```
!pip install -q faiss-cpu sentence-transformers transformers accelerate gradio==4.29.0 opendata
```

```
import os
import json
import numpy as np
import pandas as pd
import faiss
import torch
from typing import List, Tuple, Dict, Any

from sentence_transformers import SentenceTransformer
from transformers import pipeline, AutoTokenizer, AutoModelForSeq2SeqLM
import gradio as gr

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", DEVICE)
```

```
Using device: cpu
```

```
import zipfile
import pandas as pd
import os

# Path to the uploaded zip file
zip_path = "/content/archive (1).zip"
extract_dir = "/content/dataset"

# Extract the ZIP
os.makedirs(extract_dir, exist_ok=True)
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

# Find and load the "Training Dataset.csv"
csv_path = None
for root, dirs, files in os.walk(extract_dir):
    for file in files:
        if file.lower().endswith(".csv") and "training" in file.lower():
            csv_path = os.path.join(root, file)
            break

if csv_path:
    df = pd.read_csv(csv_path)
    print(f" Loaded file: {csv_path}")
    print("Shape:", df.shape)
    df.head()
else:
    raise FileNotFoundError(" Could not find 'Training Dataset.csv' in the ZIP.")
```

```
 Loaded file: /content/dataset/Training Dataset.csv
Shape: (614, 13)
```

```
assert df is not None, "No DataFrame loaded. Use either the upload cell or Kaggle cell."
df_original = df.copy()
df.fillna("Unknown", inplace=True)

print("Columns:", list(df.columns))
```

```
print("Columns:", list(df.columns))
df.head()
```

```
Columns: ['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome', 'Coapp
/tmp/ipython-input-9-4003636802.py:3: FutureWarning: Setting an item of incompatible dtype is deprecated and wi
  df.fillna("Unknown", inplace=True)
```

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmou |
|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | Unknc |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 12 |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 6 |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 12 |
| 4 | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 14 |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
def row_to_doc(row: pd.Series) -> str:
    return " | ".join([f"{col}: {row[col]}" for col in df.columns])

docs = df.apply(row_to_doc, axis=1).tolist()
print(f"Prepared {len(docs)} row-documents.")
```

```
Prepared 614 row-documents.
```

```
EMBED_MODEL_NAME = "sentence-transformers/all-MiniLM-L6-v2"
INDEX_PATH = "/content/faiss_index.bin"
EMBEDS_PATH = "/content/doc_embeds.npy"

embedder = SentenceTransformer(EMBED_MODEL_NAME, device=DEVICE)

if os.path.exists(INDEX_PATH) and os.path.exists(EMBEDS_PATH):
    print("Loading cached FAISS index & embeddings…")
    doc_embeds = np.load(EMBEDS_PATH)
    index = faiss.read_index(INDEX_PATH)
else:
    print("Encoding documents…")
    doc_embeds = embedder.encode(docs, convert_to_numpy=True, batch_size=256, show_progress_bar

    print("Building FAISS index…")
    d = doc_embeds.shape[1]
    index = faiss.IndexFlatIP(d)  # cosine if normalized
    index.add(doc_embeds)

    faiss.write_index(index, INDEX_PATH)
    np.save(EMBEDS_PATH, doc_embeds)
    print("Saved FAISS + embeddings to disk.")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%  349/349 [00:00<00:00, 17.5kB/s]
config_sentence_transformers.json: 100%  116/116 [00:00<00:00, 8.09kB/s]
README.md: 10.5k/? [00:00<00:00, 540kB/s]
sentence_bert_config.json: 100%  53.0/53.0 [00:00<00:00, 3.16kB/s]
config.json: 100%  612/612 [00:00<00:00, 39.8kB/s]
model.safetensors: 100%  90.9M/90.9M [00:01<00:00, 77.9MB/s]
tokenizer_config.json: 100%  350/350 [00:00<00:00, 27.7kB/s]
vocab.txt: 232k/? [00:00<00:00, 3.29MB/s]
tokenizer.json: 466k/? [00:00<00:00, 7.65MB/s]
special_tokens_map.json: 100%  112/112 [00:00<00:00, 2.44kB/s]
config.json: 100%  190/190 [00:00<00:00, 6.29kB/s]
Encoding documents…
Batches: 100%  3/3 [00:31<00:00, 9.33s/it]
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1750: FutureWarning: `encoder_attention_mask
  return forward_call(*args, **kwargs)
Building FAISS index…
Saved FAISS + embeddings to disk.
```

```
GEN_MODEL_NAME = "google/flan-t5-base"
generator = pipeline(
    task="text2text-generation",
    model=GEN_MODEL_NAME,
    device=0 if DEVICE == "cuda" else -1,
    max_new_tokens=256,
    do_sample=True,
    temperature=0.3,
    top_p=0.95
)
```

```
config.json: 1.40k/? [00:00<00:00, 77.6kB/s]
model.safetensors: 100%  990M/990M [00:35<00:00, 19.9MB/s]
generation_config.json: 100%  147/147 [00:00<00:00, 6.14kB/s]
tokenizer_config.json: 2.54k/? [00:00<00:00, 75.3kB/s]
spiece.model: 100%  792k/792k [00:00<00:00, 1.96MB/s]
tokenizer.json: 2.42M/? [00:00<00:00, 27.8MB/s]
special_tokens_map.json: 2.20k/? [00:00<00:00, 49.4kB/s]
Device set to use cpu
```

```
def retrieve(query: str, k: int = 5) -> Tuple[List[int], List[str]]:
    q_emb = embedder.encode([query], convert_to_numpy=True, normalize_embeddings=True)
    scores, idxs = index.search(q_emb, k)
    idxs = idxs[0].tolist()
    ctxs = [docs[i] for i in idxs]
    return idxs, ctxs

SYSTEM_PROMPT = """You are a data-savvy assistant answering questions strictly using the provid
```

```
If the question asks for statistics or aggregations you cannot compute exactly, say so and expl
Always be concise, but include useful details.
When helpful, mention the row indices you used as sources.
If the user asks something outside the dataset, say you don't know because it's outside the pro
"""

def build_prompt(question: str, contexts: List[str]) -> str:
    joined = "\n\n".join([f"[Context {i+1}]\n{c}" for i, c in enumerate(contexts)])
    return f"""{SYSTEM_PROMPT}

Context rows:
{joined}

Question: {question}
Answer:"""

def rag_answer(question: str, k: int = 5) -> Dict[str, Any]:
    idxs, ctxs = retrieve(question, k=k)
    prompt = build_prompt(question, ctxs)
    out = generator(prompt)[0]["generated_text"].strip()
    return {"answer": out, "indices": idxs, "contexts": ctxs}



with gr.Blocks(theme=gr.themes.Soft()) as demo:
    gr.Markdown("# Loan Dataset RAG Chatbot")
    gr.Markdown("Ask anything about the dataset. The bot retrieves the most relevant rows and an
    gr.Markdown("> *What are the common traits of rejected loans?*\n> *How many rows are there?*

    chatbot = gr.Chatbot(height=400)
    msg = gr.Textbox(placeholder="Ask a question about the dataset…")
    k_slider = gr.Slider(1, 15, value=5, step=1, label="Top-k rows to retrieve")
    clear = gr.ClearButton([chatbot, msg])

    def respond(user_message, chat_history, k):

        structured = try_structured_tools(user_message)
        if structured is not None:
            answer = structured
            idxs = []
            ctxs = []
        else:
            result = rag_answer(user_message, k=int(k))
            answer = result["answer"]
            idxs = result["indices"]
            ctxs = result["contexts"]

        sources = ""
        if len(idxs) > 0:
            sources = "\n\n**Sources (row indices):** " + ", ".join(map(str, idxs))

        chat_history = chat_history + [(user_message, answer + sources)]
        return "", chat_history

    msg.submit(respond, [msg, chatbot, k_slider], [msg, chatbot])

demo.launch(debug=False, share=True)
```

```
IMPORTANT: You are using gradio version 4.29.0, however version 4.44.1 is available, please upgrade.
--------
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Running on public URL: https://b63782af753238d3f1.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from Term
```

# Loan Dataset RAG Chatbot

Ask anything about the dataset. The bot retrieves the most relevant rows and answers with context. Try questions like:

> *What are the common traits of rejected loans? How many rows are there? Distribution of ApplicantIncome? Show rows where Credit_History is 0 and Loan_Status is N*

Chatbot