Beginner's Guide — Codebasics RAG AI Agent using Vector Database

This guide will help you build a Retrieval-Augmented Generation (RAG) chatbot using **n8n**, **Pinecone**, and **Google Gemini** that answers questions about the **Codebasics Bootcamp 5.0** brochure.

Workflow 1 — Document Ingestion (Indexing)

Purpose: Load the Bootcamp brochure into Pinecone so the chatbot can retrieve it later.

Step 1 — Trigger the Workflow

- **Node**: *Execute Workflow* (Trigger)
- **Description**: Starts the indexing process manually whenever you update the brochure.

Step 2 — Download the brochure from Google Drive

- **Node**: *Google Drive → Download File*
- **Description**: Fetches the brochure PDF from Google Drive as a binary file.
- **Key Parameters**:
    - Credential: Google Drive account 2
    - Resource: File → Operation: Download
    - File: Final Codebasics DA Bootcamp 5.0 + Parameters.pdf

Step 3 — Load the file into workflow memory

- **Node**: *Default Data Loader*
- **Description**: Converts the binary PDF into a format ready for text processing.
- **Key Parameters**:
    - Type of Data: Binary
    - Mode: Load All Input Data
    - Data Format: Auto-detect by MIME type

Step 4 — Split the text into smaller chunks

- **Node**: *Recursive Character Text Splitter*
- **Description**: Splits the brochure text into manageable chunks for embedding.
- **Key Parameters**:
    - Chunk Size: 1000
    - Chunk Overlap: 0

Step 5 — Generate embeddings from text chunks

- **Node**: *Google Gemini (PaLM) — Embeddings*

- **Description**: Turns each text chunk into a vector representation.
- **Key Parameters**:
    - Model: models/text-embedding-004
    - Output Dimension: 768

Step 6 — Store vectors in Pinecone

- **Node**: *Pinecone Vector Store → Insert Documents*
- **Description**: Saves embeddings into Pinecone for later retrieval.
- **Key Parameters**:
    - Index: sample-codebasics-index1
    - Namespace: Codebasics DA Bootcamp 5.0
    - Metric: cosine
    - Dimensions: 768

Workflow 2 — Conversational Agent (Querying)

Purpose: Accepts a user question, retrieves relevant info from Pinecone, and responds in a friendly tone.

Step 1 — Trigger when chat message is received

- **Node**: *Chat Message Received*
- **Description**: Entry point for all user questions.

Step 2 — Store conversation history

- **Node**: *Memory (Simple Memory)*
- **Description**: Saves recent chat turns for context.
- **Key Parameters**:
    - Session ID: {{ $json.sessionId }}
    - Context Window Length: 10

Step 3 — Set up the AI Agent

- **Node**: *Codebasics AI Agent*
- **Description**: Orchestrates tools, retrieves info from Pinecone, and answers questions.
- **Key Parameters**:
    - Prompt (User Message): {{ $json.chatInput }}
    - System Message: *(Full friendly Bootcamp rep message you provided earlier)*

Step 4 — Add the Chat Model

- **Node**: *Google Gemini (PaLM) — Chat Model*
- **Description**: Generates natural responses based on retrieved info.
- **Key Parameters**:
    - Model: models/gemini-2.0-flash-thinking-exp-01-21

Step 5 — Create the Vector Store Tool

- **Node**: *Answer Questions with Vector Store*
- **Description**: Uses the query to search Pinecone for relevant brochure chunks.
- **Key Parameters**:
    - Description of Data: {{ $json.chatInput }}
    - Limit: 4

Step 6 — Retrieve documents from Pinecone

- **Node**: *Pinecone Vector Store — Retrieve Documents*
- **Description**: Pulls the most relevant chunks from Pinecone for the answer.
- **Key Parameters**:
    - Index: sample-codebasics-index1
    - Namespace: Codebasics DA Bootcamp 5.0

Step 7 — Add Wikipedia as fallback

- **Node**: *Wikipedia*
- **Description**: Provides general background if Pinecone returns no relevant data.

Step 8 — Output the final answer

- **Node**: *Output / Return to Chat*
- **Description**: Sends the chatbot's response back to the user interface.

Important Notes for Beginners

- Always run **Workflow 1** before Workflow 2.
- Pinecone **index dimensions** must match embedding output dimensions (768 in this setup).
- Changing **chat model** (e.g., OpenAI) requires updating model names and parameters.
- n8n Cloud and local versions have slightly different node settings.
- Store API keys in n8n credentials, not hardcoded fields.