Develop an embedding model for a provided dataset sample.

Steps for developing an embedding model that facilitate querying the dataset using both textual and numerical descriptors utilizing Retreival Augmented Generation (RAG):

1. Loading the csv dataset
2. Extract only 4 columns namely: Order_date Days_from_order_to_shipment, Product_Category and Status
3. Split the dataset into chunks. Chunks are made for each row.
4. Embedding the chunked data into vectors using Sentence transformer's embedding model
5. Store the embedded data in FAISS database.
6. Prompt Engineering
7. Build a RAG Chain: Integrate our LLM with our FAISS retriever and put it all together using Langchain
8. Generate the response using RAG chain.

GPU Utilized: Nvidia L4 GPU

Importing necessary libraries

In [ ]:

```
!pip install -q -U torch datasets transformers tensorflow langchain sentence_transformers faiss-cpu
!pip install -q accelerate==0.21.0 peft==0.4.0 bitsandbytes==0.40.2 trl==0.4.7
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 779.1/779.1 MB 1.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 542.0/542.0 kB 56.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.0/9.0 MB 104.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 589.8/589.8 MB 1.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 817.7/817.7 kB 67.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 171.5/171.5 kB 23.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.0/27.0 MB 59.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 176.2/176.2 MB 9.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 168.1/168.1 MB 5.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 16.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 26.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 20.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 388.9/388.9 kB 45.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.3/5.3 MB 105.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 88.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 111.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 72.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 93.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 299.3/299.3 kB 33.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.5/115.5 kB 16.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.4/49.4 kB 7.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 311.2/311.2 kB 35.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.0/53.0 kB 8.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 141.1/141.1 kB 20.3 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
fastai 2.7.14 requires torch<2.3,>=1.10, but you have torch 2.3.0 which is incompatible.
tf-keras 2.15.1 requires tensorflow<2.16,>=2.15, but you have tensorflow 2.16.1 which is incompatible.
torchaudio 2.2.1+cu121 requires torch==2.2.1, but you have torch 2.3.0 which is incompatible.
torchtext 0.17.1 requires torch==2.2.1, but you have torch 2.3.0 which is incompatible.
torchvision 0.17.1+cu121 requires torch==2.2.1, but you have torch 2.3.0 which is incompatible.
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 244.2/244.2 kB 5.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.9/72.9 kB 10.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 92.5/92.5 MB 16.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.4/77.4 kB 12.4 MB/s eta 0:00:00
```

Dependencies

In [ ]:

```
import os
import torch
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    pipeline
)
from datasets import load_dataset
from peft import LoraConfig, PeftModel

from langchain.text_splitter import CharacterTextSplitter



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

from langchain.prompts import PromptTemplate
from langchain.schema.runnable import RunnablePassthrough
from langchain.llms import HuggingFacePipeline
from langchain.chains import LLMChain
import pandas as pd
```

Log in to hugging face to get access to load LLM(Mistral-7b) using Access Token

In [ ]:

```
from huggingface_hub import login
login()
```

```
VBox(children=(HTML(value='<center> <img\nsrc=https://huggingface.co/front/assets/huggingface_logo-noborder.sv…
```

Load Model and Tokenizer (Mistral 7b-Instruct-v0.2)

In [ ]:

```
model_name='mistralai/Mistral-7B-Instruct-v0.2'

tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: 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/setting
s/tokens), set it as secret in your Google Colab and restart your session.
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(

tokenizer_config.json:   0%|          | 0.00/1.46k [00:00<?, ?B/s]

tokenizer.model:   0%|          | 0.00/493k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/1.80M [00:00<?, ?B/s]

special_tokens_map.json:   0%|          | 0.00/72.0 [00:00<?, ?B/s]
```

Activate 4-bit precision base model loading and set up quantization config: Model Quantization is a technique used to reduce the size of large neural networks, including large language models (LLMs) by modifying the precision of their weights.

In [ ]:

```python
use_4bit = True

# Compute dtype for 4-bit base models
bnb_4bit_compute_dtype = "float16"

# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4-bit base models (double quantization)
use_nested_quant = False


compute_dtype = getattr(torch, bnb_4bit_compute_dtype)

bnb_config = BitsAndBytesConfig(
    load_in_4bit=use_4bit,
    bnb_4bit_quant_type=bnb_4bit_quant_type,
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=use_nested_quant,
)

# Check GPU compatibility with bfloat16
if compute_dtype == torch.float16 and use_4bit:
    major, _ = torch.cuda.get_device_capability()
    if major >= 8:
        print("=" * 80)
        print("Your GPU supports bfloat16: accelerate training with bf16=True")
        print("=" * 80)
```

```
================================================================================
Your GPU supports bfloat16: accelerate training with bf16=True
================================================================================
```

Load quantized Mistal 7B

In [ ]:

```python
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
)
```

```
config.json:   0%|          | 0.00/596 [00:00<?, ?B/s]

`low_cpu_mem_usage` was None, now set to True since model is quantized.

model.safetensors.index.json:   0%|          | 0.00/25.1k [00:00<?, ?B/s]

Downloading shards:   0%|          | 0/3 [00:00<?, ?it/s]

model-00001-of-00003.safetensors:   0%|          | 0.00/4.94G [00:00<?, ?B/s]

model-00002-of-00003.safetensors:   0%|          | 0.00/5.00G [00:00<?, ?B/s]

model-00003-of-00003.safetensors:   0%|          | 0.00/4.54G [00:00<?, ?B/s]

Loading checkpoint shards:   0%|          | 0/3 [00:00<?, ?it/s]

generation_config.json:   0%|          | 0.00/111 [00:00<?, ?B/s]

You are calling `save_pretrained` to a 4-bit converted model, but your `bitsandbytes` version doesn't support i
t. If you want to save 4-bit models, make sure to have `bitsandbytes>=0.41.3` installed.
```

Type *Markdown* and LaTeX: α^2

Count number of trainable parameters

In [ ]:

```python
def print_number_of_trainable_model_parameters(model):
    trainable_model_params = 0
    all_model_params = 0
    for _, param in model.named_parameters():
        all_model_params += param.numel()
        if param.requires_grad:
            trainable_model_params += param.numel()
    return f"trainable model parameters: {trainable_model_params}\nall model parameters: {all_model_params}\nper

print(print_number_of_trainable_model_parameters(model))
```

```
trainable model parameters: 262410240
all model parameters: 3752071168
percentage of trainable model parameters: 6.99%
```

Build Mistral text generation pipeline: Utilize temperature, repetition penalty parameters for better response and max_new_tokens to determine the length of tokens

In [ ]:
```
text_generation_pipeline = pipeline(
    model=model,
    tokenizer=tokenizer,
    task="text-generation",
    temperature=0.1,
    repetition_penalty=1.1,
    return_full_text=True,
    max_new_tokens=1000,
)
mistral_llm = HuggingFacePipeline(pipeline=text_generation_pipeline)
```

###Load and filter data

In [ ]:
```
import pandas as pd

# Load the CSV data using pandas
file_path = "/content/new_file.csv"
data = pd.read_csv(file_path)

# Select only the desired columns
selected_columns = ['Order_date', 'Days_from_order_to_shipment', 'Product_Category', 'Status']
filtered_data = data[selected_columns]

# Save the filtered data to a new CSV file
output_file_path = "/content/filtered_data.csv"
filtered_data.to_csv(output_file_path, index=False)
```

In [ ]:
```
# Print the first few rows of the filtered data
filtered_data.head()
```

Out[10]:

	Order_date	Days_from_order_to_shipment	Product_Category	Status
0	2023-07-01	0	Apparel	Order received today
1	2023-07-01	0	Apparel	Preparing for Shipment
2	2023-07-01	2	Apparel	Order has been shipped today
3	2023-07-01	0	Apparel	Delivered today
4	2023-07-01	0	Cosmetics & Personal Care	Order received today

Load filtered data using CSVLoader from langchain.document_loaders.csv_loader for easy splitting into small chunks

In [ ]:
```
from langchain.document_loaders.csv_loader import CSVLoader


loader = CSVLoader(file_path= "/content/filtered_data.csv")
data = loader.load()
```

###Show data[0] to know how what data is in the first row

In [ ]:
```
data[0]
```

Out[12]:
```
Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Apparel\nStatu
s: Order received today', metadata={'source': '/content/filtered_data.csv', 'row': 0})
```

Load and chunk documents. The most common approach to chunking is to define a fixed size of chunks and whether there should be any overlap between them.

In [ ]:
```
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,  # Adjusted to a larger chunk size for more context
    chunk_overlap=0,
    length_function=len
)

chunked_documents = text_splitter.transform_documents(data)
len(chunked_documents)
```

Out[13]:
```
100
```

In [ ]:
```
chunked_documents[0]
```

Out[14]:
```
Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Apparel\nStatu
s: Order received today', metadata={'source': '/content/filtered_data.csv', 'row': 0})
```

Load embedding model (all-MiniLM-L6-v2 model) to generate embeddings of the chunked data and store in vector store using FAISS (Facebook AI Similarity Search)

In [ ]:
```
from langchain.embeddings import CacheBackedEmbeddings, HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
from langchain.storage import LocalFileStore

store = LocalFileStore("./cache/")

embed_model_id = 'sentence-transformers/all-MiniLM-L6-v2'

core_embeddings_model = HuggingFaceEmbeddings(
    model_name=embed_model_id
)

embedder = CacheBackedEmbeddings.from_bytes_store(
    core_embeddings_model, store, namespace=embed_model_id
)

vector_store = FAISS.from_documents(chunked_documents, embedder)
```

```
modules.json:   0%|          | 0.00/349 [00:00<?, ?B/s]

config_sentence_transformers.json:   0%|          | 0.00/116 [00:00<?, ?B/s]

README.md:   0%|          | 0.00/10.7k [00:00<?, ?B/s]

sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/612 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/90.9M [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/350 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]

1_Pooling/config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]
```

Store all the embedded vector data to "retreiver" variable and set k=25. When a query is provided to the LLM, top 25 relevent documents that has the highest similarity with the query are retrieved from the retreiver.

In [ ]:
```
retriever = vector_store.as_retriever(search_kwargs={"k": 25})
```

Create PromptTemplate: Prompt is provided with proper instructions so that the model can understand what type of response we need exactly. Here 2- shot prompting is employed by providing 2 examples of how we want the response to be generated.

In [47]:

```
prompt_template = """
### [INST] Instruction: You are an AI Assistant. You are given the user question. Analyze the dataset to identify

- Treat the question as case-insensitive.
- Handle potential synonyms and common variations in the input text.
- Include capabilities to process both textual and numerical data.
- Ensure the output format strictly adheres JSON structure followed by "Generataed Output".
- Pay special attention to ensuring that the row ids returned are exact matches or closest matches based on the c

For example for the input "Apparel Products", the output should be like:

Generated Output: "column_name": "Product_Category", "value": "Apparel", "row_ids": row0, row1, row2,row3,row15,r

Another example for the input "Cosmetics", the output should be like:
Generated Output: "column_name": "Product_Category", "value": "Cosmetics & Personal Care", "row_ids": row5, row6,
The output should be in exactly the same format as above provided examples.
{context}

### QUESTION:
{question} [/INST]
 """
```

Create a prompt template taking context and question as input variables and llm chain using langchain framework. Langchain is used to connect different components with LLM to create workflows.

In [48]:

```
# Create prompt from prompt template
prompt = PromptTemplate(
    input_variables=["context", "question"],
    template=prompt_template,
)

# Create llm chain
llm_chain = LLMChain(llm=mistral_llm, prompt=prompt)
```

##Create a RAG Chain: **Now the context will be the information extracted from the retreiver. We need to combine the llm_chain with the retriever to create a RAG chain. Input = "Electronics"**

In [49]:

```
rag_chain = (
 {"context": retriever, "question": RunnablePassthrough()}
    | llm_chain
)

result = rag_chain.invoke("Electronics")
```

```
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:492: UserWarning: `do_sa
mple` is set to `False`. However, `temperature` is set to `0.1` -- this flag is only used in sample-based gener
ation modes. You should set `do_sample=True` or unset `temperature`.
  warnings.warn(
```

Simple response generated for the query "Electronics"

In [50]:
```
print(result['text'])
```

```
### [INST] Instruction: You are an AI Assistant. You are given the user question. Analyze the dataset to identi
fy the most relevant column and the corresponding value of the question given. Then, search the dataset to find
all rows where the question matches the value exactly or semantically. Output the results in a JSON format incl
uding the most relevant column name of the question, the value, and the IDs of all matching rows.

- Treat the question as case-insensitive.
- Handle potential synonyms and common variations in the input text.
- Include capabilities to process both textual and numerical data.
- Ensure the output format strictly adheres JSON structure followed by "Generataed Output".
- Pay special attention to ensuring that the row ids returned are exact matches or closest matches based on the
dataset analysis.

For example for the input "Apparel Products", the output should be like:

Generated Output: "column_name": "Product_Category", "value": "Apparel", "row_ids": row0, row1, row2,row3,row1
5,row16,row17,row18,row19,row20

Another example for the input "Cosmetics", the output should be like:
Generated Output: "column_name": "Product_Category", "value": "Cosmetics & Personal Care", "row_ids": row5, row
6, row7, row8, row9
The output should be in exactly the same format as above provided examples.
[Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Electronics\n
Status: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 92}), Document(page_content='Orde
r_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Electronics\nStatus: In transit', metadat
a={'source': '/content/filtered_data.csv', 'row': 93}), Document(page_content='Order_date: 2023-07-01\nDays_fro
m_order_to_shipment: 0\nProduct_Category: Electronics\nStatus: In transit', metadata={'source': '/content/filte
red_data.csv', 'row': 94}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProd
uct_Category: Electronics\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 95}),
Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 2\nProduct_Category: Electronics\nS
tatus: Order has been shipped today', metadata={'source': '/content/filtered_data.csv', 'row': 91}), Document(p
age_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Electronics\nStatus: Del
ivered today', metadata={'source': '/content/filtered_data.csv', 'row': 96}), Document(page_content='Order_dat
e: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Electronics\nStatus: Order received today', me
tadata={'source': '/content/filtered_data.csv', 'row': 89}), Document(page_content='Order_date: 2023-07-01\nDay
s_from_order_to_shipment: 0\nProduct_Category: Electronics\nStatus: Preparing for Shipment', metadata={'sourc
e': '/content/filtered_data.csv', 'row': 90}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_t
o_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_dat
a.csv', 'row': 42}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Cat
egory: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 43}), Docume
nt(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatu
s: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 68}), Document(page_content='Order_dat
e: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata=
{'source': '/content/filtered_data.csv', 'row': 69}), Document(page_content='Order_date: 2023-07-01\nDays_from_
order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filter
ed_data.csv', 'row': 70}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProdu
ct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 71}),
Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\n
Status: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 72}), Document(page_content='Orde
r_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metada
ta={'source': '/content/filtered_data.csv', 'row': 73}), Document(page_content='Order_date: 2023-07-01\nDays_fr
om_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/fil
tered_data.csv', 'row': 74}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nPr
oduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 7
5}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Ga
mes\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 76}), Document(page_content
='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit',
metadata={'source': '/content/filtered_data.csv', 'row': 77}), Document(page_content='Order_date: 2023-07-01\nD
ays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/conte
nt/filtered_data.csv', 'row': 78}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment:
0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'ro
w': 79}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys
& Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 80}), Document(page_cont
ent='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: Delivered
today', metadata={'source': '/content/filtered_data.csv', 'row': 44}), Document(page_content='Order_date: 2023-
07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: Delivered today', metadata={'sou
rce': '/content/filtered_data.csv', 'row': 81})]

### QUESTION:
Electronics [/INST]
  Generated Output: {
    "column_name": "Product_Category",
    "value": "Electronics",
    "row_ids": ["91", "92", "93", "94", "95"]
  }
```

In [51]:

```
import time


# Sample query
query = "Toys"

# Start timing
start_time = time.time()

# Generate response using the rag_chain
rag_chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | llm_chain
)

result = rag_chain.invoke(query)

# Stop timing
end_time = time.time()

# Calculate elapsed time
elapsed_time = end_time - start_time

# Extract the response text
response_text = result['text']

# Count tokens in the response
num_tokens = len(tokenizer.tokenize(response_text))

# Calculate tokens per second
tokens_per_second = num_tokens / elapsed_time

# Output the results
print(f"Response: {response_text}")
print(f"Tokens per Second: {tokens_per_second}")
```

```
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:492: UserWarning: `do_sa
mple` is set to `False`. However, `temperature` is set to `0.1` -- this flag is only used in sample-based gener
ation modes. You should set `do_sample=True` or unset `temperature`.
  warnings.warn(
```

```
Response:
### [INST] Instruction: You are an AI Assistant. You are given the user question. Analyze the dataset to identi
fy the most relevant column and the corresponding value of the question given. Then, search the dataset to find
all rows where the question matches the value exactly or semantically. Output the results in a JSON format incl
uding the most relevant column name of the question, the value, and the IDs of all matching rows.

- Treat the question as case-insensitive.
- Handle potential synonyms and common variations in the input text.
- Include capabilities to process both textual and numerical data.
- Ensure the output format strictly adheres JSON structure followed by "Generataed Output".
- Pay special attention to ensuring that the row ids returned are exact matches or closest matches based on the
dataset analysis.

For example for the input "Apparel Products", the output should be like:

Generated Output: "column_name": "Product_Category", "value": "Apparel", "row_ids": row0, row1, row2,row3,row1
5,row16,row17,row18,row19,row20

Another example for the input "Cosmetics", the output should be like:
Generated Output: "column_name": "Product_Category", "value": "Cosmetics & Personal Care", "row_ids": row5, row
6, row7, row8, row9
The output should be in exactly the same format as above provided examples.
[Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games
\nStatus: Delivered today', metadata={'source': '/content/filtered_data.csv', 'row': 44}), Document(page_conten
t='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: Delivered to
day', metadata={'source': '/content/filtered_data.csv', 'row': 81}), Document(page_content='Order_date: 2023-07
-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source':
'/content/filtered_data.csv', 'row': 42}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_sh
ipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.cs
v', 'row': 43}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Categor
y: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 68}), Document(p
age_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In
transit', metadata={'source': '/content/filtered_data.csv', 'row': 69}), Document(page_content='Order_date: 202
3-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'sourc
e': '/content/filtered_data.csv', 'row': 70}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_t
o_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_dat
a.csv', 'row': 71}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Cat
egory: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 72}), Docume
nt(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatu
s: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 73}), Document(page_content='Order_dat
e: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata=
{'source': '/content/filtered_data.csv', 'row': 74}), Document(page_content='Order_date: 2023-07-01\nDays_from_
order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filter
ed_data.csv', 'row': 75}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProdu
ct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 76}),
Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\n
Status: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 77}), Document(page_content='Orde
r_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metada
ta={'source': '/content/filtered_data.csv', 'row': 78}), Document(page_content='Order_date: 2023-07-01\nDays_fr
om_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/fil
tered_data.csv', 'row': 79}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nPr
oduct_Category: Toys & Games\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 8
0}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Ga
mes\nStatus: Order received today', metadata={'source': '/content/filtered_data.csv', 'row': 39}), Document(pag
e_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: Orde
r received today', metadata={'source': '/content/filtered_data.csv', 'row': 65}), Document(page_content='Order_
date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: Preparing for Shipmen
t', metadata={'source': '/content/filtered_data.csv', 'row': 40}), Document(page_content='Order_date: 2023-07-0
1\nDays_from_order_to_shipment: 0\nProduct_Category: Toys & Games\nStatus: Preparing for Shipment', metadata=
{'source': '/content/filtered_data.csv', 'row': 66}), Document(page_content='Order_date: 2023-07-01\nDays_from_
order_to_shipment: 2\nProduct_Category: Toys & Games\nStatus: Order has been shipped today', metadata={'sourc
e': '/content/filtered_data.csv', 'row': 41}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_t
o_shipment: 2\nProduct_Category: Toys & Games\nStatus: Order has been shipped today', metadata={'source': '/con
tent/filtered_data.csv', 'row': 67}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipmen
t: 0\nProduct_Category: Apparel\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row':
17}), Document(page_content='Order_date: 2023-07-01\nDays_from_order_to_shipment: 0\nProduct_Category: Apparel
\nStatus: In transit', metadata={'source': '/content/filtered_data.csv', 'row': 18})]

### QUESTION:
Toys [/INST]
  Generated Output: {
    "column_name": "Product_Category",
    "value": "Toys & Games",
    "row_ids": ["44", "81", "42", "43", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79",
"80", "39", "65"]
  }
Tokens per Second: 326.7716270378078
```

As we can see, the generated response contain multiple sentences that are not the requirement for us. We only want column name, value and row id in a json format in our response. So, I have stored the whole response in json file and extract only required objects from the json output.

In [52]:

```
import json

# Assuming result['text'] is the text you want to save
data = {'text': result['text']}  # Store it in a dictionary if it's not already

# Specify the filename where you want to store the output
filename = 'output2.json'

# Open the file in write mode and write the JSON data
with open(filename, 'w') as file:
    json.dump(data, file)

print("Data has been saved to", filename)
```

```
Data has been saved to output2.json
```

Save the final response generated in json file and print the final result.

In [53]:

```
import json
import re

def clean_data(data):
    # Normalize the data by removing newlines and excessive whitespace
    return ' '.join(data.split())

def extract_generated_output(data):
    # Simplify the data for easier regex handling
    cleaned_data = clean_data(data)
    # print("Cleaned Data Sample:", cleaned_data[:500])  # Show a sample of the cleaned data

    # Regex pattern to find the JSON object following "Generated Output:"
    # pattern = r'### Question: (\{.*?\})'
    pattern = r'Generated Output: (\{.*?\})'
    match = re.search(pattern, cleaned_data)
    if match:
        # Convert the extracted string to a valid JSON object
        json_output = json.loads(match.group(1).replace("'", '"'))  # Ensure double quotes for JSON
        return json_output
    else:
        return None

def load_json(filename):
    with open(filename, 'r') as file:
        data = json.load(file)
        return data['text']

def save_json(data, filename):
    with open(filename, 'w') as file:
        json.dump(data, file, indent=4)

# Load the content from the JSON file
content = load_json('/content/output2.json')

# Extract the 'Generated Output' part
extracted_data = extract_generated_output(content)

# Save the extracted data to a new JSON file if it was found
if extracted_data:
    save_json(extracted_data, 'extracted_output2.json')

    print(json.dumps(extracted_data, indent=4))
else:
    print("No generated output found.")
```


```
{
    "column_name": "Product_Category",
    "value": "Toys & Games",
    "row_ids": [
        "44",
        "81",
        "42",
        "43",
        "68",
        "69",
        "70",
        "71",
        "72",
        "73",
        "74",
        "75",
        "76",
        "77",
        "78",
        "79",
        "80",
        "39",
        "65"
    ]
}
```


Conclusion: As we can see, we have acheived the result in the format: column_name, value, row_id as per our requirement.