Challenging Task-5

T.Akash Reddy

21MIS1128

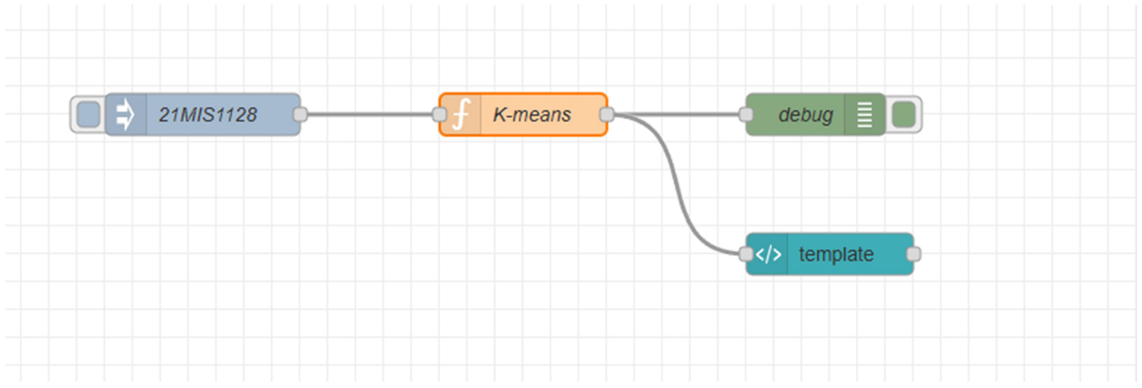
Problem:

Consider a sample dataset that have 8 data points representing temperature and humidity readings from environmental sensors. Perform K-means clustering algorithm for the given data set using Euclidean Distance. Display the sample out put in node red browser as table shown below. Consider there are three clusters for the given dataset.

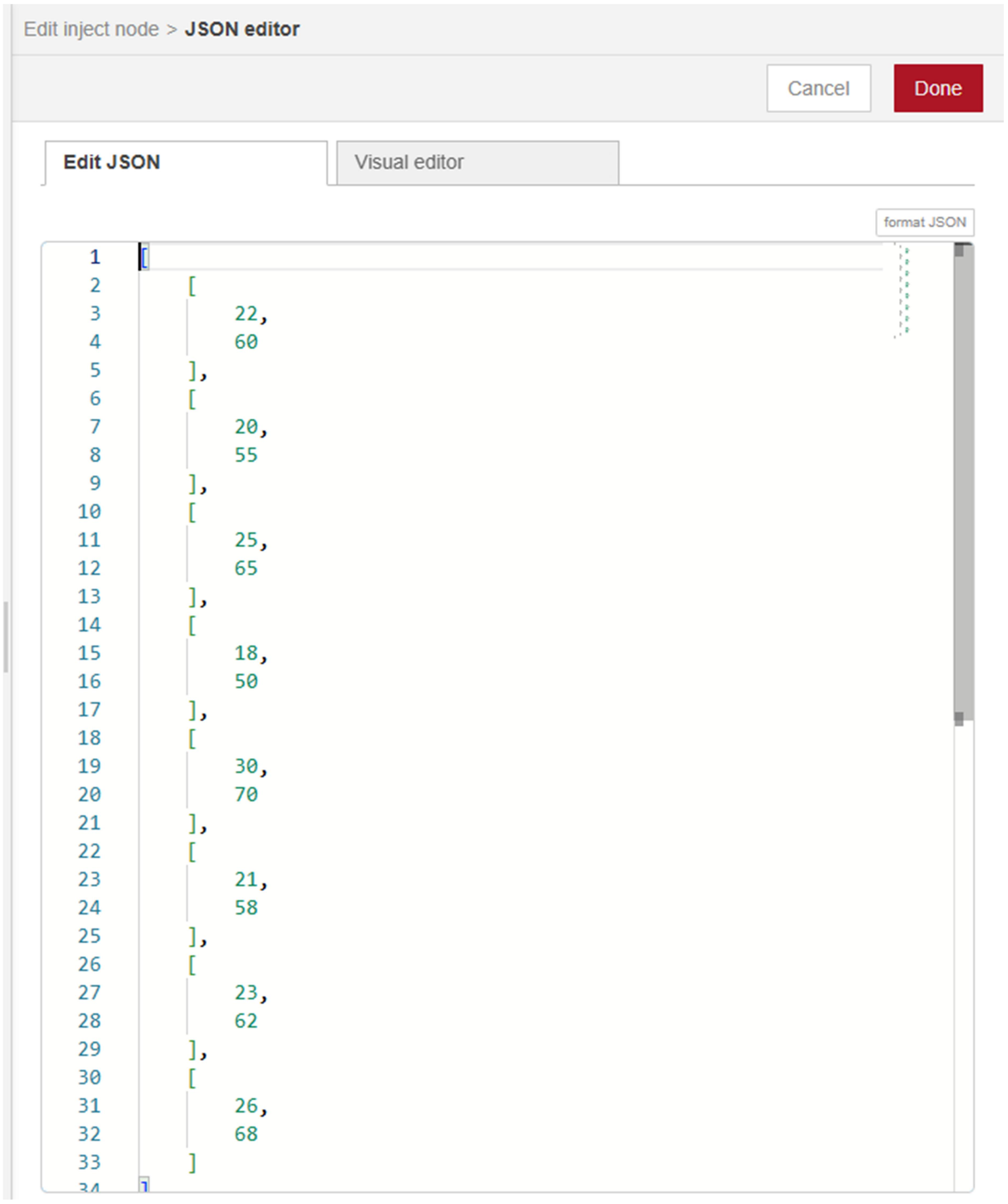
Procedure:

- Add an inject node with the input data as json format.
- Then add a function node and add the code for kmeans clustering that supports manhattan distance.
- Next add a template node in which we have to write the html code that shows a 2d graph where x axis shows cluster points and y axis shows the data points.

Node Red Flow:



Inject node:



```
[
    [
        22,
        60
    ],
    [
        20,
        55
    ],
    [
        25,
        65
    ],
    [
        18,
        50
    ],
    [
        30,
        70
    ],
    [
        21,
        58
    ],
    [
        23,
        62
    ],
    [
        26,
        68
    ]
]
```

Function node code:

```
const data = msg.payload;
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 a.reduce((sum, val, i) => sum + Math.abs(val - b[i]), 0);
}

let assignments = [];

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

    // Assign each point to 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.map(({ point, cluster }) => ({ point,
cluster })),
};
```

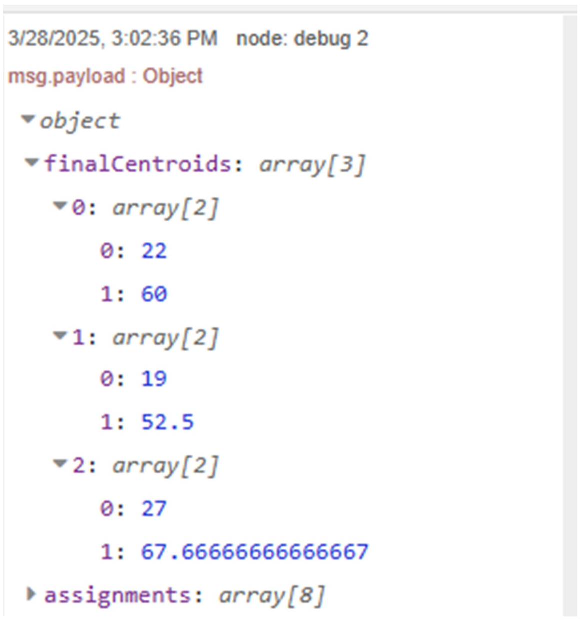
```
let c = msg.payload.finalCentroids;  // Example: {0: [22,60], 1:
[45,80], 2: [70,30]}
let a = msg.payload.assignments;  // Example: [{point: [22,60],
cluster: 0}, ...]

let d = [];

// Add centroids with a distinct color
Object.entries(c).forEach(([cluster, coord]) => {
    d.push({ x: coord[0], y: coord[1], label: `Cluster ${cluster}
(Centroid)`, type: "centroid" });
});

// Add assigned points
a.forEach(({ point, cluster }) => {
    d.push({ x: point[0], y: point[1], label: `Cluster ${cluster}`,
type: "point", cluster });
});

msg.payload = d;
return msg;


return msg;
```

Template node code: Displays 2d graph with x and y axis

```
<canvas id="scatterChart" width="1000px" height="500px"></canvas>

<script>
    (function(scope) {
        let ctx =
document.getElementById("scatterChart").getContext("2d");

        scope.$watch('msg.payload', function(data) {
            if (!data) return;

            let centroids = data.filter(d => d.type === "centroid");
            let points = data.filter(d => d.type === "point");

            let colors = ["red", "blue", "green", "orange", "purple"];
// Define cluster colors
```

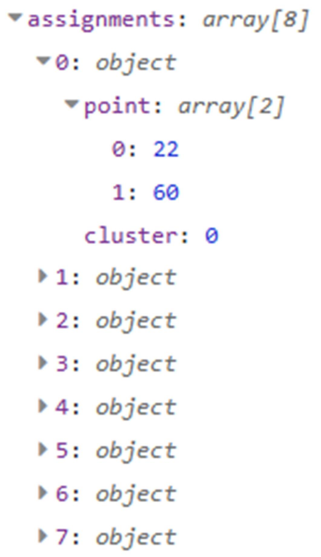
```
            let datasets = [];

            // Add centroids dataset
            datasets.push({
                label: "Centroids",
                data: centroids.map(d => ({ x: d.x, y: d.y })),
                backgroundColor: "black",
                pointRadius: 7,
                pointStyle: "triangle"
            });

            // Add points dataset per cluster
            let clusters = [...new Set(points.map(p => p.cluster))];
            clusters.forEach((cluster, index) => {
                datasets.push({
                    label: `Cluster ${cluster}`,
                    data: points.filter(p => p.cluster ===
cluster).map(d => ({ x: d.x, y: d.y })),
                    backgroundColor: colors[index % colors.length],
                    pointRadius: 5
                });
            });

            // Create or update the chart
            if (scope.chart) {
                scope.chart.data.datasets = datasets;
                scope.chart.update();
            } else {
                scope.chart = new Chart(ctx, {
                    type: 'scatter',
                    data: { datasets },
                    options: {
                        responsive: true,
                        scales: {
                            x: { title: { display: true, text: "X-Axis
(Centroids)" } },
                            y: { title: { display: true, text: "Y-Axis
(Points)" } }
                        }
                    }
                });
            }
        });
    })(scope);
</script>
```

Debug output: Shows the centroids and data points within each cluster



```
3/28/2025, 3:02:36 PM   node: debug 2
msg.payload : Object
▾object
 ▾finalCentroids: array[3]
   ▾0: array[2]
      0: 22
      1: 60
   ▾1: array[2]
      0: 19
      1: 52.5
   ▾2: array[2]
      0: 27
      1: 67.66666666666667
 ▸assignments: array[8]
```

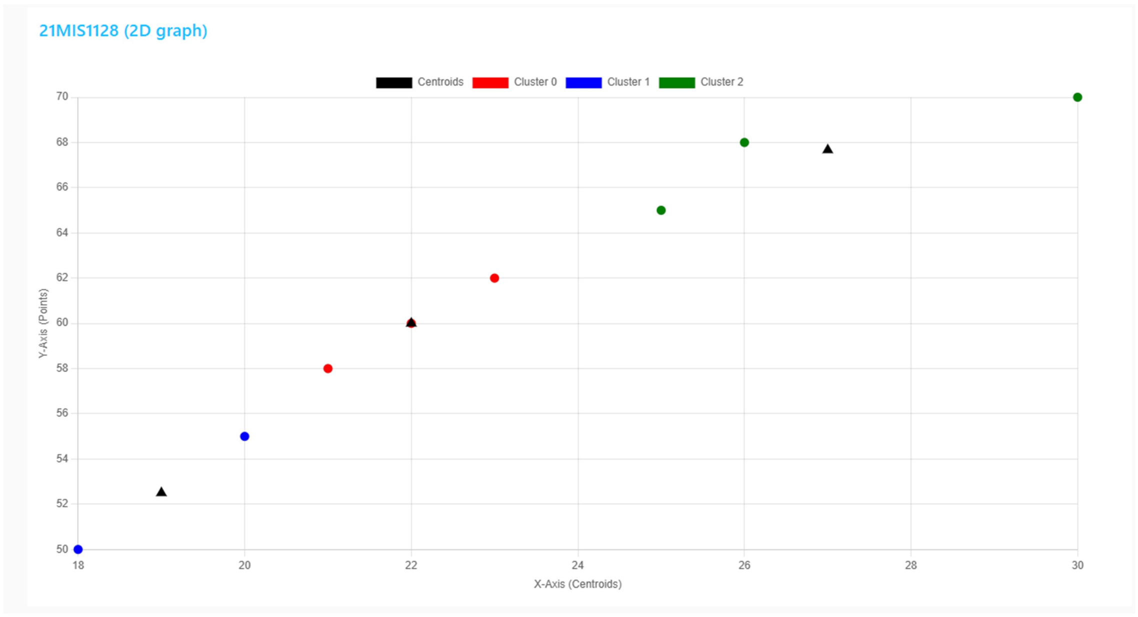
These are the centroids.



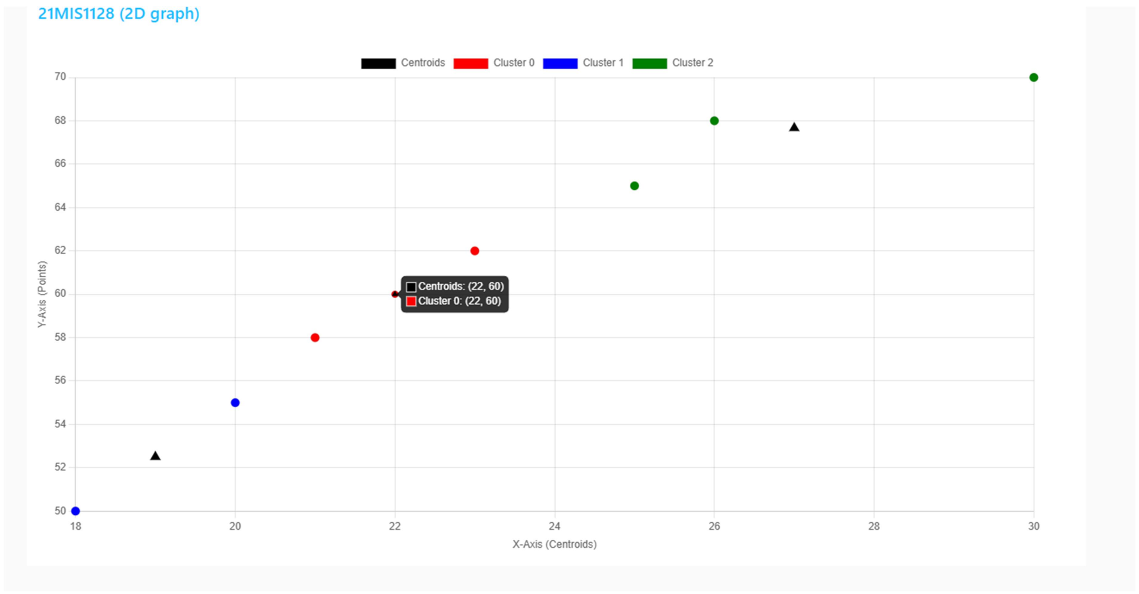
```
▾assignments: array[8]
  ▾0: object
    ▾point: array[2]
       0: 22
       1: 60
     cluster: 0
  ▸1: object
  ▸2: object
  ▸3: object
  ▸4: object
  ▸5: object
  ▸6: object
  ▸7: object
```

This shows the cluster for each data points.

Chart Output:



- Here black color triangles represents the centroids.
- Red, Blue,Green color circles represents the cluster 0,1,2.
- Graph helps us understand data points under different clusters.



- Here we can see the data point 22,60 is in the cluster 0 which exactly matches with our debug output in the same way we can check for other data points.

Json code:

```
[
   {
      "id": "636a9cd3bd38a9a7",
      "type": "tab",
      "label": "Flow 2",
      "disabled": false,
      "info": "",
      "env": []
   },
   {
      "id": "80d847b41d379e91",
      "type": "function",
      "z": "636a9cd3bd38a9a7",
      "name": "K-means",
      "func": "const data = msg.payload;\nconst k = 3; // number of
clusters\nconst maxIterations = 10;\n\n// Randomly initialize centroids (first k
points)\nlet centroids = data.slice(0, k);\n\nfunction distance(a, b) {\n    return
a.reduce((sum, val, i) => sum + Math.abs(val - b[i]), 0);\n}\n\nlet assignments
= [];\n\nfor (let i = 0; i < maxIterations; i++) {\n    let clusters = Array.from({
length: k }, () => []);\n    assignments = [];\n\n    // Assign each point to 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\nmsg.payload = {\n    finalCentroids: centroids,\n    assignments:
assignments.map(({ point, cluster }) => ({ point, cluster })),\n};\n\nlet c =
msg.payload.finalCentroids;  // Example: {0: [22,60], 1: [45,80], 2: [70,30]}\nlet
a = msg.payload.assignments;  // Example: [{point: [22,60], cluster: 0},
...]\n\nlet d = [];\n\n// Add centroids with a distinct
color\nObject.entries(c).forEach(([cluster, coord]) => {\n    d.push({ x: coord[0],
y: coord[1], label: `Cluster ${cluster} (Centroid)`, type: \"centroid\" });\n});\n\n//
Add assigned points\na.forEach(({ point, cluster }) => {\n    d.push({ x: point[0],
y: point[1], label: `Cluster ${cluster}`, type: \"point\", cluster
});\n});\n\nmsg.payload = d;\nreturn msg;\n\n\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 620,
        "y": 220,
        "wires": [
            [
                "f72bde566935353f",
                "c75f0236e8a35d53"
            ]
        ]
    },
    {
        "id": "16a7decf19d309a3",
        "type": "inject",
        "z": "636a9cd3bd38a9a7",
        "name": "21MIS1128",
        "props": [
            {
                "p": "payload"
            },
            {
                "p": "topic",
                "vt": "str"
            }
        ],
        "repeat": "",
        "crontab": "",
        "once": false,
        "onceDelay": 0.1,
        "topic": "",
        "payload":
"[[22,60],[20,55],[25,65],[18,50],[30,70],[21,58],[23,62],[26,68]]",
        "payloadType": "json",
        "x": 390,
        "y": 220,
        "wires": [
            [
                "80d847b41d379e91"
            ]
        ]
    },
    {
        "id": "f72bde566935353f",
        "type": "debug",
        "z": "636a9cd3bd38a9a7",
        "name": "debug",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 830,
        "y": 220,
        "wires": []
    },
    {
        "id": "c75f0236e8a35d53",
        "type": "ui_template",
        "z": "636a9cd3bd38a9a7",
        "group": "0a8e951cbc9c3edb",
        "name": "",
        "order": 0,
        "width": 0,
        "height": 0,
        "format": "<canvas id=\"scatterChart\" width=\"1000px\"
height=\"500px\"></canvas>\n\n<script>\n    (function(scope) {\n        let ctx =
document.getElementById(\"scatterChart\").getContext(\"2d\");\n\n
scope.$watch('msg.payload', function(data) {\n            if (!data) return;\n\n
let centroids = data.filter(d => d.type === \"centroid\");\n            let points =
data.filter(d => d.type === \"point\");\n\n            let colors = [\"red\", \"blue\",
\"green\", \"orange\", \"purple\"]; // Define cluster colors\n\n            let datasets
= [];\n\n            // Add centroids dataset\n            datasets.push({\n
label: \"Centroids\",\n                data: centroids.map(d => ({ x: d.x, y: d.y })),\n
backgroundColor: \"black\",\n                pointRadius: 7,\n                pointStyle:
\"triangle\"\n            });\n\n            // Add points dataset per cluster\n            let
clusters = [...new Set(points.map(p => p.cluster))];\n
clusters.forEach((cluster, index) => {\n                datasets.push({\n
label: `Cluster ${cluster}`,\n                    data: points.filter(p => p.cluster ===
cluster).map(d => ({ x: d.x, y: d.y })),\n                    backgroundColor:
colors[index % colors.length],\n                    pointRadius: 5\n                });\n
});\n\n            // Create or update the chart\n            if (scope.chart) {\n
scope.chart.data.datasets = datasets;\n                scope.chart.update();\n
} else {\n                scope.chart = new Chart(ctx, {\n                    type:
```

```
'scatter',\n                data: { datasets },\n                options: {\n
responsive: true,\n                    scales: {\n                        x: { title: {
display: true, text: \"X-Axis (Centroids)\" } },\n                        y: { title: {
display: true, text: \"Y-Axis (Points)\" } }\n                    }\n                }\n
});\n           }\n        });\n    })(scope);\n</script>",
        "storeOutMessages": true,
        "fwdInMessages": true,
        "resendOnRefresh": true,
        "templateScope": "local",
        "className": "",
        "x": 840,
        "y": 320,
        "wires": [
            []
        ]
    },
    {
        "id": "0a8e951cbc9c3edb",
        "type": "ui_group",
        "name": "21MIS1128 (2D graph)",
        "tab": "e1010ad4089e4617",
        "order": 1,
        "disp": true,
        "width": "6",
        "collapse": false,
        "className": ""
    },
    {
        "id": "e1010ad4089e4617",
        "type": "ui_tab",
        "name": "Kmeans",
        "icon": "dashboard",
        "order": 6,
        "disabled": false,
        "hidden": false
    }
]
```