import os

from mistralai import Mistral

from dotenv import load\_dotenv

from qdrant\_client.models import PointStruct

from qdrant\_client.models import Distance, VectorParams

from qdrant\_client import QdrantClient

Import required

load\_dotenv()

qdrant\_client = QdrantClient(":memory:")

Loading environment variables and setting up qdrant location

api\_key = os.getenv("MISTRALAI\_API\_KEY")

client = Mistral(api\_key=api\_key)

Initializing api setup for mistral

def generate\_context(url):

model = "pixtral-12b-2409"

messages = [

{

"role": "user",

"content": [

{

"type": "text",

"text": "Extract the text from the image precisely, extract every text."

},

{

"type": "image\_url",

"image\_url": url

}

]

}

]

*# Get the chat response*

chat\_response = client.chat.complete(

model=model,

messages=messages

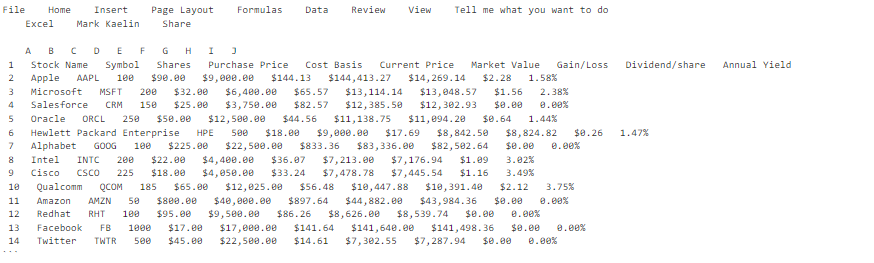
)

*# return the context*

return chat\_response.choices[0].message.content

Generate context from the given image using **pixtral-12b-2409** model. Context will then be used to feed into the Qdrant database.

**Context Generated** :



def embed(context, model = "mistral-embed"):

embeddings\_batch\_response = client.embeddings.create(

model=model,

inputs= context

)

return embeddings\_batch\_response

Used to generate embeddings and return the batch of embeddings.

def generate\_embeddings(context: str):

model = "mistral-embed"

context = context.split('\n')

data = []

client = Mistral(api\_key=api\_key)

embeddings\_batch\_response = embed(context, model)

for i in range(len(embeddings\_batch\_response.data)):

temp = []

temp.append(context[i])

temp.append(embeddings\_batch\_response.data[i].embedding)

data.append(temp)

return data

Prepare the embedding the the required format of list of [raw\_context, embeddings]

def initialize\_qdrant(length: int):

vector\_size = length

*# Define the vectors configuration*

vector\_params = VectorParams(

size=vector\_size, *# Size of the vectors*

distance=Distance.COSINE *# Choose distance metric (COSINE, EUCLID, or IP)*

)

*# Create the collection with the specified configuration*

if "CHATBOT" not in qdrant\_client.get\_collections().collections[0].name:

qdrant\_client.create\_collection(

collection\_name="CHATBOT",

vectors\_config=vector\_params *# Specify vector configuration*

)

Initialise the Qdrant collection for continuous use, collection used is “CHATBOT”.

def qdrant\_entry(final\_data):

points=[PointStruct( id=i, vector=final\_data[i][1],payload={'raw\_context':final\_data[i][0] }) for i in range(len(final\_data))]

qdrant\_client.upsert(collection\_name="CHATBOT", points=points)

print(qdrant\_client.get\_collections())

Enter the context and final\_data earlier derived in Qdrant database in form of Points.

def query\_qdrant(query, collection\_name='CHATBOT', limit=4):

query\_vector=embed([query]).data[0].embedding

result = qdrant\_client.search(

collection\_name = collection\_name,

query\_vector=query\_vector,

limit = limit,

with\_vectors = False

)

*# search\_result=[]*

return result

Query from Qdrant database the top 4 relevant element in the database with a given query,

Firstly convert the query into embedding and then push the same in query\_quadrant usecase.

def prepare\_llm\_context(result):

*# result[0].payload['raw\_context']*

context =[]

for i in range(len(result)):

context.append(result[i].payload['raw\_context'])

return context

From the response of query\_qdrant use it to form a context which will then be provided to mistral text responder.

if \_\_name\_\_ == "\_\_main\_\_":

choice = input("Enter 'query' for rag query \n 'entry' for rag entry")

if choice == 'entry':

url: str = "https://assets.techrepublic.com/uploads/2017/04/aexcelpowerbi.png"

context = generate\_context(url)

final\_data = generate\_embeddings(context)

*# final\_data[1][1]--->size*

initialize\_qdrant(len(final\_data[0][1]))

qdrant\_entry(final\_data)

else:

query = input("Enter the query : ")

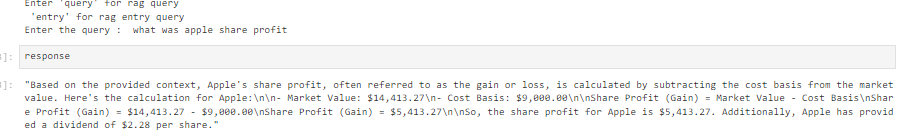
result = query\_qdrant(query)

context = prepare\_llm\_context(result)

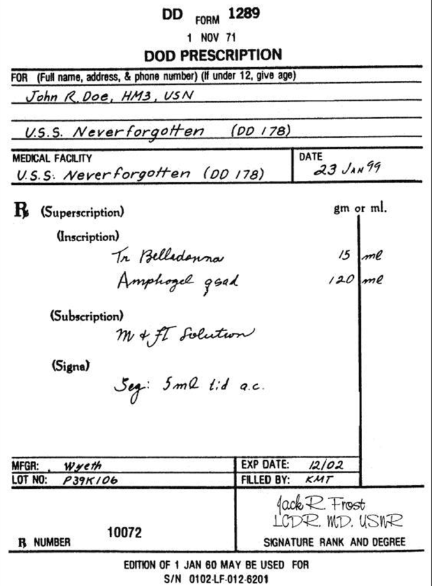
response = query\_llm(context, query)

Main control of the program, use it according to need of entry of more data or make a query over the existing data, change of url is required as per the entry changes.

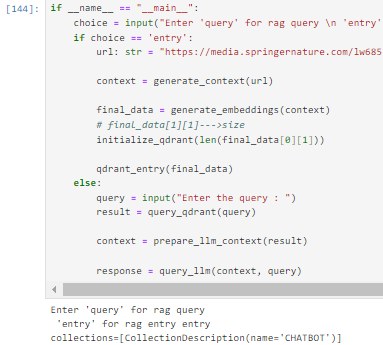
Output :



**Bill 1** :

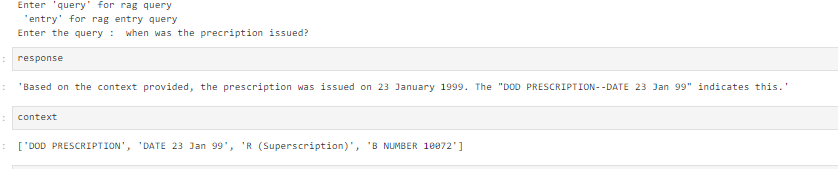


**Output :**

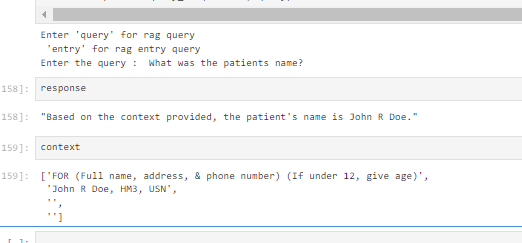


Parsed bill :

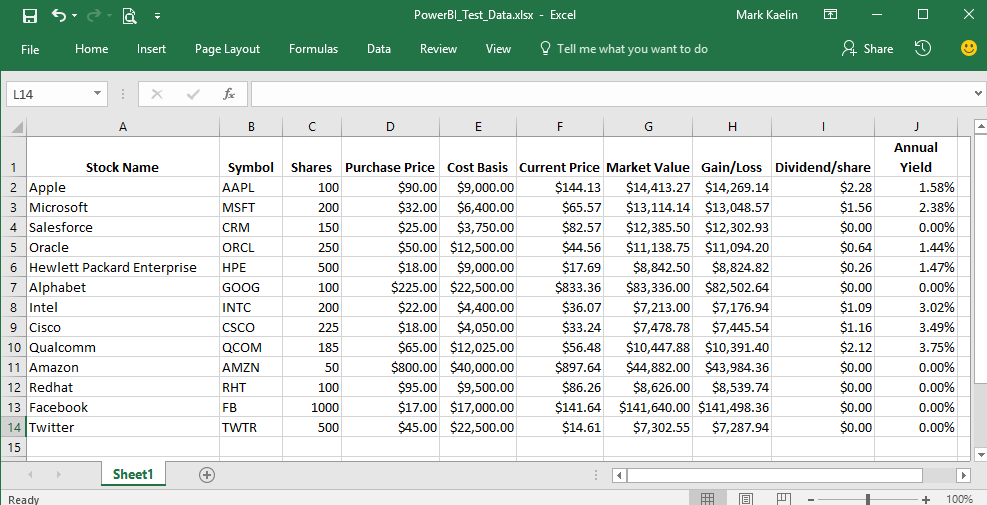




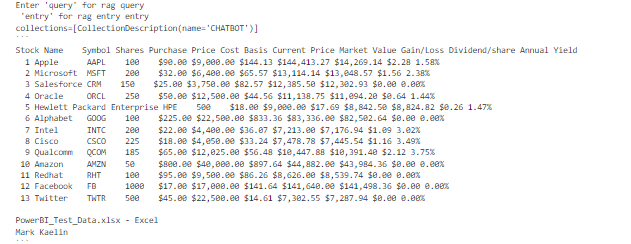
'Based on the context provided, the prescription was issued on 23 January 1999. The "DOD PRESCRIPTION--DATE 23 Jan 99" indicates this.'



**Bill 2 :**

****

**Entry Outputs:**

****