Code

Task 1&2 :-

// Input values from the form submission with checks for undefined or empty strings

let companySize = inputData["Company Size"] ? inputData["Company Size"].trim() : ""; // Trim spaces if any

let annualBudget = inputData["Annual Budget"] ? inputData["Annual Budget"].trim() : "";

let industry = inputData["Industry"] ? inputData["Industry"].trim() : "";

let urgency = inputData["Urgency of Need"] ? inputData["Urgency of Need"].trim() : ""; // Trim spaces for urgency

console.log("Company Size:", companySize); // Debug log

console.log("Annual Budget:", annualBudget); // Debug log

console.log("Industry:", industry); // Debug log

console.log("Urgency:", urgency); // Debug log

// Initialize lead score

let leadScore = 0;

// Scoring logic for Company Size

if (companySize === "1- 50 employees") {

leadScore += 5;

} else if (companySize === "51- 200 employees") {

leadScore += 10;

} else if (companySize === "201- 1000 employees") {

leadScore += 15;

} else if (companySize === "1000+ employees") {

leadScore += 20;

} else {

leadScore += 0; // Default score for unknown company size

}

// Scoring logic for Annual Budget

if (annualBudget === "Less than $10,000") {

leadScore += 5;

} else if (annualBudget === "$10,000 - $50,000") {

leadScore += 10;

} else if (annualBudget === "$50,001 - $100,000") {

leadScore += 15;

} else if (annualBudget === "More than $100,000") {

leadScore += 20;

} else {

leadScore += 0; // Default score for unknown or missing budget range

}

// Scoring logic for Industry

if (industry === "Technology") {

leadScore += 15;

} else if (industry === "Finance") {

leadScore += 10;

} else if (industry === "Healthcare") {

leadScore += 20;

} else if (industry === "Retail") {

leadScore += 10;

} else {

leadScore += 5; // Default score for "Other" industry

}

// Scoring logic for Urgency of Need

if (urgency === "Immediate") {

leadScore += 20;

} else if (urgency === "Short-term") {

leadScore += 30;

} else if (urgency === "Medium- term") {

leadScore += 35;

} else if (urgency === "Long- term") {

leadScore += 40;

} else {

leadScore += 0; // Default score for unknown or missing urgency

}

// Output the total lead score

console.log("Total Lead Score:", leadScore); // Debug log

output = { totalScore: leadScore };

Task - 3

Code for assignedRepIndex   
// Input data: Row number and the number of sales reps

const rowNumber = inputData.rowNumber; // This is the row number that triggers the Zap (from the Google Sheets row)

const numSalesReps = inputData.numSalesReps; // Total number of sales reps (from your SalesRepList)

const assignedRepIndex = (rowNumber - 1) % numSalesReps; // Round-robin logic

return { assignedRepIndex: assignedRepIndex + 1 }; // Adding 1 to ensure it starts from 1, not 0

Code for assignedSalesRep

// List of sales reps

const salesReps = ['Alice', 'Bob', 'Charlie']; // Modify this list with your actual sales reps

// Get the sales rep based on the index

const assignedSalesRep = salesReps[inputData.assignedRepIndex - 1]; // Subtract 1 because the index starts at 1

return { assignedSalesRep: assignedSalesRep };

Code for assigning category to the comments

// Input data from the trigger

const comments = inputData.comments || ""; // Get the Comments field, or set it to an empty string if missing

let category = "Uncategorized"; // Default category

// Define keywords and their corresponding categories

const keywordMap = [

{ keyword: "pricing", category: "Pricing Request" },

{ keyword: "support", category: "Support Request" },

{ keyword: "demo", category: "Demo Request" },

{ keyword: "integration", category: "Integration Request" },

{ keyword: "urgent", category: "High Priority" }

];

// Check the comments for each keyword

keywordMap.forEach(item => {

const regex = new RegExp(\\b${item.keyword}\\b, "i"); // Create a case-insensitive regex for the keyword

if (regex.test(comments)) {

category = item.category; // Assign the corresponding category

}

});

// Return the category for further steps

return { category: category };

code to Calculate the follow-up date

// Input variables

const currentDateString = inputData.current\_date; // Current date string

const urgency = inputData.urgency || "Medium-term"; // Default urgency if not provided

// Validate the input date string

if (!currentDateString || isNaN(Date.parse(currentDateString))) {

throw new Error("Invalid date string provided. Please check the current\_date input.");

}

// Convert the valid date string to a Date object

const currentDate = new Date(currentDateString);

let daysToAdd = 0;

// Map urgency levels to days

switch (urgency.toLowerCase()) {

case "immediate":

daysToAdd = 1;

break;

case "short-term":

daysToAdd = 3;

break;

case "medium-term":

daysToAdd = 7;

break;

case "long-term":

daysToAdd = 14;

break;

default:

daysToAdd = 7; // Default to Medium-term

}

// Calculate the follow-up date

const followUpDate = new Date(currentDate);

followUpDate.setDate(followUpDate.getDate() + daysToAdd);

// Format the follow-up date as an ISO string (optional)

const followUpDateISO = followUpDate.toISOString();

// Output the follow-up date and additional info

return {

follow\_up\_date: followUpDateISO,

urgency: urgency,

days\_added: daysToAdd

};