Title: Crowdfunding Insights with Open-Source RAG
Author: Roshan Salunke
Date: 20-Nov-2025

Github Repo: https://github.com/roshan9900/ImpactGuruCrowdfundingRAG

---

1. Objective

Build a **Retrieval-Augmented Generation (RAG)** system to analyze open-source crowdfunding data and extract actionable insights. The system should include:

- Data collection and preparation
- Summarized documents for RAG
- Open-source embedding and LLM integration
- Interactive retrieval and insight generation

---

2. Data Preparation

2.1 Data Sourcing

- **Dataset:** Scraped publicly available crowdfunding campaigns from Ketto and ImpactGuru.
- **Features included:**
  - Campaign title, description, category
  - Fundraising goal, raised amount
  - Start and end dates
  - Donor count, updates, comments
- **File:** `ketto_campaigns.csv`

Justification: Ketto/ImpactGuru campaigns are open and provide rich numeric and textual data.

2.2 Cleaning & Enrichment

Steps performed:

1. **Loading:**

```
import pandas as pd

df = pd.read_csv("ketto_campaigns.csv")
```

2. **Handling missing values:**

```
df.fillna({"description": "No description provided", "raised_amount": 0},
inplace=True)
```

3. **Text standardization:**

- Lowercased all text
- Removed special characters
- Normalized whitespace

```
df["description"] = df["description"].str.lower().str.replace(r"\s+", " ",
regex=True)
```

4. **Numeric handling:**

- Converted `raised_amount` and `goal_amount` to numeric types
- Calculated **% raised** for quick insight

```
df["raised_percent"] = (df["raised_amount"] / df["goal_amount"] *
100).round(2)
```

5. **Output:** `ketto_campaigns_enriched.csv`

2.3 Creating Documents for RAG

- **Objective:** Provide ≥10 summarized documents suitable for retrieval-based QA.
- **Method:** Chunked campaigns by category and summarized descriptions.

```
from summariser import generate_text_summary, generate_numeric_summary

generate_numeric_summary(df, "numeric_summary.csv")
generate_text_summary(df, "text_summary.txt")
```

- **Documents stored in `docs_json/`** in JSON format with keys:
  - `title`, `category`, `numeric_summary`, `text_summary`, `goal_amount`, `raised_amount`

---

3. RAG System Implementation

3.1 Embedding Model

- **Model chosen:** `sentence-transformers/all-MiniLM-L6-v2`
- **Justification:**
  - Open-source and lightweight
  - Produces high-quality embeddings for semantic similarity
  - Efficient for small/medium datasets (≤50K documents)

Embedding code:

```
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS

embedding_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vectorstore = FAISS.from_documents(docs, embedding=embedding_model)
```

3.2 Retrieval

- **Top-N retriever:** Returns the top 5 most relevant documents based on cosine similarity.

```
query = "Medical campaigns that raised >50% of goal"
retrieved_docs = vectorstore.similarity_search(query, k=5)
```

3.3 LLM Integration

- **Models chosen:**
  - `LLaMA-Pro 8B Instruct`
  - `Mistral 7B Instruct`
- **Justification:**
  - Open-source, no cloud API needed
  - Instruct-tuned for Q&A
  - Run locally for privacy and cost-efficiency
- **Cons / Observations:**
  - LLaMA and Mistral were **slow for inference** on larger document sets.
  - To optimize, we integrated **Groq API** to accelerate generation and handle real-time queries efficiently.
- **Example inference:**

```
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(model_name="llama-pro-8b-instruct", temperature=0)
response = llm("Summarize the top retrieved campaigns")
```

3.4 Insight Function

Function: `get_insight_from_rag(query: str)`

```
def get_insight_from_rag(query: str):
    # Retrieve top documents
    docs = vectorstore.similarity_search(query, k=5)

    # Prepare context
    context = "\n\n".join([doc.page_content for doc in docs])

    # Prompt LLM
    prompt = f"Using the following campaign data, provide
insights:\n{context}\nQuestion: {query}"

    # Generate answer using Groq-accelerated LLM
    answer = llm(prompt)
```

```
    return answer
```

- **Usage:**

```
query = "Which campaigns raised the most funds in 2025?"
print(get_insight_from_rag(query))
```

3.5 Interactive Interface

- **Streamlit integration:** `streamlit.py`
- Provides a **web-based GUI** to query campaigns, select top-N retrieval, and visualize summaries.
- Makes the system **accessible to non-technical users**.

```
streamlit run streamlit.py
```

---

4. Deliverables

- **GitHub repo** containing:
  - Python scripts (`scrapper.py`, `summariser.py`, `rag.py`, `local_rag.py`, `raghybrid.py`, `infer.py`, `streamlit.py`)
  - Dataset (`ketto_campaigns.csv`, `ketto_campaigns_enriched.csv`)
  - Summaries (`numeric_summary.csv`, `text_summary.txt`, `overall_summary.txt`)
  - FAISS vectorstore (`faiss_index/`)
  - Local LLM models (`models/`)
  - `requirements.txt`
- **RAG system:** Fully functional and able to answer queries with insights from crowdfunding campaigns.
- **Optimizations:** Groq API acceleration and Streamlit for interactive queries.

---

5. Highlights

- Fully **local RAG system** with open-source LLMs.
- Handles **structured numeric data and unstructured text** seamlessly.
- Integrated **Groq for accelerated inference** to overcome LLaMA/Mistral slowness.
- **Streamlit interface** for interactive, user-friendly queries.
- Supports **hybrid retrieval**, combining semantic similarity with metadata filtering.