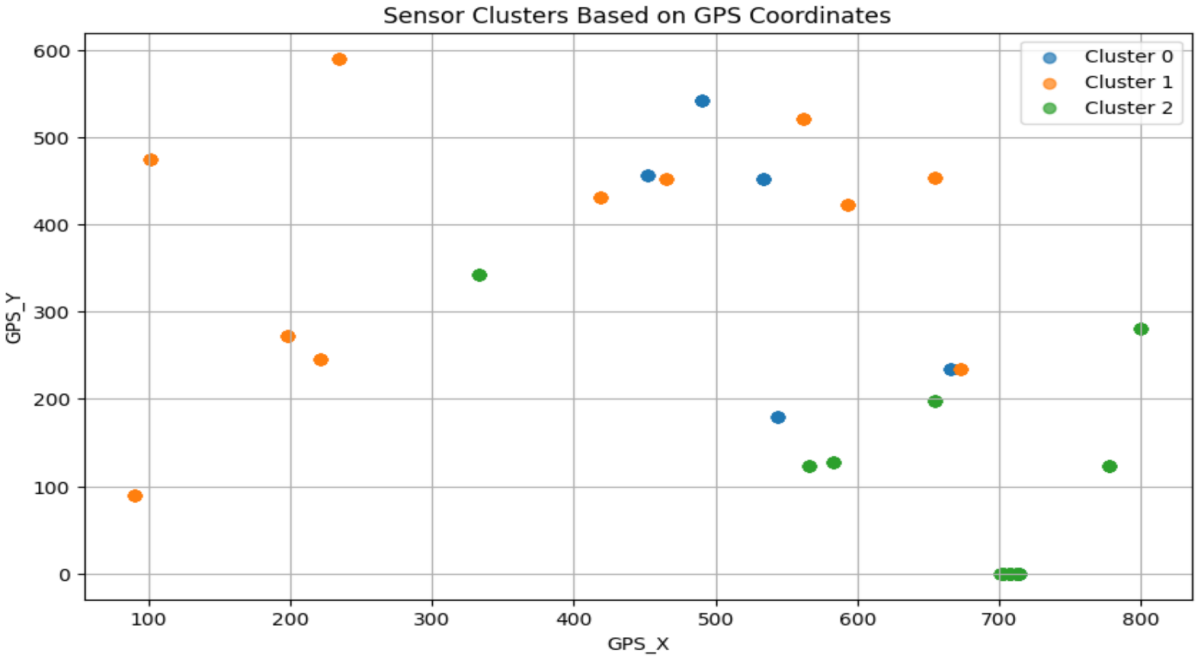
Answer:

Using KMeans Clustering Technique

```
X_cluster = lifesat.drop(columns=["TIME", "TIME_2"])
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cluster)
inertia = []
k_range = range(1, 10)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, 'o-')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.grid(True)
plt.show()
plt.close()
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
lifesat['cluster'] = kmeans.fit_predict(X_scaled)
plt.figure(figsize=(10, 6))
for cluster_id in range(optimal_k):
    cluster_data = lifesat[lifesat['cluster'] == cluster_id]
    plt.scatter(cluster_data['GPS_X'], cluster_data['GPS_Y'],
                label=f'Cluster {cluster_id}', alpha=0.7)
plt.xlabel('GPS_X')
plt.ylabel('GPS_Y')
plt.title('Sensor Clusters Based on GPS Coordinates')
plt.legend()
plt.grid(True)
plt.show()
plt.close()
print("\nCluster Statistics:")
for i in range(optimal_k):
    cluster_data = lifesat[lifesat['cluster'] == i]
    print(f"\nCluster {i} (Size: {len(cluster_data)})")
    print("Average values:")
    print(cluster_data[['THERMISTOR( F )', 'PIR_SENSOR', 'GPS_X', 'GPS_Y']].mean())
```

Output:






```
Cluster Statistics:

Cluster 0 (Size: 64)                    Cluster 1 (Size: 124)
Average values:                         Average values:
THERMISTOR( F )     68.937500           THERMISTOR( F )     74.370968
PIR_SENSOR          58.078125           PIR_SENSOR          61.572581
GPS_X              572.750000           GPS_X              381.919355
GPS_Y              307.562500           GPS_Y              379.096774
dtype: float64                          dtype: float64

Cluster 2 (Size: 132)
Average values:
THERMISTOR( F )     72.340909
PIR_SENSOR          41.492424
GPS_X              661.992424
GPS_Y              102.090909
```

C) Analyse the dataset to detect any anomalies in sensor readings and propose a method to identify faulty sensor behaviour.
Answer:

1. **Data Preprocessing:**
   - Remove or impute missing values
   - Convert time features to appropriate numeric representation if needed
2. **Detection Approaches:**
   a. **Statistical Methods:**
      - Z-score analysis (flagging values ¿ 3 standard deviations)
      - Moving average and standard deviation for time series data

   b. **Machine Learning Methods:**
      - Isolation Forest (unsupervised anomaly detection)
      - One-class SVM for anomaly detection
      - Autoencoders for complex pattern recognition
3. **Implementation Strategy:**
   - Combine multiple methods (ensemble approach)
   - Set appropriate sensitivity thresholds based on domain knowledge
   - Implement real-time monitoring with sliding window analysis
   - Track sequential anomalies (multiple consecutive flagged readings)
4. **Validation:**
   - Cross-verify between different detection methods
   - Manually inspect a sample of flagged readings
   - Update models regularly with feedback

Python Code:

```
sensor_data = lifesat.drop(columns=['TIME', 'TIME_2', 'cluster'])
z_scores = np.abs((sensor_data - sensor_data.mean()) / sensor_data.std())
threshold = 3
anomalies_zscore = lifesat[np.any(z_scores > threshold, axis=1)]
print(f"Z-score method detected {len(anomalies_zscore)} anomalies")
iso_forest = IsolationForest(contamination=0.05, random_state=42)
lifesat['anomaly'] = iso_forest.fit_predict(X_scaled)
anomalies_iforest = lifesat[lifesat['anomaly'] == -1]
print(f"Isolation Forest detected {len(anomalies_iforest)} anomalies")
if len(anomalies_iforest) > 0:
    print("\nExample anomalous readings:")
    print(anomalies_iforest[['PIR_SENSOR', 'GPS_X', 'GPS_Y', 'THERMISTOR( F )']].head())
plt.figure(figsize=(10, 6))
plt.scatter(lifesat['GPS_X'], lifesat['GPS_Y'], c=lifesat['anomaly'], cmap='viridis',
    alpha=0.7)
plt.colorbar(label='Anomaly (-1) vs Normal (1)')
plt.xlabel('GPS_X')
plt.ylabel('GPS_Y')
plt.title('Anomaly Detection Results')
plt.grid(True)
plt.show()
plt.close()
```

Output:

PIR_SENSOR	GPS_X	GPS_Y	THERMISTOR (F)
89	666	234	58
89	666	234	58
89	666	234	58
89	666	234	58
89	666	234	58