Startel telecom

1. Converting unstructured data to structured:

Uploaded PySpark script (startel_csv.py) to S3 and submitted it as an EMR Spark step, which triggered a distributed Spark job on EMR cluster. The job read unstructured invoice PDFs directly from S3 using Spark's binaryFile reader, extracted raw text from each PDF with PyMuPDF, cleaned and normalized that text, and then applied regex-based parsing to extract structured fields (such as customer ID, name, city, plan, billing month, usage charges, GST amount, and bill due (computed as usage + GST)). Spark processed all invoices in parallel across the cluster, consolidated the results into a single DataFrame, and finally wrote the structured output back to S3 as a single CSV (inside a Spark output folder). This converted raw, unstructured invoice files into an analytics-ready structured dataset.

2. Data Analysis:

In the next stage of the project, the structured invoice CSV produced by the EMR Spark pipeline is loaded directly from S3 into a local Python analytics layer using pandas. This data is cleaned and enriched by splitting billing month into month and year, ordering time correctly, ranking plans, and deriving customer behavior features such as upgrades, downgrades, churn status, revenue metrics, and city-level summaries.

3. Model building:

a) **<u>These processed datasets power two parallel paths:</u>**
a deterministic analytics engine that answers numerical and trend-based questions (revenue, churn, plan movement, top customers, cities, etc.) using pandas logic, and a Retrieval-Augmented Generation (RAG) layer where curated analytical summaries are embedded using OpenAI embeddings, stored persistently in a Chroma vector database, and retrieved at query time to answer descriptive or explanatory questions.

b) **<u>A hybrid assistant orchestrates this flow by first detecting user intent:</u>**
analytical questions are answered directly from computed data, while open-ended questions are answered by combining retrieved context from the vector database with an LLM response, resulting in a complete analytics + AI question-answering system over the Startel invoice data.

4. Final part

- In the final step, a Streamlit-based web interface is built to expose the entire analytics and AI pipeline to end users in an interactive and intuitive way. The Streamlit app acts as the presentation layer, where a user types a natural-language question about Startel's telecom data. The app sends this question to the hybrid assistant, which internally loads the structured invoice data, detects whether the question is analytical or descriptive, and then routes it either to the rule-based.
- analytics engine (for exact numerical answers like revenue, churn, or plan changes) or to the
- RAG pipeline (for explanatory answers using embedded context from city and customer summaries).
- The final answer is displayed on the UI, and for transparency and trust, the app also shows the retrieved contextual passages used by the RAG system. This completes the end-to-end flow from raw invoices to a user-facing AI analytics assistant.
- The entire system is exposed through a production-ready Flask REST API. The API receives user questions, invokes the hybrid assistant, and returns responses as JSON. CORS is enabled to support browser-based clients.A lightweight frontend built using HTML, CSS, and JavaScript allows users to interact with the system. The frontend communicates with the backend via REST APIs and serves purely as a presentation layer.