Challenging Task – 5

Name: Gangireddy Madhurima

Reg Number: 21MIS1155

Task: Implement K-Means Clustering in Node-RED

Aim :

To implement K-Means Clustering in Node-RED using temperature and humidity data to classify the data points into two clusters and visualize the results on the dashboard.

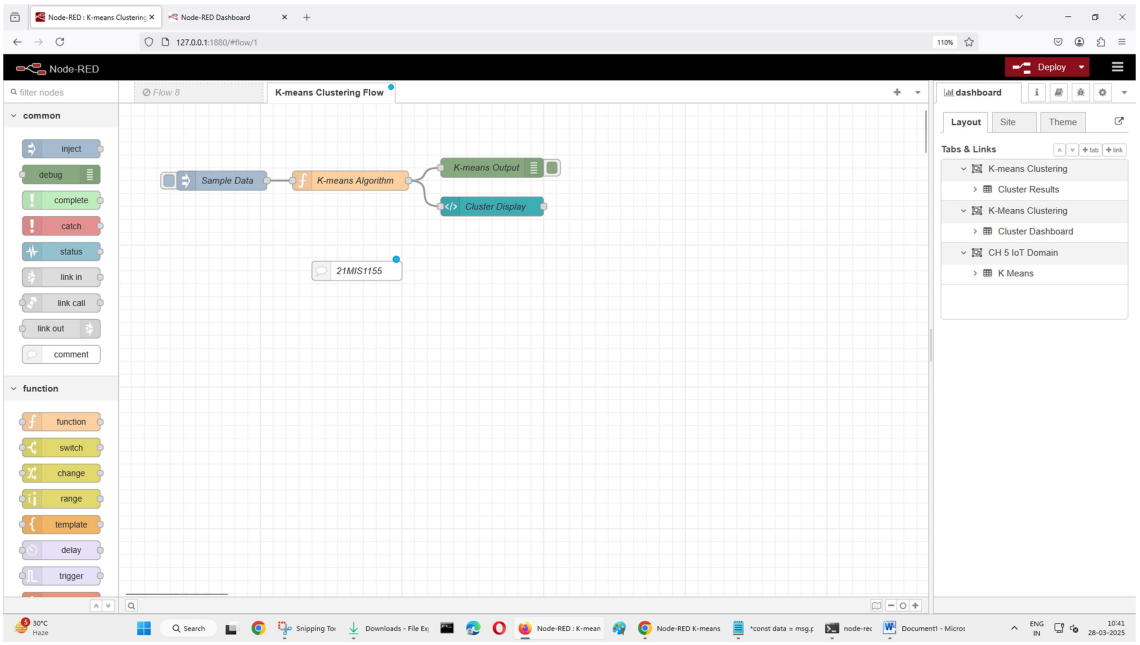
Procedure:

1. Open Node-RED
2. **Inject Node**: Add an Inject node to input temperature and humidity values.
3. **Function Node:** Add a Function node and insert the K-Means clustering code.
4. Input the temperature and humidity data, perform clustering, and assign clusters.
5. **Debug Node:** Add a Debug node to output the cluster results in the debug window.
6. **Dashboard Visualization**: Add a UI Table or UI Text node to display temperature, humidity, and cluster assignments on the dashboard.
7. **Deploy and Test:** Deploy the flow, click Inject, and observe the clustering results in the debug console and dashboard.

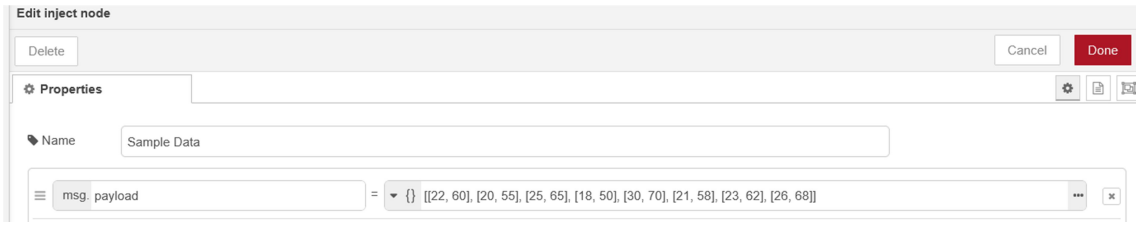
Input:

Temperature (°C)	Humidity (%)	Cluster
22	60	?
20	55	?
25	65	?
18	50	?
30	70	?
21	58	?
23	62	?
26	68	?

Node-RED Set Up:



1. **Inject Node** – Takes Input(i.e., Temperature and Humidity Values)



2. **Function Node –** For K-Means Clustering for Points

```
const data = msg.payload;

// Check if data is defined and is an array
if (!data || !Array.isArray(data)) {
    node.error("Invalid data: Payload must be an array of points.");
    return;
}

const k = 3; // Number of clusters
const maxIterations = 10;

// Randomly initialize centroids (first k points)
let centroids = data.slice(0, k);

function distance(a, b) {
    return Math.sqrt(
        a.reduce((sum, val, i) => sum + Math.pow(val - b[i], 2), 0)
    );
}

let assignments = [];

for (let i = 0; i < maxIterations; i++) {
    let clusters = Array.from({ length: k }, () => []);
    assignments = [];
```

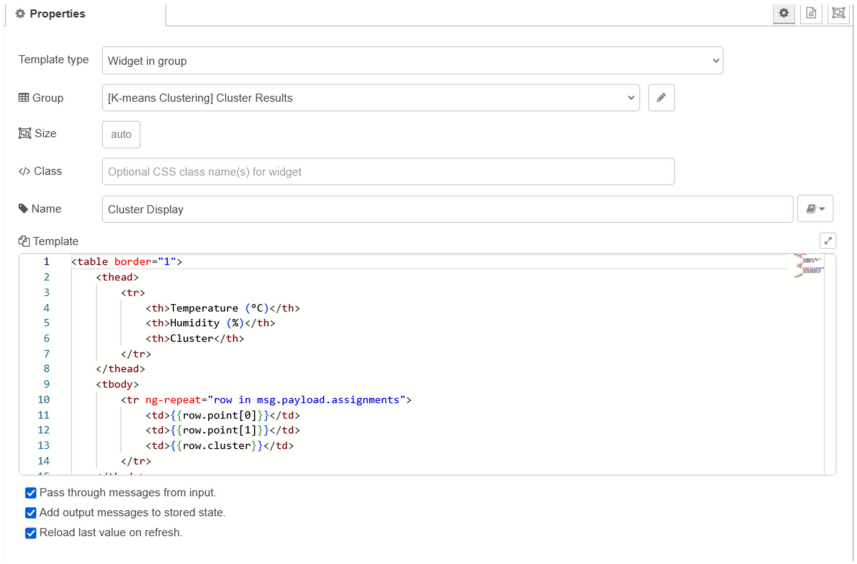
```
    // Assign each point to the nearest centroid
    data.forEach(point => {
        let distances = centroids.map(c => distance(point, c));
        let clusterIndex = distances.indexOf(Math.min(...distances));
        clusters[clusterIndex].push(point);
        assignments.push({ point, cluster: clusterIndex });
    });

    // Recalculate centroids
    centroids = clusters.map(cluster => {
        let len = cluster.length;
        if (len === 0) return Array(data[0].length).fill(0); // Prevent NaN
        let sum = cluster.reduce((acc, point) => {
            return acc.map((val, idx) => val + point[idx]);
        }, Array(data[0].length).fill(0));
        return sum.map(val => val / len);
    });
}

// Output includes final centroids and point-cluster mapping
msg.payload = {
    finalCentroids: centroids,
    assignments: assignments
};

return msg;
```

3. Dashboard – HTML Template



HTML Code:

```
<table border="1">
    <thead>
        <tr>
            <th>Temperature (°C)</th>
            <th>Humidity (%)</th>
            <th>Cluster</th>
        </tr>
    </thead>
    <tbody>
        <tr ng-repeat="row in msg.payload.assignments">
            <td>{{row.point[0]}}</td>
            <td>{{row.point[1]}}</td>
```

```
            <td>{{row.cluster}}</td>
        </tr>
    </tbody>
</table>
```

Json Code:

```
[
    {
        "id": "1",
        "type": "tab",
        "label": "K-means Clustering Flow",
        "disabled": false,
        "info": ""
    },
    {
        "id": "2",
        "type": "inject",
        "z": "1",
        "name": "Sample Data",
        "props": [
            {
                "p": "payload"
            }
        ],
        "repeat": "",
        "crontab": "",
        "once": false,
        "onceDelay": 0.1,
        "topic": "",
        "payload": "[[22, 60], [20, 55], [25, 65], [18, 50], [30, 70], [21, 58], [23, 62], [26, 68]]",
        "payloadType": "json",
        "x": 160,
        "y": 120,
        "wires": [
            [
                "3"
            ]
        ]
    },
    {
        "id": "3",
        "type": "function",
        "z": "1",
        "name": "K-means Algorithm",
        "func": "const data = msg.payload;\n\n// Check if data is defined and is an array\nif (!data ||
!Array.isArray(data)) {\n    node.error(\"Invalid data: Payload must be an array of points.\");\n        return;\n
}\n\n        const k = 3; // Number of clusters\n        const maxIterations = 10;\n\n        // Randomly initialize
centroids (first k points)\n        let centroids = data.slice(0, k);\n\n        function distance(a, b) {\n
return Math.sqrt(\n                    a.reduce((sum, val, i) => sum + Math.pow(val - b[i], 2), 0)\n
);\n                        }\n\n                        let assignments = [];\n\n                         for (let i
= 0; i < maxIterations; i++) {\n                            let clusters = Array.from({ length: k }, () => []);\n
assignments = [];\n\n                                    // Assign each point to the nearest centroid\n
data.forEach(point => {\n                                                let distances = centroids.map(c =>
distance(point, c));\n                                                     let clusterIndex =
distances.indexOf(Math.min(...distances));\n
clusters[clusterIndex].push(point);\n
assignments.push({ point, cluster: clusterIndex });\n
});\n\n                                                                                    // Recalculate centroids\n
centroids = clusters.map(cluster => {\n
let len = cluster.length;\n
if (len === 0) return Array(data[0].length).fill(0); // Prevent NaN\n
let sum = cluster.reduce((acc, point) => {\n
return acc.map((val, idx) => val + point[idx]);\n
}, Array(data[0].length).fill(0));\n
return sum.map(val => val / len);\n
});\n
}\n\n
// Output includes final centroids and point-cluster mapping\n
msg.payload = {\n
finalCentroids: centroids,\n
assignments: assignments\n
};\n\n
```

```
return msg;\n
",
        "outputs": 1,
        "timeout": "",
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 360,
        "y": 120,
        "wires": [
            [
                "4",
                "5"
            ]
        ]
    },
    {
        "id": "4",
        "type": "debug",
        "z": "1",
        "name": "K-means Output",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "false",
        "statusVal": "",
        "statusType": "auto",
        "x": 580,
        "y": 100,
        "wires": []
    },
    {
        "id": "5",
        "type": "ui_template",
        "z": "1",
        "group": "6",
        "name": "Cluster Display",
        "order": 0,
        "width": 0,
        "height": 0,
        "format": "<table border=\"1\">\n    <thead>\n        <tr>\n            <th>Temperature (°C)</th>\n
<th>Humidity (%)</th>\n            <th>Cluster</th>\n        </tr>\n    </thead>\n    <tbody>\n        <tr ng-
repeat=\"row in msg.payload.assignments\">\n            <td>{{row.point[0]}}</td>\n
<td>{{row.point[1]}}</td>\n            <td>{{row.cluster}}</td>\n        </tr>\n    </tbody>\n</table>",
        "storeOutMessages": true,
        "fwdInMessages": true,
        "resendOnRefresh": true,
        "templateScope": "local",
        "className": "",
        "x": 540,
        "y": 180,
        "wires": [
            []
        ]
    },
    {
        "id": "30e150169db808c0",
        "type": "comment",
        "z": "1",
        "name": "21MIS1155",
        "info": "",
        "x": 370,
        "y": 260,
        "wires": []
    },
    {
        "id": "6",
        "type": "ui_group",
        "name": "Cluster Results",
        "tab": "7",
        "order": 1,
        "disp": true,
```

```
        "width": "6",
        "collapse": false
    },
    {
        "id": "7",
        "type": "ui_tab",
        "name": "K-means Clustering",
        "icon": "dashboard",
        "order": 1,
        "disabled": false,
        "hidden": false
    }
]
```

Output :