Code Explanation for Sensor Data Processing

1. Where to Input Sensor Data

The sensor data should be placed in the `field` array.

Example: The `field` array is declared and initialized like this:

```
140             num = 0;
141         } else {
142             row = 0;
143             col = 0;
144             for( int a= 0;a<MAX_ROWS;a++){
145                 col = 0;
146                 // your data which format you wanna put here lik ax,ay,az
147             }
148 
149             //char tempoarray[MAX_COLS][MAX_FIELD_LENGTH];
150             while (row < MAX_ROWS && field[row][0] != '\0') {
151                 col = 0;
152                 float ax,ay,az,gx,gy,gz;
153                 ax = field[row][0];
154                 ay = field[row][1];
155                 az = field[row][2];
156                 gx = field[row][3];
157                 gy = field[row][4];
158                 gz = field[row][5];
```

Replace the sensor reading in line no **146**. Each row in `field` should contain one line of sensor data:

For example

```
for (int a = 0; a < 200; a++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    field[a][0] = a.acceleration.x;
    field[a][1] = a.acceleration.y;
    field[a][2] = a.acceleration.z;
    field[a][3] = g.gyro.x;
    field[a][4] = g.gyro.y;
    field[a][5] = g.gyro.z;
    delay(1000); // delay for 1 sec for each data
```

2. Where to Process Data

The `while(1)` loop processes the sensor data. The key logic is:

```
while (row < MAX_ROWS && field[row][0] != '\0') {

        col = 0;

        float ax,ay,az,gx,gy,gz;

        ax = field[row][0];

        ay = field[row][1];

        az = field[row][2];

        gx = field[row][3];

        gy = field[row][4];

        gz = field[row][5];
```

Ensure the `field` array has the correct sensor data before this loop begins. Also, verify the tokenization logic matches the input format.

3. How the Code Processes Sensor Values

Each sensor reading is passed to the `Health()` function. This function should return:
- `1` or `2`: For significant activity (e.g., estrus).
- `0`: For normal conditions.

```
if (Health(ax, ay, az, gx, gy, gz) == 1 || Health(ax, ay, az, gx, gy, gz) == 2) {
   numberOne++;
} else {
   numberZero++;
}
```

4. How to Handle Data for Daily Trends

Every 108 loops (representing 4 hours of data):
- **First Three Days:** Store `numberOne` counts into a circular queue and compare values using the `CompareData()` function after three days.
- **Subsequent Days:** Replace the oldest data in the queue with new counts and compare values for warnings or normal behavior.

5. What the Output Represents

The `CompareData()` function compares the daily trends. It returns:
- `1`: For a warning condition (e.g., estrus).

- `0`: For normal behavior.

```
int CompareData(int firstDay, int secondDay, int thirdDay) {
  if (firstDay > (secondDay + (0.1 * secondDay))) {
    return 1; // Warning
  }
  return 0; // OK
}
```

The result is stored in a doubly linked list (`result_data`) for further processing.

Summary Checklist for Developers

1. **Input Sensor Data:** Update the `field` array with real sensor data in the correct format.
2. **Verify Processing:** Check the `Health()` function logic to ensure it processes sensor values correctly.
3. **Adjust Trend Analysis:** Modify the `CompareData()` logic if needed.
4. **Review Output:** Results in `result_data` should reflect correct warnings or normal behavior.