## **🧠 1. Turn Unstructured Data Into Strategic AI Decisions**

### **🧩 What Is Unstructured Data?**

* Data that doesn't fit neatly into tables:  
   E.g., call transcripts, meeting notes, email threads, chat logs, free-form survey responses, etc.

### **🤖 How Can AI Help?**

* Use **LLMs** (Large Language Models) to:  
  + Summarize
  + Extract key themes
  + Detect sentiment
  + Tag decisions, risks, blockers, etc.

### **🛠️ How to Build This Agent**

#### **Tools:**

* **LangChain**: For document loaders + prompt templates
* **Pinecone/FAISS**: For vector search across long documents
* **OpenAI / Anthropic**: As your LLM brain

#### **Workflow:**

1. 📂 Load documents → using LangChain’s DocumentLoader
2. 📌 Chunk + embed → using TextSplitter + OpenAIEmbeddings
3. 🔍 Query via RAG → use RetrievalQA or ConversationalRetrievalChain
4. 🎯 Output: Summary of top issues, actions, blockers

#### **Example Use Case:**

“Analyze the last 10 sales team calls and summarize 3 major customer objections.”

## **👥 2. Define and Deploy Agent Roles for Strategy Workflows**

### **🧠 Why Use Multiple Roles?**

Just like in human teams, agents can be specialized to:

* 🔎 Research → Gather raw insights
* 📊 Analyze → Pull trends, patterns
* 🖋️ Scribe → Draft summaries, notes
* 🧭 Strategist → Propose actions, recommendations

### **🛠️ How to Do This (CrewAI or LangGraph)**

#### **Framework: Role-Based Agents**

Each agent:

* Has a role (defined in natural language)
* Uses a shared memory or toolset
* Passes control after task completion

#### **Example Setup:**

| **Agent** | **Description** |
| --- | --- |
| ResearchAgent | Uses web search and document retrieval |
| InsightAgent | Extracts patterns or KPIs |
| WriterAgent | Creates summaries, executive reports |
| StrategyAgent | Proposes next best action using templates |

#### **Implementation Tip:**

Use **CrewAI** or **LangGraph** to create these roles and define their task, tools, and input/output.

## **🧩 3. Orchestrate Multiple AI Agents to Collaborate on Strategy**

### **🧠 What is Orchestration?**

Think of this like a relay race:

* Agents take turns based on workflow logic
* They **share memory** or documents
* There's a **Planner** and an **Executor**

### **🛠️ Using LangGraph:**

* Use **LangGraph's state machine**:  
  + Define each agent as a node
  + Define transitions based on success/failure or task output
* Perfect for **sequential** or **looped** workflows (like OKR reviews)

#### **Example:**

“Let’s simulate an AI-led OKR planning session.”

1. ResearchAgent → pulls last quarter's performance
2. InsightAgent → finds success areas + gaps
3. StrategyAgent → proposes new OKRs
4. FeedbackAgent → refines based on manager style

## **🤝 4. Design Agent Handoffs That Replicate Human Collaboration**

### **🎯 Why This Matters:**

Most LLM pipelines fail when trying to handle **multi-turn, multi-agent, multi-tool logic**.

Handoffs = how one agent’s output becomes another’s input.

### **🛠️ Best Practices:**

* Use **structured outputs** (e.g., JSON instead of plain text)
* Clearly define:  
  + input\_template
  + expected output format
* Use **shared memory**, like:  
  + LangChain’s ConversationBuffer
  + VectorStore memory for long documents

## **🔧 Beginner Tools to Try (Plug-and-Play)**

| **Tool** | **Use** | **Docs** |
| --- | --- | --- |
| LangChain Hub | Prebuilt agent chains | hub.langchain.com |
| CrewAI | Role-based agents in teams | crewai.dev |
| LangGraph | Graph-based agent flow | langgraph.readthedocs.io |
| OpenAI Function Calling | For structured output | [platform.openai.com/docs](https://platform.openai.com/docs) |

## **🧪 Beginner Lab Idea (30–60 min)**

**“Build a Mini-Agent Team to Review a Business Report”**

1. Load a PDF with business updates.
2. Use ResearchAgent to extract metrics.
3. InsightAgent to generate insights.
4. StrategyAgent to recommend next steps.

Sample Project:

# **🧠 Business Report Analysis Pipeline with LangChain**

This example notebook demonstrates how to build a modular, production-ready pipeline using LangChain. It reads a business PDF report, extracts key metrics, generates insights, and recommends strategic next steps using LLMs and retrieval-augmented generation (RAG).

## **📚 Sections**

### **1. Installation & Setup**

Install required libraries:

pip install langchain-community unstructured[all-docs] pdfminer.six faiss-cpu langchain\_openai sentence-transformers

Ensure you restart your Colab runtime after installation.

### **2. PDF Loading & Chunking**

# Download PDF, load with UnstructuredPDFLoader

# Split into semantic chunks using RecursiveCharacterTextSplitter

chunks = load\_and\_split\_pdf(URL, chunk\_size=1000, overlap=200)

This step handles large files efficiently and tags each chunk with page metadata.

### **3. Research Agent**

metrics = research\_agent(chunks)

Uses SentenceTransformer embeddings and FAISS vectorstore to extract the top 4 relevant text chunks for the query:  
 **“List key metrics or numbers in this report.”**

### **4. Insight Agent**

insights = insight\_agent(metrics)

Converts the metrics text into higher-level insights using a chained ChatOpenAI call with a prompt template:

“Here are the metrics: {metrics} … derive trends.”

### **5. Strategy Agent**

strategies = strategy\_agent(insights)

Uses insights text to generate **three strategic next steps** via another chat model chain.

### **6. End-to-End Orchestration**

metrics, insights, strategies = run\_pipeline(URL)

Runs all agents in sequence. Outputs are printed and saved to:

all\_results.txt

## **🔧 Key Features**

* ✅ **Modular Agents** – each agent is an independent, reusable function
* ✅ **Local Embeddings** – uses SentenceTransformer to avoid API model restrictions
* ✅ **Chat-Compatible LLMs** – chains use ChatOpenAI(model="gpt-3.5-turbo") for compatibility
* ✅ **Metadata Tracking** – chunks retain source and page info
* ✅ **Robust & Shareable** – logging, error resilience, and output files make it learner-friendly

## **🔄 Customization Ideas**

* Plug in different LLMs ⚙️ (gpt-4, Anthropic Claude, etc.)
* Override chunk size and overlap for finer control
* Swap in other embedding models (open-source or paid)
* Add evaluation (via LangSmith) or interactive UI layers
* Deploy pipeline as a FastAPI or Gradio service

## **🚀 How to Run**

1. Copy the full code into a Colab notebook
2. Run cells in order: install → load PDF → run pipeline
3. Inspect outputs in notebook and check all\_results.txt
4. Share with learners for experimentation or expansion