**Project Scope Document**

**Project Title:**  
Amazon Review - RAG Question Answering & Recommendation System

**Project Overview**

The objective of this project is to develop a fully-functional, reliable, and entirely free-to-use artificial intelligence (AI) application capable of answering user queries accurately and providing personalized product recommendations based solely on real Amazon customer review data.

**Key Functionalities:**

* Question Answering: Providing precise and relevant answers to user questions based on review data.
* Product Recommendations: Suggesting products tailored to user interests inferred from their queries and Amazon reviews.

**Technology Stack**

This project will exclusively utilize open-source tools, libraries, and technologies.

**Programming Language:**

* Python 3.11

**Data Handling & Machine Learning Libraries:**

* PySpark (for large-scale data preprocessing)
* Pandas
* PyTorch
* Hugging Face Transformers
* Sentence-BERT (sentence-transformers)
* LangChain (for implementing Retrieval-Augmented Generation or RAG, combining retrieved review data with generative AI)
* FAISS (Facebook AI Similarity Search - for efficient embedding retrieval)

**Web Interface & API:**

* FastAPI (Backend API)
* Streamlit (Interactive frontend application)

**Experiment Tracking & Monitoring:**

* MLflow (Open-source experiment and model tracking)

**Containerization:**

* Docker
* Docker Compose

*Note: AWS or any paid cloud services are excluded.*

**Dataset**

The project will utilize the Amazon US Reviews Dataset available publicly from Hugging Face (Dataset: polinaeterna/amazon\_us\_reviews).

* Format: Parquet file
* Contains authentic customer reviews from multiple product categories sold on Amazon.

**Project Phases & Deliverables**

**Phase 1: Data Acquisition & Exploratory Data Analysis (EDA)**

* Acquire and verify the Amazon review dataset.
* Perform exploratory analysis using Pandas.
* Document insights, data distributions, and potential issues.

**Deliverables:**

* Verified raw dataset
* Documented exploratory analysis notebook (01\_EDA.ipynb)

**Phase 2: Data Cleaning & Preprocessing (PySpark)**

* Develop and run preprocessing pipeline using PySpark.
  + Data cleaning: handle missing, noisy, or irrelevant content.
  + Text normalization: tokenization, stop-word removal, stemming, and lemmatization.
* Validate data quality.

**Deliverables:**

* Cleaned, preprocessed data
* PySpark preprocessing scripts
* Data validation checks (Pytest/Pandera)

**Phase 3: Embedding Generation & Vector Storage**

* Generate text embeddings using Sentence-BERT.
* Store embeddings in FAISS for quick retrieval.

**Deliverables:**

* Embedding generation script
* FAISS embedding index
* Embedding query functions

**Phase 4: Retrieval-Augmented Generation (RAG) Engine**

* Implement Retrieval-Augmented Generation (RAG) pipeline using LangChain and Hugging Face LLMs (Mistral-7B or FLAN-T5).
  + Retrieve review context via embedding search.
  + Generate accurate answers based on retrieved contexts.

**Deliverables:**

* RAG implementation script (rag\_engine.py)
* Evaluation and benchmarking notebook (06\_Evaluation.ipynb)

**Phase 5: Product Recommendation System**

* Develop content-based recommendation model using cosine similarity.
* Generate transparent recommendation rationales linked to reviews.

**Deliverables:**

* Recommendation system script (recommender.py)
* Evaluation notebook including recommendation quality metrics (NDCG@5)

**Phase 6: API & Streamlit User Interface**

* Develop FastAPI endpoints for:
  + Question Answering (/ask\_review)
  + Product Recommendations (/recommend)
* Create a Streamlit frontend for interactive user engagement.

**Deliverables:**

* FastAPI backend scripts
* Interactive Streamlit application (streamlit\_app/app.py)

**Phase 7: Containerization & Local Deployment**

* Dockerize the entire solution to simplify local setup and deployment.

**Deliverables:**

* Dockerfile
* docker-compose.yml
* Step-by-step setup instructions

**Phase 8: Project Documentation & Final Demo**

* Prepare comprehensive project documentation including:
  + Project overview
  + Detailed setup and usage instructions
  + Architectural diagrams
  + Links to scripts, notebooks, and evaluations

**Deliverables:**

* README.md file
* Complete project repository (GitHub)

**Evaluation & Success Metrics**

* Question Answering Accuracy: Achieve a minimum 20% improvement over traditional keyword-based searches (e.g., BM25).
* Recommendation Quality: Achieve NDCG@5 score of at least 0.65.
* Performance Target: Latency less than 150ms (95th percentile) for local deployments.

**Recommended Timeline**

(Adjustable according to student's pace, not tied to strict sprints):

| **Phase** | **Description** | **Estimated Duration** |
| --- | --- | --- |
| 1 | Data Acquisition & EDA | 5 days |
| 2 | Data Cleaning & PySpark Preprocessing | 7 days |
| 3 | Embeddings & FAISS Indexing | 7 days |
| 4 | RAG Question-Answering Engine | 10 days |
| 5 | Product Recommendation Engine | 7 days |
| 6 | FastAPI & Streamlit Development | 5 days |
| 7 | Dockerization & Local Deployment | 3 days |
| 8 | Documentation & Final Demo | 4 days |
|  | **Total Estimated Time** | ~48 days (~7 weeks) |

**Risks & Mitigation Strategies**

* Embedding Drift: Periodically evaluate embeddings and retrain if accuracy declines.
* Latency Issues: Optimize FAISS vector index using efficient indexing (e.g., HNSW indexing).
* Technical Complexity: Clearly document scripts and modularize components for maintainability and debugging ease.

**Project Directory Structure**

css

CopyEdit

amazon\_review\_intelligence\_suite/

├── README.md

├── requirements.txt

├── data/

│ ├── raw/

│ ├── processed/

│ └── embeddings/

├── notebooks/

│ ├── 01\_EDA.ipynb

│ ├── 02\_Cleaning.ipynb

│ ├── 03\_Embedding\_Benchmark.ipynb

│ ├── 04\_RAG\_Pipeline.ipynb

│ ├── 05\_RecSys\_Baseline.ipynb

│ ├── 06\_Evaluation.ipynb

│ └── 07\_API\_Demo.ipynb

├── src/

│ ├── data\_loader.py

│ ├── text\_preprocess.py (PySpark)

│ ├── embedding.py

│ ├── rag\_engine.py

│ ├── recommender.py

│ └── utils/

│ └── logger.py

├── api/

│ └── main.py

├── streamlit\_app/

│ └── app.py

├── infra/

│ ├── Dockerfile

│ └── docker-compose.yml

└── tests/