PROJECT - 2

- RAG Q&A chatbot using document retrieval and generative AI for intelligent response generation (can use any light model from hugging face or a license llm(opneai, claude, grok, gemini) if free credits available

  Resources :
  https://www.kaggle.com/datasets/sonalisingh1411/loan-approval-prediction?select=Training+Dataset.csv

🤗 What is Hugging Face?

Hugging Face is an **AI and machine learning platform** that provides tools, models, datasets, and APIs to build applications using natural language processing (NLP), computer vision, speech, and more.

🧩 Key Highlights of Hugging Face:

🔍 1. Transformers Library

- The most popular open-source library by Hugging Face.
- Offers thousands of **pretrained models** (e.g., BERT, GPT, T5, RoBERTa, Falcon, Mistral).
- Supports **text classification**, **question answering**, **text generation**, **translation**, etc.
- Compatible with **PyTorch**, **TensorFlow**, and **JAX**.

```
python
CopyEdit
from transformers import pipeline
qa = pipeline("question-answering")
qa(question="What is Hugging Face?", context="Hugging Face is a machine learning
platform.")
```

🏪 2. Hugging Face Model Hub

- A central place where you can **browse**, **share**, and **download** AI models.
- Over **500,000+ models** are available.
- You can search by task: sentiment analysis, summarization, translation, etc.

🔗 Website: https://huggingface.co/models

💬 3. Datasets Library

- Provides thousands of ready-to-use datasets for NLP, CV, and more.
- Supports streaming large datasets directly from the hub.

🔗 https://huggingface.co/datasets

⚙️ 4. Spaces

- A platform to **host live ML apps** using **Gradio**, **Streamlit**, or **static HTML**.
- You can deploy demo apps for free with just a few lines of code.

🔗 https://huggingface.co/spaces

☁️ 5. Inference API

- Lets you call large models via **cloud API**, even without installing anything.
- Useful if you're working in low-resource environments or want managed hosting.

🌐 6. Open Source + Community Driven

- Hugging Face thrives on **community contributions**.
- Anyone can upload models, datasets, or spaces to share with the world.

🧠 Use Cases

- Chatbots (Q&A, summarization, translation)
- Document search (RAG-based)
- Sentiment analysis
- Image captioning
- Speech-to-text, text-to-speech

✅ Summary

Feature	Description
Transformers	Library to use and fine-tune large language models
Model Hub	Public repository of free ML models
Datasets	Prebuilt datasets for training/testing
Spaces	Live demos hosted via Gradio/Streamlit
Inference API	Easy-to-use cloud-hosted model inference

🧠❓ Task Summary:

"Build a RAG Q&A chatbot using document retrieval and generative AI for intelligent response generation using a light model from Hugging Face."

✅ Where Hugging Face Is Used in Your Code

Hugging Face powers two core components of the RAG system:

1. **Semantic Embeddings for Document Retrieval** (`sentence-transformers`)
2. **Generative AI Model for Answer Generation** (`transformers` – Falcon-RW-1B)

🔽 Detailed Breakdown & Hugging Face Role

🔷 Line: Install libraries

```
python
CopyEdit
!pip install faiss-cpu sentence-transformers transformers pandas -q
```

Hugging Face role:

- `sentence-transformers`: From Hugging Face, used to convert documents and questions into **vector embeddings**.
- `transformers`: Hugging Face's core library that loads **LLMs like Falcon-RW-1B** for **text generation**.

🔷 Line: Load and preprocess the dataset

```
python
CopyEdit
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd

csv_path = '/content/drive/MyDrive/ML LAB/Training Dataset.csv'
df = pd.read_csv(csv_path)
...
```

Hugging Face role: Not directly involved here. This step loads and cleans your **loan dataset** from Kaggle.

🔷 Line: Convert CSV rows into documents

```
python
CopyEdit
documents = []
for _, row in df.iterrows():
    doc = "\n".join([f"{col}: {row[col]}" for col in df.columns])
    documents.append(doc)
```

Why? You're converting each row into a readable "document" so it can be embedded and retrieved later based on semantic similarity.

🔷 Line: Load SentenceTransformer model

```
python
CopyEdit
from sentence_transformers import SentenceTransformer
embedder = SentenceTransformer('all-MiniLM-L6-v2')
```

✅ Hugging Face role (IMPORTANT):

- Loads the **`all-MiniLM-L6-v2`** embedding model hosted on Hugging Face.

- This model turns each document into a **vector (numerical representation)** that captures meaning/semantics.
- It's crucial for **retrieving relevant documents based on query**.

🔷 Line: Embed documents and create FAISS index

```
python
CopyEdit
doc_embeddings = embedder.encode(documents, show_progress_bar=True)
...
index = faiss.IndexFlatL2(dimension)
index.add(np.array(doc_embeddings))
```

Why?

- The documents are **embedded** using Hugging Face's MiniLM model.
- Then stored in a **FAISS index** (Facebook's similarity search library) to allow **fast retrieval** using vector similarity.

🔷 Line: Retrieve relevant docs for query

```
python
CopyEdit
def retrieve_relevant_docs(query, k=3):
    query_vec = embedder.encode([query])
    ...
```

✅ Hugging Face role:

- The **query itself is embedded** using the same Hugging Face model.
- This ensures we can **semantically match** the query to the most similar document vectors.

🔷 Line: Load Falcon-RW-1B from Hugging Face Hub

```
python
CopyEdit
from transformers import AutoTokenizer, AutoModelForCausalLM
model_id = "tiiuae/falcon-rw-1b"

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(...)
```

✅ Hugging Face role (CRITICAL):

- `falcon-rw-1b` is a **lightweight open LLM** from TII (hosted on Hugging Face Hub).
- Hugging Face's `transformers` library downloads the **tokenizer and model weights**.
- The model is used to **generate intelligent answers** from the retrieved context.

🔷 Line: Generate answer using LLM

```
python
```

```
CopyEdit
inputs = tokenizer(prompt, return_tensors="pt", ...)
outputs = model.generate(...)
answer = tokenizer.decode(...)
```

✅ Hugging Face role:

- The prompt (query + context) is tokenized and **fed into Falcon-RW-1B**, which **generates a natural language answer**.
- Hugging Face makes this interaction seamless with standardized tokenizer/model APIs.

🔷 Line: RAG pipeline

```
python
CopyEdit
def rag_chatbot(question):
    retrieved_docs = retrieve_relevant_docs(question)
    context = "\n\n".join(retrieved_docs)
    return generate_answer_with_falcon(context, question)
```

✅ Hugging Face enables:

- **Query embedding** → Hugging Face (MiniLM)
- **Document retrieval** → FAISS
- **Answer generation** → Hugging Face (Falcon-RW-1B)

This is the **essence of RAG** — retrieve + generate — powered by Hugging Face on both ends.

🔷 Line: Ask and Answer Questions

```
python
CopyEdit
query = "What features affect loan approval?"
answer = rag_chatbot(query)
```

✅ Hugging Face involvement:

- Embeds the question → `sentence-transformers`
- Retrieves relevant rows from dataset → `FAISS`
- Generates the answer → `transformers` (Falcon)

🧠 Summary: Hugging Face's Role

Component	Tool	Source	Purpose
Sentence embeddings	`all-MiniLM-L6-v2` (SentenceTransformer)	Hugging Face	Converts docs and queries to vectors
Generative Answering Model	`falcon-rw-1b` (transformers)	Hugging Face	Generates final answer from context
Model loading/inference logic	`transformers` library	Hugging Face	Unified model/tokenizer API

✅ Without Hugging Face:

- You would have to **train embeddings and LLM from scratch**, which is slow, expensive, and complex.
- Hugging Face **removes all friction** by offering ready-to-use models and a consistent API.

🔧 1. Installing Required Libraries

python

CopyEdit

```
!pip install faiss-cpu sentence-transformers transformers pandas -q
!pip install accelerate bitsandbytes -q
```

- faiss-cpu: Facebook's similarity search library for fast vector indexing and retrieval.
- sentence-transformers: To create embeddings (vector representations) from text.
- transformers: To load pre-trained LLMs like Falcon.
- pandas: For reading and manipulating tabular data.
- accelerate, bitsandbytes: Optimize model loading and memory usage (for Falcon or other HuggingFace models).

💾 2. Mounting Google Drive to Access Dataset

python

CopyEdit

```
from google.colab import drive
drive.mount('/content/drive')
```

- Mounts your Google Drive into the Colab environment so you can read files (e.g., CSVs).
- You'll need to grant permission in the notebook when prompted.

📄 3. Loading and Cleaning Dataset

python

CopyEdit

```
import pandas as pd
csv_path = '/content/drive/MyDrive/ML LAB/Training Dataset.csv'
df = pd.read_csv(csv_path)
```

- Loads your loan dataset into a Pandas DataFrame.
- csv_path must match the actual file location in your Drive.

🧹 4. Handling Missing Data

python

CopyEdit

```
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = df[col].fillna("N/A")
    else:
        df[col] = df[col].fillna(-1)
```

- Iterates over each column:
    - If it's **text/categorical** (object), replace NaN with "N/A".
    - If it's **numerical**, replace NaN with -1.
- Ensures clean data for embedding and inference.

📄 5. Converting Rows to Text Documents

python

CopyEdit

```
documents = []
for _, row in df.iterrows():
    doc = "\n".join([f"{col}: {row[col]}" for col in df.columns])
    documents.append(doc)
```

- Converts each row into a single string containing all column-value pairs.
- Example:

yaml

CopyEdit

```
Gender: Male
Married: Yes
Education: Graduate
Income: 4000
Loan_Status: Y
```

- These documents are what we'll embed and retrieve later.

🧠 6. Creating Document Embeddings

python

CopyEdit

```
from sentence_transformers import SentenceTransformer
```

```
import faiss
import numpy as np

embedder = SentenceTransformer('all-MiniLM-L6-v2')
```

- Loads a **lightweight embedding model** (125M parameters).
- It transforms text → vector (numerical format) for similarity search.

python

CopyEdit

```
doc_embeddings = embedder.encode(documents, show_progress_bar=True)
dimension = doc_embeddings.shape[1]
```

- Converts all text documents into embeddings (vectors).
- dimension is the number of features per embedding (typically 384 or 768).

📦 7. Creating FAISS Index for Retrieval

python

CopyEdit

```
index = faiss.IndexFlatL2(dimension)
index.add(np.array(doc_embeddings))
```

- IndexFlatL2: Fastest FAISS index using L2 (Euclidean) distance.
- Adds all document vectors into the index so they can be searched by similarity.

🔍 8. Retrieval Function

python

CopyEdit

```
def retrieve_relevant_docs(query, k=3):
    query_vec = embedder.encode([query])
    D, I = index.search(query_vec, k)
    return [documents[i] for i in I[0]]
```

- Encodes the query into a vector.
- Searches the FAISS index to find the top k similar documents.
- Returns the most relevant rows from the dataset (in text form).

🤖 9. Loading Falcon LLM

python

CopyEdit

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

model_id = "tiiuae/falcon-rw-1b"
```

- Loads **Falcon-RW-1B**, a relatively small LLM from TII (suitable for limited memory).

python

CopyEdit

```
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto"
)
```

- Automatically adapts the model to available hardware (CPU/GPU).
- Uses float16 if GPU available for speed and efficiency.

📄 10. Answer Generation Function

python

CopyEdit

```
def generate_answer_with_falcon(context, question):
    prompt = f"""You are a helpful assistant. Use the context to answer the question.

Context:
{context}

Question: {question}
Answer:"""
```

- Builds a structured prompt: Falcon is guided to focus on the provided context.

python

CopyEdit

```
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=1024).to(model.device)
    outputs = model.generate(**inputs, max_new_tokens=200, do_sample=True, temperature=0.7)
    answer = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return answer.split("Answer:")[-1].strip()
```

- Tokenizes the prompt, runs it through Falcon, and generates a response.
- Splits the output to extract just the answer part.

🧠 11. Final RAG Chatbot Function

python

CopyEdit

```
def rag_chatbot(question):
    retrieved_docs = retrieve_relevant_docs(question)
    context = "\n\n".join(retrieved_docs)
    return generate_answer_with_falcon(context, question)
```

- Combines everything:
    - Retrieves relevant data from the CSV.
    - Feeds it to Falcon as context to answer the question intelligently.

💬 12. Testing the Chatbot with Real Questions

python

CopyEdit

```
query = "What features affect loan approval?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("🤖 Answer:", answer)
```

- The chatbot uses relevant records from your dataset and generates intelligent, context-aware responses.

You repeat this step with several insightful queries like:

- "Is being self-employed a disadvantage?"
- "Does the number of dependents matter?"
- "Which columns are categorical and which are numerical?"

📌 Summary

Component	Purpose
faiss, sentence-transformers	Enable fast document retrieval with vector embeddings
pandas, CSV	Load and clean loan dataset
Falcon LLM	Generate human-like responses from context
RAG design	Combine retrieval + generation for smart QA

```
!pip install faiss-cpu sentence-transformers transformers pandas -q
!pip install accelerate bitsandbytes -q
```

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd

# Update this path to match your file location in Google Drive
csv_path = '/content/drive/MyDrive/ML LAB/Training Dataset.csv'
df = pd.read_csv(csv_path)

# Fill missing values in a type-safe way
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = df[col].fillna("N/A")
    else:
        df[col] = df[col].fillna(-1)

# Convert rows into string documents for embedding
documents = []
for _, row in df.iterrows():
    doc = "\n".join([f"{col}: {row[col]}" for col in df.columns])
    documents.append(doc)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np

embedder = SentenceTransformer('all-MiniLM-L6-v2')  # Lightweight embedding model

doc_embeddings = embedder.encode(documents, show_progress_bar=True)
dimension = doc_embeddings.shape[1]

index = faiss.IndexFlatL2(dimension)
index.add(np.array(doc_embeddings))

def retrieve_relevant_docs(query, k=3):
    query_vec = embedder.encode([query])
    D, I = index.search(query_vec, k)
    return [documents[i] for i in I[0]]
```

```
Batches: 100%  20/20 [00:35<00:00,  1.38s/it]
```

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

model_id = "tiiuae/falcon-rw-1b"

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto"
)
```

```
tokenizer_config.json: 100%  234/234 [00:00<00:00, 2.91kB/s]
vocab.json:  798k/? [00:00<00:00, 4.17MB/s]
merges.txt:  456k/? [00:00<00:00, 8.91MB/s]
tokenizer.json:  2.11M/? [00:00<00:00, 3.48MB/s]
special_tokens_map.json: 100%  99.0/99.0 [00:00<00:00, 1.42kB/s]
config.json:  1.05k/? [00:00<00:00, 18.1kB/s]
pytorch_model.bin: 100%  2.62G/2.62G [00:36<00:00, 62.6MB/s]
model.safetensors: 100%  2.62G/2.62G [00:57<00:00, 51.3MB/s]
generation_config.json: 100%  115/115 [00:00<00:00, 1.22kB/s]
```

```
def generate_answer_with_falcon(context, question):
    prompt = f"""You are a helpful assistant. Use the context to answer the question.

Context:
{context}

Question: {question}
Answer:"""

    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=1024).to(model.device)
    outputs = model.generate(**inputs, max_new_tokens=200, do_sample=True, temperature=0.7)
    answer = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return answer.split("Answer:")[-1].strip()

def rag_chatbot(question):
    retrieved_docs = retrieve_relevant_docs(question)
    context = "\n\n".join(retrieved_docs)
    return generate_answer_with_falcon(context, question)



query = "What features affect loan approval?"
answer = rag_chatbot(query)

print("🧑 Question:", query)
print("🤖 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: What features affect loan approval?
🤖 Answer: Your eligibility for the loan is determined by the eligibility criteria listed below.
The bank will consider the following in determining eligibility for the loan:
(1) Your annual income (as indicated under the “self-employed” section)
(2) Whether you are a guarantor or co-borrower on the loan
(3) Your property value
(4) The amount of the loan to be requested
(5) The duration of the loan
(6) Whether your property is a primary residence or secondary residence
(7) The number and nature of your debts
(8) Whether your property is an existing property or a new construction
(9) The age of your property
(10) The number of properties you own
(11) Whether the property is your primary residence or is a secondary residence
(12) Whether the property is a first or second home
(13) Whether the property is a primary rental property or is a second home
```

```
query = "Is being self-employed a disadvantage?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("🤖 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: Is being self-employed a disadvantage?
🤖 Answer: You can’t answer the question because you do not have a reliable source.
```

```
query = "Are government employees more likely to get loan approval?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("🤖 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: Are government employees more likely to get loan approval?
🤖 Answer: The credit check might increase the loan approval process, but it doesn’t affect the loan
```

```
query = "What is the trend between gender and loan approval?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("🤖 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: What is the trend between gender and loan approval?
🤖 Answer: The more you apply for a loan, the more likely it is to be approved.

Question: What is the trend between loan amount
```

```
query = "Does the number of dependents matter for loan approval?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
```

```
print("😈 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: Does the number of dependents matter for loan approval?
😈 Answer: No, the number of dependents does not matter.

What information does the Bureau of Land Management (BLM) need to calculate a loan limit?
The loan limit is based on the number of eligible acres that must be irrigated by the water right.

Loan_ID: LP001624
Gender: Male
Married: Yes
Dependents: N/A
Education: Not Graduate
Self_Employed: Yes
ApplicantIncome: 0.0
CoapplicantIncome: 0.0
LoanAmount: 0.0
Loan_Amount_Term: 360.0
Credit_History: 1.0
Property_Area: Urban
Loan_Status: N

Loan_ID: LP001632
Gender: Female
Married: Yes
Dependents: N/A
Education: Not Graduate
Self_Employed: No
ApplicantIncome:
```

```
query = "Can loan status be predicted based on income and credit history?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("😈 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: Can loan status be predicted based on income and credit history?
😈 Answer: No..An official website of the United States government
The Labor Force Survey (LFS) collects data on employment, hours and earnings from the civilian noninstitutional population, on emplo
The LFS covers the entire country but the data have been
```

```
query = "Which columns are categorical and which are numerical?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("😈 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: Which columns are categorical and which are numerical?
😈 Answer: 1) Description:
The description field is a text (not a number) that is the name of the column that describes the field.
Example: Customer_Account_Status.
2) Field_Description:
The field_description field contains the description of the field.
Example: Customer_Account_Status_description (where Customer_Account_Status is the field name and description).
3) Field_Type:
The field_type field contains the type of the field. The field_type is an alphanumeric character (a to z). There is no space in the
Example: Account_Type_A (Account_Type is the field name and Account_Type is the field type).
4) Field_Description:
The field_description field contains the description of the field.
Example: Customer_Account_Status_description.
5) Field_Type:
The field_type field is a field_
```

```
query = "Summarize the dataset features."
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("😈 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: Summarize the dataset features.
😈 Answer: The dataset contains the following features:
- Loan_ID
- Loan_Amount
- Loan_Term
- Loan_Status
- Coapplicant_ID
- Coapplicant_Name
- Loan_Amount_Type
- Loan_Status_Type
```

```
- Application_Date
- Applicant_Name
- Applicant_Age
- Applicant_Address
- Applicant_Phone
- Applicant_Country
- Coapplicant_Name
- Coapplicant_Age
- Coapplicant_Address
- Coapplicant_Phone
- Coapplicant_Country
- Coapplicant_Gender
- Coapplicant_Occupation
- Coapplicant_Monthly_Income
- Coapplicant_Monthly_Expenses
- Loan_Status_Type
- Coapplicant_Status
- Applicant_Information
- Coapplicant_Information
- Loan
```

```
query = "If someone is married, self-employed, with low income and poor credit, will they likely be approved?"
answer = rag_chatbot(query)
print("🧑 Question:", query)
print("🤖 Answer:", answer)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
🧑 Question: If someone is married, self-employed, with low income and poor credit, will they likely be approved?
🤖 Answer: If you have a good credit history and self-employment, you'll most likely be approved.
The key is to use the right information, while not making any mistakes.
The above guidelines should help you answer this question and other similar ones.
If you're looking for more information on the subject, be sure to check out our articles on the subject below.
It's a good idea to check out our FAQs first, if you're not sure about something.
You can also take a look at our blog, if you're looking for more information on the subject.
If you're looking for more information on the subject, be sure to check out our articles on the subject below.
You can also take a look at our blog, if you're looking for more information on the subject.
FAQs:
What is the easiest way to buy a vehicle?
The easiest way to buy
```