PLEASE NOTE:

We already have a very powerful product estimator with our proprietary, fine-tuned LLM. Most people would be very satisfied with that! The main reason we're adding these extra steps is to deepen your expertise with RAG and with Agentic workflows.

Step 1: Import necessary libraries. These include standard Python packages, numerical computing (numpy), data handling (pickle, datasets), Chroma DB for vector storage, HuggingFace tools, and embedding models.

[ ]:
```
# imports
import os
import re
import math
import json
from tqdm import tqdm
import random
from dotenv import load_dotenv
from huggingface_hub import login
import numpy as np
import pickle
from sentence_transformers import SentenceTransformer, util
from datasets import load_dataset
import chromadb
from sklearn.manifold import TSNE
import plotly.graph_objects as go
```

Step 2: Setup environment variables and database path for API keys and Chroma storage.

[ ]:
```
# environment setup
load_dotenv(override=True)
os.environ['OPENAI_API_KEY'] = os.environ.get('OPENAI_API_KEY',␣
↪'your-key-if-not-using-env')
os.environ['HF_TOKEN'] = os.environ.get('HF_TOKEN', 'your-key-if-not-using-env')
DB = 'products_vectorstore'
```

Step 3: Log in to HuggingFace hub using the token to access models and datasets.

[ ]:
```
# HuggingFace login
hf_token = os.environ['HF_TOKEN']
login(hf_token, add_to_git_credential=True)
```

Step 4: Load preprocessed training data from pickle files.

[ ]:
```
# Load training data
with open('../week6/train.pkl', 'rb') as f:
    train = pickle.load(f)
```

Step 5: Initialize Chroma persistent client, delete existing collection if any, and create a new collection for product vectors.

[ ]:
```
# Create Chroma datastore
client = chromadb.PersistentClient(path=DB)
if 'products' in [c.name for c in client.list_collections()]:
    client.delete_collection('products')
collection = client.create_collection('products')
```

Step 6: Load sentence transformer embedding model for vectorizing product descriptions.

[ ]:
```
# Initialize embedding model
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
```

Step 7: Define a helper function to extract product descriptions from the training data.

[ ]:
```
# Prepare text for vectorization
def description(item):
    return item.prompt.replace('How much does this cost to the nearest dollar?\n\n', '').split('\n\nPrice is $')[0]
```

Step 8: Loop through training data, vectorize descriptions, and add them to the Chroma collection.

[ ]:
```
# Populate Chroma collection
NUMBER_OF_DOCUMENTS = len(train)
for i in range(0, NUMBER_OF_DOCUMENTS, 1000):
    docs = [description(item) for item in train[i:i+1000]]
    vectors = model.encode(docs).astype(float).tolist()
    metas = [{'category': item.category, 'price': item.price} for item in train[i:i+1000]]
    ids = [f'doc_j' for j in range(i, i+len(docs))]
    collection.add(ids=ids, documents=docs, embeddings=vectors, metadatas=metas)
```