# 4-Day Generative AI POC Plan – Priority Banking Customer Insights (Detailed Version)

## Purpose

Create a proof-of-concept (POC) web application that enables Relationship Managers (RMs) to retrieve and analyze customer data using natural language queries, combining enterprise SQL and PDF data with OpenAI's LLM via Azure services.

---

## Success Criteria

- Web UI with interactive chat
- LLM answers grounded in customer data
- Outputs include summary, product usage, cross-sell opportunities, red flags
- Hosted on Azure with working demo link or recording

---

## Day 1 – Foundations & Environment Setup

### Learning Blocks

#### L1 – Generative AI Primer (09:00–10:30)

- **Objective**: Understand the mechanics behind LLMs and Retrieval-Augmented Generation (RAG).
- **Topics**:
- What are LLMs (ChatGPT, GPT-4o, embeddings, context windows)?
- What is RAG and why it matters?
- Prompt engineering basics (system vs user roles, few-shot prompts)
- **Resources**:
- Microsoft Learn: *"Introduction to Azure OpenAI"*
- DeepLearning.AI short course: *ChatGPT Prompt Engineering*
- Anthropic blog post: *"RAG Explained Simply"*

#### L2 – Python Refresher (10:45–12:30)

- **Objective**: Gain comfort with Python basics used for GenAI POCs.
- **Topics**:
- Virtual environments, file handling, APIs
- JSON manipulation, HTTP requests, simple classes
- **Hands-On**:
- Set up `venv`, install with `pip install requests pandas openai`
- Create a script that calls OpenAI's GPT with a test prompt
- **Resources**:
- Real Python "Python in 90 minutes"
- W3Schools Python Tutorial

#### L3 – Azure Fundamentals (13:30–15:00)

- **Objective**: Understand core Azure services relevant to the POC.
- **Topics**:
- Azure AI Search
- Azure Blob Storage
- Azure SQL Database
- Azure App Service and Resource Groups
- **Resources**:
- Microsoft Learn: *"Intro to Azure AI Services"*
- Azure AI 900 prep (videos, summaries)

### POC Build Block

#### P1 – Environment Setup (15:15–18:00)

- **Goal**: Set up local Python project and Azure environment.
- **Tasks**:
- Create Azure Resource Group and services:

```
az group create -n rg-genai-poc -l eastus
az cognitiveservices account create \
  --kind OpenAI --location eastus --name aoai-poc \
  --sku S0 --resource-group rg-genai-poc
az search service create --name aisearch-poc --sku basic --resource-group rg-
genai-poc
az sql server create --name sql-poc-srv --resource-group rg-genai-poc \
  --location eastus --admin-user admin --admin-password <Password>
```

1. Set up Python project:
2. `python -m venv .venv`
3. `pip install fastapi streamlit openai azure-search-documents langchain pyodbc tiktoken`
4. Create `.env` file for storing API keys
5. Create GitHub repo and push initial code
6. Launch Jupyter Notebook to test simple queries to GPT-4o

---

## Day 2 – Data Loading & Indexing

### Learning Blocks

#### L4 – Azure SQL and Blob Storage Access (09:00–10:30)

- **Goal**: Learn how to extract enterprise data from SQL and PDFs
- **Resources**:
- Microsoft: *"Using pyodbc with Azure SQL"*
- Azure Blob SDK Quickstart (Python)

#### L5 – Embeddings, Chunking and Vector Stores (10:45–12:00)

- **Goal**: Understand how to turn raw data into semantically searchable chunks
- **Concepts**:
- Chunking PDF and SQL data (by paragraph/row)
- Embedding using OpenAI's `text-embedding-ada-002`
- Vector storage and retrieval with Azure AI Search

### POC Build Block

#### P2 – ETL and Indexing (13:00–17:30)

- **Tasks**:
- **SQL Extraction**

```
import pyodbc
conn = pyodbc.connect("Driver={ODBC Driver 17 for SQL Server};Server=sql-poc-srv.database.windows.net;Database=bankdb;UID=admin;PWD=...")
cursor = conn.cursor()
cursor.execute("SELECT * FROM vw_CustomerProductFacts")
rows = cursor.fetchall()
```

1. **PDF Ingestion**
2. Use PyMuPDF or Azure Form Recognizer to extract tables and key-value fields
3. **Chunking & Embedding**

```
import openai
chunks = ["Customer has FD and Credit Card...", ...]
response = openai.Embedding.create(model="text-embedding-ada-002", input=chunks)
```

1. **Store in Azure Search**
2. Use `SearchClient.upload_documents()`
3. **Test query**: manually search for "What loans does customer 001 have?"

---

## Day 3 – Backend API and RAG Integration

### Learning Blocks

#### L6 – RAG with Azure AI Search (09:00–10:00)

- **Goal**: Learn how to retrieve context and call LLM to answer questions
- **Resources**: MS Docs – *RAG with Python*, LangChain RAG template

#### L7 – FastAPI Basics (10:15–11:30)

- **Goal**: Create backend API with FastAPI to receive customer queries and return LLM answers

### POC Build Block

#### P3 – RAG Backend API (12:30–18:00)

- **Steps**:
- Create FastAPI service with `POST /ask`
- Search Azure AI index using query & customer ID
- Format context for GPT prompt
- Call OpenAI ChatCompletion API
- Return result to client

```
@app.post("/ask")
async def ask(payload: AskRequest):
    results = search_client.search(payload.query, filter=f"customerId eq
'{payload.customer_id}'")
    context = "\n".join([doc["content"] for doc in results])
    prompt = f"Answer as RM assistant.\nContext:\n{context}\n---\nQ:
{payload.query}\nA:"
    response = openai.ChatCompletion.create(model="gpt-4o-mini",
messages=[{"role":"user", "content": prompt}])
    return {"answer": response.choices[0].message.content}
```

1. Test with Postman or Python `requests`

---

## Day 4 – Front-End & Deployment

### Learning Block

#### L8 – Streamlit Frontend (09:00–10:00)

- **Goal**: Build an intuitive web UI with chat interface
- **Features**:
- Customer ID input
- Initial overview question sent automatically
- Chat history UI
- Backend call to `/ask`

### POC Build Blocks

#### P4 – RM Chat UI (10:15–13:00)

```
import streamlit as st
import requests
if "history" not in st.session_state:
    st.session_state["history"] = []
cust_id = st.text_input("Customer ID")
query = st.text_input("Ask a question")
if st.button("Send"):
    response = requests.post("http://localhost:8000/ask",
json={"customer_id": cust_id, "query": query})
    st.session_state.history.append((query, response.json()["answer"]))
for q, a in st.session_state.history:
    st.chat_message("user").write(q)
    st.chat_message("assistant").write(a)
```

**P5 – Deployment (14:00–16:30)**

- Containerize backend with Docker
- Push to Azure Container Registry
- Create Web App with Azure CLI and deploy container
- Configure custom DNS if needed

**P6 – Demo Recording & Testing (16:45–18:00)**

- Test sample scenarios (e.g., "What products are underperforming?" or "Suggest next-best product")
- Record demo video (Loom, OBS Studio)

---

## Day 5 – Enhancements (Optional)

- Generate cross-sell offers using GPT reasoning:

```
prompt = f"Based on this product list, what products should this customer be
offered next?\nProducts:\n{product_list}"
```

- Add persona selector (e.g., Aggressive, Balanced)
- Export chat to PDF using `reportlab`

---

## Appendix – Budget Summary

| Resource | SKU | Cost/Unit | Est. Use | Total |
|---|---|---|---|---|
| OpenAI GPT-4o | \$0.06 / 1k tokens | 30k tokens | \$1.80 | |
| Embeddings (ada-002) | \$0.0001 / 1k | 3M tokens | \$0.30 | |
| Azure AI Search | S1 | \$200/mo × 0.15 | \$30 | |
| Azure SQL | Basic | \$5/mo × 0.15 | \$0.75 | |
| Azure Storage | 10 GB | \$0.02/GB-mo × 0.15 | \$0.03 | |
| Azure Web App | B1 | \$56/mo × 0.15 | \$8.40 | |
| **Total (est)** | | | | **≈ \$41–47** |

---

## Appendix – Resources

- Azure AI Studio
- Microsoft Learn for Azure AI
- LangChain Docs
- Streamlit Docs

---

### You're Ready to Start

Let me know if you'd like templates for:

- PDF parsing
- FastAPI structure
- Sample customer data
- Deployment YAMLs