In [ ]:
```
!pip install pypdf --quiet
```

In [ ]:
```
from langchain.chains import ConversationalRetrievalChain
from langchain.document_loaders import TextLoader, PyPDFLoader
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.llms import OpenAI
from langchain.memory import ConversationBufferMemory
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import Chroma
import glob
```

Step-1 :Data Importing

In [ ]:
```
#Example of ConversationalRetrievalChain

#Document Loading
!wget https://raw.githubusercontent.com/venkatareddykonasani/Datasets/master/C
#Unzip and Overwrite
!unzip -o All_Cibil_Docs.zip -d All_Docs_Set1
#listdown all documents content/All_Docs_Set1 and store them
pdf_files = glob.glob("All_Docs_Set1/*.pdf")
print(pdf_files)
```

In [ ]:
```
full_text = ""
for pdf_file in pdf_files:
    loader = PyPDFLoader(pdf_file)
    pages = loader.load()
    print(pdf_file, len(pages))
    for page in pages:
        full_text += page.page_content

print(full_text[1:1000])
print("Lines" , len(full_text.split("\n")))
print("Words" , len(full_text.split(" ")))
print("Charecters", len(full_text))
```

Step-2:Split the data into Chunks

In [ ]:
```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=300, chunk_overlap=0
chunks = text_splitter.split_text(full_text)
print(len(chunks))
print(chunks[0])
```

Step-3: Creating embeddings and Storing in Vector Stores

In [ ]:
```
!pip install ChromaDB --quiet
```

In [ ]:
```
embeddings = OpenAIEmbeddings()
cibil_db=Chroma.from_texts(chunks,
                                embeddings,
                                persist_directory="cibil_db")
cibil_db.persist()
```

In [ ]:
```
retriever = cibil_db.as_retriever()
result=retriever.get_relevant_documents(query="What is the CIBIL Score?")
for i in range(len(result)):
  print(result[i].page_content)
```

Step-4: Conversation and Retrieval Chain

In [ ]:
```
from langchain.chat_models import ChatOpenAI, cohere
from langchain.llms import Cohere
```

In [ ]:
```
llm= ChatOpenAI(temperature=0, max_tokens=512)
#llm=Cohere(temperature=0, max_tokens=512)
memory = ConversationBufferMemory(memory_key='chat_history', return_messages=T
conversational_RAG = ConversationalRetrievalChain.from_llm(llm=llm,
                                                           retriever=cibil_db.as_ret
                                                           memory=memory,
                                                           #verbose=True
                                                           )
```

Step-5 : Conversation

In [ ]:
```
user_input=input("Your message:")
while user_input!="quit":
    response=conversational_RAG({"question": user_input})
    print("AI message ==>", response["answer"])
    user_input=input("Your message: ")
```

RAG ChatBotTool

In [ ]:
```
%%writefile requirements.txt
langchain
langchain_community
PyPDF2
python-dotenv
streamlit
openai
faiss-cpu
altair
tiktoken
huggingface-hub
InstructorEmbedding
```

```
sentence-transformers
pydantic
```

In [ ]:
```
!pip install -r requirements.txt --quiet
```

In [ ]:
```
%%writefile app.py

import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import CharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chat_models import ChatOpenAI
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory


def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PdfReader(pdf)
        for page in pdf_reader.pages:
            text += page.extract_text()
    return text

def get_text_chunks(text):
  text_splitter = CharacterTextSplitter(
      separator="\n",
      chunk_size=1000,
      chunk_overlap=200,
      length_function=len
  )
  chunks = text_splitter.split_text(text)
  return chunks

def get_embeddings(text_chunks):
  embeddings = OpenAIEmbeddings()
  return embeddings

def get_convesrational_chain(vectorstore):
  llm = ChatOpenAI(temperature=0)
  memory = ConversationBufferMemory(memory_key='chat_history', return_messages
  convesrational_chain = ConversationalRetrievalChain.from_llm(
      llm=llm,
      retriever=vectorstore.as_retriever(),
      memory=memory
  )
  return convesrational_chain


def handle_userinput(user_question):
  response = st.session_state.convesrational_chain({"question": user_question}
  st.session_state.chat_history = response['chat_history']
```

```
  for i, message in enumerate(st.session_state.chat_history):
    if i % 2 == 0 :
      st.write(':man_in_tuxedo:', message.content)
    else:
      st.write(':robot_face:', message.content)

def main():

    st.set_page_config(page_title="Chat with Documents", page_icon=":books:")
    styl = f"""
      <style>
          .stTextInput {{
            position: fixed;
            bottom: 3rem;
          }}
      </style>
      """
    st.markdown(styl, unsafe_allow_html=True)

    #st.write(css, unsafe_allow_html=True)
    st.title("AI Chat-Bot for Your Documents :books:")

    if "convesrational_chain" not in st.session_state:
        st.session_state.convesrational_chain = None
    if "chat_history" not in st.session_state:
        st.session_state.chat_history = None


    user_input=st.text_input("Upload files, Hit Submit button, then ask questi
    if user_input:
      handle_userinput(user_input)


    with st.sidebar:
        pdf_docs=st.file_uploader("Upload your documents here" , accept_multip
        if st.button("Submit"):
          with st.spinner("Processing..."):

            # Get PDF Data
            raw_text=get_pdf_text(pdf_docs)

            #Divide the Data into Chunks
            chunked_array=get_text_chunks(raw_text)

            #Create embeddings
            embeddings=get_embeddings(chunked_array)

            #Create vectorstore
            vectorstore=FAISS.from_texts(chunked_array,embeddings)

            #crate a converstaion chain
            st.session_state.convesrational_chain=get_convesrational_chain(vec
```

```
if __name__ == "__main__":
    main()
```

In [ ]:
```
!streamlit run app.py & npx localtunnel --port 8501 & curl ipv4.icanhazip.com
#Doesn't work in Chrome sometimes - Due to CSS related issues; Try Forefox
```