

Documentation: Sales Insights Automation with n8n + LangChain

Why n8n + LangChain?

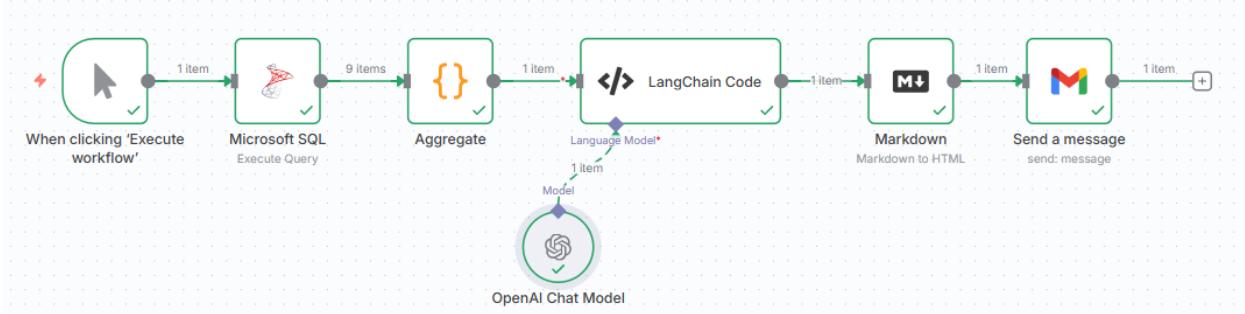
Business users often need quick **executive insights** from raw data, not just numbers, but **summaries, highlights, and trends**.

- **n8n** makes it easy to connect data sources (SQL Server, APIs, CSV, etc.), process data, and automate workflows without heavy coding.
- **LangChain** enables the integration of an **LLM (Large Language Model)** into the workflow to transform raw data into narrative **insights, summaries, and Q&A bots**.

Together, this creates a **no-code + AI-powered data pipeline**:

1. **n8n** handles the data extraction and orchestration
2. **LangChain** handles intelligent summarization & reasoning

Workflow Overview for Sales Insights demo



1. Trigger Node (Manual / Scheduled)

- Start the workflow manually or via a cron schedule.
- Useful for daily/weekly reporting.

2. Microsoft SQL Node

- Connect to the Sales dataset with 200 rows (via SQL database).
- Executes the query:

```
SELECT
    Region,
    Product,
    SUM([Revenue]) AS TotalRevenue,
    SUM([Units Sold]) AS TotalUnits
FROM Sales
WHERE Date BETWEEN '2025-01-01' AND '2025-03-31'
GROUP BY Region, Product
ORDER BY TotalRevenue DESC;
```

- Outputs aggregated metrics by **region** and **product**.

3. Aggregate Node (Code)

- Combine all SQL rows into a **single item** for the AI.
- Prepare a **datasetText** string that's easy for an LLM to understand.
- Example code:

```
const allData = items.map(item => ({
    Region: item.json.Region,
    Product: item.json.Product,
    TotalRevenue: parseFloat(item.json.TotalRevenue),
    TotalUnits: parseInt(item.json.TotalUnits)
}));

return [
    json: {
        completeDataset: allData,
        totalRecords: allData.length,
        datasetText: allData
            .map(row => `${row.Region} | ${row.Product} | Revenue: ${row.TotalRevenue} | Units: ${row.TotalUnits}`)
            .join('\n')
    }
];
```

- Instead of 9 separate rows to the AI, **one consolidated input** with structured text is produced.

4. OpenAI Chat Model Node

- Provide the LLM (e.g., GPT-4-mini).
- Connected to LangChain Code as the “AI engine”.

5. LangChain Code Node

- This is the module responsible for **intelligence operations**.
- Uses LangChain’s PromptTemplate + SummarizationChain.
- Example workflow:

```
// Get aggregated sales data from previous node
const salesData = $input.item.json.completeDataset;

if (!salesData || salesData.length === 0) {
  throw new Error("No sales data provided to the node.");
}

// Convert data into a readable string for the LLM
const salesText = salesData
  .map(row => `${row.Region} | ${row.Product} | Revenue: ${row.TotalRevenue} | Units: ${row.TotalUnits}`).join("\n");

// Prepare summarization prompts
const { PromptTemplate } = require("@langchain/core/prompts");
const { loadSummarizationChain } = require("langchain/chains");

// Get the AI model connected to this LangChain Code node
const llmSummary = await this.getInputConnectionData('ai_languageModel', 0);

const summaryTemplate = `
You are a sales analyst. Analyze the following sales data and write an executive summary for management:
- Highlight top-performing regions by revenue
- Highlight best-selling products by units
- Identify any interesting trends or anomalies

Sales data:
-----
{text}
-----
Provide a concise executive summary and also suggest 2-3 actionable insights or recommendations for the management team.
`;

const SUMMARY_PROMPT = PromptTemplate.fromTemplate(summaryTemplate);

const summaryRefineTemplate = `
You are a sales analyst. You have an existing summary: {existing_answer}
Use the new data below to refine the summary, ensuring the key insights and recommendations are clear and concise.

Sales data:
-----
{text}
-----
If the new data does not change the insights, keep the original summary. Return the refined summary and actionable recommendations.
`;

const SUMMARY_REFINE_PROMPT = PromptTemplate.fromTemplate(summaryRefineTemplate);

// Load summarization chain (refine type)
const summarizeChain = loadSummarizationChain(llmSummary, {
  type: "refine",
  verbose: true,
  questionPrompt: SUMMARY_PROMPT,
  refinePrompt: SUMMARY_REFINE_PROMPT,
});

// Run the chain
const summary = await summarizeChain.run([{ pageContent: salesText }]);

// Return summary
return [{ json: { summary } }];
```

- **Step-by-step breakdown:**

1. Import LangChain chain & prompt tools.
 - const { PromptTemplate } = require("@langchain/core/prompts");
 - const { loadSummarizationChain } = require("langchain/chains");
2. Access the LLM connection from the OpenAI node.
 - const llmSummary = await this.getInputConnectionData('ai_languageModel', 0);
3. Define a **prompt template** with clear business instructions.
 - const summaryTemplate = ` ... `;
 - const SUMMARY_PROMPT = PromptTemplate.fromTemplate(summaryTemplate);
 - const summaryRefineTemplate = ` ... `;
 - const SUMMARY_REFINE_PROMPT = PromptTemplate.fromTemplate(summaryRefineTemplate);
 - The refine prompt tells it: *"update the summary if the new data changes insights, otherwise keep it the same."*
4. Run the **summarization chain** on the aggregated dataset.
 - const summarizeChain = loadSummarizationChain(llmSummary, {
 type: "refine",
 verbose: true,
 questionPrompt: SUMMARY_PROMPT,
 refinePrompt: SUMMARY_REFINE_PROMPT,
});
 - const summary = await summarizeChain.run([{ pageContent: salesText }]);
5. Return the structured summary as JSON.
 - return [{ json: { summary } }];

Thus, the whole flow:

1. Take sales data in JSON.
2. Convert it to readable text.
3. Pass it through a **summarization chain** with a detailed prompt.
4. Return an executive summary + insights.

6. Markdown & Email Nodes

- Convert output from LangChain Code node to HTML to send via **Gmail**.

Key Takeaways

- **n8n** automates the pipeline (data → processing → AI → delivery).
- **LangChain** turns structured numbers into **business insights & narratives**.
- This setup is reusable for:
 - Sales reports
 - Customer support insights
 - Operational dashboards

Example: AI Output

Executive Summary

1. Top Region–Product Revenue Generators
 - EMEA – Widget C: \$113,081 (3,580 units)
 - Americas – Widget A: \$108,757 (3,177 units)
 - APAC – Widget B: \$ 90,208 (2,722 units)

2. Overall Best-Selling Products (Units)
 - Widget A: 8,167 units
 - Widget C: 8,114 units
 - Widget B: 6,967 units

3. Regional Revenue Rankings (all products)
 - Americas: \$249.8 K
 - EMEA: \$247.4 K
 - APAC: \$235.1 K

Key Trends & Anomalies

- Product–Region Affinities: EMEA strongly favors Widget C, APAC favors Widget B, and Americas favors Widget A.
- Price Variance: Average revenue per unit ranges from ~\$27.8 (Widget B in EMEA) up to ~\$34.2 (Widget A in Americas), suggesting uneven pricing or discounting strategies.
- High-Margin, Low-Volume: Widget B commands above-average unit prices but trails in volume—an opportunity to boost sales through targeted promotions.

Actionable Recommendations

1. Regional Marketing Focus
 - EMEA: Double down on Widget C with loyalty programs and upsell bundles.
 - APAC: Invest in demand generation for Widget B through localized campaigns.
 - Americas: Leverage Widget A's popularity to cross-sell Widget C (combine high-margin with high-volume appeal).
2. Price Optimization & Promotions
 - Evaluate pricing structure for low-price, high-volume combos (e.g., Widget B in EMEA) to better align margins.
 - Deploy time-limited discounts on underperforming region–product pairings to stimulate incremental units without eroding brand value.
3. Inventory & Supply Chain Alignment
 - Rebalance stock allocations toward high-demand products in each region to prevent stockouts (e.g., Widget A in Americas, Widget B in APAC).
 - Monitor lead times for high-margin SKUs and ensure on-time replenishment to capture full market potential.

Dataset & SQL Results

The image shows four separate SQL query results displayed in a grid. Each result is presented in a tabular format with columns labeled 'Product', 'Region', 'Total_Revenue', and 'Total_Units_Sold' (except for the first result which lacks a header row).

	Product	Region	Total_Revenue	Total_Units_Sold
1	Widget C	EMEA	113081	3580
2	Widget A	Americas	108757	3177
3	Widget B	APAC	90208	2722
4	Widget A	APAC	80429	2574
5	Widget A	EMEA	79511	2416
6	Widget B	Americas	70883	2277
7	Widget C	Americas	70142	2518
8	Widget C	APAC	64454	2016
9	Widget B	EMEA	54762	1968

The results are as follows:

- Result 1 (Top 1000 rows):

```
SELECT TOP (1000) [Date]
 ,[Region]
 ,[Product]
 ,[Units_Sold]
 ,[Revenue]
 FROM [SalesDemo].[dbo].[Sales]
```

Date	Region	Product	Units_Sold	Revenue	
184	2025-03-20	EMEA	Widget B	178	2136
185	2025-03-12	Americas	Widget A	100	3000
186	2025-01-24	APAC	Widget A	68	2108
187	2025-01-06	EMEA	Widget A	135	4590
188	2025-01-26	Americas	Widget A	80	4000
189	2025-03-17	EMEA	Widget A	167	4843
190	2025-01-19	Americas	Widget B	94	1598
191	2025-02-12	APAC	Widget A	59	767
192	2025-02-03	EMEA	Widget A	125	4500
193	2025-01-18	APAC	Widget A	148	5180
194	2025-03-15	APAC	Widget B	127	2413
195	2025-01-16	Americas	Widget A	124	4588
196	2025-03-24	EMEA	Widget C	34	612
197	2025-03-29	EMEA	Widget B	27	810
198	2025-01-03	APAC	Widget B	81	1053
199	2025-02-02	APAC	Widget B	167	3340
200	2025-02-20	EMEA	Widget A	146	6424
- Result 2:

```
SELECT
Product,
Region,
SUM(Revenue) AS Total_Revenue,
SUM(Units_Sold) AS Total_Units_Sold
FROM Sales
GROUP BY Product, Region
ORDER BY Total_Revenue DESC;
```

Product	Region	Total_Revenue	Total_Units_Sold
Widget C	EMEA	113081	3580
Widget A	Americas	108757	3177
Widget B	APAC	90208	2722
Widget A	APAC	80429	2574
Widget A	EMEA	79511	2416
Widget B	Americas	70883	2277
Widget C	Americas	70142	2518
Widget C	APAC	64454	2016
Widget B	EMEA	54762	1968
- Result 3:

```
SELECT Top(3)
Product,
SUM(Units_Sold) AS TotalUnits
FROM [SalesDemo].[dbo].[Sales]
GROUP BY Product
ORDER BY TotalUnits DESC;
```

Product	TotalUnits
Widget A	8167
Widget C	8114
Widget B	6967
- Result 4:

```
SELECT Top(3)
Region,
SUM(Revenue) AS TotalRevenue
FROM [SalesDemo].[dbo].[Sales]
GROUP BY Region
ORDER BY TotalRevenue DESC;
```

Region	TotalRevenue
Americas	249782
EMEA	247354
APAC	235091