Data Engineer - Technical Assignment

Company: Lemici IQ
Time Limit: 48-72 hours
Submission Format: GitHub repository (private with access granted)

Important Instructions

1. **Code Quality Matters**: We value clean, well-structured code with proper error handling
2. **Document Your Thinking**: Include a README explaining your approach, design decisions, and trade-offs
3. **Be Honest**: We expect candidates to use resources (documentation, research papers, LLMs), but we want to see YOUR thinking and reasoning
4. **Commit Regularly**: Make frequent commits to show your development process
5. **Be Ready to Explain**: You'll walk us through your solution in a follow-up technical discussion

---

Assignment Overview

You'll build a **Document Intelligence System** that combines EDA, ML fundamentals, and LLM capabilities. This assignment is designed to test:

- Your data analysis and critical thinking skills
- Understanding of ML fundamentals (not just using libraries)
- Deep knowledge of LLM architecture and RAG systems
- Ability to debug and handle edge cases
- Code quality and system design thinking

Part 1: Exploratory Data Analysis (EDA)

Dataset

We've provided you with a dataset of customer support tickets.

Your Task:

1. Perform comprehensive EDA and create a Jupyter notebook with:
    - Data quality assessment (missing values, duplicates, anomalies)

- Statistical analysis of resolution times across categories and priorities
- Text analysis of customer messages (length distribution, common patterns)
- Correlation between features and customer satisfaction
- At least 3 interesting insights with visualizations

2. **Critical Thinking Questions** (Answer in notebook):
   - You notice that 15% of "urgent" tickets have lower resolution times than "high" priority tickets. What could explain this? How would you investigate?
   - The satisfaction score for the "billing" category is lower despite faster resolution times. Why might this be? What data would you collect to confirm your hypothesis?
   - You find that agent responses are sometimes copy-pasted (exact duplicates). How does this affect your analysis? Should you remove them?

What We're Looking For:

- Beyond basic `.describe()` - we want thoughtful analysis
- Understanding of when to use different statistical tests
- Ability to question data quality and identify real-world issues
- Insights that show domain understanding, not just technical skills

---

Part 2: Machine Learning Fundamentals

Task

Build a classifier to predict ticket priority (low, medium, high, urgent) based on the customer message.

Requirements:

1. Implement at least 3 different algorithms (e.g., Logistic Regression, Random Forest, XGBoost)
2. Proper train-test split with stratification
3. Feature engineering from text (TF-IDF, word counts, message length, sentiment if relevant)
4. Model evaluation with appropriate metrics (accuracy, precision, recall, F1, confusion matrix)
5. Hyperparameter tuning with clear reasoning for parameter choices

Critical Thinking Questions (Answer in code comments or separate doc):

- Why did you choose these specific algorithms? What assumptions does each make?
- Your Random Forest performs better than Logistic Regression. Why might this be? What does this tell you about the data?
- The model predicts "urgent" very poorly (low recall). What are 3 concrete ways you would improve this?
- You notice the model performs worse on tickets with >200 words. Why? How would you fix this?
- If this model goes to production and predicts wrong priority, what's the business impact? How would you monitor for model drift?

What We're Looking For:

- Understanding WHY algorithms work, not just HOW to use them
- Proper evaluation methodology (not just accuracy)
- Ability to debug model performance issues
- Awareness of real-world deployment considerations

Part 3: LLM & RAG System

Task

Build a **Customer Support Assistant** using LLMs and RAG to automatically suggest responses to new tickets.

System Requirements

1. **Document Processing Pipeline:**
    - Ingest historical agent responses as knowledge base
    - Implement intelligent chunking strategy (explain why you chose your approach)
    - Generate embeddings and store in vector database

2. **RAG Implementation:**
    - Given a new customer query, retrieve relevant historical responses
    - Use an LLM to generate a suggested response
    - Implement proper prompt engineering with context management

3. **Technical Requirements:**
    - Use any vector DB (Pinecone, Chroma, FAISS, Weaviate)

- Use at least one open-source LLM (Llama, Mistral) OR commercial API (OpenAI, Anthropic)
- Implement proper error handling and edge case management
- Add basic evaluation metrics for retrieval quality

Critical Deep-Dive Questions (Answer in a separate `RAG_DESIGN.md` file):

Architecture & Design:

1. Explain your chunking strategy. Why did you choose this size/overlap? What happens if chunks are too small or too large?
2. Which embedding model did you use and why? How would you evaluate if it's working well for this domain?
3. You retrieve 5 similar documents - how do you decide which ones to include in the LLM context? What's your re-ranking strategy?
4. How does your system handle queries that have no relevant historical data?

LLM Understanding:

5. Explain how the transformer attention mechanism helps in understanding customer queries. Draw or describe it in your own words.

6. What's the difference between the embedding model and the LLM model? Why do we need both in RAG?

7. Your LLM sometimes generates responses that contradict the retrieved documents. Why does this happen? How would you mitigate it?

8. If you're using GPT-4 via API, what's happening between your request and the response? Explain the inference process.

Prompt Engineering:

9. Share your final prompt. Explain each part of it and why it's there. 1

0. How did you handle the scenario where retrieved context is 10,000 tokens but your context window is 4,096? Show your solution.

11. What techniques did you try for better responses? (Few-shot examples, chain-of-thought, structured outputs?)

Production Considerations:

12. Your RAG system will handle 1000 queries/day. Where are your bottlenecks? How would you optimize?

13. An agent reports the suggested response was completely wrong and upset a customer. How do you debug this? What data do you need?

14. How would you evaluate if your RAG system is actually helpful to support agents? What metrics matter?

What We're Looking For:

- Deep understanding of how RAG works, not just using LangChain blindly
- Awareness of LLM fundamentals (attention, embeddings, context windows)
- Thoughtful prompt engineering with clear reasoning
- Understanding of real-world challenges (hallucination, retrieval quality, latency)
- Ability to debug and improve the system systematically

Part 4: Agent Implementation (Bonus)

Task

Extend your RAG system to be an **autonomous agent** that can:

1. Decide whether to search the knowledge base OR escalate to human
2. If searching, formulate a better search query if first attempt fails
3. Track conversation history for multi-turn interactions

Requirements:

- Implement decision-making logic (not just sequential steps)
- Show the agent's "thinking process" in logs
- Handle failure cases gracefully

Questions:

1. How does your agent decide when to escalate vs. search again?
2. What happens if the agent gets stuck in a loop? How do you prevent it?
3. Explain the difference between an LLM agent and a simple if-else system. Where does the "intelligence" come from?

Part 5: Code Quality & Documentation

Your repository should include:

1. Clean, modular code with proper functions/classes
2. Requirements.txt or environment.yml with all dependencies

3. README.md with:
    - Setup instructions
    - How to run each part
    - Your design decisions and trade-offs
    - What you would improve with more time
4. Unit tests for critical functions (at least 3-4 tests)
5. Clear git commit history showing your development process

Submission Guidelines

What to Submit:

1. **GitHub Repository Link** containing:
    - All code (notebooks + Python scripts)
    - `README.md` with setup and execution instructions
    - `EDA_INSIGHTS.md` with your EDA findings and critical thinking answers
    - `ML_ANALYSIS.md` with ML model analysis and answers
    - `RAG_DESIGN.md` with RAG system design and deep-dive answers
    - Sample `.env.example` if using API keys (don't commit actual keys!)
    - Requirements file for dependencies

2. **Demo Video** (5-10 minutes):
    - Quick walkthrough of your solution
    - Show your RAG system in action with 2-3 example queries
    - Explain one technical challenge you faced and how you solved it

3. **One-Pager Summary**:
    - What you built
    - Key technical decisions
    - What you learned
    - What you would do differently

FAQ

Q: Can I use LLMs (ChatGPT, Claude) to help with coding?
A: Yes, but we want to see YOUR thinking in the design decisions and critical analysis. The code is less important than understanding WHY things work.

Q: Can I use LangChain/LlamaIndex?
A: Yes, but you must understand what's happening under the hood. We'll ask you to explain the internals.

Q: What if I can't finish everything in 48 hours?
A: Focus on quality over quantity. Better to do Part 1-3 well than rush through everything poorly. Document what you would add with more time.

Q: Can I use a different dataset?
A: No, we need everyone to work on the same problem for fair comparison.

Q: Do I need to deploy this?
A: No, but having a simple FastAPI endpoint is a plus. Focus on the core logic first.

Questions?

Contact: people@lemici.com with subject line "Data Engineer Assignment - [Your Name]"

Good luck! We're excited to see your work. 🚀